



VisionLabs LUNA PLATFORM 5

Administrator manual

v.5.67.0

Contents

Glossary	69
1 Introduction	72
2 General concepts	73
2.1 Services	73
2.1.1 General services	73
2.1.2 Additional services	74
2.1.3 Third-party services	76
2.2 Authorization system	77
2.3 Approaches at work	79
2.3.1 Parallel performing of requests	79
2.3.2 Sequential performing of requests	81
2.4 Detection	83
2.4.0.1 Image format requirements	83
2.4.1 Face detection and estimation process	83
2.4.2 Body detection and estimation process	85
2.4.3 Services interaction during detection	86
2.4.4 Requests to perform detection	86
2.4.4.1 Requests “create handler” and “generate events”	87
2.4.4.2 Request “detect faces”	87
2.4.4.3 Request “sdk”	88
2.4.5 Requests to perform estimation	88
2.4.6 Detection results	88
2.4.7 Redetection	88
2.5 Extraction	90
2.5.1 Extraction process	90
2.5.2 Specifics of extracting data from faces	91
2.5.2.1 Temporary attributes	91
2.5.3 Specifics of extracting data from bodies	92
2.5.4 Services interaction during extraction	92
2.5.5 Requests to perform extraction	93
2.5.5.1 Requests “create handler” and “generate events”	93
2.5.5.2 Request “extract attributes”	94
2.5.5.3 Request “sdk”	95
2.5.6 Extraction results	95
2.6 Matching	96
2.6.1 Matching process	97

2.6.2	Service interaction during matching	97
2.6.3	Filtering of matching results	98
2.6.4	Requests to perform matching	99
2.6.4.1	Requests “create handler” and “generate events”	99
2.6.4.2	Requests “matching faces” and “matching bodies”	99
2.6.4.3	Request “raw matching”	100
2.6.5	Matching results	100
2.6.6	Verification	100
2.6.7	Matching a large set of descriptors	100
2.7	Stored data and LP objects	102
2.7.1	Source images	102
2.7.1.1	Source images usage	103
2.7.1.2	Source images saving	103
2.7.1.3	Source images deletion	103
2.7.2	Sample object	104
2.7.2.1	Samples usage	104
2.7.2.2	Samples creation and saving	104
2.7.2.3	Samples saving disabling	105
2.7.2.4	Samples deletion	105
2.7.2.5	Getting information about samples	105
2.7.3	Descriptor	105
2.7.4	Attribute object	106
2.7.4.1	Attributes usage	106
2.7.4.2	Attributes creation and saving	107
2.7.4.3	Attributes time to live	107
2.7.4.4	Attributes extraction disabling	107
2.7.4.5	Attributes saving disabling	108
2.7.4.6	Attributes deletion	108
2.7.4.7	Getting information about attributes	108
2.7.5	Face object	108
2.7.5.1	Faces usage	110
2.7.5.2	Faces creation and saving	110
2.7.5.3	Requests for working with faces	110
2.7.6	List object	111
2.7.6.1	Lists usage	111
2.7.6.2	Requests for working with lists	111
2.7.7	Event object	112
2.7.7.1	Events usage	114
2.7.7.2	Events creation and saving	115

2.7.7.3	Events deletion	116
2.7.7.4	Getting information about events	116
2.7.7.5	Use “multipart/form-data” schema when generating event	116
2.7.8	Handler object	116
2.7.8.1	Static handler	119
2.7.8.2	Dynamic handler	119
2.7.8.3	Lambda handler	119
2.7.8.4	Requests for working with handlers	119
2.7.9	Verifier object	120
2.7.9.1	Requests for working with verifiers	120
2.7.10	Other objects	121
2.8	Sending notifications via Sender	122
2.9	Licenses	123
2.10	Admin	124
2.11	Configurator	124
2.12	Tasks	124
2.13	Backports	127
2.13.1	Restrictions for working with Backport services	128
2.14	Sdk resource	129
3	Accounts, tokens and authorization types	130
3.1	Account	130
3.1.1	Account type	130
3.2	Token	131
3.2.1	Permissions set in token	131
3.3	Viewing other account data	133
3.4	Authorization types for accessing resources	133
3.5	Credentials verifier	134
3.6	Logging account information	134
4	Estimated data	136
4.1	Gender and age by face image	137
4.2	Face parameters	138
4.2.1	Eyes attributes	138
4.2.2	Face landmarks	139
4.2.3	Distance between eyes	140
4.2.4	Red eyes effect	140
4.2.5	Gaze	142
4.2.6	Glasses	144
4.2.7	Eyebrows	145

4.2.8	Mouth attributes	146
4.2.8.1	Smile state	147
4.2.9	Image quality	149
4.2.9.1	Illumination uniformity according to ICAO standard	153
4.2.10	Image background	154
4.2.10.1	Background lightness	154
4.2.10.2	Background uniformity	155
4.2.11	Dynamic range according to ICAO standard	156
4.2.12	Natural light	156
4.2.13	Face color type	158
4.2.14	Head pose	159
4.2.15	Vertical and horizontal face position	161
4.2.16	Head width and height	162
4.2.17	Face width and height	163
4.2.18	Indents from image edges	163
4.2.19	Mask	164
4.2.20	Emotions	165
4.2.21	Shoulders position	166
4.2.22	Headwear	167
4.2.23	Radial distortion (Fisheye effect)	168
4.3	Image parameters	169
4.3.1	Image format	169
4.3.2	Image size	170
4.3.3	Image width and height	170
4.3.4	Aspect ratio	171
4.3.5	EXIF metadata	172
4.4	Body parameters	172
4.4.1	Gender and age by body image	172
4.4.2	Upper body	173
4.4.3	Lower body	173
4.4.4	Backpack	173
4.5	Liveness	174
4.6	Deepfake	174
4.7	People count	176
5	Services interaction	178
6	Services description	183
6.1	General information about services	183
6.1.1	Worker processes	183

6.1.2	Automatic configurations reload	183
6.1.2.1	Restrictions	184
6.1.2.2	Enable automatic configuration reload	184
6.1.2.3	Configurations update process	184
6.1.3	Database migration execution	185
6.1.3.1	Single command	185
6.1.3.2	Running from container	185
6.2	API service	187
6.3	Remote SDK service	188
6.3.1	Remote SDK with GPU	188
6.3.2	Aggregation	188
6.3.3	Descriptor formats	189
6.3.4	Create objects using external data	190
6.3.5	Checking images for compliance with standards	190
6.3.6	Enable/disable several estimators and detectors	191
6.4	Handlers service	193
6.4.1	Send events to third-party service	193
6.5	Image Store service	194
6.5.1	Buckets description	194
6.5.2	Use S3-compatible storage	195
6.5.3	Object TTL	195
6.5.3.1	Configuring basic TTL bucket policy	196
6.5.3.2	Configuring TTL for specific objects	196
6.5.3.3	Adding TTL to existing objects	196
6.5.3.4	Supported cloud providers	197
6.5.3.5	Migration to apply TTL to objects in S3	197
6.5.3.6	Expiration of TTL	198
6.5.3.7	Search for expiring objects	198
6.5.4	External samples	198
6.6	Accounts service	200
6.6.1	JWT Tokens algorithms	200
6.6.1.1	Default algorithm	200
6.6.1.2	Use ES256 algorithm	200
6.6.1.3	Impact of changing algorithm type	201
6.7	Faces service	202
6.8	Matching services	203
6.8.1	Python Matcher	203
6.8.2	Python Matcher Proxy	203

6.8.3	List caching	203
6.8.3.1	Workers cache	204
6.9	Events service	205
6.9.1	Database for Events	205
6.9.2	Geo position	205
6.9.2.1	Geo position filter	205
6.9.2.2	Filter performance	206
6.9.2.3	Filter features	207
6.9.3	Events creation	208
6.9.4	Events meta-information	208
6.9.4.1	Search by “meta” field	209
6.9.4.2	Important notes	210
6.10	Sender service	211
6.11	Tasks service	214
6.11.1	General information about tasks	214
6.11.2	Clustering task	215
6.11.3	Reporter task	216
6.11.4	Exporter task	216
6.11.5	Cross-matching task	217
6.11.6	Linker task	219
6.11.7	Garbage collection task	220
6.11.8	Additional extraction task	220
6.11.9	ROC-curve calculating task	222
6.11.10	Estimator task	223
6.11.11	Task processing	226
6.11.12	Running scheduled tasks	226
6.11.12.1	Examples of Cron expressions	227
6.11.13	Send notification about task and subtask status changes	228
6.11.14	Additional protection for passwords and tokens	228
6.11.14.1	Add encryption when updating	229
6.12	Admin service	230
6.12.1	Admin user interface	230
6.12.1.1	Accounts tab	231
6.12.1.2	Tasks tab	232
6.12.1.3	Schedules tab	233
6.12.1.4	Info tab	235
6.13	Configurator service	236
6.13.1	Configurator UI	237
6.13.1.1	Settings	237

6.13.1.2	Use tagged settings	240
6.13.1.3	Limitations	240
6.13.1.4	Groups	241
6.13.2	Settings dump	241
6.13.2.1	Receive settings dump	241
6.13.2.2	Apply settings dump	241
6.13.3	Limitations file	242
6.13.3.1	Receive limitation file	242
6.14	Licenses service	243
6.14.1	General information	243
6.14.2	License information	243
6.14.2.1	Expiration date	244
6.14.2.2	Faces limit	244
6.14.2.3	OneShotLiveness	245
6.14.2.4	Body parameters estimation	245
6.14.2.5	People count estimation	246
6.14.2.6	Image check by ISO/IEC 19794-5:2011 standard	246
6.15	Lambda service	246
6.15.1	Before start	247
6.15.1.1	Code and archive requirements	247
6.15.1.2	Environment requirements	247
6.15.1.3	Lambda service configuration	248
6.15.1.4	Configuring lambda entities	248
6.15.2	Lambda types	248
6.15.2.1	Handlers-lambda	249
6.15.2.2	Standalone-lambda	250
6.15.2.3	Tasks-lambda	250
6.15.3	Create lambda	251
6.15.4	Create handler for Handlers-lambda	252
6.15.5	Use lambda	252
6.15.6	Create GPU-enabled lambda	253
6.15.7	Update lambda	254
6.16	Backport 3	255
6.16.1	Backport 3 new resources	255
6.16.1.1	Liveness estimation	255
6.16.1.2	Handlers	255
6.16.2	Backport 3 architecture	256
6.16.3	Backport 3 features and restrictions	257

6.16.4	Garbage collection (GC) module	257
6.16.4.1	Script execution pipeline	257
6.16.4.2	Script launching	258
6.17	Backport 4	259
6.17.1	Backport 4 architecture	259
6.17.2	Backport 4 features and restrictions	259
6.17.3	Backport 4 User Interface	261
6.17.3.1	Lists/faces page	262
6.17.3.2	Handlers page	265
6.17.3.3	Events page	266
6.17.4	Common information	267
6.17.5	Matching dialog	267
6.18	Resource consumption by services	270
7	Additional Information	271
7.1	OneShotLiveness description	271
7.1.1	Liveness check results	271
7.1.2	Requests for estimating Liveness	271
7.1.2.1	Liveness requirements	272
7.1.3	Liveness thresholds	273
7.1.3.1	Quality threshold	273
7.1.3.2	Liveness threshold	274
7.1.3.3	Changing thresholds on “/handlers” and “/verifiers” resources	274
7.1.3.4	Changing thresholds on «/Liveness» and «/sdk» resources	274
7.2	Video analytics	275
7.2.1	People counting video analytics	275
7.2.2	Human tracking video analytics	276
7.2.2.1	roi	277
7.2.2.2	droi	279
7.3	Filters	280
7.3.1	Filters by comparison operators	280
7.3.2	Now-time filters	280
7.4	User interfaces	282
7.4.1	LUNA CLEMENTINE 2.0 Web Service	282
7.4.2	Backport 3 and Backport 4 service interfaces	282
7.4.3	Other interfaces	282
7.5	Disableable services	284
7.5.1	Disabling services process	284
7.5.2	Image Store service disabling consequences	284
7.5.2.1	Image Store service disabling consequences for Backport 3	285

7.5.3	Handlers service disabling consequences	285
7.5.4	Events service disabling consequences	285
7.5.5	Tasks service disabling consequences	286
7.5.6	Sender service disabling consequences	286
7.5.7	Disabling additional services	286
7.6	Nuances of working with services	287
7.6.1	Auto-orientation of rotated image	287
7.6.1.1	Auto-orientation based on EXIF data	287
7.6.1.2	Auto-orientation based on Configurator setting	287
7.6.2	Saving source images	288
7.7	Neural networks	289
7.7.1	Change neural network used	290
7.7.1.1	Launch Additional extraction task	291
7.7.1.2	Change neural network model in settings	291
7.7.2	Use non-delivery neural network model	291
7.7.2.1	Unzip neural networks	292
7.7.2.2	Assign rights to neural networks	292
7.7.2.3	Copy neural network and configuration file to Remote SDK container	292
7.8	Logging	294
7.9	Image check	295
7.9.1	Image check according to ISO/IEC 19794-5:2011	296
7.9.2	Image check according to specified conditions	296
7.9.3	Comparison table of available checks	297
7.10	Services health checks	300
7.11	Upload images from folder	301
7.11.1	General information about the script	301
7.11.2	Script usage	301
7.11.3	Install dependencies	302
7.11.4	GC script launching	303
7.12	Client library	305
7.12.1	General information	305
7.12.2	Library installation example	307
7.13	Plugins	308
7.13.1	Event plugins	308
7.13.2	Background plugins	308
7.13.3	Matching plugins	308
7.13.3.1	General plugin processing pipeline	310
7.13.3.2	Matching cost	311
7.13.3.3	Matching targets	311

7.13.3.4	Plugin data source	313
7.13.4	Plugins usage	314
7.13.4.1	Adding plugins to the directory manually	314
7.13.4.2	Building new Docker container with plugin	314
7.14	Monitoring	317
7.14.1	Send data to InfluxDB	317
7.14.1.1	Data being sent	318
7.14.1.2	View monitoring data	320
7.14.1.3	Requests and estimations statistics gathering	320
7.14.2	Export metrics in Prometheus format	321
7.14.2.1	Type of metrics	321
7.14.3	Configuring metrics collection for Prometheus	323
7.14.4	LUNA Dashboards	324
7.14.4.1	Manual installation of LUNA dashboards	326
7.14.5	Grafana Loki	329
7.14.5.1	Promtail	330
7.15	Databases	331
7.15.1	Manual creation of services databases in PostgreSQL	331
7.15.1.1	PostgreSQL user creation	331
7.15.1.2	Configurator DB creation	331
7.15.1.3	Accounts DB creation	332
7.15.1.4	Handlers DB creation	332
7.15.1.5	Backport 3 DB creation	332
7.15.1.6	Faces DB creation	333
7.15.1.7	Events DB creation	333
7.15.1.8	Tasks DB creation	334
7.15.1.9	Lambda DB creation	334
7.15.2	Build VLMatch for PostgreSQL	335
7.15.3	Add VLMatch function for Faces DB in PostgreSQL	336
7.15.3.1	Add VLMatch function to Faces database	336
7.15.4	Add VLMatch function for Events DB in PostgreSQL	336
7.15.4.1	Add VLMatch function to Events database	337
7.15.5	VLMatch for Oracle	337
7.16	Collect information for technical support	339
7.16.1	Collect services logs	339
7.16.2	Collect additional information	339
8	Recommendations	342
8.1	Resource optimization	342

8.2	Advanced PostgreSQL setting	344
8.2.1	Recommended values for settings	344
9	Sequence diagrams	345
9.1	Samples creation diagrams	346
9.1.1	Sample creation diagram	346
9.1.2	Get information about samples and save samples	347
9.2	Attributes diagrams	348
9.2.1	Temporary attributes extraction diagram	348
9.2.2	Diagram of attribute creation using external data	350
9.2.3	Attributes information diagrams	352
9.3	Faces and lists diagrams	353
9.3.1	Face creation diagram	353
9.3.2	Faces and Lists information and management	354
9.4	Matching diagrams	356
9.4.1	Matching using Python Matcher	357
9.4.1.1	Matching by DB	357
9.4.1.2	Matching by list	358
9.5	Handlers diagrams	359
9.5.1	Handlers management requests	359
9.6	Events diagrams	360
9.6.1	Event creation general diagram	360
9.6.2	Get statistics on events and events information	362
9.7	Tasks diagrams	363
9.7.1	General diagram of task creation and execution	364
9.7.1.1	Starting task creation	364
9.7.1.2	Splitting tasks into subtasks	366
9.7.1.3	Processing each subtask	366
9.7.1.4	Merging results and completing processing	367
9.7.2	General diagram of task cancellation	368
9.7.3	Clustering task diagrams	369
9.7.3.1	Clustering task creating	369
9.7.3.2	Splitting Clustering task into subtask	369
9.7.3.3	Clustering subtask processing	369
9.7.3.4	Clustering task processing completion	371
9.7.4	Linker task diagrams	371
9.7.4.1	Linker task creation	371
9.7.4.2	Splitting Linker task into subtasks	375
9.7.4.3	Linker subtasks processing	375
9.7.4.4	Linker task completing processing	376

9.7.5	Garbage collection task diagram	376
9.7.5.1	Garbage collection task creating	376
9.7.5.2	Splitting Garbage collection task into subtasks	377
9.7.5.3	Garbage collection task processing	377
9.7.5.4	Garbage collection task completing processing	378
9.7.6	Reporter task diagram	378
9.7.6.1	Reporter task creating	378
9.7.6.2	Splitting Reporter task into subtasks	378
9.7.6.3	Reporter subtask processing	378
9.7.6.4	Reporter task completing processing	380
9.7.7	Exporter task diagram	380
9.7.7.1	Exporter task creating	380
9.7.7.2	Splitting Exporter task into subtask	381
9.7.7.3	Exporter subtask processing	381
9.7.7.4	Exporter task completing processing	382
9.7.8	Cross-matching task diagrams	382
9.7.8.1	Cross-matching task creating	382
9.7.8.2	Splitting Cross-matching task into subtasks	382
9.7.8.3	Cross-matching subtasks processing	383
9.7.8.4	Cross-matching task completing processing	384
9.7.9	Estimator task diagram	384
9.7.9.1	Estimator task creating	384
9.7.9.2	Splitting Estimator task into subtask	384
9.7.9.3	Estimator task processing	385
9.7.9.4	Estimator task completing processing	386
9.7.10	Additional extraction task diagram	386
9.7.10.1	Additional extraction task creation	387
9.7.10.2	Splitting Additional extraction task into subtasks	388
9.7.10.3	Additional extraction subtask processing	388
9.7.10.4	Additional extraction task completing processing	389
9.7.11	Tasks information diagrams	389
9.8	Lambda diagrams	391
9.8.1	Lambda creation diagram	391
9.8.2	Lambda processing diagram	394

10 Database description 398

10.1	Faces database description	398
10.1.1	Attribute table model	399
10.1.2	Descriptor table model	400
10.1.3	Face table model	401

10.1.4	List table model	401
10.1.5	List_face table model	402
10.1.6	Unlink_attributes_log table model	402
10.1.7	Sample table model	402
10.1.8	Unlink_deletion_log table model	403
10.1.9	Requests_cache table model	403
10.1.10	Luna-faces_migrations table model	403
10.2	Events database description	404
10.2.1	Event table model	404
10.2.2	Face_detect_result table model	408
10.2.3	Body_detect_result table model	409
10.2.4	Face_descriptor table model	410
10.2.5	Body_descriptor table model	410
10.2.6	Event_match_result table model	410
10.2.7	Face_match_result table model	411
10.2.8	Location table model	412
10.2.9	Tag table model	412
10.2.10	Attach_result table model	412
10.3	Tasks database	414
10.3.1	Task table model	414
10.3.2	Subtask table model	416
10.3.3	Task_error table model	416
10.3.4	Schedule table model	417
10.3.5	Luna-tasks_migrations table model	417
10.4	Handlers database	418
10.4.1	Handler table model	418
10.4.2	Verifier table model	419
10.4.3	Luna-handlers_migrations table model	419
10.5	Configurator database	420
10.5.1	Limitation table model	421
10.5.2	Settings table model	421
10.5.3	Tag table model	421
10.5.4	Group table model	421
10.5.5	Group_limitation table model	422
10.5.6	Configs_migration table model	422
10.5.7	Luna-conf_migrations table model	422
10.6	Backport3 database	423
10.6.1	Account table model	423
10.6.2	Account_token model table	423

10.6.3	Person table model	423
10.6.4	Persons_list table model	424
10.6.5	Descriptors_list table model	424
10.6.6	List_person table model	424
10.6.7	Person_face table model	425
10.6.8	Luna-backport3_migrations table model	425
10.6.9	Handler table model	425
10.7	Accounts database	426
10.7.1	Account table model	426
10.7.2	Token table model	427
10.7.3	Luna-accounts_migration table model	427
10.8	Lambda database	427
10.8.1	Lambda table model	428
10.8.2	Luna-lambda_migration table model	429

11 API errors 430

11.1	General errors	430
11.1.1	Code 0 returned	430
11.1.2	Code 1 returned	430
11.2	HTTP client errors	430
11.2.1	Code 3 returned	430
11.2.2	Code 4 returned	431
11.2.3	Code 5 returned	431
11.2.4	Code 6 returned	431
11.2.5	Code 7 returned	432
11.2.6	Code 8 returned	432
11.2.7	Code 9 returned	433
11.2.8	Code 10 returned	433
11.2.9	Code 11 returned	433
11.2.10	Code 12 returned	434
11.2.11	Code 13 returned	434
11.2.12	Code 14 returned	434
11.2.13	Code 15 returned	435
11.2.14	Code 16 returned	435
11.2.15	Code 17 returned	435
11.2.16	Code 18 returned	436
11.2.17	Code 19 returned	436
11.2.18	Code 20 returned	436
11.2.19	Code 21 returned	436
11.2.20	Code 22 returned	437

11.2.21	Code 23 returned	437
11.2.22	Code 24 returned	437
11.3	Legacy errors	438
11.3.1	Code 5101 returned	438
11.3.2	Code 5102 returned	438
11.4	Database errors	438
11.4.1	Code 10015 returned	438
11.4.2	Code 10016 returned	439
11.4.3	Code 10017 returned	439
11.4.4	Code 10018 returned	439
11.5	API service errors	440
11.5.1	Code 11009 returned	440
11.5.2	Code 11020 returned	440
11.5.3	Code 11027 returned	440
11.5.4	Code 11028 returned	441
11.5.5	Code 11029 returned	441
11.5.6	Code 11030 returned	441
11.5.7	Code 11031 returned	442
11.5.8	Code 11032 returned	442
11.5.9	Code 11034 returned	442
11.5.10	Code 11035 returned	443
11.5.11	Code 11036 returned	443
11.5.12	Code 11037 returned	443
11.5.13	Code 11038 returned	444
11.5.14	Code 11039 returned	444
11.5.15	Code 11040 returned	444
11.5.16	Code 11041 returned	445
11.5.17	Code 11042 returned	445
11.5.18	Code 11043 returned	445
11.5.19	Code 11044 returned	446
11.5.20	Code 11045 returned	446
11.5.21	Code 11046 returned	446
11.5.22	Code 11047 returned	447
11.5.23	Code 11048 returned	447
11.5.24	Code 11049 returned	447
11.5.25	Code 11050 returned	447
11.5.26	Code 11051 returned	448
11.5.27	Code 11052 returned	448
11.5.28	Code 11053 returned	448

11.5.29	Code 11055 returned	449
11.5.30	Code 11056 returned	449
11.5.31	Code 11057 returned	449
11.5.32	Code 11058 returned	449
11.5.33	Code 11059 returned	450
11.5.34	Code 11060 returned	450
11.5.35	Code 11061 returned	450
11.5.36	Code 11062 returned	451
11.5.37	Code 11063 returned	451
11.5.38	Code 11064 returned	451
11.5.39	Code 11065 returned	451
11.5.40	Code 11066 returned	452
11.5.41	Code 11067 returned	452
11.5.42	Code 11068 returned	452
11.5.43	Code 11069 returned	453
11.5.44	Code 11070 returned	453
11.5.45	Code 11071 returned	453
11.5.46	Code 11072 returned	454
11.6	REST API common errors	454
11.6.1	Code 12002 returned	454
11.6.2	Code 12003 returned	454
11.6.3	Code 12005 returned	455
11.6.4	Code 12010 returned	455
11.6.5	Code 12012 returned	455
11.6.6	Code 12013 returned	456
11.6.7	Code 12014 returned	456
11.6.8	Code 12016 returned	457
11.6.9	Code 12017 returned	457
11.6.10	Code 12021 returned	457
11.6.11	Code 12022 returned	458
11.6.12	Code 12023 returned	458
11.6.13	Code 12024 returned	459
11.6.14	Code 12025 returned	459
11.6.15	Code 12027 returned	459
11.6.16	Code 12028 returned	460
11.6.17	Code 12029 returned	460
11.6.18	Code 12030 returned	460
11.6.19	Code 12031 returned	461
11.6.20	Code 12032 returned	461

11.6.21	Code 12033 returned	461
11.6.22	Code 12034 returned	462
11.6.23	Code 12035 returned	462
11.6.24	Code 12036 returned	462
11.6.25	Code 12037 returned	463
11.6.26	Code 12038 returned	463
11.6.27	Code 12039 returned	464
11.6.28	Code 12040 returned	464
11.6.29	Code 12041 returned	464
11.6.30	Code 12042 returned	465
11.6.31	Code 12043 returned	465
11.6.32	Code 12044 returned	465
11.6.33	Code 12045 returned	465
11.6.34	Code 12046 returned	466
11.6.35	Code 12047 returned	466
11.6.36	Code 12048 returned	466
11.6.37	Code 12049 returned	467
11.7	Image Store service errors	467
11.7.1	Code 13003 returned	467
11.7.2	Code 13004 returned	467
11.7.3	Code 13005 returned	468
11.7.4	Code 13006 returned	468
11.7.5	Code 13007 returned	468
11.7.6	Code 13008 returned	469
11.7.7	Code 13009 returned	469
11.8	Admin service errors	469
11.8.1	Code 15012 returned	469
11.8.2	Code 15013 returned	469
11.8.3	Code 15014 returned	470
11.8.4	Code 15015 returned	470
11.9	Image Processing Errors	470
11.9.1	Code 18001 returned	470
11.9.2	Code 18002 returned	471
11.9.3	Code 18003 returned	471
11.10	Image Store Service Errors (Local Storage)	471
11.10.1	Code 19001 returned	471
11.10.2	Code 19002 returned	471
11.10.3	Code 19003 returned	472
11.10.4	Code 19004 returned	472

11.10.5	Code 19005 returned	472
11.10.6	Code 19006 returned	473
11.10.7	Code 19007 returned	473
11.10.8	Code 19008 returned	473
11.10.9	Code 19009 returned	474
11.10.10	Code 19010 returned	474
11.10.11	Code 19011 returned	474
11.10.12	Code 19012 returned	475
11.10.13	Code 19013 returned	475
11.11	Image Store Service Errors (S3 storage)	475
11.11.1	Code 20001 returned	475
11.11.2	Code 20002 returned	476
11.11.3	Code 20003 returned	476
11.11.4	Code 20004 returned	476
11.11.5	Code 20005 returned	477
11.11.6	Code 20006 returned	477
11.11.7	Code 20007 returned	477
11.11.8	Code 20008 returned	478
11.11.9	Code 20009 returned	478
11.11.10	Code 20010 returned	478
11.11.11	Code 20011 returned	479
11.11.12	Code 20012 returned	479
11.11.13	Code 20013 returned	479
11.11.14	Code 20014 returned	480
11.11.15	Code 20015 returned	480
11.11.16	Code 20016 returned	480
11.11.17	Code 20017 returned	481
11.11.18	Code 20018 returned	481
11.11.19	Code 20019 returned	481
11.11.20	Code 20020 returned	482
11.12	Faces service errors	482
11.12.1	Code 22001 returned	482
11.12.2	Code 22002 returned	482
11.12.3	Code 22003 returned	483
11.12.4	Code 22004 returned	483
11.12.5	Code 22005 returned	483
11.12.6	Code 22009 returned	484
11.12.7	Code 22010 returned	484
11.12.8	Code 22011 returned	484

11.12.9	Code 22012 returned	485
11.12.10	Code 22013 returned	485
11.12.11	Code 22015 returned	485
11.12.12	Code 22016 returned	486
11.12.13	Code 22017 returned	486
11.12.14	Code 22018 returned	487
11.12.15	Code 22020 returned	487
11.12.16	Code 22021 returned	487
11.12.17	Code 22022 returned	488
11.12.18	Code 22023 returned	488
11.12.19	Code 22024 returned	488
11.12.20	Code 22025 returned	488
11.12.21	Code 22026 returned	489
11.12.22	Code 22027 returned	489
11.13	Events service errors	489
11.13.1	Code 23001 returned	489
11.13.2	Code 23002 returned	490
11.13.3	Code 23003 returned	490
11.13.4	Code 23004 returned	490
11.13.5	Code 23005 returned	490
11.13.6	Code 23006 returned	491
11.13.7	Code 23007 returned	491
11.13.8	Code 23008 returned	491
11.13.9	Code 23009 returned	492
11.13.10	Code 23010 returned	492
11.13.11	Code 23011 returned	492
11.14	Configurator service errors	493
11.14.1	Code 27001 returned	493
11.14.2	Code 27002 returned	493
11.14.3	Code 27003 returned	493
11.14.4	Code 27004 returned	494
11.14.5	Code 27005 returned	494
11.14.6	Code 27006 returned	494
11.14.7	Code 27007 returned	495
11.14.8	Code 27008 returned	495
11.14.9	Code 27009 returned	495
11.14.10	Code 27010 returned	496
11.14.11	Code 27011 returned	496
11.14.12	Code 27012 returned	496

11.15	Tasks service errors	497
11.15.1	Code 28000 returned	497
11.15.2	Code 28001 returned	497
11.15.3	Code 28002 returned	497
11.15.4	Code 28003 returned	498
11.15.5	Code 28004 returned	498
11.15.6	Code 28005 returned	498
11.15.7	Code 28006 returned	499
11.15.8	Code 28007 returned	499
11.15.9	Code 28008 returned	499
11.15.10	Code 28009 returned	499
11.15.11	Code 28010 returned	500
11.15.12	Code 28011 returned	500
11.15.13	Code 28012 returned	500
11.15.14	Code 28013 returned	500
11.15.15	Code 28014 returned	501
11.15.16	Code 28015 returned	501
11.15.17	Code 28016 returned	502
11.15.18	Code 28017 returned	502
11.15.19	Code 28018 returned	502
11.15.20	Code 28019 returned	503
11.15.21	Code 28020 returned	503
11.15.22	Code 28021 returned	503
11.15.23	Code 28022 returned	503
11.15.24	Code 28023 returned	504
11.15.25	Code 28024 returned	504
11.15.26	Code 28025 returned	504
11.15.27	Code 28026 returned	505
11.15.28	Code 28027 returned	505
11.15.29	Code 28028 returned	505
11.15.30	Code 28029 returned	505
11.15.31	Code 28030 returned	506
11.15.32	Code 28031 returned	506
11.15.33	Code 28032 returned	506
11.15.34	Code 28033 returned	507
11.15.35	Code 28034 returned	507
11.15.36	Code 28035 returned	507
11.15.37	Code 28036 returned	508
11.15.38	Code 28037 returned	508

11.15.39	Code 28038 returned	508
11.15.40	Code 28039 returned	508
11.15.41	Code 28040 returned	509
11.15.42	Code 28041 returned	509
11.16	Sender service errors	510
11.16.1	Code 29001 returned	510
11.16.2	Code 29002 returned	510
11.16.3	Code 29003 returned	510
11.16.4	Code 29004 returned	511
11.16.5	Code 29005 returned	511
11.17	Python Matcher service errors	511
11.17.1	Code 31000 returned	511
11.17.2	Code 31001 returned	512
11.17.3	Code 31002 returned	512
11.17.4	Code 31003 returned	512
11.17.5	Code 31005 returned	513
11.17.6	Code 31006 returned	513
11.17.7	Code 31007 returned	513
11.17.8	Code 31008 returned	514
11.18	Licenses service errors	514
11.18.1	Code 33001 returned	514
11.18.2	Code 33002 returned	514
11.18.3	Code 33003 returned	515
11.18.4	Code 33004 returned	515
11.18.5	Code 33005 returned	515
11.18.6	Code 33006 returned	516
11.18.7	Code 33007 returned	516
11.19	Handlers service errors	516
11.19.1	Code 34000 returned	516
11.19.2	Code 34001 returned	517
11.19.3	Code 34002 returned	517
11.19.4	Code 34003 returned	518
11.19.5	Code 34004 returned	518
11.19.6	Code 34005 returned	519
11.19.7	Code 34006 returned	519
11.19.8	Code 34007 returned	519
11.19.9	Code 34008 returned	520
11.20	Backport 4 service errors	520
11.20.1	Code 35000 returned	520

11.20.2	Code 35001 returned	520
11.20.3	Code 35002 returned	521
11.21	Backport 3 service errors	521
11.21.1	Code 4003 returned	521
11.21.2	Code 11002 returned	521
11.21.3	Code 11004 returned	522
11.21.4	Code 11011 returned	522
11.21.5	Code 11012 returned	522
11.21.6	Code 11018 returned	523
11.21.7	Code 11022 returned	523
11.21.8	Code 12001 returned	523
11.21.9	Code 12018 returned	523
11.21.10	Code 22007 returned	524
11.21.11	Code 22008 returned	524
11.21.12	Code 36001 returned	524
11.21.13	Code 36002 returned	525
11.21.14	Code 36003 returned	525
11.21.15	Code 36004 returned	525
11.21.16	Code 36005 returned	525
11.22	Application server errors	526
11.22.1	Code 37001 returned	526
11.22.2	Code 37002 returned	526
11.22.3	Code 37003 returned	527
11.22.4	Code 37004 returned	527
11.22.5	Code 37005 returned	527
11.23	Healthcheck error	527
11.23.1	Code 38001 returned	527
11.24	Cached Matcher error	528
11.24.1	Code 40001 returned	528
11.25	Accounts service errors	528
11.25.1	Code 41001 returned	528
11.25.2	Code 41002 returned	528
11.25.3	Code 41003 returned	529
11.25.4	Code 41004 returned	529
11.25.5	Code 41005 returned	529
11.25.6	Code 41006 returned	529
11.25.7	Code 41007 returned	530
11.25.8	Code 41008 returned	530
11.25.9	Code 41009 returned	530

11.25.10 Code 41010 returned	531
11.26 Lambda service errors	531
11.26.1 Code 42001 returned	531
11.26.2 Code 42002 returned	531
11.26.3 Code 42003 returned	532
11.26.4 Code 42004 returned	532
11.26.5 Code 42005 returned	532
11.26.6 Code 42006 returned	532
11.26.7 Code 42007 returned	533
11.26.8 Code 42008 returned	533
11.27 Remote SDK service errors	533
11.27.1 Code 43001 returned	533
11.27.2 Code 43002 returned	534
11.27.3 Code 43003 returned	534
11.27.4 Code 43004 returned	534
11.27.5 Code 43005 returned	535
11.27.6 Code 43006 returned	535
11.27.7 Code 43007 returned	535
11.27.8 Code 43008 returned	536
11.28 SDK errors	536
11.28.1 Code 99999 returned	536
11.28.2 Code 100001 returned	536
11.28.3 Code 100002 returned	537
11.28.4 Code 100003 returned	537
11.28.5 Code 100004 returned	537
11.28.6 Code 100005 returned	537
11.28.7 Code 100006 returned	538
11.28.8 Code 100007 returned	538
11.28.9 Code 100008 returned	538
11.28.10 Code 100009 returned	538
11.28.11 Code 100010 returned	539
11.28.12 Code 100011 returned	539
11.28.13 Code 100012 returned	539
11.28.14 Code 100013 returned	540
11.28.15 Code 100014 returned	540
11.28.16 Code 100015 returned	540
11.28.17 Code 100016 returned	540
11.28.18 Code 100017 returned	541
11.28.19 Code 100018 returned	541

11.28.20Code 100019 returned	541
11.28.21Code 100020 returned	541
11.28.22Code 100021 returned	542
11.28.23Code 100022 returned	542
11.28.24Code 100023 returned	542
11.28.25Code 100024 returned	543
11.28.26Code 100025 returned	543
11.28.27Code 100026 returned	543
11.28.28Code 100027 returned	543
11.28.29Code 100028 returned	544
11.28.30Code 100029 returned	544
11.28.31Code 100030 returned	544
11.28.32Code 100031 returned	544
11.28.33Code 100032 returned	545
11.28.34Code 100033 returned	545
11.28.35Code 100034 returned	545
11.28.36Code 100035 returned	546
11.28.37Code 110001 returned	546
11.28.38Code 110002 returned	546
11.28.39Code 110003 returned	546
11.28.40Code 110004 returned	547
11.28.41Code 110005 returned	547
11.28.42Code 110006 returned	547
11.28.43Code 110007 returned	547
11.28.44Code 110008 returned	548
11.28.45Code 110009 returned	548
11.28.46Code 110011 returned	548
11.28.47Code 110012 returned	549
11.28.48Code 110013 returned	549
11.28.49Code 110014 returned	549
11.28.50Code 110015 returned	549
11.28.51Code 110016 returned	550
11.28.52Code 110017 returned	550
11.28.53Code 110018 returned	550
11.28.54Code 110019 returned	550
11.28.55Code 110020 returned	551
11.28.56Code 110021 returned	551
11.28.57Code 110022 returned	551
11.28.58Code 110023 returned	552

11.28.59	Code 110024 returned	552
11.28.60	Code 1100010 returned	552
11.29	RPC errors	552
11.29.1	Code 120004 returned	552
11.29.2	Code 120005 returned	553
11.29.3	Code 120006 returned	553
11.30	Estimation errors	553
11.30.1	Code 200003 returned	553
11.30.2	Code 200004 returned	554
11.30.3	Code 200005 returned	554

12 Configuration parameters of services 555

12.1	Configurator configuration	555
12.1.1	LUNA_CONFIGURATOR_DB section	555
12.1.1.1	db_type	555
12.1.1.2	db_host	555
12.1.1.3	db_port	555
12.1.1.4	db_user	555
12.1.1.5	db_password	556
12.1.1.6	db_name	556
12.1.1.7	connection_pool_size	556
12.1.1.8	dsn	556
12.1.2	LUNA_CONFIGURATOR_LOGGER section	557
12.1.2.1	log_level	557
12.1.2.2	log_time	557
12.1.2.3	log_to_stdout	558
12.1.2.4	log_to_file	558
12.1.2.5	folder_with_logs	558
12.1.2.6	max_log_file_size	558
12.1.2.7	multiline_stack_trace	559
12.1.2.8	format	559
12.1.3	LUNA_CONFIGURATOR_HTTP_SETTINGS section	559
12.1.3.1	request_timeout	559
12.1.3.2	response_timeout	559
12.1.3.3	request_max_size	560
12.1.3.4	keep_alive_timeout	560
12.1.4	INFLUX_MONITORING section	560
12.1.4.1	send_data_for_monitoring	560
12.1.4.2	use_ssl	560
12.1.4.3	organization	560

12.1.4.4	token	561
12.1.4.5	bucket	561
12.1.4.6	host	561
12.1.4.7	port	561
12.1.4.8	flushing_period	561
12.1.5	LUNA_SERVICE_METRICS section	561
12.1.5.1	enabled	561
12.1.5.2	metrics_format	562
12.1.5.3	extra_labels	562
12.1.6	Other	562
12.1.6.1	storage_time	562
12.2	API service configuration	563
12.2.1	LUNA_CONFIGURATOR section	563
12.2.1.1	use_configurator	563
12.2.1.2	luna_configurator_origin	563
12.2.1.3	luna_configurator_api	563
12.2.2	INFLUX_MONITORING section	563
12.2.2.1	send_data_for_monitoring	564
12.2.2.2	use_ssl	564
12.2.2.3	organization	564
12.2.2.4	token	564
12.2.2.5	bucket	564
12.2.2.6	host	564
12.2.2.7	port	564
12.2.2.8	flushing_period	565
12.2.3	LUNA_API_LOGGER section	565
12.2.3.1	log_level	565
12.2.3.2	log_time	565
12.2.3.3	log_to_stdout	565
12.2.3.4	log_to_file	565
12.2.3.5	folder_with_logs	566
12.2.3.6	max_log_file_size	566
12.2.3.7	multiline_stack_trace	566
12.2.3.8	format	566
12.2.4	LUNA_FACES_ADDRESS section	567
12.2.4.1	origin	567
12.2.4.2	api_version	567
12.2.5	LUNA_FACES_TIMEOUTS section	567
12.2.5.1	connect	567

12.2.5.2	request	568
12.2.5.3	sock_connect	568
12.2.5.4	sock_read	568
12.2.6	LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section	568
12.2.6.1	origin	568
12.2.6.2	api_version	568
12.2.6.3	bucket	569
12.2.7	LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section	569
12.2.7.1	connect	569
12.2.7.2	request	569
12.2.8	LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section	569
12.2.8.1	origin	569
12.2.8.2	api_version	570
12.2.8.3	bucket	570
12.2.9	LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section	570
12.2.9.1	connect	570
12.2.9.2	request	570
12.2.10	LUNA_IMAGE_STORE_IMAGES_ADDRESS section	570
12.2.10.1	origin	571
12.2.10.2	api_version	571
12.2.10.3	bucket	571
12.2.11	LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section	571
12.2.11.1	connect	571
12.2.11.2	request	571
12.2.12	LUNA_IMAGE_STORE_OBJECTS_ADDRESS section	572
12.2.12.1	origin	572
12.2.12.2	api_version	572
12.2.12.3	bucket	572
12.2.13	LUNA_IMAGE_STORE_OBJECTS_TIMEOUTS section	572
12.2.13.1	connect	572
12.2.13.2	request	573
12.2.14	LUNA_SENDER_ADDRESS section	573
12.2.14.1	origin	573
12.2.14.2	api_version	573
12.2.15	ADDITIONAL_SERVICES_USAGE section	573
12.2.15.1	luna_events	573
12.2.15.2	luna_tasks	574
12.2.15.3	luna_handlers	574
12.2.15.4	luna_sender	574

12.2.15.5	luna_matcher_proxy	574
12.2.15.6	luna_image_store	575
12.2.15.7	luna_lambda	575
12.2.16	LUNA_EVENTS_ADDRESS section	575
12.2.16.1	origin	575
12.2.16.2	api_version	576
12.2.17	LUNA_EVENTS_TIMEOUTS section	576
12.2.17.1	connect	576
12.2.17.2	request	576
12.2.17.3	sock_connect	576
12.2.17.4	sock_read	577
12.2.18	LUNA_HANDLERS_ADDRESS section	577
12.2.18.1	origin	577
12.2.18.2	api_version	577
12.2.19	LUNA_HANDLERS_TIMEOUTS section	577
12.2.19.1	connect	577
12.2.19.2	request	578
12.2.19.3	sock_connect	578
12.2.19.4	sock_read	578
12.2.20	LUNA_PYTHON_MATCHER_ADDRESS section	578
12.2.20.1	origin	578
12.2.20.2	api_version	578
12.2.21	LUNA_PYTHON_MATCHER_TIMEOUTS section	579
12.2.21.1	connect	579
12.2.21.2	request	579
12.2.21.3	sock_connect	579
12.2.21.4	sock_read	579
12.2.22	LUNA_MATCHER_PROXY_ADDRESS section	579
12.2.22.1	origin	580
12.2.22.2	api_version	580
12.2.23	LUNA_PYTHON_MATCHER_PROXY_TIMEOUTS section	580
12.2.23.1	connect	580
12.2.23.2	request	580
12.2.23.3	sock_connect	580
12.2.23.4	sock_read	581
12.2.24	LUNA_TASKS_ADDRESS section	581
12.2.24.1	origin	581
12.2.24.2	api_version	581

12.2.25	LUNA_TASKS_TIMEOUTS section	581
12.2.25.1	connect	581
12.2.25.2	request	582
12.2.25.3	sock_connect	582
12.2.25.4	sock_read	582
12.2.26	LUNA_LICENSES_ADDRESS section	582
12.2.26.1	origin	582
12.2.26.2	api_version	582
12.2.27	LUNA_ACCOUNTS_ADDRESS section	583
12.2.27.1	origin	583
12.2.27.2	api_version	583
12.2.28	LUNA_ACCOUNTS_TIMEOUTS section	583
12.2.28.1	connect	583
12.2.28.2	request	583
12.2.28.3	sock_connect	584
12.2.28.4	sock_read	584
12.2.29	LUNA_REMOTE_SDK_ADDRESS section	584
12.2.29.1	origin	584
12.2.29.2	api_version	584
12.2.30	LUNA_REMOTE_SDK_TIMEOUTS section	584
12.2.30.1	connect	585
12.2.30.2	request	585
12.2.30.3	sock_connect	585
12.2.30.4	sock_read	585
12.2.31	LUNA_LAMBDA_ADDRESS section	585
12.2.31.1	origin	585
12.2.31.2	api_version	586
12.2.32	LUNA_LAMBDA_TIMEOUTS section	586
12.2.32.1	connect	586
12.2.32.2	request	586
12.2.32.3	sock_connect	586
12.2.32.4	sock_read	587
12.2.33	EXTERNAL_LUNA_API_ADDRESS section	587
12.2.33.1	origin	587
12.2.33.2	api_version	587
12.2.34	LUNA_API_HTTP_SETTINGS section	588
12.2.34.1	request_timeout	588
12.2.34.2	response_timeout	588
12.2.34.3	request_max_size	588

12.2.34.4	keep_alive_timeout	588
12.2.35	LUNA_SERVICE_METRICS section	588
12.2.35.1	enabled	589
12.2.35.2	metrics_format	589
12.2.35.3	extra_labels	589
12.2.36	Other	589
12.2.36.1	luna_api_active_plugins	589
12.2.36.2	luna_api_plugins_settings	590
12.2.36.3	allow_luna_account_auth_header	591
12.2.36.4	storage_time	591
12.2.36.5	default_face_descriptor_version	591
12.3	Admin service configuration	592
12.3.1	LUNA_CONFIGURATOR section	592
12.3.1.1	use_configurator	592
12.3.1.2	luna_configurator_origin	592
12.3.1.3	luna_configurator_api	592
12.3.2	INFLUX_MONITORING section	592
12.3.2.1	send_data_for_monitoring	593
12.3.2.2	use_ssl	593
12.3.2.3	organization	593
12.3.2.4	token	593
12.3.2.5	bucket	593
12.3.2.6	host	593
12.3.2.7	port	593
12.3.2.8	flushing_period	594
12.3.3	LUNA_ADMIN_LOGGER section	594
12.3.3.1	log_level	594
12.3.3.2	log_time	594
12.3.3.3	log_to_stdout	594
12.3.3.4	log_to_file	594
12.3.3.5	folder_with_logs	595
12.3.3.6	max_log_file_size	595
12.3.3.7	multiline_stack_trace	595
12.3.3.8	format	595
12.3.4	LUNA_ACCOUNTS_ADDRESS section	596
12.3.4.1	origin	596
12.3.4.2	api_version	596
12.3.5	LUNA_API_ADDRESS section	596
12.3.5.1	origin	596

12.3.5.2	api_version	597
12.3.6	LUNA_FACES_ADDRESS section	597
12.3.6.1	origin	597
12.3.6.2	api_version	597
12.3.7	LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section	597
12.3.7.1	origin	597
12.3.7.2	api_version	598
12.3.7.3	bucket	598
12.3.8	LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section	598
12.3.8.1	origin	598
12.3.8.2	api_version	598
12.3.8.3	bucket	598
12.3.9	LUNA_IMAGE_STORE_IMAGES_ADDRESS section	599
12.3.9.1	origin	599
12.3.9.2	api_version	599
12.3.9.3	bucket	599
12.3.10	LUNA_IMAGE_STORE_TASK_RESULT_ADDRESS section	599
12.3.10.1	origin	599
12.3.10.2	api_version	600
12.3.10.3	bucket	600
12.3.11	LUNA_SENDER_ADDRESS section	600
12.3.11.1	origin	600
12.3.11.2	api_version	600
12.3.12	LUNA_EVENTS_ADDRESS section	600
12.3.12.1	origin	601
12.3.12.2	api_version	601
12.3.13	LUNA_TASKS_ADDRESS section	601
12.3.13.1	origin	601
12.3.13.2	api_version	601
12.3.14	LUNA_HANDLERS_ADDRESS section	601
12.3.14.1	origin	602
12.3.14.2	api_version	602
12.3.15	LUNA_REMOTE_SDK_ADDRESS section	602
12.3.15.1	origin	602
12.3.15.2	api_version	602
12.3.16	LUNA_PYTHON_MATCHER_ADDRESS section	602
12.3.16.1	origin	603
12.3.16.2	api_version	603

12.3.17	LUNA_MATCHER_PROXY_ADDRESS section	603
12.3.17.1	origin	603
12.3.17.2	api_version	603
12.3.18	LUNA_LICENSES_ADDRESS section	603
12.3.18.1	origin	604
12.3.18.2	api_version	604
12.3.19	LUNA_LAMBDA_ADDRESS section	604
12.3.19.1	origin	604
12.3.19.2	api_version	604
12.3.20	LUNA_ADMIN_TIMEOUTS section	604
12.3.20.1	connect	605
12.3.20.2	request	605
12.3.20.3	sock_connect	605
12.3.20.4	sock_read	605
12.3.21	ADDITIONAL_SERVICES_USAGE section	605
12.3.21.1	luna_events	605
12.3.21.2	luna_tasks	606
12.3.21.3	luna_handlers	606
12.3.21.4	luna_sender	606
12.3.21.5	luna_matcher_proxy	606
12.3.21.6	luna_image_store	607
12.3.21.7	luna_lambda	607
12.3.22	LUNA_ADMIN_HTTP_SETTINGS section	607
12.3.22.1	request_timeout	607
12.3.22.2	response_timeout	607
12.3.22.3	request_max_size	608
12.3.22.4	keep_alive_timeout	608
12.3.23	LUNA_SERVICE_METRICS section	608
12.3.23.1	enabled	608
12.3.23.2	metrics_format	608
12.3.23.3	extra_labels	609
12.3.24	Other	609
12.3.24.1	luna_admin_active_plugins	609
12.4	Faces service configuration	610
12.4.1	LUNA_CONFIGURATOR section	610
12.4.1.1	use_configurator	610
12.4.1.2	luna_configurator_origin	610
12.4.1.3	luna_configurator_api	610

12.4.2	LUNA_FACES_DB section	610
12.4.2.1	db_type	611
12.4.2.2	db_host	611
12.4.2.3	db_port	611
12.4.2.4	db_user	611
12.4.2.5	db_password	611
12.4.2.6	db_name	611
12.4.2.7	connection_pool_size	612
12.4.2.8	dsn	612
12.4.3	LUNA_ATTRIBUTES_DB section	613
12.4.3.1	user	613
12.4.3.2	password	613
12.4.3.3	host	613
12.4.3.4	port	613
12.4.3.5	number	614
12.4.3.6	sentinel > master_name	614
12.4.3.7	sentinel > sentinels	614
12.4.3.8	sentinel > user	614
12.4.3.9	sentinel > password	614
12.4.4	ATTRIBUTES_STORAGE_POLICY section	614
12.4.4.1	default_ttl	615
12.4.4.2	max_ttl	615
12.4.5	LUNA_LICENSES_ADDRESS section	615
12.4.5.1	origin	615
12.4.5.2	api_version	615
12.4.6	LUNA_FACES_LOGGER section	615
12.4.6.1	log_level	615
12.4.6.2	log_time	616
12.4.6.3	log_to_stdout	616
12.4.6.4	log_to_file	616
12.4.6.5	folder_with_logs	616
12.4.6.6	max_log_file_size	616
12.4.6.7	multiline_stack_trace	617
12.4.6.8	format	617
12.4.7	INFLUX_MONITORING section	617
12.4.7.1	send_data_for_monitoring	618
12.4.7.2	use_ssl	618
12.4.7.3	organization	618
12.4.7.4	token	618

12.4.7.5	bucket	618
12.4.7.6	host	618
12.4.7.7	port	618
12.4.7.8	flushing_period	619
12.4.8	LUNA_FACES_HTTP_SETTINGS section	619
12.4.8.1	request_timeout	619
12.4.8.2	response_timeout	619
12.4.8.3	request_max_size	619
12.4.8.4	keep_alive_timeout	619
12.4.9	LUNA_SERVICE_METRICS section	619
12.4.9.1	enabled	620
12.4.9.2	metrics_format	620
12.4.9.3	extra_labels	620
12.4.10	Other	620
12.4.10.1	database_number	620
12.4.10.2	default_face_descriptor_version	621
12.4.10.3	use_material_views	621
12.4.10.4	luna_faces_db_ping_max_count	621
12.4.10.5	storage_time	621
12.4.10.6	luna_faces_active_plugins	622
12.5	Image Store service configuration	623
12.5.1	LUNA_CONFIGURATOR section	623
12.5.1.1	use_configurator	623
12.5.1.2	luna_configurator_origin	623
12.5.1.3	luna_configurator_api	623
12.5.2	LUNA_IMAGE_STORE_LOGGER section	623
12.5.2.1	log_level	624
12.5.2.2	log_time	624
12.5.2.3	log_to_stdout	624
12.5.2.4	log_to_file	624
12.5.2.5	folder_with_logs	624
12.5.2.6	max_log_file_size	625
12.5.2.7	multiline_stack_trace	625
12.5.2.8	format	625
12.5.3	INFLUX_MONITORING section	625
12.5.3.1	send_data_for_monitoring	626
12.5.3.2	use_ssl	626
12.5.3.3	organization	626
12.5.3.4	token	626

12.5.3.5	bucket	626
12.5.3.6	host	626
12.5.3.7	port	627
12.5.3.8	flushing_period	627
12.5.4	LUNA_IMAGE_STORE_HTTP_SETTINGS section	627
12.5.4.1	request_timeout	627
12.5.4.2	response_timeout	627
12.5.4.3	request_max_size	627
12.5.4.4	keep_alive_timeout	628
12.5.5	S3	628
12.5.5.1	host	628
12.5.5.2	region	628
12.5.5.3	aws_public_access_key	628
12.5.5.4	aws_secret_access_key	629
12.5.5.5	authorization_signature	629
12.5.5.6	request_timeout	629
12.5.5.7	connect_timeout	629
12.5.5.8	verify_ssl	629
12.5.6	LUNA_SERVICE_METRICS section	630
12.5.6.1	enabled	630
12.5.6.2	metrics_format	630
12.5.6.3	extra_labels	630
12.5.7	Other	630
12.5.7.1	storage_type	630
12.5.7.2	local_storage	631
12.5.7.3	default_image_extension	631
12.5.7.4	luna_image_store_active_plugins	631
12.6	Tasks service configuration	632
12.6.1	LUNA_CONFIGURATOR section	632
12.6.1.1	use_configurator	632
12.6.1.2	luna_configurator_origin	632
12.6.1.3	luna_configurator_api	632
12.6.2	LUNA_TASKS_DB section	632
12.6.2.1	db_type	633
12.6.2.2	db_host	633
12.6.2.3	db_port	633
12.6.2.4	db_user	633
12.6.2.5	db_password	633
12.6.2.6	db_name	633

12.6.2.7	connection_pool_size	634
12.6.2.8	dsn	634
12.6.3	LUNA_TASKS_LOGGER section	635
12.6.3.1	log_level	635
12.6.3.2	log_time	635
12.6.3.3	log_to_stdout	635
12.6.3.4	log_to_file	635
12.6.3.5	folder_with_logs	636
12.6.3.6	max_log_file_size	636
12.6.3.7	multiline_stack_trace	636
12.6.3.8	format	636
12.6.4	INFLUX_MONITORING section	637
12.6.4.1	send_data_for_monitoring	637
12.6.4.2	use_ssl	637
12.6.4.3	organization	637
12.6.4.4	token	637
12.6.4.5	bucket	638
12.6.4.6	host	638
12.6.4.7	port	638
12.6.4.8	flushing_period	638
12.6.5	LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section	638
12.6.5.1	origin	638
12.6.5.2	api_version	639
12.6.5.3	bucket	639
12.6.6	LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section	639
12.6.6.1	origin	639
12.6.6.2	api_version	639
12.6.6.3	bucket	639
12.6.7	LUNA_IMAGE_STORE_IMAGES_ADDRESS section	640
12.6.7.1	origin	640
12.6.7.2	api_version	640
12.6.7.3	bucket	640
12.6.8	LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section	640
12.6.8.1	connect	640
12.6.8.2	request	641
12.6.9	LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section	641
12.6.9.1	connect	641
12.6.9.2	request	641

12.6.10	LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section	641
12.6.10.1	connect	641
12.6.10.2	request	642
12.6.11	LUNA_IMAGE_STORE_TASK_RESULT_ADDRESS section	642
12.6.11.1	origin	642
12.6.11.2	api_version	642
12.6.11.3	bucket	642
12.6.12	LUNA_IMAGE_STORE_TASK_RESULT_TIMEOUTS section	642
12.6.12.1	connect	643
12.6.12.2	request	643
12.6.13	LUNA_TASKS_LOAD_EXTERNAL_ARCHIVE_TIMEOUTS section	643
12.6.13.1	connect	643
12.6.13.2	request	643
12.6.13.3	sock_connect	643
12.6.13.4	sock_read	644
12.6.14	LUNA_FACES_ADDRESS section	644
12.6.14.1	origin	644
12.6.14.2	api_version	644
12.6.15	LUNA_FACES_TIMEOUTS section	644
12.6.15.1	connect	644
12.6.15.2	request	645
12.6.15.3	sock_connect	645
12.6.15.4	sock_read	645
12.6.16	LUNA_PYTHON_MATCHER_ADDRESS section	645
12.6.16.1	origin	645
12.6.16.2	api_version	645
12.6.17	LUNA_MATCHER_PROXY_ADDRESS section	646
12.6.17.1	origin	646
12.6.17.2	api_version	646
12.6.18	LUNA_TASKS_TIMEOUTS section	646
12.6.18.1	connect	646
12.6.18.2	request	646
12.6.18.3	sock_connect	647
12.6.18.4	sock_read	647
12.6.19	LUNA_EVENTS_ADDRESS section	647
12.6.19.1	origin	647
12.6.19.2	api_version	647
12.6.20	LUNA_EVENTS_TIMEOUTS section	647
12.6.20.1	connect	648

12.6.20.2	request	648
12.6.20.3	sock_connect	648
12.6.20.4	sock_read	648
12.6.21	LUNA_HANDLERS_ADDRESS section	648
12.6.21.1	origin	648
12.6.21.2	api_version	649
12.6.22	LUNA_HANDLERS_TIMEOUTS section	649
12.6.22.1	connect	649
12.6.22.2	request	649
12.6.22.3	sock_connect	649
12.6.22.4	sock_read	650
12.6.23	PLATFORM_LIMITS section	650
12.6.23.1	cross_match > short_array_filter_limit	650
12.6.23.2	cross_match > array_filter_limit	650
12.6.23.3	cross_match > result_candidate_limit	651
12.6.23.4	cross_match > general_limit	651
12.6.24	EXTERNAL_LUNA_API_ADDRESS section	651
12.6.24.1	origin	652
12.6.24.2	api_version	652
12.6.25	LUNA_TASKS_HTTP_SETTINGS section	652
12.6.25.1	request_timeout	652
12.6.25.2	response_timeout	652
12.6.25.3	request_max_size	652
12.6.25.4	keep_alive_timeout	653
12.6.26	TASKS_REDIS_DB_ADDRESS section	653
12.6.26.1	user	653
12.6.26.2	password	653
12.6.26.3	host	653
12.6.26.4	port	653
12.6.26.5	number	653
12.6.26.6	sentinel > master_name	654
12.6.26.7	sentinel > sentinels	654
12.6.26.8	sentinel > user	654
12.6.26.9	sentinel > password	654
12.6.27	ADDITIONAL_SERVICES_USAGE section	654
12.6.27.1	luna_events	654
12.6.27.2	luna_handlers	655
12.6.27.3	luna_sender	655
12.6.27.4	luna_matcher_proxy	655

12.6.27.5	luna_image_store	655
12.6.27.6	luna_lambda	656
12.6.28	LUNA_SERVICE_METRICS section	656
12.6.28.1	enabled	656
12.6.28.2	metrics_format	656
12.6.28.3	extra_labels	656
12.6.29	Other	657
12.6.29.1	storage_time	657
12.6.29.2	max_error_count_per_task	657
12.6.29.3	tasks_to_faces_requests_concurrency	657
12.6.29.4	tasks_to_image_store_requests_concurrency	657
12.6.29.5	tasks_to_handlers_requests_concurrency	657
12.6.29.6	luna_tasks_active_plugins	658
12.7	Events service configuration	659
12.7.1	LUNA_CONFIGURATOR section	659
12.7.1.1	use_configurator	659
12.7.1.2	luna_configurator_origin	659
12.7.1.3	luna_configurator_api	659
12.7.2	LUNA_EVENTS_LOGGER section	659
12.7.2.1	log_level	660
12.7.2.2	log_time	660
12.7.2.3	log_to_stdout	660
12.7.2.4	log_to_file	660
12.7.2.5	folder_with_logs	660
12.7.2.6	max_log_file_size	661
12.7.2.7	multiline_stack_trace	661
12.7.2.8	format	661
12.7.3	LUNA_EVENTS_DB section	661
12.7.3.1	db_type	662
12.7.3.2	db_host	662
12.7.3.3	db_port	662
12.7.3.4	db_user	662
12.7.3.5	db_password	662
12.7.3.6	db_name	662
12.7.3.7	connection_pool_size	663
12.7.3.8	dsn	663
12.7.4	INFLUX_MONITORING section	664
12.7.4.1	send_data_for_monitoring	664
12.7.4.2	use_ssl	664

12.7.4.3	organization	664
12.7.4.4	token	664
12.7.4.5	bucket	665
12.7.4.6	host	665
12.7.4.7	port	665
12.7.4.8	flushing_period	665
12.7.5	LUNA_EVENTS_HTTP_SETTINGS section	665
12.7.5.1	request_timeout	665
12.7.5.2	response_timeout	665
12.7.5.3	request_max_size	666
12.7.5.4	keep_alive_timeout	666
12.7.6	LUNA_SERVICE_METRICS section	666
12.7.6.1	enabled	666
12.7.6.2	metrics_format	666
12.7.6.3	extra_labels	666
12.7.7	Other	667
12.7.7.1	storage_time	667
12.7.7.2	default_face_descriptor_version	667
12.7.7.3	default_body_descriptor_version	667
12.7.7.4	save_events_timeout	667
12.7.7.5	luna_events_active_plugins	667
12.8	Sender service configuration	669
12.8.1	LUNA_CONFIGURATOR section	669
12.8.1.1	use_configurator	669
12.8.1.2	luna_configurator_origin	669
12.8.1.3	luna_configurator_api	669
12.8.2	LUNA_SENDER_LOGGER section	669
12.8.2.1	log_level	670
12.8.2.2	log_time	670
12.8.2.3	log_to_stdout	670
12.8.2.4	log_to_file	670
12.8.2.5	folder_with_logs	670
12.8.2.6	max_log_file_size	671
12.8.2.7	multiline_stack_trace	671
12.8.2.8	format	671
12.8.3	REDIS_DB_ADDRESS section	671
12.8.3.1	user	671
12.8.3.2	password	672
12.8.3.3	host	672

12.8.3.4	port	672
12.8.3.5	number	672
12.8.3.6	channel	672
12.8.3.7	sentinel > master_name	673
12.8.4	LUNA_SERVICE_METRICS section	673
12.8.4.1	enabled	673
12.8.4.2	metrics_format	673
12.8.4.3	extra_labels	673
12.8.4.4	sentinel > sentinels	673
12.8.4.5	sentinel > user	674
12.8.4.6	sentinel > password	674
12.8.5	INFLUX_MONITORING section	674
12.8.5.1	send_data_for_monitoring	674
12.8.5.2	use_ssl	674
12.8.5.3	organization	674
12.8.5.4	token	674
12.8.5.5	bucket	675
12.8.5.6	host	675
12.8.5.7	port	675
12.8.5.8	flushing_period	675
12.8.6	LUNA_SENDER_HTTP_SETTINGS section	675
12.8.6.1	request_timeout	675
12.8.6.2	response_timeout	675
12.8.6.3	request_max_size	676
12.8.6.4	keep_alive_timeout	676
12.8.7	Other	676
12.8.7.1	luna_sender_active_plugins	676
12.9	Licenses service configuration	677
12.9.1	LUNA_CONFIGURATOR section	677
12.9.1.1	use_configurator	677
12.9.1.2	luna_configurator_origin	677
12.9.1.3	luna_configurator_api	677
12.9.2	LUNA_LICENSES_LOGGER section	677
12.9.2.1	log_level	678
12.9.2.2	log_time	678
12.9.2.3	log_to_stdout	678
12.9.2.4	log_to_file	678
12.9.2.5	folder_with_logs	678
12.9.2.6	max_log_file_size	679

12.9.2.7	multiline_stack_trace	679
12.9.2.8	format	679
12.9.3	INFLUX_MONITORING section	679
12.9.3.1	send_data_for_monitoring	680
12.9.3.2	use_ssl	680
12.9.3.3	organization	680
12.9.3.4	token	680
12.9.3.5	bucket	680
12.9.3.6	host	680
12.9.3.7	port	681
12.9.3.8	flushing_period	681
12.9.4	LICENSE_VENDOR section	681
12.9.4.1	vendor	681
12.9.4.2	server_address	681
12.9.4.3	license_id	682
12.9.5	LUNA_LICENSES_HTTP_SETTINGS section	682
12.9.5.1	request_timeout	682
12.9.5.2	response_timeout	682
12.9.5.3	request_max_size	682
12.9.5.4	keep_alive_timeout	682
12.9.6	LUNA_SERVICE_METRICS section	683
12.9.6.1	enabled	683
12.9.6.2	metrics_format	683
12.9.6.3	extra_labels	683
12.9.7	Other	683
12.9.7.1	luna_licenses_active_plugins	683
12.10	Python Matcher service configuration	685
12.10.1	LUNA_CONFIGURATOR section	685
12.10.1.1	use_configurator	685
12.10.1.2	luna_configurator_origin	685
12.10.1.3	luna_configurator_api	685
12.10.2	LUNA_PYTHON_MATCHER_LOGGER section	685
12.10.2.1	log_level	686
12.10.2.2	log_time	686
12.10.2.3	log_to_stdout	686
12.10.2.4	log_to_file	686
12.10.2.5	folder_with_logs	686
12.10.2.6	max_log_file_size	687
12.10.2.7	multiline_stack_trace	687

12.10.2.8	format	687
12.10.3	INFLUX_MONITORING section	687
12.10.3.1	send_data_for_monitoring	688
12.10.3.2	use_ssl	688
12.10.3.3	organization	688
12.10.3.4	token	688
12.10.3.5	bucket	688
12.10.3.6	host	688
12.10.3.7	port	689
12.10.3.8	flushing_period	689
12.10.4	LUNA_FACES_DB section	689
12.10.4.1	db_type	689
12.10.4.2	db_host	689
12.10.4.3	db_port	689
12.10.4.4	db_user	690
12.10.4.5	db_password	690
12.10.4.6	db_name	690
12.10.4.7	connection_pool_size	690
12.10.4.8	dsn	690
12.10.5	LUNA_ATTRIBUTES_DB section	691
12.10.5.1	user	691
12.10.5.2	password	691
12.10.5.3	host	692
12.10.5.4	port	692
12.10.5.5	number	692
12.10.5.6	sentinel > master_name	692
12.10.5.7	sentinel > sentinels	692
12.10.5.8	sentinel > user	693
12.10.5.9	sentinel > password	693
12.10.6	ADDITIONAL_SERVICES_USAGE section	693
12.10.6.1	luna_events	693
12.10.6.2	luna_handlers	693
12.10.6.3	luna_sender	693
12.10.6.4	luna_matcher_proxy	694
12.10.6.5	luna_image_store	694
12.10.6.6	luna_lambda	694
12.10.7	LUNA_EVENTS_DB section	695
12.10.7.1	db_type	695
12.10.7.2	db_host	695

12.10.7.3	db_port	695
12.10.7.4	db_user	695
12.10.7.5	db_password	695
12.10.7.6	db_name	696
12.10.7.7	connection_pool_size	696
12.10.7.8	dsn	696
12.10.8	LUNA_PYTHON_MATCHER_ADDRESS section	697
12.10.8.1	origin	697
12.10.8.2	api_version	697
12.10.9	LUNA_PYTHON_MATCHER_HTTP_SETTINGS section	697
12.10.9.1	request_timeout	698
12.10.9.2	response_timeout	698
12.10.9.3	request_max_size	698
12.10.9.4	keep_alive_timeout	698
12.10.10	PLATFORM_LIMITS section	698
12.10.10.1	match > array_filter_limit	699
12.10.10.2	match > reference_limit	699
12.10.10.3	match > candidate_limit	699
12.10.10.4	match > result_candidate_limit	699
12.10.10.5	cross_match > short_array_filter_limit	700
12.10.10.6	cross_match > array_filter_limit	700
12.10.10.7	cross_match > result_candidate_limit	700
12.10.10.8	cross_match > general_limit	700
12.10.11	DESCRIPTORS_CACHE section	701
12.10.11.1	cache_enabled	701
12.10.11.2	updating_cache_interval	701
12.10.11.3	matching_settings > thread_count	701
12.10.11.4	matching_settings > tasks_count	701
12.10.11.5	matching_settings > batch_size	702
12.10.11.6	rpc_settings > timeouts > connect_timeout	702
12.10.11.7	rpc_settings > timeouts > request_timeout	702
12.10.11.8	rpc_settings > timeouts > response_timeout	702
12.10.11.9	rpc_settings > pool_size	702
12.10.11.10	cached_data > faces_lists > exclude	702
12.10.11.11	cached_data > faces_lists > include	703
12.10.12	LUNA_SERVICE_METRICS section	703
12.10.12.1	enabled	703
12.10.12.2	metrics_format	703
12.10.12.3	extra_labels	703

12.10.13	Other	703
12.10.13.1	storage_time	703
12.10.13.2	luna_python_matcher_active_plugins	704
12.10.13.3	default_face_descriptor_version	704
12.10.13.4	default_body_descriptor_version	704
12.11	Python Matcher Proxy service configuration	705
12.11.1	LUNA_CONFIGURATOR section	705
12.11.1.1	use_configurator	705
12.11.1.2	luna_configurator_origin	705
12.11.1.3	luna_configurator_api	705
12.11.2	LUNA_PYTHON_MATCHER_PROXY_LOGGER section	705
12.11.2.1	log_level	706
12.11.2.2	log_time	706
12.11.2.3	log_to_stdout	706
12.11.2.4	log_to_file	706
12.11.2.5	folder_with_logs	706
12.11.2.6	max_log_file_size	707
12.11.2.7	multiline_stack_trace	707
12.11.2.8	format	707
12.11.3	INFLUX_MONITORING section	707
12.11.3.1	send_data_for_monitoring	708
12.11.3.2	use_ssl	708
12.11.3.3	organization	708
12.11.3.4	token	708
12.11.3.5	bucket	708
12.11.3.6	host	708
12.11.3.7	port	709
12.11.3.8	flushing_period	709
12.11.4	PLATFORM_LIMITS section	709
12.11.4.1	match > array_filter_limit	709
12.11.4.2	match > reference_limit	709
12.11.4.3	match > candidate_limit	710
12.11.4.4	match > result_candidate_limit	710
12.11.4.5	cross_match > short_array_filter_limit	710
12.11.4.6	cross_match > array_filter_limit	710
12.11.4.7	cross_match > result_candidate_limit	711
12.11.4.8	cross_match > general_limit	711
12.11.5	LUNA_PYTHON_MATCHER_DB section	711
12.11.5.1	db_type	711

12.11.5.2	db_host	711
12.11.5.3	db_port	712
12.11.5.4	db_user	712
12.11.5.5	db_password	712
12.11.5.6	db_name	712
12.11.5.7	connection_pool_size	712
12.11.5.8	dsn	713
12.11.6	LUNA_PROXY_TO_PYTHON_MATCHER_TIMEOUTS section	714
12.11.6.1	connect	714
12.11.6.2	request	714
12.11.6.3	sock_connect	714
12.11.6.4	sock_read	714
12.11.7	LUNA_PYTHON_MATCHER_PROXY_HTTP_SETTINGS section	714
12.11.7.1	request_timeout	715
12.11.7.2	response_timeout	715
12.11.7.3	request_max_size	715
12.11.7.4	keep_alive_timeout	715
12.11.7.5	luna_python_matcher_proxy_active_plugins	715
12.11.8	LUNA_FACES_DB section	716
12.11.8.1	db_type	716
12.11.8.2	db_host	716
12.11.8.3	db_port	716
12.11.8.4	db_user	716
12.11.8.5	db_password	716
12.11.8.6	db_name	717
12.11.8.7	connection_pool_size	717
12.11.8.8	dsn	717
12.11.9	LUNA_EVENTS_DB section	718
12.11.9.1	db_type	718
12.11.9.2	db_host	718
12.11.9.3	db_port	718
12.11.9.4	db_user	719
12.11.9.5	db_password	719
12.11.9.6	db_name	719
12.11.9.7	connection_pool_size	719
12.11.9.8	dsn	719
12.11.10	LUNA_ATTRIBUTES_DB section	720
12.11.10.1	user	720
12.11.10.2	password	720

12.11.10.3	host	721
12.11.10.4	port	721
12.11.10.5	number	721
12.11.10.6	sentinel > master_name	721
12.11.10.7	sentinel > sentinels	721
12.11.10.8	sentinel > user	722
12.11.10.9	sentinel > password	722
12.11.11	ADDITIONAL_SERVICES_USAGE section	722
12.11.11.1	luna_events	722
12.11.11.2	luna_handlers	722
12.11.11.3	luna_sender	722
12.11.11.4	luna_matcher_proxy	723
12.11.11.5	luna_image_store	723
12.11.11.6	luna_lambda	723
12.11.12	LUNA_SERVICE_METRICS section	724
12.11.12.1	enabled	724
12.11.12.2	metrics_format	724
12.11.12.3	extra_labels	724
12.11.13	Other	724
12.11.13.1	storage_time	724
12.11.13.2	default_face_descriptor_version	725
12.11.13.3	default_body_descriptor_version	725
12.12	Handlers service configuration	726
12.12.1	LUNA_CONFIGURATOR section	726
12.12.1.1	use_configurator	726
12.12.1.2	luna_configurator_origin	726
12.12.1.3	luna_configurator_api	726
12.12.2	LUNA_HANDLERS_DB section	726
12.12.2.1	db_type	727
12.12.2.2	db_host	727
12.12.2.3	db_port	727
12.12.2.4	db_user	727
12.12.2.5	db_password	727
12.12.2.6	db_name	727
12.12.2.7	connection_pool_size	728
12.12.2.8	dsn	728
12.12.3	INFLUX_MONITORING section	729
12.12.3.1	send_data_for_monitoring	729
12.12.3.2	use_ssl	729

12.12.3.3	organization	729
12.12.3.4	token	729
12.12.3.5	bucket	730
12.12.3.6	host	730
12.12.3.7	port	730
12.12.3.8	flushing_period	730
12.12.4	LUNA_HANDLERS_LOGGER section	730
12.12.4.1	log_level	730
12.12.4.2	log_time	730
12.12.4.3	log_to_stdout	731
12.12.4.4	log_to_file	731
12.12.4.5	folder_with_logs	731
12.12.4.6	max_log_file_size	731
12.12.4.7	multiline_stack_trace	732
12.12.4.8	format	732
12.12.5	LUNA_REMOTE_SDK_ADDRESS section	732
12.12.5.1	origin	732
12.12.5.2	api_version	733
12.12.6	LUNA_REMOTE_SDK_TIMEOUTS section	733
12.12.6.1	connect	733
12.12.6.2	request	733
12.12.6.3	sock_connect	733
12.12.6.4	sock_read	733
12.12.7	LUNA_FACES_ADDRESS section	734
12.12.7.1	origin	734
12.12.7.2	api_version	734
12.12.8	LUNA_FACES_TIMEOUTS section	734
12.12.8.1	connect	734
12.12.8.2	request	734
12.12.8.3	sock_connect	735
12.12.8.4	sock_read	735
12.12.9	LUNA_LAMBDA_UNIT_TIMEOUTS section	735
12.12.9.1	connect	735
12.12.9.2	request	735
12.12.9.3	sock_connect	735
12.12.9.4	sock_read	736
12.12.10	LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section	736
12.12.10.1	origin	736
12.12.10.2	api_version	736

12.12.10.3 bucket	736
12.12.11 LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section	736
12.12.11.1 connect	737
12.12.11.2 request	737
12.12.12 LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section	737
12.12.12.1 origin	737
12.12.12.2 api_version	737
12.12.12.3 bucket	737
12.12.13 LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section	738
12.12.13.1 connect	738
12.12.13.2 request	738
12.12.14 LUNA_IMAGE_STORE_IMAGES_ADDRESS section	738
12.12.14.1 origin	738
12.12.14.2 api_version	739
12.12.14.3 bucket	739
12.12.15 LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section	739
12.12.15.1 connect	739
12.12.15.2 request	739
12.12.16 ADDITIONAL_SERVICES_USAGE section	739
12.12.16.1 luna_events	739
12.12.16.2 luna_handlers	740
12.12.16.3 luna_sender	740
12.12.16.4 luna_matcher_proxy	740
12.12.16.5 luna_image_store	741
12.12.16.6 luna_lambda	741
12.12.17 LUNA_EVENTS_ADDRESS section	741
12.12.17.1 origin	741
12.12.17.2 api_version	741
12.12.18 LUNA_EVENTS_TIMEOUTS section	742
12.12.18.1 connect	742
12.12.18.2 request	742
12.12.18.3 sock_connect	742
12.12.18.4 sock_read	742
12.12.19 LUNA_PYTHON_MATCHER_ADDRESS section	742
12.12.19.1 origin	743
12.12.19.2 api_version	743
12.12.20 LUNA_PYTHON_MATCHER_TIMEOUTS section	743
12.12.20.1 connect	743
12.12.20.2 request	743

12.12.20.3 sock_connect	743
12.12.20.4 sock_read	744
12.12.21 LUNA_MATCHER_PROXY_ADDRESS section	744
12.12.21.1 origin	744
12.12.21.2 api_version	744
12.12.22 LUNA_PYTHON_MATCHER_PROXY_TIMEOUTS section	744
12.12.22.1 connect	744
12.12.22.2 request	745
12.12.22.3 sock_connect	745
12.12.22.4 sock_read	745
12.12.23 REDIS_DB_ADDRESS section	745
12.12.23.1 user	745
12.12.23.2 password	745
12.12.23.3 host	746
12.12.23.4 port	746
12.12.23.5 number	746
12.12.23.6 channel	746
12.12.23.7 sentinel > master_name	746
12.12.23.8 sentinel > sentinels	746
12.12.23.9 sentinel > user	747
12.12.23.10 sentinel > password	747
12.12.24 FETCH_EXTERNAL_IMAGE_TIMEOUTS section	747
12.12.24.1 connect	747
12.12.24.2 request	747
12.12.24.3 sock_connect	747
12.12.24.4 sock_request	748
12.12.25 ATTRIBUTES_STORAGE_POLICY section	748
12.12.25.1 default_ttl	748
12.12.25.2 max_ttl	748
12.12.26 LUNA_HANDLERS_LIMITS section	748
12.12.26.1 received_images_limit	748
12.12.26.2 raw_event_detections_limit	749
12.12.26.3 raw_event_arrays_limit	749
12.12.26.4 result_candidate_limit	749
12.12.27 EXTERNAL_LUNA_API_ADDRESS section	749
12.12.27.1 origin	749
12.12.27.2 api_version	750
12.12.28 LUNA_HANDLERS_HTTP_SETTINGS section	750
12.12.28.1 request_timeout	750

12.12.28.2	response_timeout	750
12.12.28.3	request_max_size	750
12.12.28.4	keep_alive_timeout	750
12.12.29	LUNA_SERVICE_METRICS section	751
12.12.29.1	enabled	751
12.12.29.2	metrics_format	751
12.12.29.3	extra_labels	751
12.12.30	Other	751
12.12.30.1	luna_handlers_active_plugins	751
12.12.30.2	storage_time	752
12.12.30.3	default_face_descriptor_version	752
12.12.30.4	default_body_descriptor_version	752
12.13	Backport 3 service configuration	753
12.13.1	LUNA_CONFIGURATOR section	753
12.13.1.1	use_configurator	753
12.13.1.2	luna_configurator_origin	753
12.13.1.3	luna_configurator_api	753
12.13.2	LUNA_BACKPORT3_DB section	753
12.13.2.1	db_type	754
12.13.2.2	db_host	754
12.13.2.3	db_port	754
12.13.2.4	db_user	754
12.13.2.5	db_password	754
12.13.2.6	db_name	754
12.13.2.7	connection_pool_size	755
12.13.2.8	dsn	755
12.13.3	INFLUX_MONITORING section	756
12.13.3.1	send_data_for_monitoring	756
12.13.3.2	use_ssl	756
12.13.3.3	organization	756
12.13.3.4	token	756
12.13.3.5	bucket	757
12.13.3.6	host	757
12.13.3.7	port	757
12.13.3.8	flushing_period	757
12.13.4	LUNA_BACKPORT3_LOGGER section	757
12.13.4.1	log_level	757
12.13.4.2	log_time	757
12.13.4.3	log_to_stdout	758

12.13.4.4	log_to_file	758
12.13.4.5	folder_with_logs	758
12.13.4.6	max_log_file_size	758
12.13.4.7	multiline_stack_trace	759
12.13.4.8	format	759
12.13.5	LUNA_API_ADDRESS section	759
12.13.5.1	origin	759
12.13.5.2	api_version	760
12.13.6	LUNA_API_TIMEOUTS section	760
12.13.6.1	connect	760
12.13.6.2	request	760
12.13.6.3	sock_connect	760
12.13.6.4	sock_read	760
12.13.7	LUNA_IMAGE_STORE_PORTRAITS_ADDRESS section	761
12.13.7.1	origin	761
12.13.7.2	api_version	761
12.13.7.3	bucket	761
12.13.8	LUNA_IMAGE_STORE_PORTRAITS_TIMEOUTS section	761
12.13.8.1	connect	761
12.13.8.2	request	762
12.13.9	BACKPORT3_EVENTS_DB_ADDRESS section	762
12.13.9.1	user	762
12.13.9.2	password	762
12.13.9.3	host	762
12.13.9.4	port	762
12.13.9.5	number	762
12.13.9.6	channel	763
12.13.9.7	sentinel > master_name	763
12.13.9.8	sentinel > sentinels	763
12.13.9.9	sentinel > user	763
12.13.9.10	sentinel > password	763
12.13.10	LUNA_BACKPORT3_HTTP_SETTINGS section	763
12.13.10.1	request_timeout	764
12.13.10.2	response_timeout	764
12.13.10.3	request_max_size	764
12.13.10.4	keep_alive_timeout	764
12.13.11	LUNA_SERVICE_METRICS section	764
12.13.11.1	enabled	764
12.13.11.2	metrics_format	765

12.13.11.3	extra_labels	765
12.13.12	Other	765
12.13.12.1	storage_time	765
12.13.12.2	luna_backport3_active_plugins	765
12.13.12.3	use_samples_as_portraits	766
12.13.12.4	backport3_enable_portraits	766
12.13.12.5	backport3_enable_ws_events	766
12.13.12.6	max_candidate_in_response	766
12.14	Backport 4 service configuration	767
12.14.1	LUNA_CONFIGURATOR section	767
12.14.1.1	use_configurator	767
12.14.1.2	luna_configurator_origin	767
12.14.1.3	luna_configurator_api	767
12.14.2	LUNA_BACKPORT4_DB section	767
12.14.2.1	db_type	768
12.14.2.2	db_host	768
12.14.2.3	db_port	768
12.14.2.4	db_user	768
12.14.2.5	db_password	768
12.14.2.6	db_name	768
12.14.2.7	connection_pool_size	769
12.14.2.8	dsn	769
12.14.3	INFLUX_MONITORING section	770
12.14.3.1	send_data_for_monitoring	770
12.14.3.2	use_ssl	770
12.14.3.3	organization	770
12.14.3.4	token	770
12.14.3.5	bucket	771
12.14.3.6	host	771
12.14.3.7	port	771
12.14.3.8	flushing_period	771
12.14.4	LUNA_BACKPORT4_LOGGER section	771
12.14.4.1	log_level	771
12.14.4.2	log_time	771
12.14.4.3	log_to_stdout	772
12.14.4.4	log_to_file	772
12.14.4.5	folder_with_logs	772
12.14.4.6	max_log_file_size	772
12.14.4.7	multiline_stack_trace	773

12.14.4.8	format	773
12.14.5	LUNA_API_ADDRESS section	773
12.14.5.1	origin	773
12.14.5.2	api_version	774
12.14.6	LUNA_API_TIMEOUTS section	774
12.14.6.1	connect	774
12.14.6.2	request	774
12.14.6.3	sock_connect	774
12.14.6.4	sock_read	774
12.14.7	LUNA_FACES_ADDRESS section	775
12.14.7.1	origin	775
12.14.7.2	api_version	775
12.14.8	LUNA_FACES_TIMEOUTS section	775
12.14.8.1	connect	775
12.14.8.2	request	775
12.14.8.3	sock_connect	776
12.14.8.4	sock_read	776
12.14.9	ATTRIBUTES_STORAGE_POLICY section	776
12.14.9.1	default_ttl	776
12.14.9.2	max_ttl	776
12.14.10	LUNA_BACKPORT4_HTTP_SETTINGS section	776
12.14.10.1	request_timeout	776
12.14.10.2	response_timeout	777
12.14.10.3	request_max_size	777
12.14.10.4	keep_alive_timeout	777
12.14.11	LUNA_SERVICE_METRICS section	777
12.14.11.1	enabled	777
12.14.11.2	metrics_format	777
12.14.11.3	extra_labels	778
12.14.12	Other	778
12.14.12.1	luna_backport4_active_plugins	778
12.15	Accounts service configuration	779
12.15.1	LUNA_CONFIGURATOR section	779
12.15.1.1	use_configurator	779
12.15.1.2	luna_configurator_origin	779
12.15.1.3	luna_configurator_api	779
12.15.2	LUNA_ACCOUNTS_DB section	779
12.15.2.1	db_type	780
12.15.2.2	db_host	780

12.15.2.3	db_port	780
12.15.2.4	db_user	780
12.15.2.5	db_password	780
12.15.2.6	db_name	780
12.15.2.7	connection_pool_size	781
12.15.2.8	dsn	781
12.15.3	INFLUX_MONITORING section	782
12.15.3.1	send_data_for_monitoring	782
12.15.3.2	use_ssl	782
12.15.3.3	organization	782
12.15.3.4	token	782
12.15.3.5	bucket	783
12.15.3.6	host	783
12.15.3.7	port	783
12.15.3.8	flushing_period	783
12.15.4	LUNA_ACCOUNTS_LOGGER section	783
12.15.4.1	log_level	783
12.15.4.2	log_time	783
12.15.4.3	log_to_stdout	784
12.15.4.4	log_to_file	784
12.15.4.5	folder_with_logs	784
12.15.4.6	max_log_file_size	784
12.15.4.7	multiline_stack_trace	785
12.15.4.8	format	785
12.15.5	LUNA_ACCOUNTS_HTTP_SETTINGS section	785
12.15.5.1	request_timeout	785
12.15.5.2	response_timeout	785
12.15.5.3	request_max_size	786
12.15.5.4	keep_alive_timeout	786
12.15.6	LUNA_SERVICE_METRICS section	786
12.15.6.1	enabled	786
12.15.6.2	metrics_format	786
12.15.6.3	extra_labels	787
12.15.7	Other	787
12.15.7.1	luna_accounts_active_plugins	787
12.15.7.2	storage_time	787
12.16	Remote SDK service configuration	788
12.16.1	LUNA_CONFIGURATOR section	788
12.16.1.1	use_configurator	788

12.16.1.2	luna_configurator_origin	788
12.16.1.3	luna_configurator_api	788
12.16.2	INFLUX_MONITORING section	788
12.16.2.1	send_data_for_monitoring	789
12.16.2.2	use_ssl	789
12.16.2.3	organization	789
12.16.2.4	token	789
12.16.2.5	bucket	789
12.16.2.6	host	789
12.16.2.7	port	789
12.16.2.8	flushing_period	790
12.16.3	LUNA_REMOTE_SDK_LOGGER section	790
12.16.3.1	log_level	790
12.16.3.2	log_time	790
12.16.3.3	log_to_stdout	790
12.16.3.4	log_to_file	790
12.16.3.5	folder_with_logs	791
12.16.3.6	max_log_file_size	791
12.16.3.7	multiline_stack_trace	791
12.16.3.8	format	791
12.16.4	LUNA_LICENSES_ADDRESS section	792
12.16.4.1	origin	792
12.16.4.2	api_version	792
12.16.5	LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section	792
12.16.5.1	origin	792
12.16.5.2	api_version	793
12.16.5.3	bucket	793
12.16.6	LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section	793
12.16.6.1	connect	793
12.16.6.2	request	793
12.16.7	LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section	794
12.16.7.1	origin	794
12.16.7.2	api_version	794
12.16.7.3	bucket	794
12.16.8	LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section	794
12.16.8.1	connect	794
12.16.8.2	request	795
12.16.9	LUNA_IMAGE_STORE_IMAGES_ADDRESS section	795
12.16.9.1	origin	795

12.16.9.2	api_version	795
12.16.9.3	bucket	795
12.16.10	LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section	795
12.16.10.1	connect	796
12.16.10.2	request	796
12.16.11	ADDITIONAL_SERVICES_USAGE section	796
12.16.11.1	luna_events	796
12.16.11.2	luna_handlers	796
12.16.11.3	luna_sender	797
12.16.11.4	luna_matcher_proxy	797
12.16.11.5	luna_image_store	797
12.16.11.6	luna_lambda	797
12.16.12	LUNA_REMOTE_SDK_RUNTIME_SETTINGS section	798
12.16.12.1	global_device_class	798
12.16.12.2	num_threads	798
12.16.12.3	num_compute_streams	798
12.16.13	LUNA_REMOTE_SDK_FACE_DETECTOR_SETTINGS section	798
12.16.13.1	runtime_settings > device_class	799
12.16.13.2	runtime_settings > num_threads	799
12.16.13.3	runtime_settings > num_compute_streams	799
12.16.13.4	estimator_settings > min_face_size	799
12.16.13.5	estimator_settings > redetect_face_target_size	799
12.16.13.6	estimator_settings > redetect_tensor_size	799
12.16.13.7	estimator_settings > redetect_score_threshold	800
12.16.13.8	estimator_settings > score_threshold	800
12.16.14	LUNA_REMOTE_SDK_GAZE_ESTIMATOR_SETTINGS section	800
12.16.14.1	runtime_settings > device_class	800
12.16.14.2	runtime_settings > num_threads	800
12.16.14.3	runtime_settings > num_compute_streams	800
12.16.15	LUNA_REMOTE_SDK_QUALITY_ESTIMATOR_SETTINGS section	800
12.16.15.1	runtime_settings > device_class	801
12.16.15.2	runtime_settings > num_threads	801
12.16.15.3	runtime_settings > num_compute_streams	801
12.16.16	LUNA_REMOTE_SDK_MOUTH_ATTRIBUTES_ESTIMATOR_SETTINGS section	801
12.16.16.1	runtime_settings > device_class	801
12.16.16.2	runtime_settings > num_threads	801
12.16.16.3	runtime_settings > num_compute_streams	801
12.16.17	LUNA_REMOTE_SDK_EMOTIONS_ESTIMATOR_SETTINGS section	802
12.16.17.1	runtime_settings > device_class	802

12.16.17.2	runtime_settings > num_threads	802
12.16.17.3	runtime_settings > num_compute_streams	802
12.16.18	LUNA_REMOTE_SDK_BASIC_ATTRIBUTES_ESTIMATOR_SETTINGS section	802
12.16.18.1	runtime_settings > device_class	802
12.16.18.2	runtime_settings > num_threads	802
12.16.18.3	runtime_settings > num_compute_streams	803
12.16.19	LUNA_REMOTE_SDK_EYES_ESTIMATOR_SETTINGS section	803
12.16.19.1	runtime_settings > device_class	803
12.16.19.2	runtime_settings > num_threads	803
12.16.19.3	runtime_settings > num_compute_streams	803
12.16.20	LUNA_REMOTE_SDK_HEAD_POSE_ESTIMATOR_SETTINGS section	803
12.16.20.1	runtime_settings > device_class	803
12.16.20.2	runtime_settings > num_threads	804
12.16.20.3	runtime_settings > num_compute_streams	804
12.16.21	LUNA_REMOTE_SDK_FACE_DESCRIPTOR_ESTIMATOR_SETTINGS section	804
12.16.21.1	runtime_settings > device_class	804
12.16.21.2	runtime_settings > num_threads	804
12.16.21.3	runtime_settings > num_compute_streams	804
12.16.22	LUNA_REMOTE_SDK_MASK_ESTIMATOR_SETTINGS section	804
12.16.22.1	runtime_settings > device_class	805
12.16.22.2	runtime_settings > num_threads	805
12.16.22.3	runtime_settings > num_compute_streams	805
12.16.23	LUNA_REMOTE_SDK_LIVENESS_ESTIMATOR_SETTINGS section	805
12.16.23.1	runtime_settings > device_class	805
12.16.23.2	runtime_settings > num_threads	805
12.16.23.3	runtime_settings > num_compute_streams	805
12.16.23.4	estimator_settings > real_threshold	806
12.16.23.5	estimator_settings > quality_threshold	806
12.16.24	LUNA_REMOTE_SDK_GLASSES_ESTIMATOR_SETTINGS section	806
12.16.24.1	runtime_settings > device_class	806
12.16.24.2	runtime_settings > num_threads	806
12.16.24.3	runtime_settings > num_compute_streams	806
12.16.25	LUNA_REMOTE_SDK_FACE_WARP_ESTIMATOR_SETTINGS section	806
12.16.25.1	runtime_settings > device_class	807
12.16.25.2	runtime_settings > num_threads	807
12.16.25.3	runtime_settings > num_compute_streams	807
12.16.26	LUNA_REMOTE_SDK_FACE_LANDMARKS68_ESTIMATOR_SETTINGS section	807
12.16.26.1	runtime_settings > device_class	807
12.16.26.2	runtime_settings > num_threads	807

12.16.26.3 runtime_settings > num_compute_streams	807
12.16.27 LUNA_REMOTE_SDK_FACE_LANDMARKS5_ESTIMATOR_SETTINGS section	808
12.16.27.1 runtime_settings > device_class	808
12.16.27.2 runtime_settings > num_threads	808
12.16.27.3 runtime_settings > num_compute_streams	808
12.16.28 LUNA_REMOTE_SDK_IMAGE_COLOR_TYPE_ESTIMATOR_SETTINGS section	808
12.16.28.1 runtime_settings > device_class	808
12.16.28.2 runtime_settings > num_threads	808
12.16.28.3 runtime_settings > num_compute_streams	809
12.16.29 LUNA_REMOTE_SDK_HEADWEAR_ESTIMATOR_SETTINGS section	809
12.16.29.1 runtime_settings > device_class	809
12.16.29.2 runtime_settings > num_threads	809
12.16.29.3 runtime_settings > num_compute_streams	809
12.16.30 LUNA_REMOTE_SDK_FACE_NATURAL_LIGHT_ESTIMATOR_SETTINGS section	809
12.16.30.1 runtime_settings > device_class	809
12.16.30.2 runtime_settings > num_threads	810
12.16.30.3 runtime_settings > num_compute_streams	810
12.16.31 LUNA_REMOTE_SDK_FISHEYE_ESTIMATOR_SETTINGS section	810
12.16.31.1 runtime_settings > device_class	810
12.16.31.2 runtime_settings > num_threads	810
12.16.31.3 runtime_settings > num_compute_streams	810
12.16.32 LUNA_REMOTE_SDK_EYEBROW_EXPRESSION_ESTIMATOR_SETTINGS section	810
12.16.32.1 runtime_settings > device_class	811
12.16.32.2 runtime_settings > num_threads	811
12.16.32.3 runtime_settings > num_compute_streams	811
12.16.33 LUNA_REMOTE_SDK_RED_EYES_ESTIMATOR_SETTINGS section	811
12.16.33.1 runtime_settings > device_class	811
12.16.33.2 runtime_settings > num_threads	811
12.16.33.3 runtime_settings > num_compute_streams	811
12.16.34 LUNA_REMOTE_SDK_FACE_DETECTION_BACKGROUND_ESTIMATOR_SETTINGS	
section	812
12.16.34.1 runtime_settings > device_class	812
12.16.34.2 runtime_settings > num_threads	812
12.16.34.3 runtime_settings > num_compute_streams	812
12.16.35 LUNA_REMOTE_SDK_IMAGE_ORIENTATION_ESTIMATOR_SETTINGS section	812
12.16.35.1 runtime_settings > device_class	812
12.16.35.2 runtime_settings > num_threads	812
12.16.35.3 runtime_settings > num_compute_streams	813

12.16.36	LUNA_REMOTE_SDK_PORTRAIT_STYLE_ESTIMATOR_SETTINGS section	813
12.16.36.1	runtime_settings > device_class	813
12.16.36.2	runtime_settings > num_threads	813
12.16.36.3	runtime_settings > num_compute_streams	813
12.16.37	LUNA_REMOTE_SDK_BODY_DETECTOR_SETTINGS section	813
12.16.37.1	runtime_settings > device_class	813
12.16.37.2	runtime_settings > num_threads	814
12.16.37.3	runtime_settings > num_compute_streams	814
12.16.37.4	estimator_settings > image_size	814
12.16.37.5	estimator_settings > redetect_score_threshold	814
12.16.37.6	estimator_settings > score_threshold	814
12.16.38	LUNA_REMOTE_SDK_BODY_DESCRIPTOR_ESTIMATOR_SETTINGS section	814
12.16.38.1	runtime_settings > device_class	815
12.16.38.2	runtime_settings > num_threads	815
12.16.38.3	runtime_settings > num_compute_streams	815
12.16.39	LUNA_REMOTE_SDK_BODY_WARP_ESTIMATOR_SETTINGS section	815
12.16.39.1	runtime_settings > device_class	815
12.16.39.2	runtime_settings > num_threads	815
12.16.39.3	runtime_settings > num_compute_streams	815
12.16.40	LUNA_REMOTE_SDK_BODY_LANDMARKS_SETTINGS section	816
12.16.40.1	runtime_settings > device_class	816
12.16.40.2	runtime_settings > num_threads	816
12.16.40.3	runtime_settings > num_compute_streams	816
12.16.41	LUNA_REMOTE_SDK_BODY_ATTRIBUTES_ESTIMATOR_SETTINGS section	816
12.16.41.1	runtime_settings > device_class	816
12.16.41.2	runtime_settings > num_threads	816
12.16.41.3	runtime_settings > num_compute_streams	817
12.16.42	LUNA_REMOTE_SDK_HUMAN_DETECTOR_SETTINGS section	817
12.16.42.1	runtime_settings > device_class	817
12.16.42.2	runtime_settings > num_threads	817
12.16.42.3	runtime_settings > num_compute_streams	817
12.16.43	LUNA_REMOTE_SDK_PEOPLE_COUNT_ESTIMATOR_SETTINGS section	817
12.16.43.1	runtime_settings > device_class	817
12.16.43.2	runtime_settings > num_threads	818
12.16.43.3	runtime_settings > num_compute_streams	818
12.16.44	LUNA_REMOTE_SDK_DEEPFAKE_ESTIMATOR_SETTINGS section	818
12.16.44.1	runtime_settings > device_class	818
12.16.44.2	runtime_settings > num_threads	818
12.16.44.3	runtime_settings > num_compute_streams	818

12.16.45	FETCH_EXTERNAL_IMAGE_TIMEOUTS section	818
12.16.45.1	connect	819
12.16.45.2	request	819
12.16.45.3	sock_connect	819
12.16.45.4	sock_request	819
12.16.46	EXTERNAL_LUNA_API_ADDRESS section	819
12.16.46.1	origin	820
12.16.46.2	api_version	820
12.16.47	LUNA_REMOTE_SDK_LIMITS section	820
12.16.47.1	received_images_limit	820
12.16.48	LUNA_REMOTE_SDK_VIDEO_SETTINGS section	820
12.16.48.1	decoder_device_class	821
12.16.48.2	decoder_worker_count	821
12.16.48.3	storage	821
12.16.48.4	max_size	821
12.16.48.5	enabled	821
12.16.48.6	metrics_format	822
12.16.48.7	extra_labels	822
12.16.49	FACE_QUALITY_SETTINGS section	822
12.16.49.1	image_format	822
12.16.49.2	illumination	823
12.16.49.3	specularity	823
12.16.49.4	blurriness	823
12.16.49.5	dark	823
12.16.49.6	light	824
12.16.49.7	head_yaw	824
12.16.49.8	head_pitch	824
12.16.49.9	head_roll	825
12.16.49.10	gaze_yaw	825
12.16.49.11	gaze_pitch	825
12.16.49.12	mouth_smiling	825
12.16.49.13	mouth_occluded	826
12.16.49.14	mouth_open	826
12.16.49.15	glasses	826
12.16.49.16	eyes	827
12.16.49.17	head_horizontal_center	827
12.16.49.18	head_vertical_center	827
12.16.49.19	head_width	827
12.16.49.20	head_height	828

12.16.49.21	eye_distance	828
12.16.49.22	image_width	828
12.16.49.23	image_height	829
12.16.49.24	aspect_ratio	829
12.16.49.25	face_width	829
12.16.49.26	face_height	829
12.16.49.27	indent_left	830
12.16.49.28	indent_right	830
12.16.49.29	indent_upper	830
12.16.49.30	indent_lower	830
12.16.49.31	image_size	831
12.16.49.32	illumination_uniformity	831
12.16.49.33	dynamic_range	831
12.16.49.34	eyebrows	832
12.16.49.35	shoulders	832
12.16.49.36	smile	832
12.16.49.37	headwear	832
12.16.49.38	natural_light	833
12.16.49.39	fish_eye	833
12.16.49.40	red_eye	833
12.16.49.41	color_type	834
12.16.49.42	background_lightness	834
12.16.49.43	background_uniformity	834
12.16.50	Other	835
12.16.50.1	luna_remote_sdk_active_plugins	835
12.16.50.2	storage_time	835
12.16.50.3	default_face_descriptor_version	835
12.16.50.4	default_body_descriptor_version	835
12.16.50.5	luna_remote_sdk_detector_type	836
12.16.50.6	luna_remote_sdk_use_auto_rotation	836
12.17	Lambda service configuration	837
12.17.1	LUNA_CONFIGURATOR section	837
12.17.1.1	use_configurator	837
12.17.1.2	luna_configurator_origin	837
12.17.1.3	luna_configurator_api	837
12.17.2	LUNA_LAMBDA_DB section	837
12.17.2.1	db_type	838
12.17.2.2	db_host	838
12.17.2.3	db_port	838

12.17.2.4	db_user	838
12.17.2.5	db_password	838
12.17.2.6	db_name	838
12.17.2.7	connection_pool_size	839
12.17.2.8	dsn	839
12.17.3	INFLUX_MONITORING section	840
12.17.3.1	send_data_for_monitoring	840
12.17.3.2	use_ssl	840
12.17.3.3	organization	840
12.17.3.4	token	840
12.17.3.5	bucket	841
12.17.3.6	host	841
12.17.3.7	port	841
12.17.3.8	flushing_period	841
12.17.4	LUNA_LAMBDA_LOGGER section	841
12.17.4.1	log_level	841
12.17.4.2	log_time	841
12.17.4.3	log_to_stdout	842
12.17.4.4	log_to_file	842
12.17.4.5	folder_with_logs	842
12.17.4.6	max_log_file_size	842
12.17.4.7	multiline_stack_trace	843
12.17.4.8	format	843
12.17.5	ADDITIONAL_SERVICES_USAGE section	843
12.17.5.1	luna_events	843
12.17.5.2	luna_handlers	844
12.17.5.3	luna_sender	844
12.17.5.4	luna_matcher_proxy	844
12.17.5.5	luna_image_store	844
12.17.5.6	luna_lambda	845
12.17.6	LUNA_LAMBDA_HTTP_SETTINGS section	845
12.17.6.1	request_timeout	845
12.17.6.2	response_timeout	845
12.17.6.3	request_max_size	845
12.17.6.4	keep_alive_timeout	846
12.17.7	LAMBDA_S3 section	846
12.17.7.1	host	846
12.17.7.2	region	846
12.17.7.3	aws_public_access_key	846

12.17.7.4	aws_secret_access_key	847
12.17.7.5	authorization_signature	847
12.17.7.6	request_timeout	847
12.17.7.7	connect_timeout	847
12.17.7.8	verify_ssl	847
12.17.7.9	bucket	848
12.17.8	CLUSTER_CREDENTIALS section	848
12.17.8.1	host	848
12.17.8.2	token	848
12.17.8.3	certificate_path	848
12.17.9	LUNA_LICENSES_ADDRESS section	848
12.17.9.1	origin	848
12.17.9.2	api_version	849
12.17.10	LUNA_SERVICE_METRICS section	849
12.17.10.1	enabled	849
12.17.10.2	metrics_format	849
12.17.10.3	extra_labels	849
12.17.11	Other	850
12.17.11.1	luna_lambda_active_plugins	850
12.17.11.2	storage_time	850
12.17.12	cluster_location	850
12.17.12.1	lambda_registry	851
12.17.12.2	lambda_insecure_registries	851
12.18	Lambda configuration	852
12.18.1	LUNA_LAMBDA_UNIT_LOGGER section	852
12.18.1.1	log_level	852
12.18.1.2	log_time	852
12.18.1.3	log_to_stdout	852
12.18.1.4	log_to_file	852
12.18.1.5	folder_with_logs	853
12.18.1.6	max_log_file_size	853
12.18.1.7	multiline_stack_trace	853
12.18.1.8	format	853
12.18.2	LUNA_LAMBDA_UNIT_HTTP_SETTINGS section	854
12.18.2.1	request_timeout	854
12.18.2.2	response_timeout	854
12.18.2.3	request_max_size	854
12.18.2.4	keep_alive_timeout	854

12.18.3	LUNA_REMOTE_SDK_ADDRESS section	855
12.18.3.1	origin	855
12.18.3.2	api_version	855
12.18.4	LUNA_REMOTE_SDK_TIMEOUTS section	855
12.18.4.1	connect	855
12.18.4.2	request	855
12.18.4.3	sock_connect	856
12.18.4.4	sock_read	856
12.18.5	LUNA_SENDER_ADDRESS section	856
12.18.5.1	origin	856
12.18.5.2	api_version	856
12.18.6	LUNA_PYTHON_MATCHER_TIMEOUTS section	856
12.18.6.1	connect	857
12.18.6.2	request	857
12.18.6.3	sock_connect	857
12.18.6.4	sock_read	857
12.18.7	LUNA_PYTHON_MATCHER_PROXY_TIMEOUTS section	857
12.18.7.1	connect	857
12.18.7.2	request	858
12.18.7.3	sock_connect	858
12.18.7.4	sock_read	858
12.18.8	LUNA_PYTHON_MATCHER_ADDRESS section	858
12.18.8.1	origin	858
12.18.8.2	api_version	859
12.18.9	LUNA_MATCHER_PROXY_ADDRESS section	859
12.18.9.1	origin	859
12.18.9.2	api_version	859
12.18.10	LUNA_IMAGE_STORE_IMAGES_ADDRESS section	859
12.18.10.1	origin	859
12.18.10.2	api_version	860
12.18.10.3	bucket	860
12.18.11	LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section	860
12.18.11.1	connect	860
12.18.11.2	request	860
12.18.12	LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section	860
12.18.12.1	origin	861
12.18.12.2	api_version	861
12.18.12.3	bucket	861

12.18.13 LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section	861
12.18.13.1 connect	861
12.18.13.2 request	861
12.18.14 LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section	862
12.18.14.1 origin	862
12.18.14.2 api_version	862
12.18.14.3 bucket	862
12.18.15 LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section	862
12.18.15.1 connect	862
12.18.15.2 request	863
12.18.16 LUNA_FACES_ADDRESS section	863
12.18.16.1 origin	863
12.18.16.2 api_version	863
12.18.17 LUNA_FACES_TIMEOUTS section	863
12.18.17.1 connect	863
12.18.17.2 request	864
12.18.17.3 sock_connect	864
12.18.17.4 sock_read	864
12.18.18 LUNA_EVENTS_ADDRESS section	864
12.18.18.1 origin	864
12.18.18.2 api_version	865
12.18.19 LUNA_EVENTS_TIMEOUTS section	865
12.18.19.1 connect	865
12.18.19.2 request	865
12.18.19.3 sock_connect	865
12.18.19.4 sock_read	865
12.18.20 INFLUX_MONITORING section	866
12.18.20.1 send_data_for_monitoring	866
12.18.20.2 use_ssl	866
12.18.20.3 organization	866
12.18.20.4 token	866
12.18.20.5 bucket	866
12.18.20.6 host	866
12.18.20.7 port	867
12.18.20.8 flushing_period	867
12.18.21 ADDITIONAL_SERVICES_USAGE section	867
12.18.21.1 luna_events	867
12.18.21.2 luna_handlers	867
12.18.21.3 luna_sender	867

12.18.21.4	luna_matcher_proxy	868
12.18.21.5	luna_image_store	868
12.18.21.6	luna_lambda	868
12.18.22	LUNA_SERVICE_METRICS section	869
12.18.22.1	enabled	869
12.18.22.2	metrics_format	869
12.18.22.3	extra_labels	869
12.18.23	Other	869
12.18.23.1	luna_lambda_unit_active_plugins	869
12.19	Tasks-lambda configuration	871
12.19.1	LUNA_LAMBDA_TASKS_UNIT_LOGGER section	871
12.19.1.1	log_level	871
12.19.1.2	log_time	871
12.19.1.3	log_to_stdout	871
12.19.1.4	log_to_file	871
12.19.1.5	folder_with_logs	872
12.19.1.6	max_log_file_size	872
12.19.1.7	multiline_stack_trace	872
12.19.1.8	format	872
12.19.2	LUNA_LAMBDA_TASKS_UNIT_HTTP_SETTINGS section	873
12.19.2.1	request_timeout	873
12.19.2.2	response_timeout	873
12.19.2.3	request_max_size	873
12.19.2.4	keep_alive_timeout	873
12.19.3	LUNA_SERVICE_METRICS section	874
12.19.3.1	enabled	874
12.19.3.2	metrics_format	874
12.19.3.3	extra_labels	874
12.19.4	Other	874
12.19.4.1	luna_lambda_tasks_unit_active_plugins	874

Glossary

Term	Description
Attributes	Basic attributes and descriptor.
Basic attributes	Age, gender, and ethnicity.
Body parameters	Body characteristics (presence of backpack, headwear, color of clothing, etc.) determined on the source image during detection.
Bounding box	Rectangle that bounds the image space with the detected face or body.
Bucket	Logical entity used to store objects.
Cooperative mode	Process of obtaining source photo images of faces or bodies from videostream with conscious person interaction with the camera.
Descriptor	Data set in closed, binary format prepared by recognition system based on the characteristic being analyzed.
Detector	Neural network used to detect either faces or bodies, or both faces and bodies in the source image.
Estimator	Neural network used to estimate a certain parameter of the face or body in the source image.
Face	Changeable objects that include information about a human face.
Face parameters	Facial characteristics (emotions, mouth parameters, head position, etc.) determined on the source image during detection.
Image parameters	Image characteristics (width and height, aspect ratio, size, etc.) determined on the source image during detection.
Landmarks	Reference points on the face or body used by recognition algorithms to localize the face or body.
Liveness	Software method that enables you to confirm whether a person in one or more images is “real” or a fraudster using a fake ID (printed face photo, video, paper or 3D mask).
Matching	The operation of matching descriptors stored in the database.
Occlusion	State of an object (eye, mouth) when it is hidden by any other object.

Term	Description
Non-cooperative mode	Process of obtaining the source photo images of faces or bodies from videostream, while a person may not interact with the camera.
Samples, Warps	Normalized (centered and cropped) image obtained after face or body detection, prior to descriptor extraction.
Verification	Comparison of two photo images of a face in order to determine belonging to the same face.

Abbreviation	Decoding
NN	Neural network
DB	Database
UI	User interface
GC	Garbage collection
LUNA PLATFORM	LP, LUNA
LUNA PLATFORM API	API
LUNA PLATFORM Accounts	Accounts
LUNA PLATFORM Faces	Faces
LUNA PLATFORM Image Store	Image Store
LUNA PLATFORM Matcher	Matcher
LUNA PLATFORM Events	Events
LUNA PLATFORM Sender	Sender
LUNA PLATFORM Remote SDK	Remote SDK
LUNA PLATFORM Handlers	Handlers
LUNA PLATFORM Python Matcher	Python Matcher
LUNA PLATFORM Python Matcher Proxy	Python Matcher Proxy
LUNA PLATFORM Backport 3	Backport 3
LUNA PLATFORM Backport 4	Backport 4
LUNA PLATFORM Admin	Admin
LUNA PLATFORM Configurator	Configurator

Abbreviation	Decoding
LUNA PLATFORM Tasks	Tasks
LUNA PLATFORM Licenses	Licenses
LUNA PLATFORM User Interface 3	User Interface 3
LUNA PLATFORM User Interface 4	User Interface 4

1 Introduction

LUNA PLATFORM is an automated face and body recognition system. LUNA PLATFORM is designed to solve the following problems:

- Images processing and analysis:
 - Face and body detection in photos.
 - Basic attributes (age, gender) and face, body, image parameters estimation (emotions, upper clothing, image ratio and other).
 - Checking images according to ISO/IEC 19794-5:2011 and non-standard conditions.
 - Presentation attacks detection (Liveness check) and image analysis for the use of Deepfake technology.
- Search for similar faces in the database (1 to 1, 1 to many and M to N).
- Storage of the received face and body descriptors in databases.
- Creation of lists to search in.
- Statistics gathering.
- Flexible request management to meet user data processing requirements.

Some of these features are licensed separately. See the [“License information”](#) section.

2 General concepts

The following sections explain general LP concepts, describe existing services and created data types. It does not describe all the requests, database structures, and other technical nuances.

2.1 Services

LP consists of several services. Communication between them occurs via HTTP requests: a service receives a request and always returns a response. Some services also communicate with each other via Redis or web sockets.

You can find information about LUNA PLATFORM 5 architecture in the [“Interaction of LP Services”](#) section.

All the services can be divided into **general** and **additional**. Using the general services, the optimal operation of the LP is ensured. Using of all general services is enabled by default in the API service settings. If necessary, some general services can be disabled (see the [“Disableable services”](#) section).

Most of the services have their database or file storage.

2.1.1 General services

Service	Description	Database	Disableable
API	The main gateway to LP. Receives requests, distributes tasks to other LP services.	None	No
Accounts	Manages accounts.	PostgreSQL/Oracle	No
Remote SDK	Detects faces in images, extracts face parameters and creates samples. Extracts descriptors from samples. Extracts basic attributes of the images.	PostgreSQL/ Oracle	No
Python Matcher	Performs matching tasks.	None	No
Faces	Creates faces, lists, and attributes. Stores these objects in the database. Allows other services to receive the required data from the database.	PostgreSQL/Oracle, Redis	No
Configurator	Stores all configurations of all the services in a single place.	PostgreSQL/ Oracle	No
Licenses	Checks your license conditions and returns information about them.	None	No

Service	Description	Database	Disableable
Handlers	Creates and stores handlers. Accepts requests for detection, estimation and extraction and redirects them to the Remote SDK service.	PostgreSQL/Oracle	Yes
Image Store	Stores samples, any objects, reports about long tasks execution, created clusters and additional metadata.	Local storage/ Amazon S3	Yes
Events	Stores data on the generated events in the database.	PostgreSQL	Yes
Admin	Enables to perform general administrative routines.	PostgreSQL/ Oracle	Yes
Tasks	Performs long tasks, such as garbage collection, extraction of descriptors with a new neural network version, clustering.	PostgreSQL/Oracle, Redis	Yes
Tasks Worker	Performs the internal work of the Tasks service.	None	Yes
Sender	Sends notifications about created events via web socket.	Redis	Yes

2.1.2 Additional services

Additional services provide more opportunities for the system to work. Launching additional services is optional.

Service	Description	Database
Backport 3	The service is used to process LUNA PLATFORM 3 requests using LUNA PLATFORM 5.	PostgreSQL/ Oracle
Backport 4	The service is used to process LUNA PLATFORM 4 requests using LUNA PLATFORM 5.	None
User Interface 3	User Interface is used to visually represent the features provided with the Backport 3 service. It does not include all the functionality available in LP 3.	None
User Interface 4	User Interface is used to visually represent the features provided with the Backport 4 service. It does not include all the functionality available in LP 4.	None

Service	Description	Database
Python Matcher Proxy	The service manages matching requests and routes them to Python Matcher or matching plugins .	None
Lambda	It works with user modules that mimic the functionality of a separate service.	PostgreSQL/ Oracle

Below is a diagram of the interaction of the general and additional services.

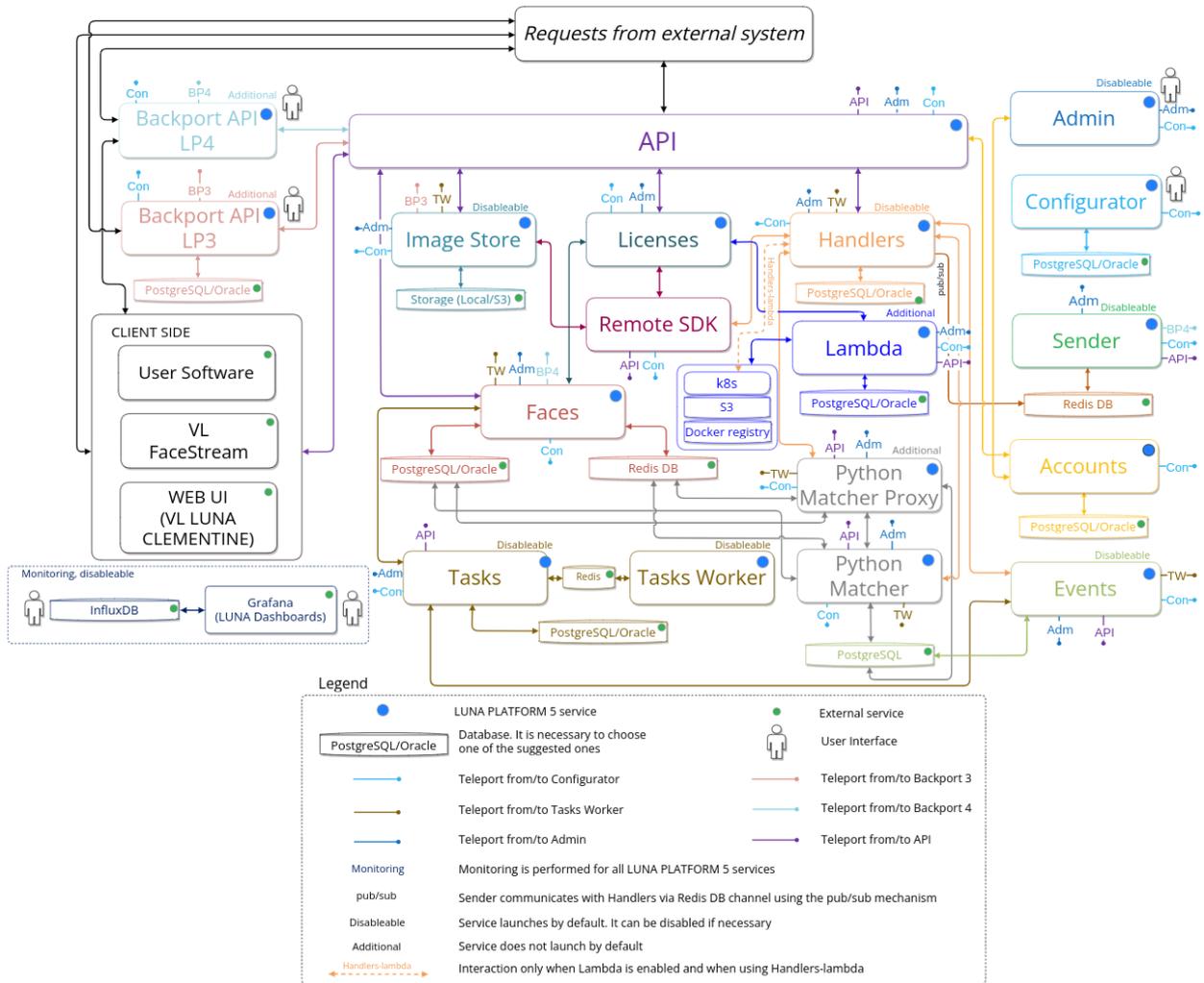


Figure 1: Simplified interaction diagram of general and additional services

This diagram does not describe the communication lines between the services. For a full description of the interaction of services, see the [“Interaction of LP services”](#) section.

2.1.3 Third-party services

There are several third-party services that are usually used with LP.

Function	Description	Supported services
Balancer	Balance requests between LP services when there are several similar services launched. For example, they can balance requests between API services or between two contours of LP upon scaling.	NGINX
Monitoring system	The monitoring system is used to monitor and control the number of processes launched for LP.	Supervisor
Monitoring database	The database is used for monitoring purposes.	InfluxDB
Monitoring visualization	Monitoring visualization is represented by a set of dashboards. You can evaluate the total load on the system or the load on individual services.	Grafana, Grafana Loki
Log rotation service	All LP services write logs and their size may grow dramatically. Log rotation service is required to delete outdated log files and free disk space.	Logrotate, etc.

These services are not described in this document. See the corresponding documentation provided by the services vendors.

2.2 Authorization system

Most requests to the LUNA PLATFORM require a mandatory account, except for requests that do not involve authorization.

Accounts

The account is required to delimit the visibility areas of objects for a particular user. Each created account has its own unique “account_id”.

The account can be created using a POST request “[create account](#)” to the API service, or by requesting “[register account](#)”, or using the Admin user interface. When creating the account, you must specify the following data: login (email), password and account type.

The account type determines what data is available to the user.

- **user** – It is type of account with which you can create objects and use only your account data.
- **advanced_user** – It is type of account for which rights similar to “user” are available, and there is access to the data of all accounts. Access to data from other accounts means the ability to receive data (GET requests), check their availability (HEAD requests) and perform comparison requests based on data from other accounts.
- **admin** – It is type of account for which rights similar to “advanced_user” are available, and there is also access to the Admin service.

Using the “Luna-Account-Id” header in the “[create account](#)” request, you can set the desired account ID.

Tokens

Token is linked to an existing account with any type and enables you to impose extended restrictions on the requests being made. For example, when creating the token, you can give the user permission only to create and modify all lists and faces, or you can prevent the use of certain handlers by specifying their ID.

The token and all its permissions are stored in the database and linked to the account by the “account_id” parameter.

When creating the token, you can set the following parameters:

- **expiration_time** – Expiration time of the token in RFC 3339 format. You can specify an infinite token expiration time using the value “null”.
- **permissions** – Permissions that are available to the user.
- **visibility_area** – Token visibility of data from other accounts.

Authorization types for accessing resources

There are three types of authorization available in LUNA PLATFORM:

- **BasicAuth** – Authorization by login and password (set during account creation).
- **BearerAuth** – Authorization by JWT token (issued after the token is created).

- **LunaAccountIdAuth** – Authorization by “Luna-Account-Id” header, which specifies the “account_id” generated after creating the account (this method was adopted as the main one before version 5.30.0).

LunaAccountIdAuth authorization has the lowest priority compared to other methods and can be disabled using the “ALLOW_LUNA_ACCOUNT_AUTH_HEADER” setting in the “OTHER” section of the API service settings in the Configurator (enabled by default). In the [OpenAPI specification](#) the “Luna-Account-Id” header is marked with the word **Deprecated**.

See detailed information about the LUNA PLATFORM 5 authorization system in the [“Accounts, tokens and authorization types”](#) section.

2.3 Approaches at work

There are three main operations in LUNA PLATFORM:

1. **Detection** is the process of recognizing a face or body in a photo and normalizing the image (creating a sample) for further work. At the stage of detection, **estimation** of face or body parameters also performs, i.e. estimation of emotions, direction of gaze, upper clothes, etc.
2. **Extraction** is the process of extracting gender and age from a face image and extracting a set of unique parameters of a face or body (descriptors) from a sample to perform further matching.
3. **Matching** is the process of comparing descriptors.

These operations are performed strictly one after the other. It is impossible to match face descriptors without first extracting them.

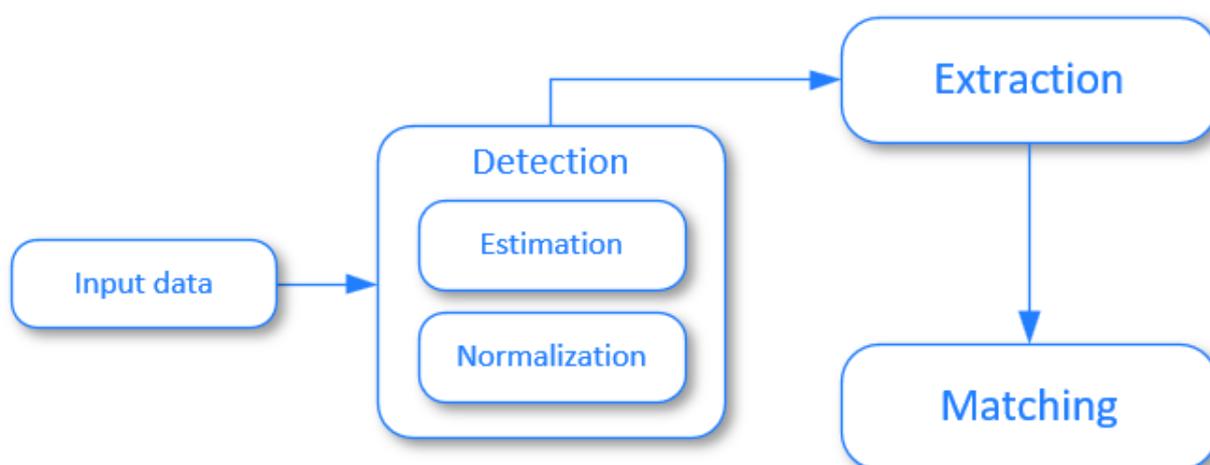


Figure 2: Basic LUNA PLATFORM operations

There are two main approaches when performing the operations described above.

2.3.1 Parallel performing of requests

The first and main approach is to set the rules of detection, estimation, extraction, matching, etc. in a single **handler** object. After that, you need to create an object **event**, which will give a result based on all the rules specified in the handler. Using this approach is the most optimal from the point of view of business logic.

With this approach, the following actions are performed:

- Using the “[create handler](#)” request, a handler is created containing information about image processing rules;

- In the “generate events” request, the received handler ID is specified, the processed image is attached and an event is generated containing information obtained by processing handler rules.

Thus, when an event is generated, detection, estimation, extraction, matching, saving to the database, and so on are performed simultaneously.

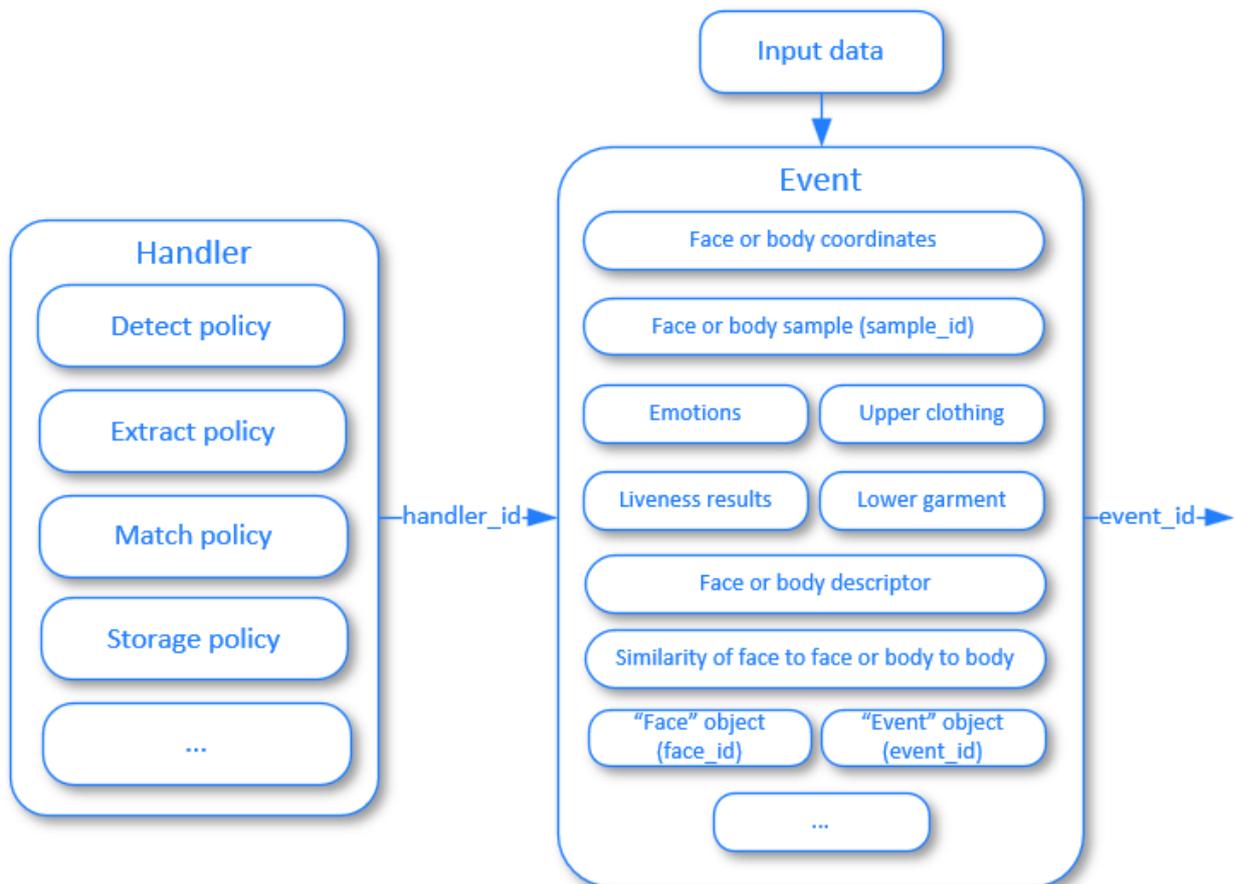


Figure 3: “Parallel performing of requests” approach

Examples of common scenarios for using handlers:

- Registration of a reference descriptor with saving to the list.
- Biometric identification of faces on the list without saving.
- Saving faces identified in the list in the database.
- Saving events only for unique faces for later counting.
- Batch identification.
- Batch import.

For some scenarios, it may be necessary to pre-create certain objects in the system using separate requests. For example, to identify a set of faces by a list, you first need to create a **list** object, link the previously created **face** objects to it, and then use handler and event.

Advantages of the approach:

- All the rules in one place.
- Flexible management of saving objects to databases, setting up save filters.
- Ability to work with packages of images located in ZIP archive, FTP server, S3-like storage, etc.
- Ability to send notifications via web sockets.
- Collecting statistics.

The classic way to use this approach is paired with the FaceStream module. FaceStream analyzes the video stream and sends the best shots from the video stream to LUNA PLATFORM for further processing. Using the specified handler, LUNA PLATFORM processes images according to the specified rules and saves objects to the specified databases.

2.3.2 Sequential performing of requests

The second approach is to sequential performing of requests, i.e. in one request you need to perform a face detection and get its result, then use this result in an extraction request and so on.

With this approach, the following actions are performed:

- Normalizing image, as well as face detection and estimation using the “[detect faces](#)” request.
- Extracting gender, age and descriptor from a normalized image using the “[extract attributes](#)” request.
- Creating a face using the “[create face](#)” request.
- Matching of face descriptors using the “[matching faces](#)” request.

As a rule, this approach is used when working with faces, but if necessary, you can also make separate requests with bodies, for example, to match body descriptors using the “[matching bodies](#)” request.

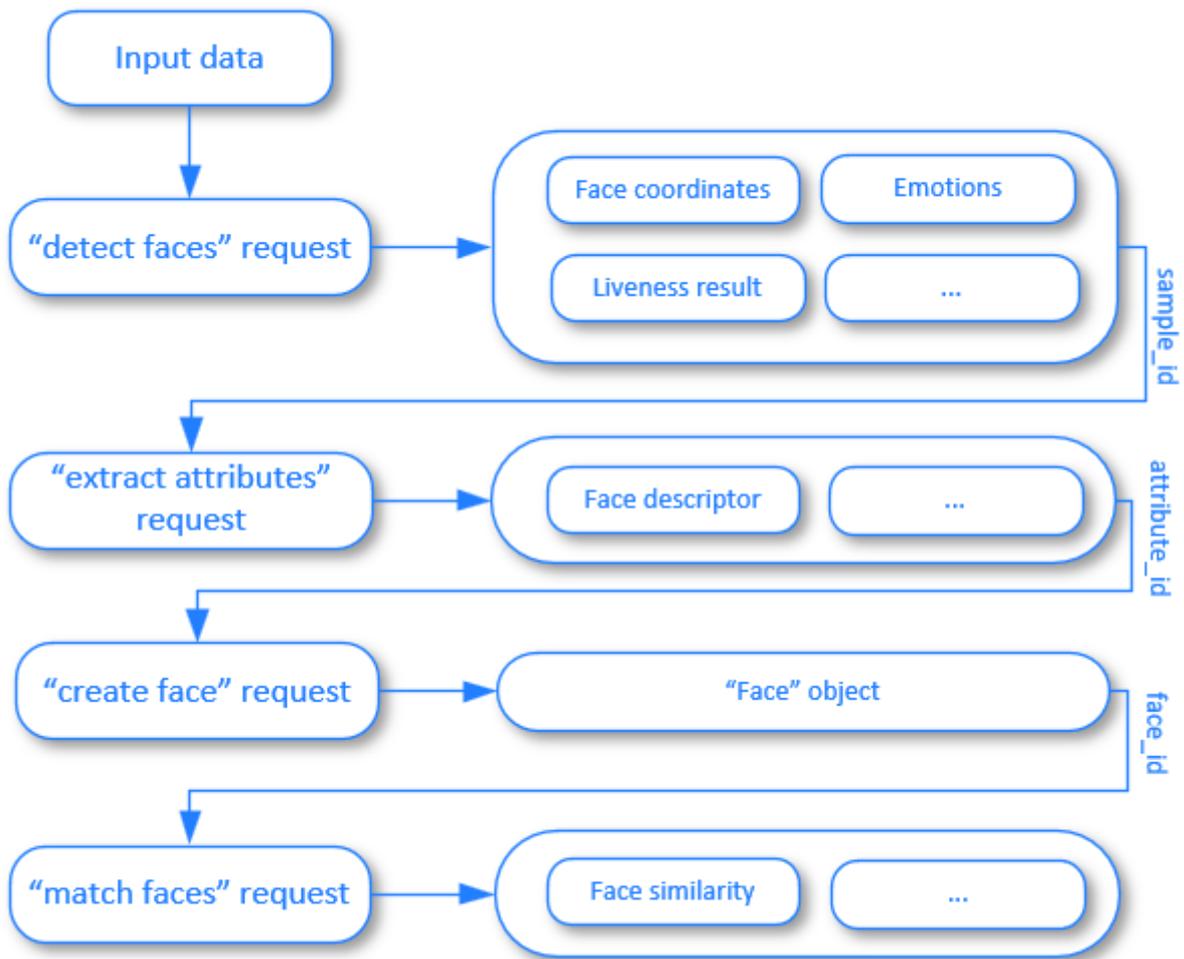


Figure 4: “Sequential performing of requests” approach

This approach is recommended for use:

- During the initial acquaintance with the system.
- If you need to measure the speed of each stage of image processing separately.
- If you need to implement a processing scenario that is difficult to implement using handlers and events.

More detailed information about these approaches will be provided below.

2.4 Detection

LP searches for all faces and/or bodies on each incoming photo image.

Detection is performed using neural networks located in the Remote SDK container.

Detailed information about neural networks is described in the [“Neural networks”](#) section.

Either the source image or a special format image obtained using the VisionLabs software (sample) can be submitted to the input.

2.4.0.1 Image format requirements

Images should be sent only in allowed formats. The image format is specified in the “Content-Type” header.

The following image formats are supported: JPG, PNG, BMP, PORTABLE PIXMAP, TIFF. Each format has its own advantages and is intended for specific tasks.

The most commonly used formats are **PNG** and **JPG**. Below is a table with their advantages and disadvantages:

Format	Compression	Advantages	Disadvantages
PNG	Lossless	Better image processing quality	More image weight
JPG	Lossy	Less image weight	Worse image processing quality

Thus, if it is not intended to save the source images in the Image Store, or the Image Store has a sufficiently large volume, then it is recommended to use PNG format to get the best image processing results.

It is not recommended to send images that are too compressed, as the quality of face and body estimations and matching will be reduced.

LUNA PLATFORM 5 supports many color models (for example, RGB, RGBA, CMYK, HSV, etc.), however, when processing an image, all of them are converted to the **RGB** color model.

The image can also be encoded in Base64 format.

For more information about the source images, see [“Source images”](#).

2.4.1 Face detection and estimation process

Below is the order of face detection in the image:

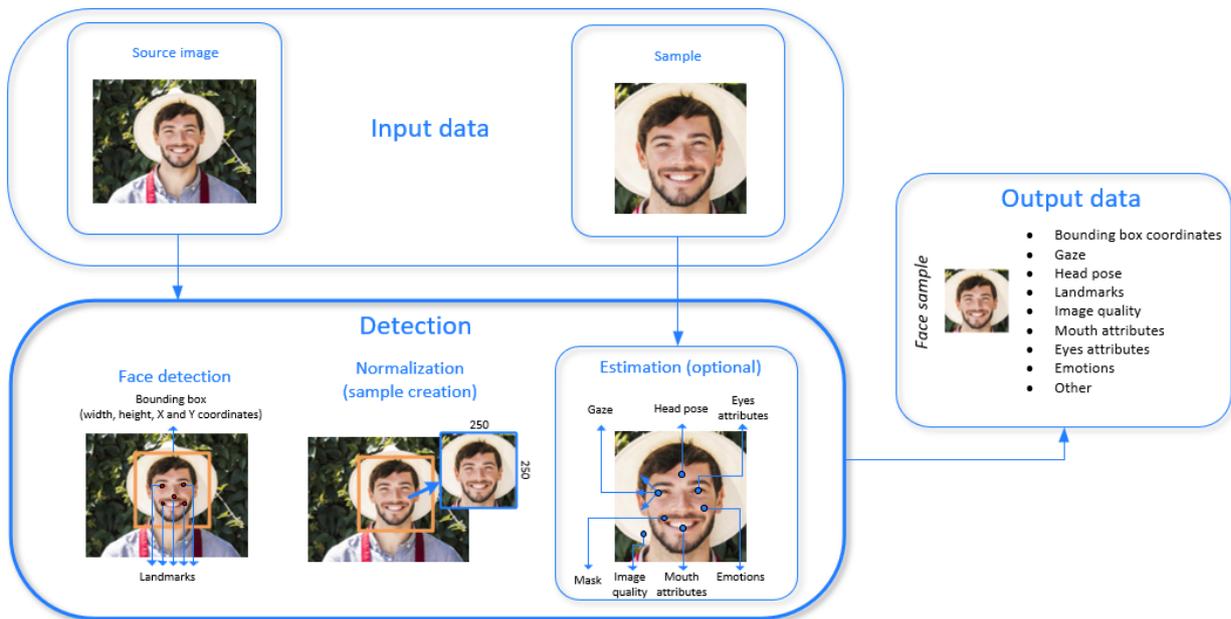


Figure 5: Face image processing order

Basic steps for face images detection:

- **Face detection.** The bounding box of the face (height, width, “X” and “Y” coordinates) and five landmarks of the face are determined.
- **Normalization.** The image is centered in a certain way and cropped to the size of 250x250 pixels required for further work. Thus, all processed images look the same. For example, the left eye is always in a frame defined by some coordinates. Such a normalized image is called a **sample**. The sample is stored in the Image Store. After creating the sample, it is assigned a special identifier “sample_id”, which is used for further image processing.

All objects created in LUNA PLATFORM are assigned identifiers. Sample — “sample_id”, face — “face_id”, event — “event_id”, etc. Identifiers are the main way to transfer data between requests and services.

For more information about samples, see [“Sample object”](#).

- **Estimation.** The following face parameters are estimated:
 - Gaze direction (rotation angles for both eyes).
 - Head pose (yaw, roll and pitch angles).
 - Sixty-eight landmarks. The number of landmarks depends on which face parameters are to be estimated.
 - Image quality (lightness, darkness, blurriness, illumination and specularity).
 - Mouth attributes (overlap, presence of smile).

- Mask presence (medical or tissue mask on the face, mask is missing, mouth is occluded).
- Emotions (anger, disgust, fear, happiness, neutrality, sadness, surprise).
- Others (see the “Face parameters” section).

If necessary, you can configure filtering by [head pose angles](#).

2.4.2 Body detection and estimation process

Below is the order of body image detection:

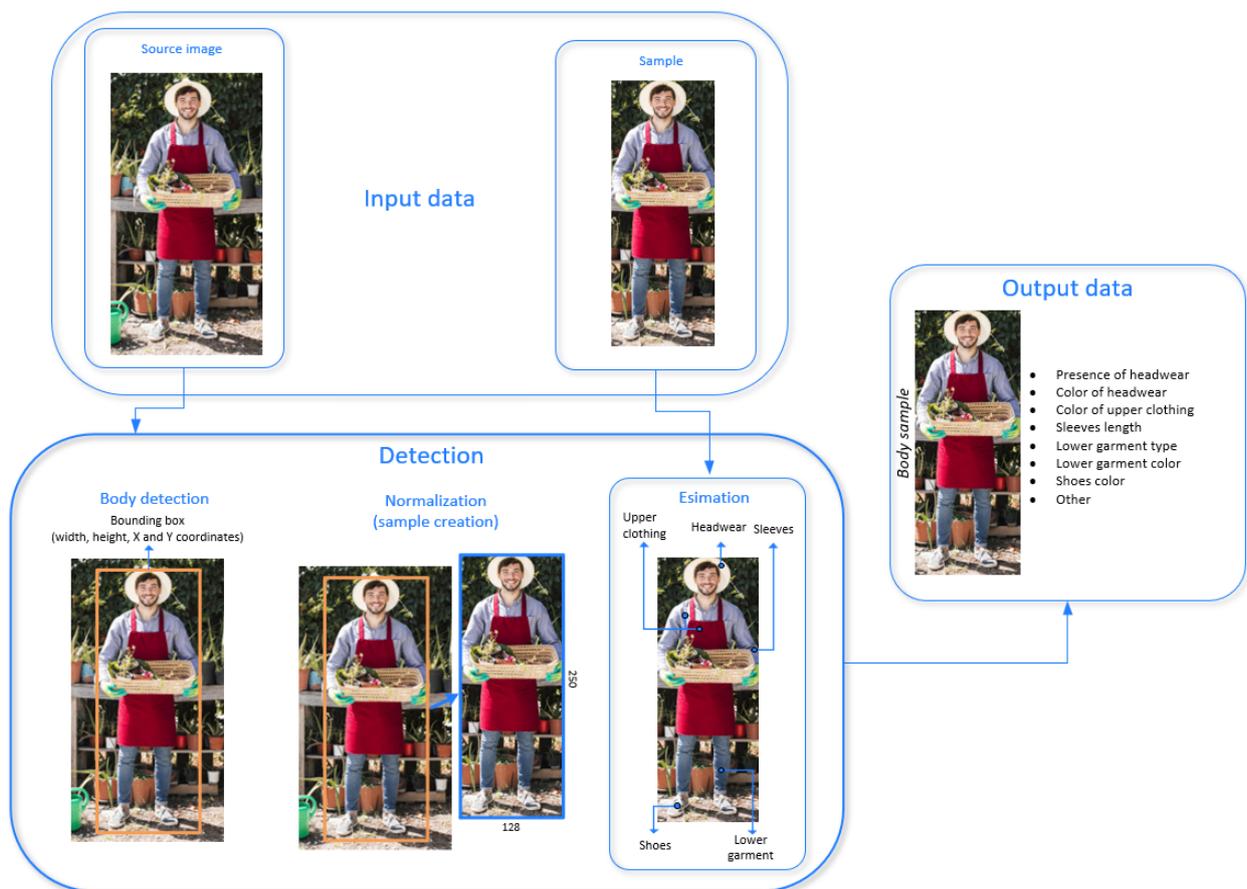


Figure 6: Body image processing order

Basic steps for body images detection:

- **Body detection.** The bounding box of the body (height, width, “X” and “Y” coordinates) is determined.
- **Normalization.** The image is centered in a certain way and cropped to the size of 128x250 pixels required for further work (otherwise, the principle of operation is similar to the process of normalization of the face).

- **Estimation.** The following body parameters are estimated:
 - Upper body (presence and color of headwear, color of upper clothing).
 - Sleeve length (long sleeves, short sleeves, unknown).
 - Lower body (type of lower garment — trousers, shorts, skirt, unknown; color of lower garment, presence and color of shoes).
 - Accessories.
 - Others (see the “Body parameters” section).

2.4.3 Services interaction during detection

Below is a diagram of the interaction of services when performing detection.

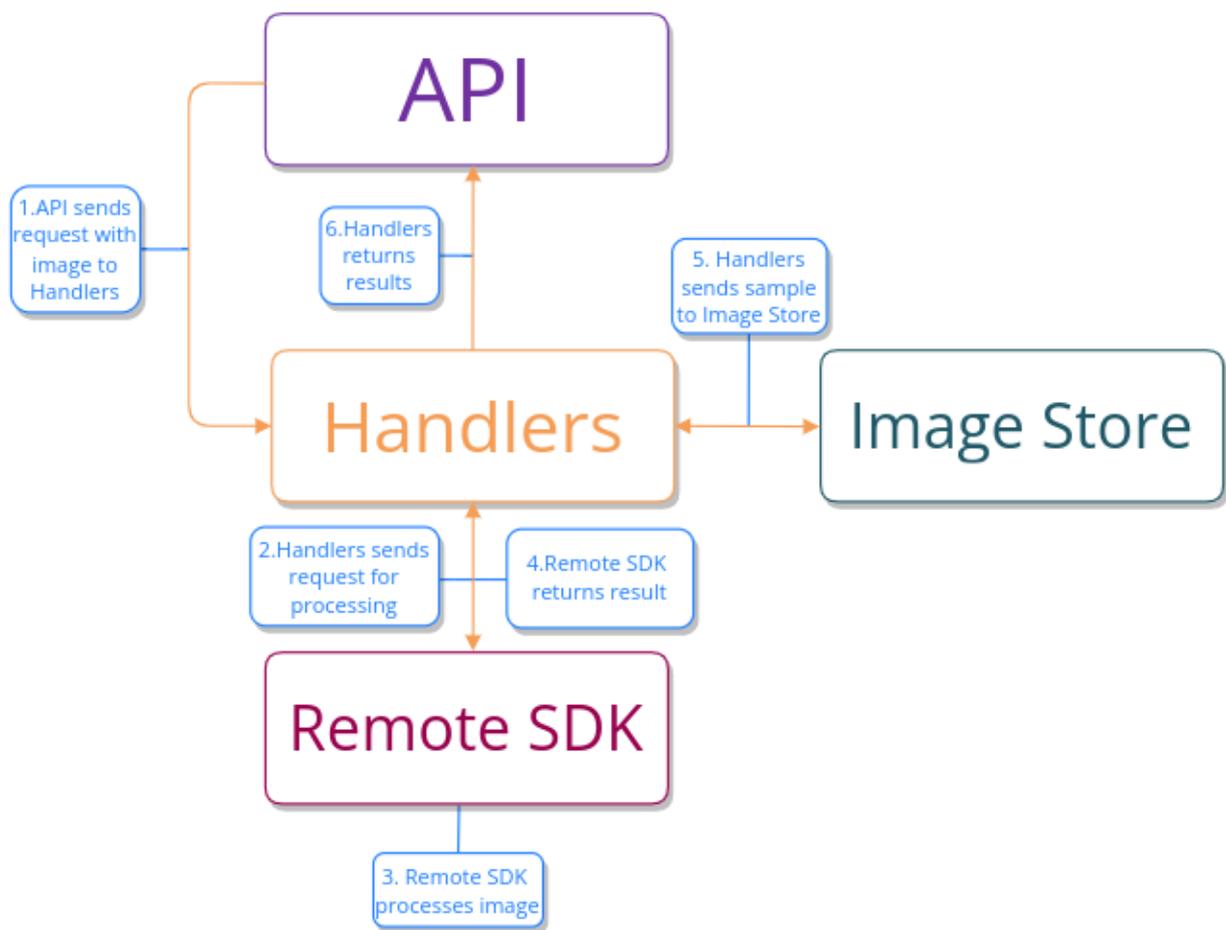


Figure 7: Services interaction during detection

2.4.4 Requests to perform detection

Below are the main requests where you can perform face or body detection.

2.4.4.1 Requests “create handler” and “generate events”

These requests are used in the approach “Parallel performing of requests”.

- Request “create handler”:

Parameter name in request body	Response body	Saving to DB
“detect_face” or “detect_body” of the “detect_policy” policy	Handler ID containing detection rules	Handler information is stored in Handlers database

- Request “generate events”:

Request body	Response body	Saving to DB
Image	Detection result	Sample is stored in the Image Store, you can disable saving using the “store_sample” parameter. If the “store_event” parameter is enabled in the handler, the detection results will be stored in the Events database in the “face_detect_result” or “body_detect_result” table

It is possible to use the “multipart/form-data” schema in the request. Using a schema enables you to send multiple images at the same time, specify additional information for an event, and more. See “[Use”multipart/form-data” schema when generating event](#)” for more information.

2.4.4.2 Request “detect faces”

The “detect faces” request is used in the approach “Sequential performing of requests”. This request cannot be used for the body image.

Request body	Response body	Saving to DB
Image	Detection result	Sample is stored in the Image Store, it is impossible to disable saving

In the request, you can send an image or specify the URL to the image. You can use a multipart/form-data schema to send several images in the request. Each of the sent images must have a unique name. The content-disposition header must contain the actual filename.

You can also specify a bounding box parameters using the multipart/form-data schema. It enables you to specify the certain zone with a face on the image. You can specify the following properties for the

bounding box:

- top left corner “x” and “y” coordinates,
- height,
- width.

2.4.4.3 Request “sdk”

The “sdk” request also allows face or body detection on the source image. Unlike other requests, its data cannot be used in the future. For more information, see [“Sdk resource”](#).

2.4.5 Requests to perform estimation

The estimation of the parameters of faces and bodies is performed in conjunction with detection. Most often, the parameters for estimation have a name like `estimate_<estimation_name>`. For example, `estimate_glasses`.

The [“Estimated data”](#) section contains a list of all possible parameters that can be estimated in LUNA PLATFORM 5.

2.4.6 Detection results

The following information is returned in responses to detection requests:

- address to the stored face/body sample
- id of the face/body sample
- coordinates of the bounding box of the face/body
- five landmarks of the face

All this data is used for further extraction and matching.

This information is supplemented depending on the enabled face/body estimation parameters.

You can read more about the detection of faces, bodies and the Remote SDK service in the [“Remote SDK service”](#) section.

2.4.7 Redetection

If necessary, you can specify detections (coordinates “x”, “y”, “width”, “height”) rather than images in the request bodies. For example, using the “application/json” schema of the [“detect faces”](#) request, you can specify an array of detections in the “face_bounding_boxes” field.

If you explicitly specify detections in the request bod instead of an image, you can indicate to LUNA PLATFORM whether repeated detection (redetection) should be performed or whether the specified

detection can be considered trusted. It is assumed that trusted detections were obtained using VisionLabs algorithms. Marking a third-party detection as trusted may affect the estimation results.

You can indicate whether the detection is trusted using the “application/json” scheme (the “trusted_detections” parameter) or the “multipart/form-data” scheme (the “X-Luna-Trusted-Detections” header) in the following requests, where detections are explicitly specified instead of images:

- “detect faces”
- “sdk”
- “iso”
- “generate events”
- “perform verification”
- “generate stream events (beta)”

2.5 Extraction

In LUNA PLATFORM, the word “extraction” means:

- Extraction of descriptors and basic attributes (gender, age, ethnicity) of faces.
- Extraction of descriptors of bodies.

It is also estimate to determine a person’s gender and age from the body image, but this method of determination is less accurate and occurs at the [estimation](#) stage. In addition, it is recommended to estimate the basic attributes of the body together with the extraction of the basic attributes of the face.

Descriptors are sets of characteristics read from faces or bodies in images. Descriptors requires much less memory than the source image.

It is impossible to restore the source image of a face or body from the descriptor due to personal data security reasons.

Descriptors are used to match faces to faces and bodies to bodies. It is impossible to match two faces or bodies in the absence of extracted descriptors. For example, if you want to match a face from a database with an input face image, you need to extract the descriptor for that image.

Descriptor is extracted using a neural network located in the Remote SDK container.

Detailed information about neural networks is described in the [“Neural networks”](#) section.

See [“Descriptor”](#) section for more information on descriptors.

2.5.1 Extraction process

Data extraction requires the following information obtained during detection:

- Sample of the face or body.
- Coordinates of the bounding box for the face or body.
- Five landmarks of the face.

The following is the order in which the extraction is performed:

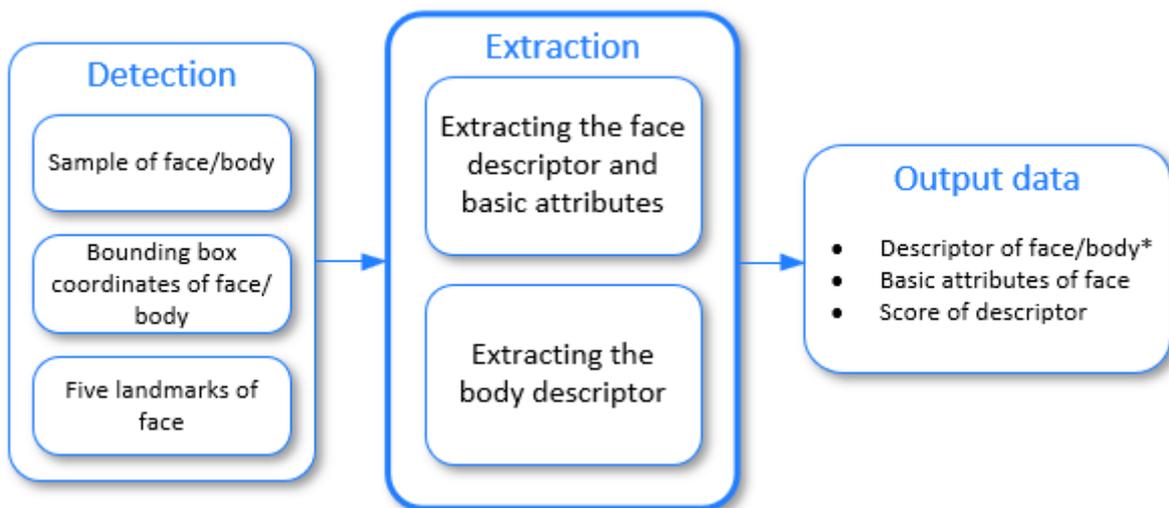


Figure 8: Extraction order

* In most extraction requests, the descriptor is saved to the database without being returned in the response body (see [“Requests to perform extraction”](#)).

2.5.2 Specifics of extracting data from faces

The descriptor and basic attributes of the face derived from the same image are stored in the database as an **attribute** object. This object may include both descriptor and basic attributes, or only one of these entities.

It is possible to extract descriptor and basic attributes using several samples of the same face at the same time. In this way, an **aggregated** descriptor and aggregated basic attributes can be obtained. The accuracy of matching and extraction of basic attributes by means of the aggregated descriptor is higher. Aggregation should be used when working with images received from webcams.

Aggregated descriptor can be obtained from images, but not from already created descriptors.

Detailed information about aggregation is described in the [“Aggregation”](#) section.

2.5.2.1 Temporary attributes

After extracting the attributes using the appropriate requests (see [“Requests to perform extraction”](#)), the extracted data is stored in the Redis database as temporary attributes.

Temporary attributes have a TTL (time to live) and are removed from the database after a specified period. In the request, you can specify a period from 1 to 86400 seconds. The default TTL is 5 minutes.

You can get information about a temporary attribute by its identifier until its TTL expires.

In order to save the attributes to the database, you need to create a **face** object and link the attributes to it. The face is created either by a separate request after the attributes are retrieved (the face is stored in the Faces DB) or during the generation of the event (the face is saved in the Faces DB or in the Events DB). You can create multiple faces and combine them into a list that can be used for matching.

See [“Face object”](#) and [“List object”](#) for details.

You can store attributes in an external database and use them in LP only when required. See [“Create objects using external data”](#).

For details on attributes, see [“Attribute object”](#).

2.5.3 Specifics of extracting data from bodies

There are no attributes for bodies. All body information can only be stored as an **event** object. Accordingly, the extracted data is not stored in the Redis database. Information about bodies can only be saved to the database of the Events service when the corresponding option is enabled.

Just like faces, events can be aggregated. Detailed information about aggregation is described in the [“Aggregation”](#) section.

2.5.4 Services interaction during extraction

Below is a diagram of the interaction of services when performing extraction.

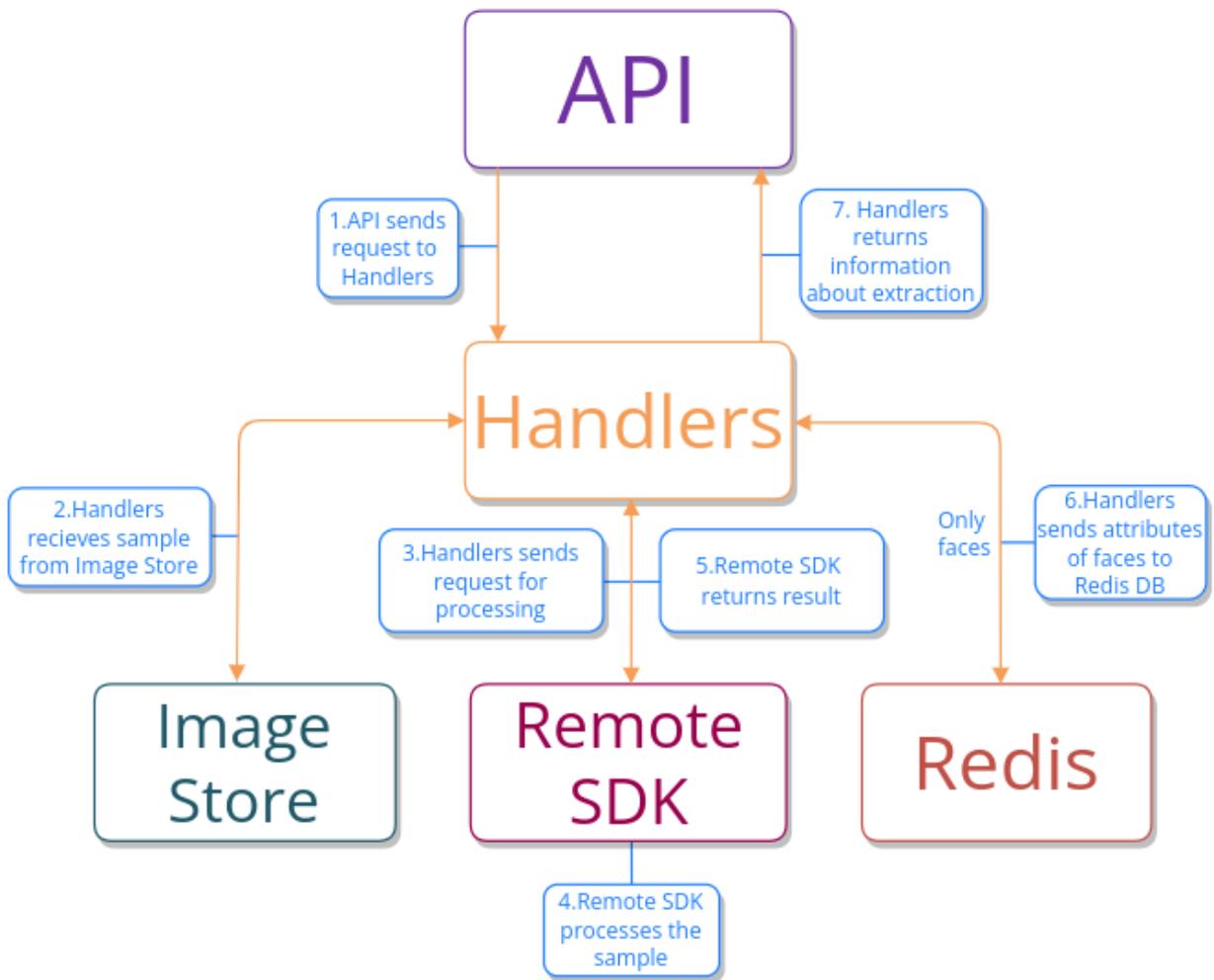


Figure 9: Services interaction during extraction

2.5.5 Requests to perform extraction

Below are the main requests where you can extract data from images of faces or bodies.

2.5.5.1 Requests “create handler” and “generate events”

These requests are used in the approach “Parallel performing of requests”.

- Request “create handler”:

Parameters name in request body	Response body	Saving to DB
Face descriptor: “extract_face_descriptor” of “extract_policy”	Handler ID containing extraction rules	Handler information is stored in Handlers database
Body descriptor: “extract_body_descriptor” of “extract_policy”	Similar to the description above	Similar to the description above
Face basic attributes: “extract_basic_attributes” of “extract_policy”	Similar to the description above	Similar to the description above

- Request [“generate events”](#):

Request body	Response body	Saving to DB
Image	Extraction results	<p>If the “store_event” parameter is enabled in the handler, then the face/body descriptor will be stored in the in the “face_descriptor” or “body_descriptor” tables of the Events database, and the basic attributes are stored in the “event” table.</p> <p>If the “store_face” parameter is enabled in the handler, then a face will be created and the corresponding face descriptor will be stored in the in the “descriptor” table of the Faces database, and the basic attributes are stored in the “event” table.</p> <p>If the “store_attribute” parameter is enabled in the handler, then the face’s descriptor and basic attributes will be stored into the Redis database for the TTL period.</p>

It is possible to use the “multipart/form-data” schema in the request. Using a schema enables you to send multiple images at the same time, specify additional information for an event, and more. See [“Use”multipart/form-data” schema when generating event”](#) for more information.

2.5.5.2 Request “extract attributes”

The [“extract attributes”](#) request is used in the [“Sequential performing of requests”](#) approach. This request cannot be used for body sample.

Query parameter	Request body	Response body	Saving to DB
“extract_descriptor” and “extract_basic_attributes”	Image	Extraction results	Temporary attributes are stored in the Redis database and removed from there after the TTL period has expired. When a face is created, the descriptor is stored in the “descriptor” table of the Faces database , and the basic attributes is stored in the “attributes” field

To save information in the Faces database, you need to link attributes to a face using the “[create face](#)” request. Otherwise, after the TTL expires, the face’s temporary attributes will be removed from the Redis database. It is not possible to save a face to the Events database using the “extract attributes” + “create face” request combination. To do this, you need to use the “Parallel performing of requests” approach (see above).

2.5.5.3 Request “sdk”

The “[sdk](#)” request also enables you to extract basic attributes or descriptor. This request returns face or body descriptor in a specific format that can be used when matching “raw” descriptors. See “[Sdk resource](#)” for details.

2.5.6 Extraction results

The following information is returned in responses to extraction requests:

- Identifier of the temporary face attribute (not returned if the “store_attribute” parameter of the “attribute_policy” is disabled in the handler).
- Address to the temporary face attribute stored in the Redis database (not returned if the “store_attribute” parameter of the “attribute_policy” is disabled in the handler).
- Face basic attributes.
- Face/body descriptor quality.
- Set of identifiers of samples for which the extraction was performed.

2.6 Matching

Given the presence of descriptors, LP enables you to search for similar faces or bodies in the database by comparing the given descriptor with descriptors stored in the Redis, Faces, Events databases or with descriptors passed directly.

The matching is done using the Python Matcher service. In response to the matching, the degree of similarity is returned, the value of which lies in the range from 0 to 1. A high degree of similarity means that two descriptors belong to the same person.

There are two types of descriptors — face descriptor and body descriptor. Body descriptor matching is not as accurate as face descriptor matching. With a large number of candidate events, the probability of false determinations of the best matches is higher than with a large number of candidate faces.

The sources of matched objects are represented as **references** (the objects you want to compare) and **candidates** (the set of objects that you want to compare with). Each reference will be matched with each of the specified candidates.

Matching cannot be performed if there is no descriptor for any of the compared faces.

You can select the following objects as references and candidates.

References:

- Attributes
- Faces
- External IDs of faces and events
- Events (face or body)
- Event track IDs
- Descriptors

Candidates:

- Faces
- Events (face or body)
- Attributes
- Descriptors

The references are specified using IDs of the corresponding objects. If a non-existent reference is set (for example, a non-existent ID is set in the “event_id” field or the “face_id” field), the corresponding error is returned.

The candidates are specified using filters. Matching results are returned for the candidates that correspond to the specified filters. If none of the candidates corresponds to the filters (for example, a non-existent ID is set in the “event_ids” field or the “face_ids” field), there will be no matching result and no error returned. The result field will be empty.

2.6.1 Matching process

Below is the process of performing matching using faces as candidates and references:

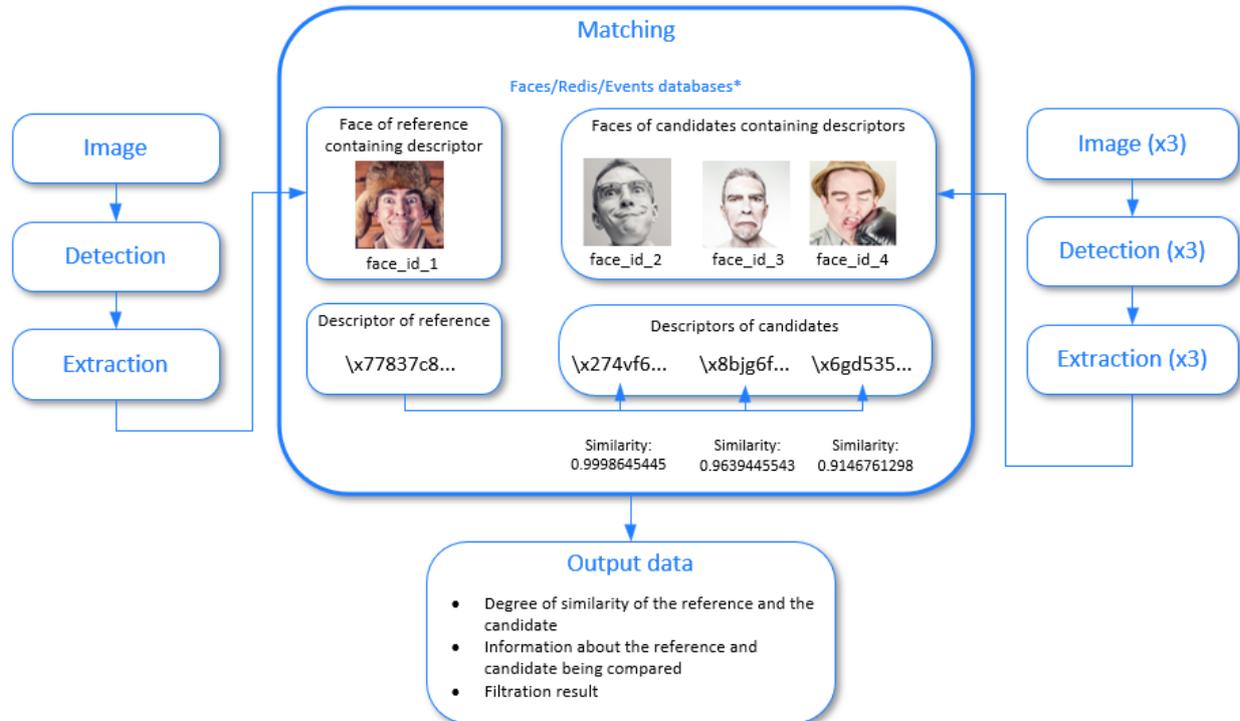


Figure 10: Matching process

* Descriptors can also be passed directly using “[raw matching](#)” request.

2.6.2 Service interaction during matching

Below is a diagram of the interaction of services when performing matching.

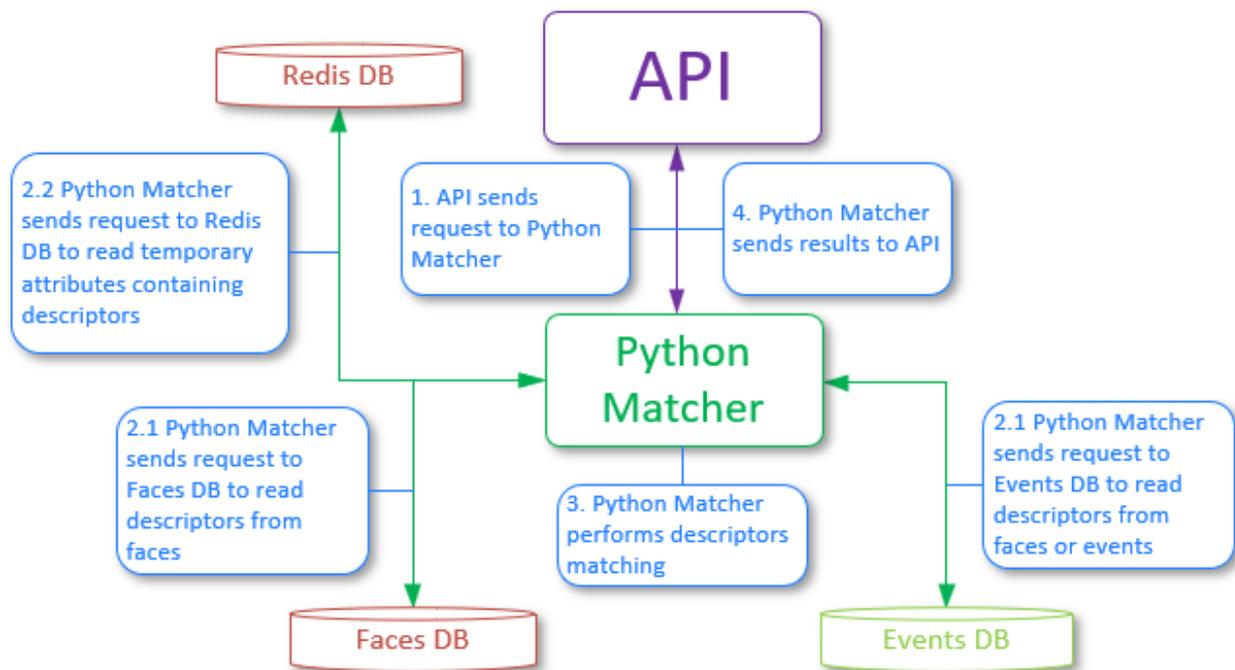


Figure 11: Service interaction during matching

See “[Matching diagrams](#)” for more service interaction scenarios when performing a matching.

2.6.3 Filtering of matching results

The results of the matching can be filtered. To filter, you need to specify the appropriate request parameters.

Several examples for events and faces filters are given below.

For events filtration one can:

- Specify camera (“source” field) and a period (“create_time__gte” and “create_time__lt” or “end_time__gte” and “end_time__lt” fields).
- Specify tags (“tags” field) for events as filters and perform matching for events with these tags only.
- Specify geographical area and perform matching with the events created in this area only. You can specify direct location (city, area, district, street, house) or a zone specified using geographical coordinates.

For faces filtration one can:

- Specify a list of external face IDs to perform matching with them.
- Specify a list ID to perform matching by list.

See the “[matcher](#)” > “[matching faces](#)” and “[matcher](#)” > “[human body matching](#)” sections in the API service reference manual for details about all the available filters.

2.6.4 Requests to perform matching

Below are the main requests where descriptors can be matched.

2.6.4.1 Requests “create handler” and “generate events”

These requests are used in the approach “Parallel performing of requests”.

- Request “create handler”:

Policy name in request body	Response body	Saving to DB
“match_policy”	Handler ID containing the matching rules	Handler information is stored in the Handlers database

- Request “generate events”:

Request body	Response body	Saving to DB
Image	Matching results	If the “store_event” parameter is enabled in the handler, then the match results will be stored in the “face_match_result” or “event_match_result” tables of the Events database

It is possible to use the “multipart/form-data” schema in the request. Using a schema enables you to send multiple images at the same time, specify additional information for an event, and more. See “[Use”multipart/form-data” schema when generating event](#)” for more information.

2.6.4.2 Requests “matching faces” and “matching bodies”

The “matching faces” and “matching bodies” requests are used in the “[Sequential performing of requests](#)”.

Request body	Response body	Saving to DB
Candidates, references and filters are specified in the parameters “candidates” and “references”	Matching results	No

2.6.4.3 Request “raw matching”

You can set descriptors as references and candidates in [SDK, Raw and XPK files](#) using the “raw matching” request.

All the descriptors data is provided with the request hence you can use this request when you need to match descriptors that are not stored in LUNA PLATFORM databases.

In LUNA PLATFORM, it is possible to extract descriptor in SDK format (see the “[Descriptor formats](#)” section).

2.6.5 Matching results

The following information is returned in responses to match requests:

- information about the reference
- information about the candidate
- the degree of similarity of each candidate with the reference
- filtered candidates

For more information about descriptor matching services, see the “[Matching services](#)” section.

2.6.6 Verification

You can use the “[/verifiers](#)” resource to create a special handler for verification. It includes several policies for the incoming images processing. See the “[Verifiers description](#)” section for details on the verifier.

Generally this request is used to match one given object with the incoming object. Use other matching requests for identification tasks.

2.6.7 Matching a large set of descriptors

Sometimes it is necessary to match a very large set of descriptors (candidates). When matching a large set of descriptors by classical brute-force matching, it is impossible to get a low latency with a high number of requests per second. Therefore, it is required to use approximation methods implemented in LIM that exchange some accuracy for high speed. These methods speed up the matching by building an index containing preprocessing data.

The work of the module is as follows: The index is created using a task containing a “list_id” with linked faces, by which the matching will be made. You can also not specify “list_id”, but set the settings for automatic indexing of all lists whose number of faces exceeds the value specified in a certain setting. After the index is created, the user sends a request to the API service, which redirects it to the Python Matcher Proxy service. The Python Matcher Proxy service determines where the matching will be performed — in the Python Matcher service or in the Indexed Matcher service. After matching, the response is returned to the user.

LUNA Index Module is licensed separately and is available on request to VisionLabs. The module delivery contains all the necessary documentation and the Docker Compose script.

2.7 Stored data and LP objects

This section describes data stored by LUNA PLATFORM 5.

This information can be useful when utilizing LUNA PLATFORM 5 according to the European Union legal system and GDPR.

This section does not describe the legal aspects of personal data utilization. You should consider which stored data can be interpreted as personal data according to your local laws and regulations.

Note that combinations of LUNA PLATFORM data may be interpreted by law as personal data, even if data fields separately do not include personal data. For example, a Face object including “user_data” and descriptor can be considered personal data.

Objects and data are stored in LP upon performing certain requests. Hence it is required to disable unnecessary data storage upon the requests execution.

It is recommended to read and understand this section before making a decision on which data should be received and stored in LUNA PLATFORM.

This document considers usage of the resources listed in the OpenAPI specification and creation of LUNA PLATFORM 5 objects only. The data created using Backport 3 and Backport 4 services is not considered in this section.

2.7.1 Source images

Photo images are general data sources for LP. They are required for [samples](#) creation and Liveness check.

You can provide images themselves or URLs to images.

After normalization of the source images, samples are saved in JPG format by default, even if the source image is sent in PNG format. If necessary, you can change the format of the stored samples using the “[default_image_extension](#)” setting of the Image Store service.

It is not recommended to send rotated images to LP as they are not processed properly and should be rotated. You can rotate the image using LP in two ways:

- By enabling the “use_exif_info” auto-orientation parameter by EXIF data in the query parameters.
- By enabling the “LUNA_HANDLERS_USE_AUTO_ROTATION” auto-orientation setting in the Configurator settings.

More information about working with rotated images can be found in the [Nuances of working with services](#) section.

See the detailed information about the requirements for the source images in the “[Image format requirements](#)” section.

2.7.1.1 Source images usage

Source images can be specified for processing when performing POST requests on the following resources:

- “/sdk”
- “/detector”
- “/handlers/{handler_id}/events”
- “/verifiers/{verifier_id}/verifications”
- “/liveness”

2.7.1.2 Source images saving

Generally, it is not required to store source images after they are processed. They can be optionally stored for system testing purposes or business cases, for example, when the source image should be displayed in a GUI.

Source images can be stored in LP:

- Using the POST request on “/images” resource.
- During the the POST request on “/handlers/{handler_id}/events” resource. Source images are stored if the “store_image” option is enabled for “image_origin_policy” of the “storage_policy” during the handler creation using the “/handlers” resource.

Samples are stored in [buckets](#) of the Image Store for an unlimited time.

Samples are stored in buckets:

- “visionlabs-samples” is the name of the bucket for faces samples.
- “visionlabs-bodies-samples” is the name of the bucket for human bodies samples.

Paths to the buckets are specified in the “bucket” parameters of “LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS” and “LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS” sections in the Configurator service.

Saving metadata with an image

In the resource “/images”, user metadata can be saved along with the image using the header X-Luna-Meta-`<user_defined_key>` with the value `<user_defined_value>`. In the Image Store bucket, metadata is saved in a separate file `<image_id>.meta.json`, which is located next to the source image.

In the response to the “get image” request, you need to specify the header `with_meta=1` to get the image metadata in the response header.

To store metadata values for multiple keys, multiple headers must be set.

2.7.1.3 Source images deletion

Source images are stored in the bucket for an unlimited period.

You can delete source images from the bucket:

- Using the DELETE request on the “/images/{image_id}” resource.
- Manually, by deleting the corresponding files from the bucket.

2.7.2 Sample object

2.7.2.1 Samples usage

Separate samples are created for face and body.

Samples are required:

- For basic attributes estimation
- For face, body and image parameters estimation
- For face and body descriptors extraction
- When changing the descriptors NN version

When the neural network is changed, you cannot use the descriptors of the previous version. A descriptor of a new version can be extracted if the source sample is preserved only.

Samples can also be used as avatars for faces and events. For example, when it is required to display the avatar in GUI.

Samples are stored in [buckets](#) of the Image Store.

Samples should be stored until all the required requests for face, body parameters estimation, basic attributes estimation, and descriptors extraction are finished.

2.7.2.2 Samples creation and saving

Samples are created upon the face and/or human body detection in the image. Samples are created during the execution of the following requests:

- POST on the “/detector” resource. Samples are created and stored implicitly. The user does not affect their creation.
- POST on the “/handlers/{handler_id}/events” resource. Samples are stored if the “store_sample” option is enabled for “face_sample_policy” and “body_sample_policy” of the “storage_policy” during the handler creation using the “/handlers” resource.
- POST on the “/verifiers/{verifier_id}/verifications” resource. Samples are stored if the “store_sample” option is enabled for “face_sample_policy” of the “storage_policy” during the verifier creation using the “/verifiers” resource. Human body samples are not stored using this resource.

External samples saving

Samples can be provided to LP directly from external VisionLabs software (e. g., FaceStream). The software itself performs sample creation from the source image.

You can manually store external samples in LP (an external sample should be provided in the requests):

- Using the POST request on the `“/samples/{sample_type}”` resource.
- When the `“warped_image”` option is set for the POST request on the `“/detector”` resource.
- When the `“image_type”` option is set to `“1”` (face sample) or `“2”` (human body sample) in the query parameters for the POST request on the `“/handlers/{handler_id}/events”` resource. The `“store_sample”` option should be enabled for `“face_sample_policy”` and `“body_sample_policy”` of the `“storage_policy”`.

2.7.2.3 Samples saving disabling

There is no possibility to disable samples saving for the request on the `“/detector”` resource. You can delete the created samples manually after the request execution.

The `“/handlers”` resource provides `“storage_policy”` that allows you to disable saving following objects:

- Face samples. Set `“face_sample_policy” > “store_sample”` to `“0”`.
- Human body samples. Set `“body_sample_policy” > “store_sample”` to `“0”`.

The `“/verifiers”` resource provides `“storage_policy”` that allows you to disable saving the following objects:

- Face samples. Set `“face_sample_policy” > “store_sample”` to `“0”`.

2.7.2.4 Samples deletion

You can use the following ways to delete face or body samples:

- Perform the DELETE request to `“/samples/faces{sample_id}”` resource to delete a face sample by its ID.
- Perform the DELETE request to `“/samples/bodies{sample_id}”` resource to delete a body sample by its ID.
- Manually delete the required face or body samples from their bucket.
- Use `“remove_samples”` parameter in the `“gc task”` when deleting events.

2.7.2.5 Getting information about samples

You can get face or body sample by ID.

- Perform the GET request to `“/samples/faces/{sample_id}”` resource to get a face sample by its ID.
- Perform the GET request to `“/samples/bodies/{sample_id}”` resource to get a body sample by its ID.

If a sample was deleted, then the system will return an error when making a GET request.

2.7.3 Descriptor

The basic description and use of descriptors is described in the `“Extraction”` section above.

See the additional information in the [“Descriptor formats”](#) section.

2.7.4 Attribute object

This object is intended to work with faces only.

It is recommended to introduce yourself with the [“Extraction”](#) section before reading.

Attributes are temporary objects that include basic attributes and face descriptors. This data is received after the sample processing.

Basic attributes include the following personal data:

- **age**. The estimated age is returned in years.
- **gender**. The estimated gender: 0 — female, 1 — male.
- **ethnicity**. The estimated ethnicity.

You can disable basic attributes extraction to avoid this personal data storage.

A descriptor cannot be considered personal data. You cannot restore the source face using a descriptor.

2.7.4.1 Attributes usage

Attribute object can be used in the following cases:

- Matching using attributes can be performed using:
 - [“/matcher/faces”](#) resource
 - [“/tasks/cross_match”](#) resource
 - [“/verifiers/{verifier_id}/verifications”](#) resource

As attributes have TTL (by default, it is set to 5 minutes), it is convenient to use them for verification or identification purpose. They will be deleted soon after you receive the result.

- Attributes may be used for ROC-curves creation using the [“/tasks/roc”](#) resource (see the [“ROC-curve calculating task”](#) section).
- You can save the data of the existing attribute to a face using the [“/faces”](#) resource.

It is not the only way to save descriptors and basic attributes to a face. You can use [“/handlers/{handler_id}/events”](#) to create a face and add the extracted descriptor and basic attributes to it.

Basic attributes saved in faces or events objects can be used for filtration of the corresponding objects during requests execution.

Descriptors are required for matching operations. You cannot compare two faces or bodies without their descriptors.

2.7.4.2 Attributes creation and saving

Attributes can be created when sending requests on the following resources:

- [“/extractor”](#)

The “extract_basic_attributes” and “extract_descriptor” query parameters should be enabled for extraction of the corresponding data. Attributes are implicitly stored after the request execution.

- [“/handlers/{handler_id}/events”](#)

The “extract_basic_attributes”, “extract_face_descriptor”, and “extract_body_descriptor” parameters should be enabled in the [handler](#) for extraction of the corresponding data. The “storage_policy” > “attribute_policy” > “store_attribute” option should be enabled in the [handler](#) for attributes storage.

- [“/verifiers/{verifier_id}/verifications”](#)

The “extract_basic_attributes” parameter should be enabled in the [verifier](#) for extraction of the corresponding data. The “storage_policy” > “attribute_policy” > “store_attribute” option should be enabled in the [verifier](#) for attributes storage.

Attributes can be created using external descriptors and external basic attributes using the [“/attributes”](#) resource.

2.7.4.3 Attributes time to live

Attributes have a TTL. After the TTL expiration, attributes are automatically deleted. Hence, it is not required to delete attributes manually.

The default TTL value can be set in the “default_ttl” parameter in the Configurator service. The maximum TTL value can be set in the “max_ttl” parameter in the Configurator service.

TTL can be directly specified in the requests on the following resources:

- [“/extractor”](#) in the “ttl” query parameter
- [“/handlers”](#) in the “storage_policy” > “attribute_storage” in the “ttl” parameter

2.7.4.4 Attributes extraction disabling

You can disable basic attributes extraction by setting the “extract_basic_attributes” parameter to “0” in the following resources:

- [“/extractor”](#)
- [“/handlers”](#)
- [“/verifiers”](#)

You can disable descriptor extraction by setting the “extract_descriptor” parameter to “0” in the following resources:

- [“/extractor”](#)
- [“/handlers”](#)

2.7.4.5 Attributes saving disabling

You can disable the “storage_policy” > “attribute_policy” > “store_attribute” parameter in the [“/handlers”](#) resource to disable attributes storage. When this handler is used for the [“/handlers/{handler_id}/events”](#) resource, attributes are not saved even for the specified TTL period.

You can disable the “storage_policy” > “attribute_policy” > “store_attribute” parameter in the [“/verifiers”](#) resource to disable attributes storage. When this verifier is used for the [“/verifiers/{verifier_id}/verifications”](#) resource, attributes are not saved even for the specified TTL period.

2.7.4.6 Attributes deletion

Attributes are automatically deleted after the TTL expiration.

Perform the DELETE request to [“/attributes/{attribute_id}”](#) resource to delete an attribute by its ID.

2.7.4.7 Getting information about attributes

You can receive information about existing temporary attributes before TTL has not expired.

- Perform the GET request to [“/attributes/{attribute_id}”](#) resource to receive information about a temporary attribute by its ID.
- Perform the GET request to [“/attributes”](#) resource to receive information about previously created attributes by their IDs.
- Perform the GET request to [“/attributes/{attribute_id}/samples”](#) resource to receive information about all the temporary attribute samples by the attribute ID.

If any attribute TTL has not expired, the attribute data is returned. Otherwise, no data is returned for this attribute in the response.

2.7.5 Face object

Faces are changeable objects that include information about a single person.

The following general data is stored in the face object:

- Descriptor (“descriptor”)
- Basic attributes (“basic_attributes”)
- Avatar (“avatar”)
- User data (“user_data”)
- External ID (“external_id”)
- Event ID (“event_id”)

- Sample ID (“sample_id”)
- List ID (“list_id”)
- Account ID (“account_id”)

See the [“Faces database description”](#) section for additional information about the face object and the data stored in it.

[Attributes](#) data can be stored in a face. Basic attributes data, descriptor data, and information about samples are saved to the Faces database and linked with the face object.

When you delete a face, the linked attributes data is also deleted.

- **Descriptor.** You should specify a descriptor for the face if you are going to use the face for comparison operations.

You cannot link a face with more than one descriptor.

- **Basic attributes:** Basic attributes can be used for displaying information in GUI.

The face object can also include IDs of samples used for the creation of attributes.

General event fields description:

- “user_data”. This field can include any information about the person.
- “avatar”. Avatar is a visual representation of the face that can be used in the user interface.
This field can include a URL to an external image or a sample that is used as an avatar for the face.
- “external_id”. The external ID of the face.

You can use the external ID to work with external systems.

You can use the external ID to specify that several faces belong to the same person. You should set the same external ID to these faces upon their creation.

You can get all the faces that have the same external ID using the [“faces” > “get faces”](#) request.

- “event_id”. This field can include an ID of the event that gave birth to this face.
- “list_id”. This field can include an ID of the [list](#) to which the face is linked.

A face can be linked to one or several lists.

- “account_id” — the account ID to which the face belongs.
- “sample_id” — One or several samples can be linked to a face. It should be the samples used for attributes extraction. All the samples must belong to the same person. If no samples are save for a face, you cannot update its descriptor to a new NN version.

IDs do not usually include personal information. However, they can be related to the object with personal data.

2.7.5.1 Faces usage

Faces usually include information about persons registered in the system. Hence, they are usually required for verification (the incoming descriptor is compared with the face descriptor) and identification (the incoming descriptor is compared with several face descriptors from the specified list) purposes.

If there are no faces stored in your system, you cannot perform the following operations with these objects:

- Matching by faces and lists, when faces are set as candidates or references in the request to the [“/matcher/faces”](#) resource.
- Matching by faces and lists, when faces are set as candidates in the matching policy of the request to the [“/handlers”](#) resource.
- Cross-matching tasks, when faces are set as candidates or references in the request to the [“/tasks/cross_match”](#) resource.
- Clustering tasks, when faces are set as filters for clustering in the request to the [“/tasks/clustering”](#) resource.
- Verification requests, when faces IDs are set in query parameters in the request to the [“/verifiers/{verifier_id}/verifications”](#) resource.
- ROC curves creation tasks ([“/tasks/roc”](#) resource).

2.7.5.2 Faces creation and saving

Faces can be created using the following requests:

- [“create face”](#)

You can specify attributes for the face using one of several ways:

- By specifying the attribute ID of the temporary attribute.
- By specifying descriptors and basic attributes (with or without samples).
- By specifying descriptors (with or without samples).
- By specifying basic attributes (with or without samples).

The last three ways are used when you need to create a face using data stored in external storage.

- [“generate events”](#)

The `“storage_policy” > “face_policy” > “store_face”` parameter should be enabled in the utilized handler.

2.7.5.3 Requests for working with faces

Below are the requests for working with faces:

Request	Description
“create face”	Create face.
“get faces”	Get information about faces according to filters. You can specify the list ID as a filter to get information about all the faces associated with the list.
“delete faces”	Delete several faces by their IDs.
“get face count”	Get information about the number of previously created faces according to filters.
“get count of faces with attributes”	Get information about the number of previously created faces with attached attributes.
“get face”	Get information about faces by its ID.
“patch face”	Update the “user_data”, “external_id”, “event_id”, “avatar” fields of an already created person.
“remove face”	Delete face by its ID.
“check if face exists”	Check if face with the specified ID exists.

2.7.6 List object

A list is an object that can include faces that correspond to a similar category, for example, customers or employees.

Lists include the “description” field that can store any required data to distinguish lists from each other.

2.7.6.1 Lists usage

You can add faces to lists for dividing the faces into groups.

Only faces can be added to lists.

Lists IDs are usually specified for matching operations as a filter for faces.

2.7.6.2 Requests for working with lists

Below are the requests for working with lists:

Request	Description
“create list”	Create list.

Request	Description
“get lists”	Get information about all previously created lists according to filters.
“delete lists”	Delete several lists by their IDs.
“get list count”	Get information about the number of previously created lists according to filters.
“get list”	Get information about the list by its ID.
“check if list exists”	Check if the list with the specified ID exists.
“update list”	Update the “user_data” field of the already created list.
“delete list”	Delete the list by its id.
“attach/detach faces to the list”	Attach/detach specific face IDs to the list.

2.7.7 Event object

Events are used in the “[Parallel performing of requests](#)” approach.

Events are immutable objects that include information about a single face and/or body. They are received after images processing using [handlers](#) or created manually.

Unlike faces, events cannot be changed after creation. The only exception is the event descriptor. It can be updated to the new neural network version.

Generally an event is created for each face/body detected in the image. If the detected face and body belong to the same person they are saved to the same event.

LP also provides possibility to create new events manually without processing using handlers. It is used in the cases when a logic for filling in event fields should differ from the logic using handlers. For example, when you want to extract descriptors only for a part of the detections and not for all the detections.

An event can be linked with a descriptor stored in the Events database.

Since there can be millions of events in the Events database, it is recommended to use a high performance column database. You can also use a relational database, but this will require additional configuration.

The following general data is stored in the event object:

- Field “source”. This field can include the source from which the face or human body was received. The value is specified during the “[generate events](#)” request execution.

- “location”. This group of parameters includes information about the location where the event occurred. The values are specified during the “[generate events](#)” request execution. The following fields are included in this group:
 - “city”
 - “area”
 - “district”
 - “street”
 - “house_number”
 - “geo_position” — latitude and longitude.
- Field “tag”. This field includes a tag (or several tags) applied during the “conditional_tags_policy” execution. The tag(s) can be specified during the “create handler” request execution or the “[generate events](#)” request execution.
- Field “emotion”. This field includes the predominant emotion estimated for the face. If necessary, you can disable the “estimate_emotions” parameter in the “[/handlers](#)” resource or “[create verifier](#)” request.
- Field “insert_time”. This field includes the time when the face appeared in the video stream. This data is usually provided by external systems.
- Field “top_similar_object_id”. This field includes the top similar object ID.
- Field “top_similar_external_id”. This field includes the external ID of the top similar candidate (event or face) with which the face is matched.
- Field “top_matching_candidates_label”. This field includes a label applied during the “match_policy” execution. The labels are specified in this policy in the “[/handlers](#)” resource.
- Field “face_id”. Events include the ID of the face to which the event gave birth.
- Field “list_id”. Events include the ID of the list to which the created face was attached.
- Field “external_id”. This external ID will be specified for the face created during the event request processing. The value is specified during the “[generate events](#)” request execution.
- Field “user_data”. This field can include any information. User data will be specified for the face created during the event request processing. The value is specified during the “[generate events](#)” request execution.
- Fields “age”, “gender”, and “ethnic_group”. The basic attributes extraction is specified in the “extract_policy” of the “[/handlers](#)” resource.
- Face and body descriptors. Events can be used for matching operations as they include descriptors.
- Information about faces, bodies and events that were used for matching with the events.
- Information about faces and human bodies detection results that include:

- IDs of created samples
- Information about bounding boxes of detected faces/bodies
- Time of face/body event detection (field “detect_time”). The time is returned from an external system.
- URL of the source image where the face/body was detected (“image_origin”)

IDs do not usually include personal information. However, they can be related to the object with personal data.

See the [“Events database description”](#) section for additional information about the event object and the data stored in it.

Attributes aggregation for events

It is possible to enable attributes aggregation for the incoming images when creating an event.

According to the specified settings faces and bodies are detected in the incoming images. If faces are detected and an aggregated descriptor creation is specified, then a single descriptor is created using all the found faces. The same logic is used for the aggregated body descriptor creation. In addition, the basic attributes are aggregated, the obtained values for the definition of Liveness, masks, emotions, upper and lower body and the body accessories.

All the information about the detected face/body and estimated properties is returned in the response separately for each image. The aggregated results are stored when an event is saved in the DB. The information about face/body detection is added to the Events database as a separate record for each image from the request.

When performing aggregation, the images in the request to the [“/handlers/{handler_id}/events”](#) resource should include only one face/body and the face/body should belong to the same person.

2.7.7.1 Events usage

Events are required to store information about persons’ occurrences in a video stream for further analysis. An external system processes a video stream and sends frames or samples with detected faces and bodies.

LP processes these frames or samples and creates events. As events include statistics about face detection time, location, basic attributes, etc., they can be used to collect statistics about the person of interest or gather general statistics using all the saved events.

Events provide the following features:

- **Matching by events**

As events store descriptors, they can be used for matching. You can match the descriptor of a person with the existing events to receive information about the person’s movements.

To perform matching, you should set events as references or candidates for matching operations (see section [“Descriptors matching”](#)).

- **Notifications via web sockets**

You can receive notifications about events using web sockets. Notifications are sent according to the specified filters. For example, when an employee is recognized, a notification can be sent to the turnstile application and the turnstile will be opened.

See the [“Sending notifications via Sender”](#) section.

- **Statistics gathering**

You can gather statistics on the existing events using a special request. It can be used to:

- Group events by frequency or time intervals.
- Filter events according to the values of their properties.
- Count the number of created events according to the filters.
- Find the minimum, maximum and average values of properties for the existing events.
- Group existing events according to the specified values of properties.

You should save generated events to the database if you need to collect statistics after the event is created.

See the [“events” > “get statistics on events”](#) request in [“APIReferenceManual.html”](#) for details.

- **User defined data**

You can add custom information to events, which can later be used to filter events. The information is passed in JSON format and written to the Events database.

See [“Events meta-information”](#) for details.

2.7.7.2 Events creation and saving

Events are created during the request to the [“/handlers/{handler_id}/events”](#) resource execution. The [“event_policy” > “store_event”](#) should be set to “1” when creating a handler using the [“/handlers”](#) resource.

For manual event creation and saving use the [“/handles/{handler_id}/events/raw”](#) resource.

The format of the generated event is similar to the format returned by the [“/handlers/{handler_id}/events”](#) resource. The [“event_id”](#) and [“url”](#) fields are not specified when creating a request. They are returned in the response after the event is created.

Notifications using web sockets are sent when events are created using this resource.

2.7.7.3 Events deletion

Events are only deleted using the garbage collection task. You should specify filters to delete all the events corresponding to the filters.

To delete an event execute the POST request to the [“/tasks/gc”](#) resource.

You can also manually delete the required events from the database.

2.7.7.4 Getting information about events

You can receive information about existing events.

- The [“events” > “get event”](#) request enables you to receive information about an event by its ID.

If the event was deleted, then the system will return an error when making a GET request.

- The [“events” > “get events”](#) request enables you to receive information about all previously created events according to the specified filters. By default, events are filtered for the last month from the current date. If any of the following filters are specified, then the default filtering will not be used.
 - List of event IDs (field “event_ids”)
 - Lower event ID boundary (field “event_id__gte”)
 - Upper event ID boundary (field “event_id__lt”)
 - Lower create time boundary (field “create_time__gte”)
 - Upper create time boundary (field “create_time__lt”)

2.7.7.5 Use “multipart/form-data” schema when generating event

In the [“generate events”](#) request, you can send an image, provide an image URL, or send a [raw descriptor](#). You can use a multipart/form-data schema to send several images in the request. Each of the sent images must have a unique name. The Content-Disposition header must contain the actual filename.

The following parameters can also be specified in the request using the “multipart/form-data” schema:

- Parameters of the bounding box of the face or body. It enables you to specify the certain zone with a face on the image
- Image detection time
- Time stamp of detection relative to the beginning of videofile
- Source image
- [User meta-information](#)

All of the above information will be saved in the event when it is generated.

2.7.8 Handler object

Handlers are used in the [“Parallel performing of requests”](#) approach.

Handler is an object that stores entry points for image processing. The entry points are called policies. They describe how the image is processed hence define the LP services used for the processing. The handler is created using the “[create handler](#)” request.

When creating a handler, it is important to ensure that only the required policies are enabled. Unnecessary policies can take into account request execution time and create gigabytes of unnecessary data on disk.

Handler policies

The table below includes all the existing handler policies. Each policy corresponds to one of the LP services listed in the “Service” column.

Table 18: Handler policies

Policy	Description	Service
detect_policy	Specifies face, body, image parameters to be estimated.	Remote SDK
extract_policy	Specifies the necessity of descriptor and basic attributes (gender, age, ethnicity) extraction. It also determines the threshold for the descriptor quality.	Remote SDK
match_policy	Specifies the array of lists for matching with the current face and additional filters for the matching process for each of the lists. The matching results can be used in “create_face_policy” and “link_to_lists_policy”.	Python Matcher
storage_policy	<p>Enables data storage in the database for:</p> <ul style="list-style-type: none"> • samples (“face_sample_policy”/“body_sample_policy”) • source images (“image_origin_policy”) • attributes (“attribute_policy”) • faces (“face_policy”) • events (“events_policy”) <p>You can specify filters for the objects saving. You can perform automatic linking to lists for faces and set TTL for attributes. You can enable and disable sending notifications via web-sockets or via webhooks (“notification_policy” and “callbacks”).</p>	Image Store, Faces, Events Handlers
conditional_tags_policy	Specifies filters for assigning tags to events.	Handlers

You can skip or disable policies if they are not required for your case. Skipped policies are executed according to the specified default values. For example, you should disable samples storage in the corresponding policy if they are not required. If you just skip the “storage_policy” in the handler, samples will be saved according to the default settings.

All the available policies are described in the [“API service reference manual”](#).

Output filtering

Many policies have filters. Filters can be specified either explicitly in the “filters” field or in parameters with the name “<estimation>_states“. If filtering is performed on the first policy, it will cause subsequent policies to stop executing as they are executed sequentially. In this case, the event will not be saved in the database, and the filtered objects will be displayed in the “filtered_detections” subblock in the response body of the event generation request.

Handlers usage

Using handler you can:

- Specify face/body detection parameters and estimated face/body parameters (see [“Source image processing and samples creation”](#)).
- Specify human body detection execution.
- Enable basic attributes and descriptors extraction (see [“Descriptor extraction and attribute creation”](#)).
- Perform descriptors comparison (see [“Descriptors matching”](#)) to receive matching results and use the result as filters for the policies.
- Configure automatic creation of faces using filters. You can specify filters according to the estimated basic attributes and matching results.
- Configure automatic attachment of the created faces to the lists using filters. You can specify filters according to the estimated basic attributes and matching results.
- Specify the objects to be saved after the image processing.
- Configure automatic tag adding for the event using filters. You can specify filters according to the estimated basic attributes and matching results.

In addition, the ability to process a batch of images using a handler is available (see the [“Estimator task”](#) section).

There are three types of handlers:

- static
- dynamic
- lambda

2.7.8.1 Static handler

Handler parameters of this type are specified when it is created, and then when [generating an event](#) or [creating an Estimator task](#), the created handler ID is specified.

2.7.8.2 Dynamic handler

Handler parameters of this type are specified when [generating an event](#) or [creating an Estimator task](#). Dynamic handlers enable you to separate technical parameters (thresholds and quality parameters that need to be hidden from frontend application users) and business logic. Dynamic handler policies are set using the “multipart/form-data” scheme in the event generation request.

Only static verifiers are available.

Dynamic handlers usage example:

You need to separate head pose thresholds from other handler parameters.

You can save the thresholds to your external database and implement the logic of automatic substitution of this data when creating events (e. g. in your frontend application).

The frontend user sends requests for events creation and specifies the required lists and other parameters. The user does not know about the thresholds and cannot change them.

2.7.8.3 Lambda handler

This handler is used when working with the Lambda service, which is intended to work with custom modules that mimic the functionality of a separate service.

All requests will be sent to the lambda handler using “lambda_id”.

If the custom Handlers-lambda supports the ability to generate an event, then the format of the body of the request and response to the resources is [“/handlers/{handler_id}/events”](#) or [“/tasks/estimator”](#) will follow the OpenAPI specification. If Handlers-lambda does not support the ability to generate an event, then a request to the resource [“/lambdas/{lambda_id}/proxy”](#) should be used with the corresponding request form.

The logic behind the request behavior is entirely dependent on the lambda written by the user.

See [“Lambda service”](#) for details.

2.7.8.4 Requests for working with handlers

Below are the requests for working with handlers:

Request	Description
“create handler”	Create handler.

Request	Description
“get handlers”	Get information about all previously created handlers according to filters.
“get handler count”	Get information about the number of previously created handlers according to filters.
“validate handler policies”	Check the handler policies before using them to create or update the handler.
“get handler”	Get information about the handler by its ID.
“replace handler”	Update the parameters of an already created handler.
“check if handler exist”	Check if the handler with the specified ID exists.
“remove handler”	Delete the handler by its ID.

2.7.9 Verifier object

Verifiers are used in the “Parallel performing of requests” approach.

The Handlers service also processes requests to create verifiers required for the verification process. They are created using the “create verifier” request.

The verifier contains a limited number of handler policies — detect_policy, extract_policy, and storage_policy.

You can specify the “verification_threshold” in the verifier.

The created verifier should be used when sending requests to:

- “/verifiers/{verifier_id}/verifications” resource. You can specify IDs of the objects with which the verification should be performed.
- “/verifiers/{verifier_id}/raw” resource. You can specify raw descriptors as candidates and references for matching. As raw descriptors are processed “verification_threshold” is the general parameter used from the specified verifier.

The response includes the “status” field. It shows, if the verification was successful or not for each pair of matched objects. It is successful if the similarity of two objects is greater than the specified “verification_threshold”.

2.7.9.1 Requests for working with verifiers

Below are the requests for working with verifiers:

Request	Description
“create verifier”	Create verifier.
“get verifiers”	Get information about all previously created verifiers according to filters.
“raw verification”	Perform verification of raw descriptors.
“perform verification”	Perform verification for the specified objects.
“count verifiers”	Get information about the number of previously created verifiers according to filters.
“get verifier”	Get information about the verifier by its ID.
“replace verifier”	Update the parameters of an already created verifier.
“check if verifier exists”	Check if the verifier with the specified ID exists.
“remove verifier”	Delete the verifier by its ID.

2.7.10 Other objects

In the Image Store service, you can store any files as objects (for example, a video file).

You can upload files using the “create objects” request. The bytes of the file must be specified in the request body, and the Content-Type header must contain the MIME type of the file (for example, video/mp4). The response to the “get object” request contains the Content-Disposition header. This header contains the file name of the attachment object (for example, video_1.mp4). The file name is generated based on the object_id and the MIME type.

If the MIME type of the file could not be determined, then the file extension will be set as .bin.

2.8 Sending notifications via Sender

You can send notifications about created events to the third-party party applications via web sockets. For example, you can configure LP to send notifications about VIP guests' arrival to your mobile phone.

When LP creates a new event, the event is added to the special DB. Then the event can be sent to the Sender service if the service is enabled.

The third-party party application should be subscribed to Sender via web sockets. The filter for the events of interest should be set for each application. Thus you will receive notifications only for the required events.

The notification about a new event is sent by Sender to the required applications.

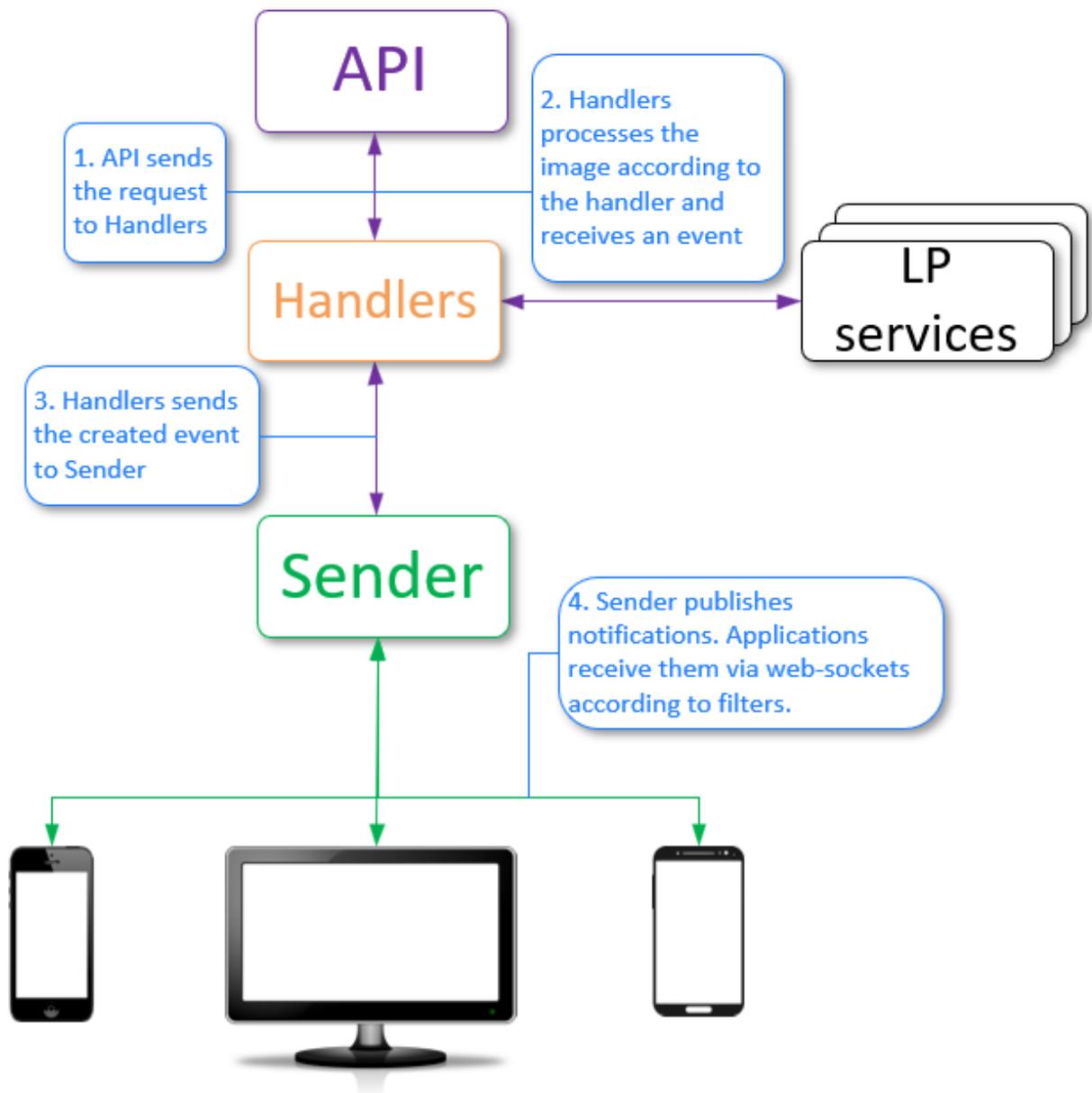


Figure 12: Sender workflow

See section “[Sender service](#)” for details about the Sender service.

2.9 Licenses

The Licenses service provides information about license terms to LP services.

See section “[Licenses service](#)” for details.

2.10 Admin

The Admin service implements tools for administrative routines.

All the requests for Admin service are described in [OpenAPI specification](#) of the Admin service.

See section [“Admin service”](#) for details about the Admin service.

2.11 Configurator

The configurator service includes the settings of all the LP services. Thus it provides the possibility to configure all the services in one place.

See section [“Configurator service”](#) for details about the Configurator service.

2.12 Tasks

Tasks requests provide additional possibilities for large data amounts processing.

The greater data array is processed, the longer it takes to finish the task. When a task is created you receive the task ID as the result of the request.

See the [“Tasks service”](#) section for details about tasks processing.

Clustering task

The clustering task provides the possibility to group events and faces that belong to the same person to clusters.

You can specify filters for choosing the objects to be processed.

Using the clustering task you can, for example, receive all the IDs of events that belong to the same person and which occurred during the specified period of time.

See the [“task processing” > “clustering task”](#) request for details about the Clustering task request creation.

See [“Clustering task”](#) for more information about this task.

Reporter task

The reporter task enables you to receive a report in CSV format with extended information about the objects grouped to clusters.

You can select columns that should be added to the report. You can receive images corresponding to each of the IDs in each cluster in the response.

See the [“task processing” > “reporter task”](#) request for details about the reporter task request creation.

See [“Reporter task”](#) for more information about this task.

Exporter task

The exporter task enables you to collect event and/or face data and export them from LP to a CSV file.

The file rows represent requested objects and corresponding samples (if they were requested).

See the [“task processing” > “exporter task”](#) request for details about the exporter task request creation.

See [“Clustering task”](#) for more information about this task.

Cross-matching task

Cross-matching means that a large number of references can be matched with a large number of candidates. Thus every reference is matched with every candidate.

Both references and candidates are specified using filters for faces and events.

See the [“task processing” > “cross-matching task”](#) request in the API service reference manual for details about the cross-matching task creation.

See [“Cross-matching task”](#) for more information about this task.

Linker task

The linker task enables you to:

- link the existing faces to lists
- create faces from the existing events and link the faces to lists

The linked faces are selected according to the specified filters.

See the [“task processing” > “linker task”](#) request in the API service reference manual for details about the cross-matching task creation.

See [“Linker task”](#) for more information about this task.

Estimator task

The estimator task enables you to perform batch processing of images using the specified policies.

In the request body, you can specify the `handler_id` of an already existing static or dynamic handler. For the dynamic `handler_id`, the ability to set the required policies is available. In addition, you can create a static handler specifying policies in the request.

The resource can accept five types of sources with images for processing:

- ZIP archive
- S3-like storage
- Network disk
- FTP server
- Samba network file system

See the [“task processing” > “estimator task”](#) request in the API service reference manual for details about the estimator task creation.

See [“Estimator task”](#) for more information about this task.

2.13 Backports

In LP 5, Backport is a mechanism of the new platform version mimicry to the older versions.

The backports are implemented for LUNA PLATFORM 3 and LUNA PLATFORM 4 by means of the Backport 3 and Backport 4 services respectively.

The backports enable you to send requests that are similar to the requests from LUNA PLATFORM 3 and LUNA PLATFORM 4 and receive responses in the appropriate format.

There are some nuances when using backports.

- Data migration to Backports is complicated.

Database structures of LUNA PLATFORM 3 and LUNA PLATFORM 4 differ from the database structure of LUNA PLATFORM 5. Therefore, additional steps should be performed for data migration and errors may occur.

- Backports have many restrictions.

Not all logic from LP 3 and LP 4 can be supported in Backports. See the [“Restrictions for working with Backport services”](#) section for more information.

- Backports are limited by available features and are not developed.

New features and resources of LUNA PLATFORM 5 are not supported when using Backports and they will never be supported. Thus, these services should only be used if it is not possible to perform a full migration to LUNA PLATFORM 5 and no new features are required.

Use case:

For example, you have a frontend application that sends requests to LUNA PLATFORM 3.

When you use the Backport 3 service, the requests to LP 3 are received by the service. They are formatted and sent to LUNA PLATFORM 5 according to its API format. LP 5 processes the request and sends the response to Backport 3. The Backport 3 service formats all the received results to the LP 3 format and sends the response.

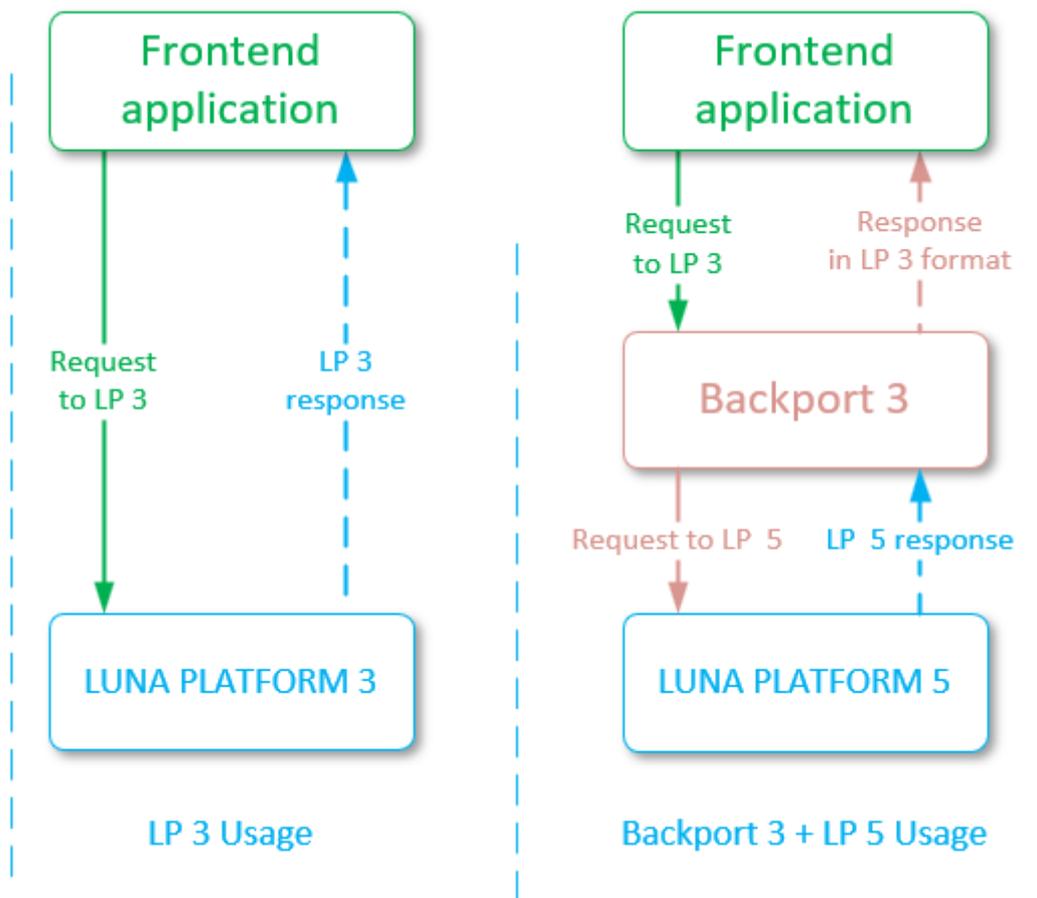


Figure 13: LP3 vs Backport 3 and LP 5

2.13.1 Restrictions for working with Backport services

As the tables of the saved data and the data itself differs for LP versions there are features and restrictions on the execution of requests. The information about features and restrictions is given in “[Backport 3](#)” and “[Backport 4](#)” sections of this document.

You can use Backport service and API service of LP 5 simultaneously following these restrictions:

- When you use Backport services, it is strongly recommended not to use the same account for requests to Backport service and the API service of LUNA PLATFORM 5.
- Perform the admin requests that do not use account ID to the LUNA PLATFORM 5 only.

2.14 Sdk resource

The sdk resource allows you to detect faces and/or human bodies and estimate attributes in input images. After the request is performed, the received data is not saved to the database and Image Store, it only returned in the response.

The sdk resource combines the capabilities of other resources and also provides additional options. For example, with `"/sdk"` request you can do the following features:

- estimate presence of glasses in the image;
- estimate liveness in the image;
- estimate face/body parameter(s);
- aggregate face/body parameter(s);
- create a face sample and return its descriptor in the SDK format encoded in Base64;
- create a human body sample and return its descriptor in the SDK format encoded in Base64;
- extract face and body descriptor(s) and return them in the response;
- set descriptor quality score threshold to filter out images that are poorly suited for further processing;
- filter by mask states;
- other.

Face or body descriptor in the SDK format encoded in Base64 can be used in requests for [matching of "raw" descriptors](#).

3 Accounts, tokens and authorization types

3.1 Account

An account is required to delimit the visibility areas of objects for a particular user. The account is necessary to perform most of the requests. Each created account has its own unique “account_id”. All account data is stored in the [Accounts service database](#) under this ID.

The account can be created using a POST request “[create account](#)” to the API service, or using the [Admin service](#) . When creating the account, you must specify the following data: login (email), password and account type.

3.1.1 Account type

The account type determines what data is available to the user.

There are three types of accounts:

- user — the type of account with which you can create objects and use only your account data.
- advanced_user — the type of account for which rights similar to “user” are available, and there is access to the data of all accounts. Access to data from other accounts means the ability to receive data (GET requests), check their availability (HEAD requests) and perform comparison requests based on data from other accounts.
- admin — the type of account for which rights similar to “advanced_user” are available, and there is also access to the [Admin service](#).

In the API service, you can work with all types of accounts, but only “advanced_user” and “user” types of accounts can be created, while in the Admin service you can create all three types.

By default, there is the “admin” type account in the system with the default login and password `root@visionlabs.ai/root`.

Using the header “Luna-Account-Id” in the “[create account](#)” request you can set the desired account ID. It should also be used if it is necessary to preserve the ability to work with data created in LP versions prior to 5.30.0 by specifying the “account_id” in the “Luna-Account-Id” header. Thus, using this parameter will link the old “account_id” to the account being created (See “Account migration” section in the LUNA PLATFORM upgrade manual for details on migration).

In response to a request to create the account, “account_id” is issued. After creating the account, you can use this ID for [LunaAccountIdAuth](#) authorization type or use [BasicAuth](#) authorization type (login/password authorization).

3.2 Token

Token is linked to an existing account with any type (“user”, “advanced_user”, “admin”) and enables you to impose extended restrictions on the requests being made. For example, when creating the token, you can give the user permission only to create and modify all lists and faces, or you can prevent the use of certain handlers by specifying their ID.

The token is created using the “[create token](#)” request to the API service.

An unlimited number of tokens can be created for each account. The token and all its restrictions are stored in the database and linked to the account by the “account_id” parameter. You cannot link one token to different accounts. To create tokens with the same permissions across different accounts, it is recommended to save the request body template for creating the account and use it.

There is no need to use both the token and the account. When working with tokens, you can restrict access to the “/accounts” resource using external means.

You can give additional restrictions on the token at any time using the “[replace token](#)” request or you can revoke the token using the “[delete token](#)” request. In this case, the token will be deleted from the database and can no longer be used for authorization.

When creating the token, you need to set the following parameters:

- expiration_time – expiration time of the token in the RFC 3339 format. You can specify an infinite token expiration time using the value “null”
- permissions – actions that the user can perform (see “[Permissions set in token](#)”)

You can also specify whether the token is visible to other accounts data using the “visibility_area” parameter (see the “[Viewing other account data](#)” section).

The response to the request is “token_id” and a Base64 encoded JWT token. After creating the token, you can use the received JWT token for [BearerAuth](#) authorization type.

3.2.1 Permissions set in token

The following permissions are available for the generated token:

Permission name	Permission description	Rights
account	Rights to use the account	view
face	Rights to use the face	creation, view, modification, deletion, matching
list	Rights to use the list	creation, view, modification, deletion
event	Rights to use the event	creation (only “ save event ” request), view, matching

Permission name	Permission description	Rights
attribute	Rights to use the attribute	creation, view, modification, deletion, matching
handler	Rights to use the handler	creation, view, modification, deletion
verifier	Rights to use the verifier	creation, view, modification, deletion
task	Rights to use the task and task schedules	creation, view, modification, deletion
face_sample	Rights to use the face sample	creation, view, deletion
body_sample	Rights to use the body sample	creation, view, deletion
object	Rights to use the object	creation, view, deletion
image	Rights to use the image	creation, view, deletion
token	Rights to use the token	view, modification, deletion
resource	Rights to use resources	"/iso", "/sdk", "/liveness"
lambda	Rights to use lambda	creation, view, modification, deletion
emit_events	Permissions to perform "generate events" requests	(see below)
custom	Custom rights	creation, view, modification, deletion

Custom rights are intended to proxy LUNA Streams requests through the LUNA API. In other cases, using user rights will have no effect. See "Proxying requests to LUNA Streams via LUNA API" in the FaceStream administrator manual.

Resources "/iso", "/sdk" and "/liveness" do not require any authorization by default.

The value [] means no permissions.

The "emit_events" permission enables you to specify whether requests can be made to the "generate events" resource, as well as blacklisting or whitelisting handler IDs. If handler IDs are blacklisted, then only their use will be prohibited. If handler IDs are present in the white list, then only their use will be allowed. The maximum number of IDs in the lists is 100. When using the "emit_events" permission, the user must not have the "creation" and "modification" Rights to use the handler.

If the "emit_events" permission is granted, then all necessary objects will be created during the generation of the event, regardless of the permissions specified in the token. For example, the "faces" type permission regulates work with faces only in requests to "faces/*" resources, but does not affect the creation of the face during event generation. Thus, when using the handler with the

“store_face” parameter and not having the “creation” permission for “faces”, the face will still be created.

When setting permissions, the following features should be taken into account:

- The “modification” permission means performing of PATCH and PUT requests.
- Linking/unlinking face to list is the “modification” permission.
- Deleting list with “with_faces” setting enabled (“delete lists” request) requires “face” > “deletion” permission.
- Linking face to list in the “create face” request requires “list” > “modification” permission.
- When performing matching, you should have the “matching” permission for the corresponding objects (event, face, attribute).
- Requests to “get statistics on events” resource require “event” > “view” permission.
- Permission “event” > “create” only grants the right to create an event using the “save event” request. To generate an event, you should use the “emit_events” permission (see above).
- Permission “event” > “create” does not apply to verifiers.

See the token coverage table for specific resources in the [developer manual](#) of the API service.

3.3 Viewing other account data

Other account details can be viewed if:

- account type is set to “advanced_user” or “admin”
- when creating the token, the “visibility_area” = “all” parameter was specified.

Account type “user” cannot be set to “visibility_area” = “all”.

If the account type is set to “advanced_user”/“admin” and the token is created with the “visibility_area” = “account” parameter set, then when logging in using the token ([BearerAuth](#)) you will no longer be able to view data from other accounts, but when logging in using login/password ([BasicAuth](#)), this option will remain.

If the account type is set to “advanced_user”/“admin” and the token is created with “visibility_area” set to “all” and then the account type is changed to “user” (using the “patch account” request), then an attempt to perform a request to view the data of other accounts using the token will result in an error.

When using the [LunaAccountIdAuth](#) authorization, the visibility area is controlled using the “Luna-Account-Id” header.

For verifiers, the ability to use the visibility area of all accounts is not available. If the “visibility_area” = “all”, only the data of your account will be visible.

3.4 Authorization types for accessing resources

There are three types of authorization available in LUNA PLATFORM:

- **BasicAuth.** Authorization by login and password (set during account creation).
- **BearerAuth.** Authorization by JWT token (issued after the token is created).
- **LunaAccountIdAuth.** Authorization by “Luna-Account-Id” header, which specifies the “account_id” generated after creating the account (this method was adopted as the main one before version 5.30.0).

LunaAccountIdAuth authorization has the lowest priority compared to other methods and can be disabled using the “ALLOW_LUNA_ACCOUNT_AUTH_HEADER” setting in the “OTHER” section of the API service settings in the Configurator (enabled by default).

In [OpenAPI specification](#) the “Luna-Account-Id” header is marked with the word **Deprecated**.

There is no need to use all three types of authorization when making requests. It is necessary to choose the preferred method depending on the required tasks.

If the authorization type is not specified in the request, an error with the status code 403 will be issued.

3.5 Credentials verifier

Using the “[verify credentials](#)” resource, you can verify existing credentials by one of the types:

- login/password verification. If verification is successful, the “account_id” and account type will be returned.
- token verification. If verification is successful, the account type and all permissions for the token will be returned.
- account ID verification. If verification is successful, the account type will be returned.

3.6 Logging account information

When executing any request, information about the corresponding account is displayed in the logs of the API service. This functionality enables you to determine who exactly executed a particular request. This may be required for information security and system administrators.

If the request was executed with BasicAuth or LunaAccountIdAuth type authorization, the following message will be displayed in the logs:

```
Request invoked by user (account_id: '270531af-e52e-4538-9181-628d9900a0db')
```

If the request was executed with BearerAuth type authorization, the following message will be displayed in the logs:

```
Request invoked by user (account_id: '270531af-e52e-4538-9181-628d9900a0db'
token_id: 'd57e16f5-e243-47d2-aa85-8b200c12d86f')
```

If the request was executed without authorization, the following message will be displayed in the logs:

```
Request invoked by user (account_id: null)
```

The logs of the Accounts service additionally display information about the creation of tokens by specific users:

```
User with account_id: '270531af-e52e-4538-9181-628d9900a0db' create token: 'd57e16f5-e243-47d2-aa85-8b200c12d86f'
```

Logging information about the creation of tokens enables you to track where the token came from and which user it belonged to, even after it was deleted.

4 Estimated data

This section lists the general parameters of faces, bodies and images estimated by LUNA PLATFORM 5 and how to get them.

You can get parameters using a some of tools and resources. Basically, getting parameters is done using the following methods:

1. **Extract gender and age** from the image. Gender and age belong to the concept of **basic attributes**.

The ["/extractor"](#), ["/sdk"](#) resources and "extract_policy" of ["/handlers"](#) and ["/verifiers"](#) resources are used to extract basic attributes from a face image.

To extract basic attributes from the face using the ["/extractor"](#) resource, you should first create a sample using the ["/detector"](#) resource. In response to the ["/extractor"](#) request, the gender and age of the person will be returned. The extracted data have a TTL (time to live) and will be removed from the database after the specified period.

See the detailed description of extracting basic attributes in the ["Descriptor extraction and attribute creation"](#) section.

When estimating parameters using the ["/sdk"](#) resource, you should send the source image to LUNA PLATFORM 5 and specify the "estimate_basic_attributes" parameter in the query parameters. In response to the request, the gender and age of the person will be received. These parameters will not be saved to the database.

To extract this parameters using the ["/handlers"](#) and ["/verifiers"](#) requests, you should use the "extract_basic_attributes" parameter of the "extract_policy".

Gender and age estimation is also available by body image. This method is not accurate and is performed by body parameters estimation (see "Perform estimation of body parameters" below).

2. **Perform estimation of face and image parameters.**

Various resources are used to estimate the parameters. The resources mainly used are ["/detector"](#), ["/sdk"](#), ["/handlers"](#) and ["/verifiers"](#).

When estimating parameters using the ["/detector"](#) resource, you need to send the source image to LUNA PLATFORM 5 and specify the estimation of the required face or image parameters in the query parameters. In response to request, a face sample will be created and the specified parameters will be given. The estimated parameters will not be saved in the database.

The method for getting parameters using the ["/sdk"](#) resource is similar to the method described above, however, the sample will not be created. The estimated parameters will also not be saved to the database.

To estimate parameters using ["/handlers"](#) and ["/verifiers"](#) resources, you should use the "detect_policy" with required parameters.

3. Perform estimation of body parameters.

The ability to perform estimation of body parameters is adjusted by a special parameter in the [LUNA PLATFORM 5 license key](#).

The `"/sdk"` and `"/handlers"` resources are used to estimate body parameters. Using these resources is similar to performing face and image estimation (see above).

4. Perform check of face and image parameters according to ISO/IEC 19794-5:2011 or custom conditions.

The ability to perform such checks is adjusted by a special parameter in the [LUNA PLATFORM 5 license key](#).

The `"/iso"` and `"/detector"` (`"estimate_face_quality"` parameter) resources and the `"face_quality"` check group of the `"detect_policy"` of the `"/handlers"` and `"/verifiers"` requests are used to perform the check.

The responses to requests show the overall result of passing all checks (`"0"` or `"1"`), as well as the results of each check.

See the detailed description of this functionality in the ["Image check"](#) section.

5. Perform Liveness estimation.

The ability to perform such estimation is adjusted by a special parameter in the [LUNA PLATFORM 5 license key](#).

6. Perform estimation of number of people in the image

All returned values and the response format depend on the resource where the estimation is performed.

In order to get results when sending requests to `"/handlers"` or `"/verifiers"` resources, you need to generate an event and perform verification on the specified handlers. See the [Handler object](#) section for more information about working with these resources.

4.1 Gender and age by face image

This estimation determines the basic attributes (gender and age) of a person in the face image.

For details on basic attributes, see [Attribute object](#).

Resources where the estimation is performed:

- `"/extractor"`

Estimation name — `"extract_basic_attributes"`.

- `"/handlers"`

Estimation name — `"policies">"extract_policy">"extract_basic_attributes"`.

- [“/verifiers”](#)

Estimation name — “policies” > “extract_policy” > “extract_basic_attributes”.

- [“/sdk”](#)

Estimation name — “estimate_basic_attributes”.

4.2 Face parameters

4.2.1 Eyes attributes

This estimation determines the following state of eyes for each of the eyes:

- “open”
- “closed”
- “occluded”

Poor quality images or ones that depict obscured eyes (for example, eyewear, hair, gestures) fall into the “occluded” category.

Iris landmarks are determined. An array of 34 landmarks is returned for each eye.

In the [“/iso”](#), [“/detector”](#) (image checking tool) and [“detect_policy”](#) > [“face_quality”](#) resources, the estimated value is also compared with threshold (according to ISO or non-standard threshold).

Resources where the estimation is performed:

- [“/detector”](#)

Estimation name — “estimate_eyes_attributes”.

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “estimate_eyes_attributes”.

- [“/verifiers”](#)

Estimation name — “policies” > “detect_policy” > “estimate_eyes_attributes”.

- [“/sdk”](#)

Estimation name — “estimate_eyes_attributes”.

Eyes attributes estimation using [image checking tools](#):

- [“/iso”](#) and [“/detector”](#) (see sections “7.2.3 Expression”, point “a”, “7.2.11 Visibility of pupils and irises” and “7.2.13 Eye patches” of the standard ISO/IEC 19794-5:2011).

Checks names — “left_eye”, “right_eye”.

- [“detect_policy”](#) > [“face_quality”](#) group of checks in the [“/handlers”](#) and [“/verifiers”](#) resources.

Checks names — “left_eye”, “right_eye”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
“left_eye” > “threshold”	[“open”]
“right_eye” > “threshold”	[“open”]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.2 Face landmarks

There are two estimations of the face landmarks:

- Estimation of 5 face landmarks
- Assessment of 68 face landmarks

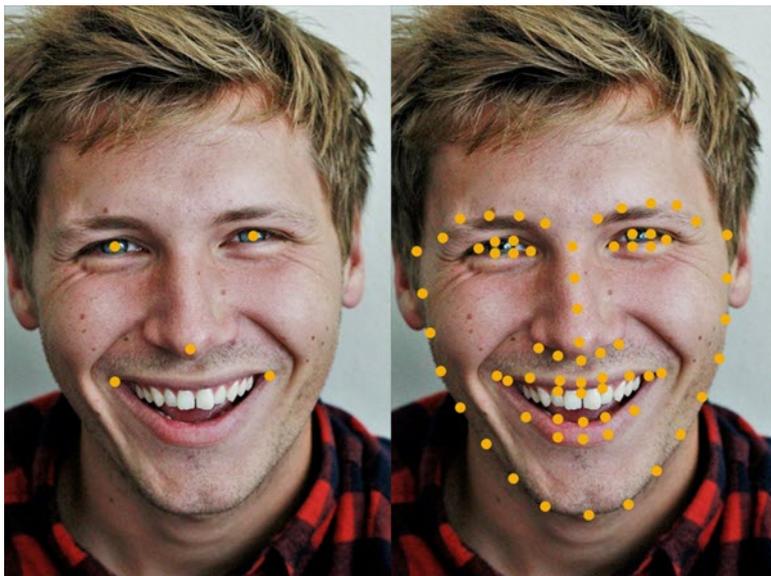


Figure 14: Estimate by 5 facial landmarks (left), estimate by 68 facial landmarks (right)

Landmarks are used for various purposes, for example, when creating the sample, when extracting the descriptor, etc. See the SDK documentation for a more detailed description of the landmarks.

- [“/detector”](#)

Estimation name — “detect_landmarks68”.

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “detect_landmarks68”.

- [“/verifiers”](#)

Estimation name — “policies” > “detect_policy” > “detect_landmarks68”.

- [“/sdk”](#)

Estimation name — “estimate_landmarks5”.

Estimation name — “estimate_landmarks68”.

4.2.3 Distance between eyes

Note: It is not possible to use a sample as an input image for this estimation.

It is possible to estimate the distance between the centers of the eyes in pixels. The estimated value is also compared with threshold (according to ISO or non-standard threshold).

This estimation can only be performed using [image checking tools](#):

- [“/iso”](#) and [“/detector”](#) (see section “5.6.5 Eye and nostril centre Landmark Points” of the standard ISO/IEC 19794-5:2011).

Check name — “eye_distance”.

- [“detect_policy”](#) > [“face_quality”](#) group of checks in the [“/handlers”](#) and [“/verifiers”](#) resources.

Check name — “eye_distance”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable range
“eye_distance” > “threshold”	[90...inf]

For the [“face_quality”](#) image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.4 Red eyes effect

This estimation determines the presence of the red eyes effect, where:

- “0” — Red eyes effect is not present in the image.
- “1” — Red eyes effect is present in the image.

The estimated value is also compared with threshold (according to ISO or non-standard threshold).

Image requirements:

For correct checking results, the following requirements should be met.

The table below shows the requirements for [image quality](#):

Parameter	Required range
“illumination”	[0.61...1]
“specularity”	[0.57...1]
“blurriness”	[0.5...1]
“dark”	[0.1...1]
“light”	[0.1...1]

The table below shows the requirement for [natural light](#):

Parameter	Required range
“natural_light”	[0.5...1]

Resources where the estimation is performed:

Red eyes effect estimation is available only using [image checking tools](#):

- “/iso” and “/detector” (see section “7.3.4 Unnatural colour” of the standard ISO/IEC 19794-5:2011)
Check name — “red_eyes”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Check name — “red_eyes”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
“red_eyes” > “threshold”	“1”

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.5 Gaze

This estimation determines the gaze. The gaze is represented by the following parameters:

- Pitch angle
- Yaw angle

A positive value of the pitch angle means looking up, and a negative value means looking down.

A positive value of yaw angle means looking to the right, and a negative value means looking to the left.

Zero position corresponds to a gaze direction orthogonally to the face plane, with the axis of symmetry parallel to the vertical camera axis.

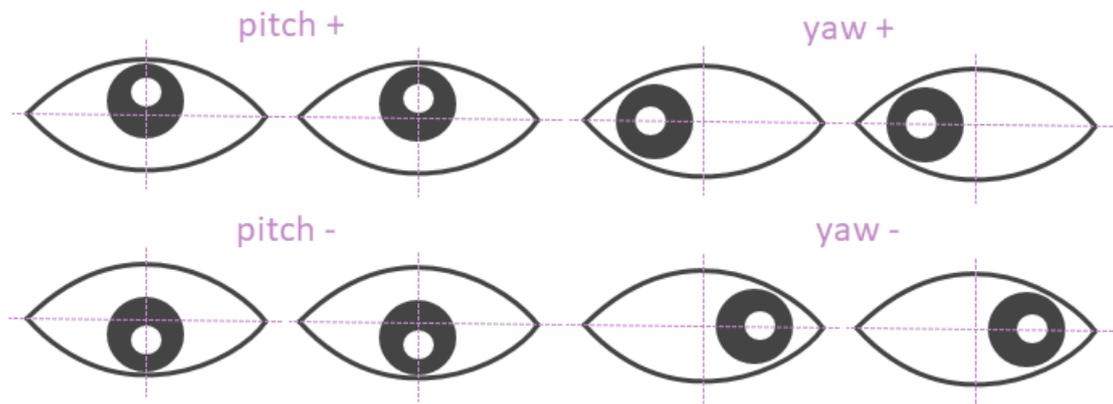


Figure 15: Gaze direction

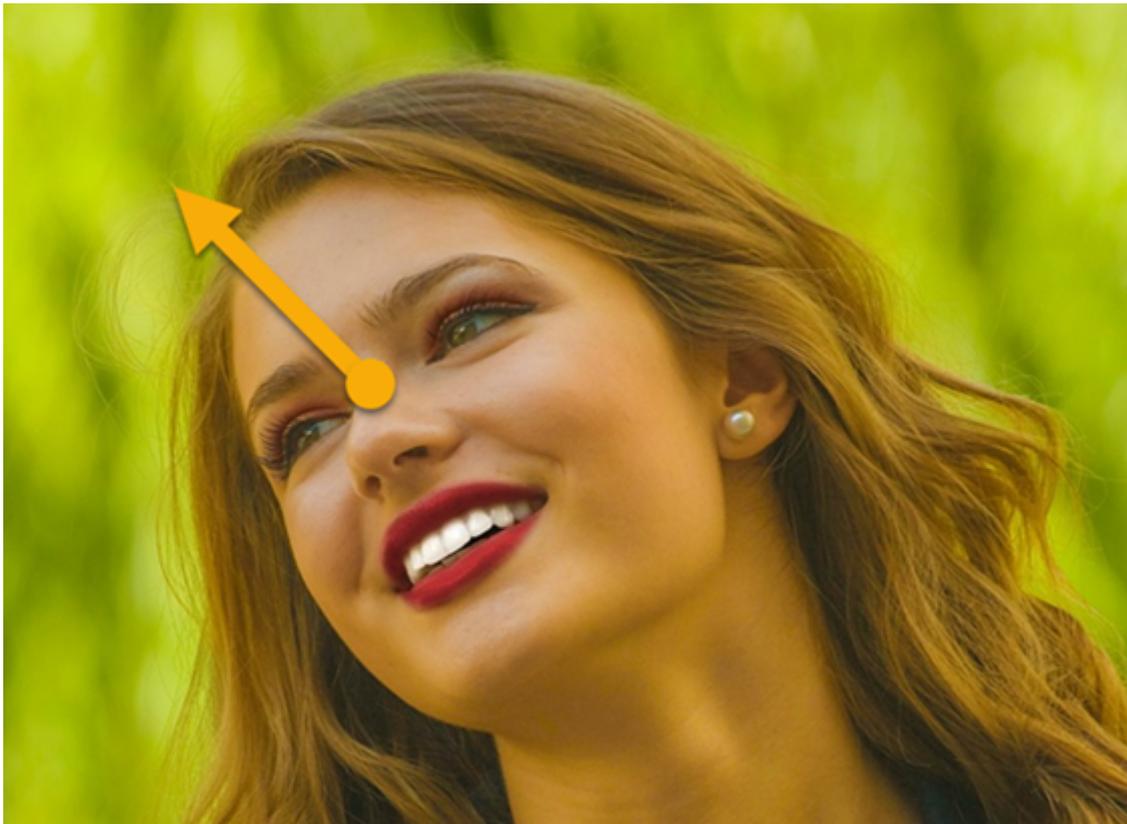


Figure 16: Gaze direction: pitch = 31.5, yaw = 31.17

In the “/iso” and “detect_policy” > “face_quality” resources, the estimated value is also compared with threshold (according to ISO or non-standard threshold).

Resources where the estimation is performed:

- “/detector”
Estimation name — “estimate_gaze”.
- “/handlers”
Estimation name — “policies” > “detect_policy” > “estimate_gaze”.
- “/verifiers”
Estimation name — “policies” > “detect_policy” > “estimate_gaze”.
- “/sdk”
Estimation name — “estimate_gaze”.

Gaze estimation using [image checking tools](#):

- “/iso” and “/detector” (see section “7.2.3 Expression” point “e” of the standard ISO/IEC 19794-5:2011)

Checks names — “gaze_yaw”, “gaze_pitch”.

- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources

Checks names — “gaze_yaw”, “gaze_pitch”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“gaze_yaw” > “threshold”	[-5...5]
“gaze_pitch” > “threshold”	[-5...-5]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.6 Glasses

This estimation determines the predominant state of the glasses from the following states:

- “sun_glasses”
- “glasses”
- “no_glasses”

In the “/iso” and “detect_policy” > “face_quality” resources, the estimated value is also compared with threshold (according to ISO or non-standard threshold).

Resources where the estimation is performed:

- “/sdk”

Estimation name — “estimate_glasses”.

Glasses estimation using [image checking tools](#):

- “/iso” and “/detector” (see section “7.2.9 Eye glasses” of the standard ISO/IEC 19794-5:2011)
Check name — “glasses”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Check name — “glasses”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable values
“glasses” > “threshold”	[“no_glasses”, “eyeglasses”]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.7 Eyebrows

This estimation determines the predominant state of the eyebrows from the following states:

- “neutral” — Eyebrows are in the usual position.
- “raised” — Eyebrows are raised.
- “squinting” — Eyes are narrowed, eyebrows are lowered.
- “frowning” — Eyebrows are frowned.

The estimated value is also compared with threshold (according to ISO or non-standard threshold).

It is possible to specify several eyebrow states as acceptable.



Figure 17: Eyebrows state. From left to right — neutral, raised, squinting, frowning

Image requirements:

For correct checking results, the following requirements should be met.

The table below shows the requirements for [head pose](#):

Parameter	Required range
pitch	[-20...20]
roll	[-20...20]

Parameter	Required range
yaw	[-20...20]

The table below shows the requirement for [face width](#):

Parameter	Required range
face_width	> 80

Resources where the estimation is performed:

Eyebrows estimation is available only using using [image checking tools](#):

- ["/iso"](#) and ["/detector"](#) (see section "7.2.3 Expression", points "d", "f" and "g" of the standard ISO/IEC 19794-5:2011)
Check name — "eyebrows_state".
- ["detect_policy"](#) > ["face_quality"](#) group of checks in the ["/handlers"](#) and ["/verifiers"](#) resources
Check name — "eyebrows_state".

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
"eyebrows_state" > "threshold"	["neutral"]

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.8 Mouth attributes

This estimation determines a probabilistic score for each of the following parameters in the range [0..1]:

- "opened"
- "smile"
- "occluded"

The probability that the mouth is opened is also determined.

In the “/iso” and “detect_policy” > “face_quality” resources, the estimated value is also compared with threshold (according to ISO or non-standard threshold).

Resources where the estimation is performed:

- “/detector”
Estimation name — “estimate_mouth_attributes”.
- “/handlers”
Estimation name — “policies” > “detect_policy” > “estimate_mouth_attributes”.
- “/verifiers”
Estimation name — “policies” > “detect_policy” > “estimate_mouth_attributes”.
- “/sdk”
Estimation name — “estimate_mouth_attributes”.

Mouth attributes estimation using [image checking tools](#):

- “/iso” and “/detector” (see section “7.2.3 Expression” point “a”, “b” and “c” of the standard ISO/IEC 19794-5:2011)
Checks names — “mouth_smiling”, “mouth_occluded”, “mouth_open”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Checks names — “mouth_smiling”, “mouth_occluded”, “mouth_open”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“mouth_occluded” > “threshold”	[0...0.5]
“mouth_smiling” > “threshold”	[0...0.5]
“mouth_opened” > “threshold”	[0...0.5]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.8.1 Smile state

This estimation determines the predominant state of a smile from the following states:

- “none” — No smile found, so no additional parameters are determined.

- “smile_lips” — Usual smile with closed lips.
- “smile_teeth” — Smile with open teeth.

The estimated value is also compared with threshold (according to ISO or non-standard threshold).

If necessary, you can specify several smile states as acceptable.

Image requirements:

For correct checking results, the following requirements should be met.

The table below shows the requirements for [head pose](#):

Parameter	Required range
“pitch”	[-20...20]
“roll”	[-10...10]
“yaw”	[-25...25]

The table below shows the requirement for [face width](#):

Parameter	Required range
“face_width”	> 80

Resources where the estimation is performed:

Smile state estimation is only available using [image checking tools](#):

- “/iso” and “/detector” (see section “7.2.3 Expression” point “a”, “b” and “c” of the standard ISO/IEC 19794-5:2011)

Check name — “smile_properties”.

- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources

Check name — “smile_properties”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
“smile_properties” > “threshold”	[“none”]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.9 Image quality

This estimation determines a probabilistic score for each of the following parameters in the range [0..1], where 0 corresponds to low quality and 1 to high quality:

- “dark” — The degree of darkness in the photo.
- “light” — The degree of brightness in the photo.
- “blur” — The degree of blurriness.
- “illumination” — The degree of uniformity in illumination. The lower the difference between light and dark zones on the face, the higher the estimated value. When illumination is evenly distributed across the face, the value is close to “1.”
- “specularity” — The degree of absence of glare. The higher the estimated value, the less glare and the better the image quality. If the estimated value is low, there are bright glares on the face.

In the “/iso” and “detect_policy” > “face_quality” resources, the estimated value is also compared with threshold (according to ISO or non-standard threshold).

Image quality is determined using a specially trained VisionLabs neural network. If necessary, you can determine the illumination of the face in the image using an algorithm that performs an estimation in accordance with the ICAO standard (see the section [”illumination uniformity according to ICAO standard](#)).

Examples are presented in the images below. Good quality images are shown on the right.



Figure 18: Blurred image (left), not blurred image (right)

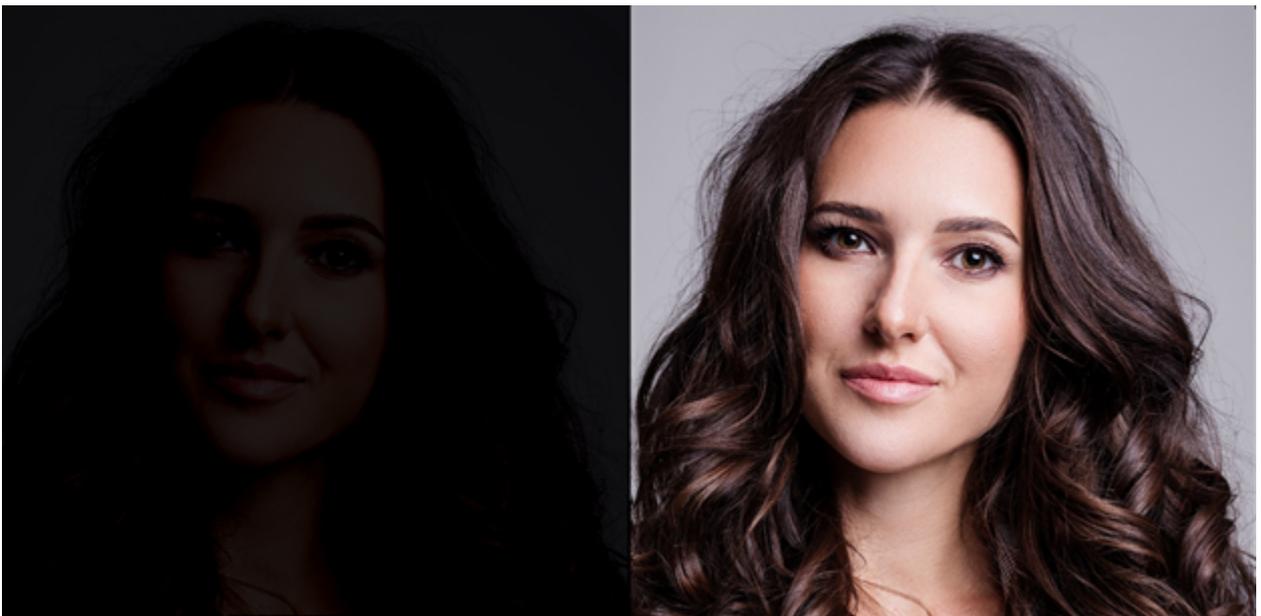


Figure 19: Dark image (left), good quality image (right)



Figure 20: Light image (left), good quality image (right)



Figure 21: Image with uneven illumination (left), image with even illumination (right)



Figure 22: Image with specularity — image contains glares on face (left), good quality image (right)

The most important image quality parameters for face recognition are darkness, light, and blur, so you should select them carefully.

The illumination and secularity parameters enable you to select images of better visual quality. Face recognition is not greatly affected by uneven illumination or glares.

Resources where the estimation is performed:

- [“/detector”](#)
Estimation name — “estimate_quality”.
- [“/handlers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_quality”.
- [“/verifiers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_quality”.
- [“/sdk”](#)
Estimation name — “estimate_quality”.

Image quality estimation using [image checking tools](#):

- [“/iso”](#) and [“/detector”](#) (see sections “7.2.7 Subject and scene lighting”, “7.3.2 Contrast and saturation”, “7.3.3 Focus and depth of field”, “7.2.8 Hot spots and specular reflections”, “7.2.12 Lighting artefacts”, “7.2.7 Subject and scene lighting” and “7.2.12 Lighting artefacts” of the standard ISO/IEC 19794-5:2011)

Checks names — “illumination_quality”, “specularity_quality”, “blurriness_quality”, “dark_quality”, “light_quality”.

- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources

Checks names — “illumination_quality”, “specularity_quality”, “blurriness_quality”, “dark_quality”, “light_quality”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“illumination_quality” > “threshold”	[0...0.3]
“specularity_quality” > “threshold”	[0...0.3]
“blurriness_quality” > “threshold”	[0.61...1]
“dark_quality” > “threshold”	[0.5...1]
“light_quality” > “threshold”	[0.57...1]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.9.1 Illumination uniformity according to ICAO standard

It is possible to estimate the uniformity of illumination according to the requirements specified in [ICAO standard](#).

The estimated value is also compared with the threshold.

In accordance with the standard, it is recommended to use color images. When using black and white images, the results may be unexpected.

The uniformity of illumination according to the ICAO standard is only available using [image checking tool](#):

- “/detector”
Check name — “illumination_uniformity”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Check name — “illumination_uniformity”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
“illumination_uniformity” > “threshold”	[0.5...1]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.10 Image background

4.2.10.1 Background lightness

Note: It is not possible to use a sample as an input image for these estimations.

This estimation determines background lightness, where:

- [0...0.1] — The background is black.
- [0.1...0.3] — The background is dark.
- [0.3...0.97] — The background is light.
- [0.97...1] — The background is white.



Figure 23: Background is dark, background_lightness = 0.13 (left), background is light background_lightness = 0.94 (right)

Resources where the estimation is performed:

Background lightness estimation is available only using [image checking tools](#):

- “/iso” and “/detector” (see section “B.2.9 Backgrounds” of the standard ISO/IEC 19794-5:2011)
Check name — “background_lightness”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Check name — “background_lightness”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable range
“background_lightness” > “threshold”	[0.2...1]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.10.2 Background uniformity

Note: It is not possible to use a sample as an input image for these estimations.

This estimation determines the degree of background uniformity, where:

- “0” — The background is not uniform.
- “1” — The background is uniform.



Figure 24: Background is non-uniform, background_uniformity = 0.004 (left), background is uniform, background_uniformity = 0.7 (right)

Resources where the estimation is performed:

Background uniformity estimation is available only using [image checking tools](#):

- “/iso” and “/detector” (see section “B.2.9 Backgrounds” of the standard ISO/IEC 19794-5:2011)
Check name — “background_uniformity”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Check name — “background_uniformity”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable range
"background_uniformity" > "threshold"	[0.5...1]

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.11 Dynamic range according to ICAO standard

This estimation determines the ratio of brightness of the lightest and darkest areas of the face according to the requirements specified in [ICAO standard](#).

The estimated value is also compared with the threshold.

Dynamic range estimation is only available using [image checking tool](#) — "detect_policy" > "face_quality" group of checks in the "/handlers" and "/verifiers" resources.

Check name — "dynamic_range".

Acceptable range for passing check:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable range
"dynamic_range" > "threshold"	[0.5...1]

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.12 Natural light

This estimation determines whether there is natural lighting on the face, where:

- "0" — The lighting is unnatural.
- "1" — The lighting is natural.

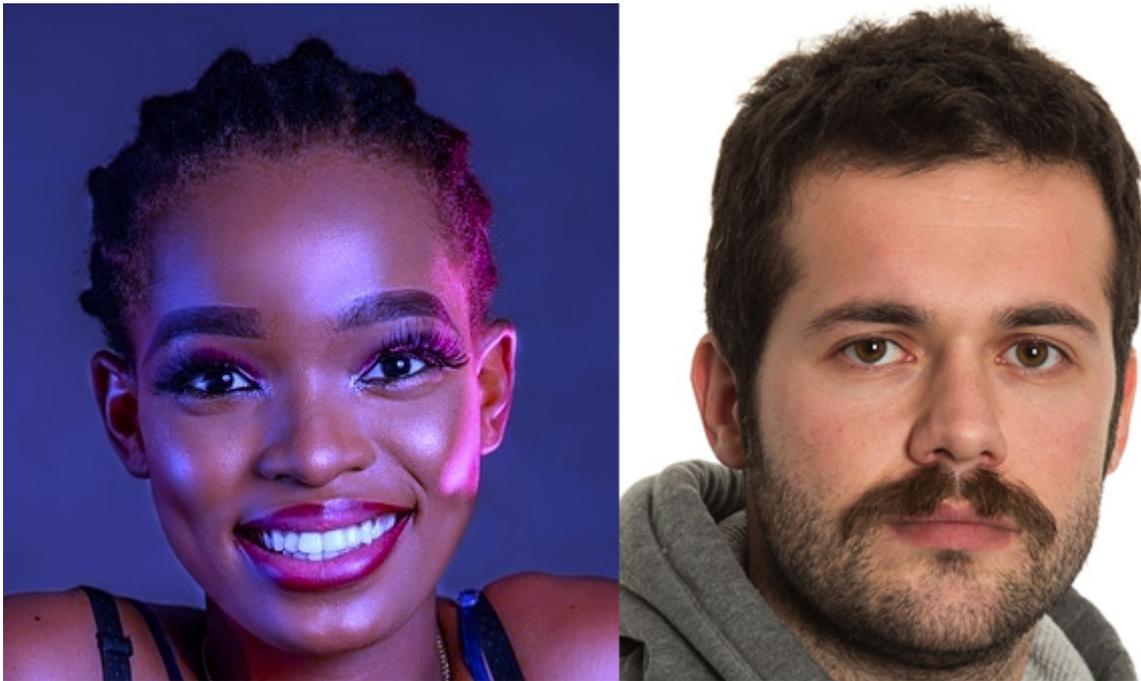


Figure 25: lighting is unnatural (left), light is natural (right)

The estimated value is also compared with threshold (according to ISO or non-standard threshold).

Image requirements:

For correct checking results, the following requirements should be met.

The table below shows the requirement for [mask](#):

Parameter	Required value
"predominant_mask"	missing

The table below shows the requirement for [image quality](#):

Parameter	Required range
"blurriness"	[0.5...1]

The table below shows the requirement for [glasses](#):

Parameter	Required values
“glasses”	“no_glasses” or “eyeglasses”

Resources where the estimation is performed:

Natural light estimation is available only using [image checking tools](#):

- [“/iso”](#) and [“/detector”](#) (see section “7.3.4 Unnatural colour” of the standard ISO/IEC 19794-5:2011)
Check name — “natural_light”.
- [“detect_policy”](#) > [“face_quality”](#) group of checks in the [“/handlers”](#) and [“/verifiers”](#) resources
Check name — “natural_light”.

Acceptable value for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
“natural_light” > “threshold”	“1”

For the [“face_quality”](#) image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.13 Face color type

This estimation determines the most likely type of face color from the following:

- “color”
- “grayscale”
- “infrared” (near infrared range)

The estimated value is also compared with threshold (according to ISO or non-standard threshold).

Resources where the estimation is performed:

Face color type estimation is only available using [image checking tools](#):

- [“/iso”](#) and [“/detector”](#) (see section “7.4.4 Use of near infra-red cameras” of the standard ISO/IEC 19794-5:2011)
Check name — “face_color_type”.
- [“detect_policy”](#) > [“face_quality”](#) group of checks in the [“/handlers”](#) and [“/verifiers”](#) resources
Check name — “face_color_type”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
<code>"face_color_type" > "threshold"</code>	<code>["color"]</code>

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.14 Head pose

This estimation determines the head pose. The pose is defined by three parameters:

- pitch angle
- roll angle
- yaw angle

The angle values are specified in the range from "-180" to "180".

A positive pitch angle indicates an upward tilt of the head, while a negative angle indicates a downward tilt of the head.

A positive roll angle indicates a rightward deviation of the head, while a negative angle indicates a leftward deviation of the head.

A positive yaw angle indicates a rightward rotation of the head, while a negative angle indicates a leftward rotation of the head.

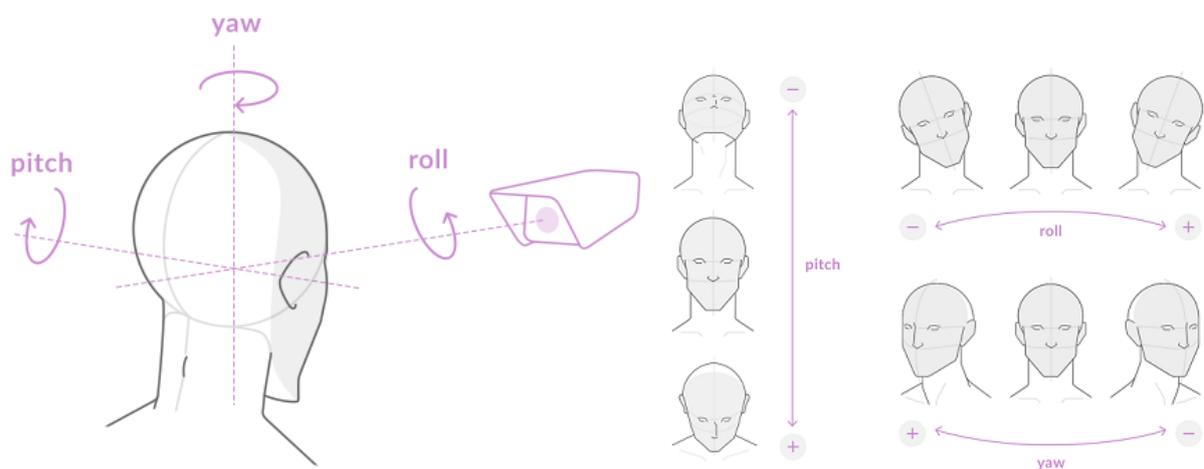


Figure 26: Head pose

In the “/iso” and “detect_policy” > “face_quality” resources, the estimated value is also compared with threshold (according to ISO or non-standard threshold).

In all the resources listed below, with the exception of “/iso”, the possibility to filtered out by head pose is available.

In the resources “/detector”, “/handlers”, “/verifiers” and “/sdk”, the threshold value is specified from “0” to “180”. The default value is “180”, which means that the head in the image can be rotated to any angle from “-180” to “180”. When you set any other value (for example, “30”), all the detections with the estimated angle less than or equal to “-30” and greater than or equal to “30” will be filtered.

For the “face_quality” field of the “/handlers” and “/verifiers” resources, the minimum and maximum thresholds are set in separate fields.

Resources where the estimation is performed:

- [“/detector”](#)
Estimation name — “estimate_head_pose”.
- [“/handlers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_head_pose”.
- [“/verifiers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_head_pose”.
- [“/sdk”](#)
Estimation name — “estimate_head_pose”.

Below are the recommended thresholds for estimation in the “/detector”, “/handlers”, “/verifiers” and “/sdk” resources.

Recommended maximum thresholds:

The table below shows the recommended maximum thresholds [head pose](#) for estimation **in cooperative mode**:

Parameter	Recommended maximum thresholds
“roll_threshold”	30
“pitch_threshold”	15
“yaw_threshold”	15

The table below shows the recommended maximum thresholds [head pose](#) for estimation **in non-cooperative mode**:

Parameter	Recommended maximum thresholds
“roll_threshold”	30
“pitch_threshold”	30
“yaw_threshold”	30

Head pose estimation using [image checking tools](#):

- “/iso” and “/detector” (see section “7.2.2 Pose” of the standard ISO/IEC 19794-5:2011)
Checks names — “head_roll”, “head_pitch”, “head_yaw”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Checks names — “head_roll”, “head_pitch”, “head_yaw”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“head_yaw” > “threshold”	[-5...5]
“head_pitch” > “threshold”	[-5...-5]
“head_roll” > “threshold”	[-8...-8]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.15 Vertical and horizontal face position

Note: It is not possible to use a sample as an input image for these estimations.

These estimations determine the position of the center point vertically and horizontally relative to the image.

The estimated values are also compared with thresholds (according to ISO or non-standard thresholds).

Resources where the estimation is performed:

Vertical and horizontal face position estimation is available only using [image checking tools](#):

- “/iso” and “/detector” (see sections “8.3.2 Horizontally centred face” and “8.3.3 Vertical position of the face” of the standard ISO/IEC 19794-5:2011)

Checks names — “head_horizontal_center”, “head_vertical_center”.

- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources

Checks names — “head_horizontal_center”, “head_vertical_center”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“head_horizontal_center” > “threshold”	[0.45...0.55]
“head_vertical_center” > “threshold”	[0.3...0.5]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.16 Head width and height

Note: It is not possible to use a sample as an input image for these estimations.

This estimations determine the vertical and horizontal head size relative to the size of the image. The estimated values are also compared with thresholds (according to ISO or non-standard thresholds).

Resources where the estimation is performed:

Head width and height estimation is available only using [image checking tools](#):

- “/iso” and “/detector” (see sections “8.3.4 Width of head” and “8.3.5 Length of head” of the standard ISO/IEC 19794-5:2011)

Checks names — “head_width”, “head_height”.

- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources

Checks names — “head_width”, “head_height”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“head_width” > “threshold”	[0.5...0.75]
“head_height” > “threshold”	[0.6...0.9]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.17 Face width and height

Note: It is not possible to use a sample as an input image for these estimations.

These estimations determine the face width and height in pixels. The estimated values are also compared with the specified thresholds.

Resources where the estimation is performed:

Face width and height estimations are only available using [image checking tool](#):

- “/detector”
Checks names — “face_width”, “face_height”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Checks names — “face_width”, “face_height”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“face_width” > “threshold”	[180...1920]
“face_height” > “threshold”	[180...inf]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.18 Indents from image edges

Note: It is not possible to use a sample as an input image for these estimations.

This estimation is determined as the indent from the image border (left, right, top, bottom) to the face border (left, right, top, bottom) in pixels. The estimated values are also compared with the specified thresholds.

Resources where the estimation is performed:

Indents from image edges estimation is only available using [image checking tool](#):

- “/detector”

Checks names — “indent_upper”, “indent_lower”, “indent_right”, “indent_left”.

- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources

Checks names — “indent_upper”, “indent_lower”, “indent_right”, “indent_left”.

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
“indent_upper” > “threshold”	[20...inf]
“indent_lower” > “threshold”	[20...inf]
“indent_right” > “threshold”	[20...inf]
“indent_left” > “threshold”	[20...inf]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.19 Mask

This estimation determines a probabilistic score for each of the following parameters in the range [0..1]:

- “medical_mask” — The probability that the person wears a medical mask.
- “missing” — The probability that the person does not wear a mask.
- “occluded” — The probability that the face is occluded by an object other than a medical mask.

The predominant state of the mask is also determined.

In addition to the three main states, the following additional properties are defined:

- “correct” — There is mask on the face, the mouth and nose are occluded by the mask.
- “mouth” — There is mask on the face and occludes only the mouth.
- “clear” — There is no mask on the face.
- “chin” — There is mask on the face and is located under the chin, without occluding the area from eyes to mouth.
- “partially” — The face is partially occluded, but not by medical mask and not by mask with full face occlusion.
- “full” — There is mask on the face, in which the face is completely occluded, for example, balaclava/ski mask.

Each main mask state corresponds to one of two additional properties. The most likely additional property is returned in the “predominant_occlusion” field:

- The “medical_mask” state corresponds to the “correct” or “mouth” property.
- The “missing” state corresponds to the “clear” or “chin” property.
- The “occluded” state corresponds to the “partially” or “full” property.

For each of the properties, a probabilistic score is returned in the range [0..1].

Additional mask properties are not stored in the database and are not filtered by them.

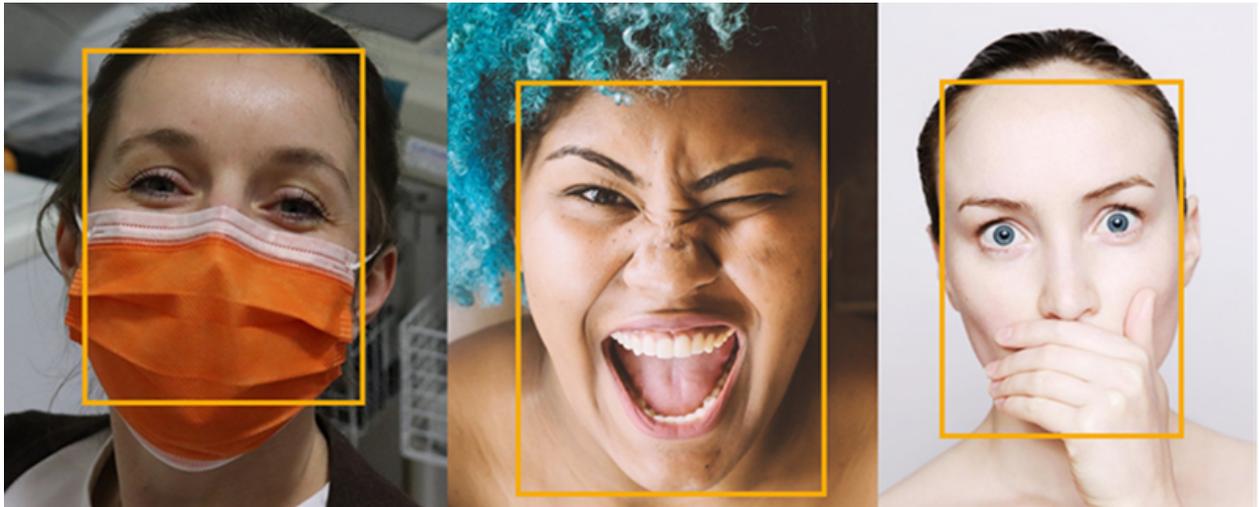


Figure 27: From left to right — correct, clear, partially

Resources where the estimation is performed:

- [“/detector”](#)
Estimation name — “estimate_mask”.
- [“/handlers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_mask”.
- [“/verifiers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_mask”.
- [“/sdk”](#)
Estimation name — “estimate_mask”.

4.2.20 Emotions

This estimation determines a probabilistic score for each of the following parameters in the range [0..1]:

- “anger”
- “disgust”

- “fear”
- “happiness”
- “neutral”
- “sadness”
- “surprise”

The predominant emotion is also estimated.

Emotions can be saved in the event object during the event creation.

Resources where the estimation is performed:

- [“/detector”](#)
Estimation name — “estimate_emotions”.
- [“/handlers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_emotions”.
- [“/verifiers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_emotions”.
- [“/sdk”](#)
Estimation name — “estimate_emotions”.

4.2.21 Shoulders position

This estimation determines the most predominant state of the shoulder position from the following:

- “non-parallel”
- “parallel”
- “hidden”

Resources where the estimation is performed:

Shoulders position estimation is available only using [image checking tools](#):

- [“/iso”](#) and [“/detector”](#) (see section “7.2.5 Shoulders” of the standard ISO/IEC 19794-5:2011)
Check name — “shoulders_position”.
- [“detect_policy” > “face_quality”](#) group of checks in the [“/handlers”](#) and [“/verifiers”](#) resources
Check name — “shoulders_position”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable range
"shoulders_position" > "threshold"	["parallel"]

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.22 Headwear

This estimation determines the most predominant type of headwear from the following:

- "none"
- "baseball_cap"
- "beanie"
- "peaked_cap"
- "shawl"
- "hat_with_ear_flaps"
- "helmet"
- "hood"
- "hat"
- "other"

The estimated value is also compared with threshold (according to ISO or non-standard threshold).

It is possible to specify several types of headwear as acceptable.

Image requirements:

The table below shows the requirements for [head pose](#):

Parameter	Required range
"pitch"	[-20...20]
"roll"	[-10...10]
"yaw"	[-25...25]

The table below shows the requirement for [face width](#):

Parameter	Required range
face_width	> 80

Resources where the estimation is performed:

Headwear estimation is available only using [image checking tools](#):

- “/iso” and “/detector” (see section “B.2.7 Head coverings” of the standard ISO/IEC 19794-5:2011)
Check name — “headwear_type”.
- “detect_policy” > “face_quality” group of checks in the “/handlers” and “/verifiers” resources
Check name — “headwear_type”.

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
“headwear_type” > “threshold”	[“none”]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.2.23 Radial distortion (Fisheye effect)

This estimation determines the presence of the Fisheye effect, where:

- “0” — The Fisheye effect is not present in the image.
- “1” — The Fisheye effect is present in the image.

The estimated value is also compared with threshold (according to ISO or non-standard threshold).

Image requirements:

For correct checking results, the following requirements should be met.

The table below shows the requirements for [head position](#):

Parameter	Required range
pitch	[-20...20]
roll	[-10...10]
yaw	[-25...25]

The table below shows the requirement for [face width](#):

Parameter	Required range
face_width	> 80

Resources where the estimation is performed:

Fisheye effect estimation is available only using [image checking tools](#):

- ["/iso](#) and ["/detector](#)" (see section "7.3.6 Radial distortion of the camera lens" of the standard ISO/IEC 19794-5:2011)

Check name — "radial_distortion".

- ["detect_policy"](#) > ["face_quality"](#) group of checks in the ["/handlers](#) and ["/verifiers](#)" resources

Check name — "radial_distortion".

Acceptable range for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable value
"radial_distortion" > "threshold"	"1"

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.3 Image parameters

4.3.1 Image format

This estimation determines the correspondence of the incoming image format to one of the following formats — "JPEG", "JPEG2000", "PNG". The estimated value is also compared with threshold (according to ISO or non-standard threshold).

Resources where the estimation is performed:

Image format estimation is available only using [image checking tools](#):

- ["/iso](#) and ["/detector](#)" (see section "7.5 Format requirements for the Frontal Image Type" of the standard ISO/IEC 19794-5:2011)

Check name — "image_format".

- ["detect_policy"](#) > ["face_quality"](#) group of checks in the ["/handlers](#) and ["/verifiers](#)" resources

Check name — "image_format".

Acceptable values for passing check:

The image passes the check if it falls within the acceptable value of the corresponding [threshold](#):

Parameter	Acceptable values
"image_format" > "threshold"	["JPEG", "JPEG2000", "PNG"]

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.3.2 Image size

This estimation determines the image size in bytes. The estimated value is also compared with the specified threshold.

Resources where the estimation is performed:

Image size estimation is only available using [image checking tool](#):

- ["/detector"](#)
Check name — "image_size".
- ["detect_policy" > "face_quality"](#) group of checks in the ["/handlers"](#) and ["/verifiers"](#) resources
Check name — "image_size".

Acceptable range for passing check:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable range
"image_size" > "threshold"	[5120...2097152]

For the "face_quality" image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.3.3 Image width and height

These estimations determine the image width and height in pixels. The estimated values are also compared with thresholds (according to ISO or non-standard thresholds).

Resources where the estimation is performed:

Image width and height estimations are available only using [image checking tools](#):

- ["/iso](#) and ["/detector](#) (see sections "5.7.4 Width" and "5.7.5 Height" of the standard ISO/IEC 19794-5:2011)

Checks names — "image_height", "image_width".

- ["detect_policy"](#) > ["face_quality"](#) group of checks in the ["/handlers](#) and ["/verifiers](#) resources

Checks names — "image_height", "image_width".

Acceptable ranges for passing checks:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable ranges
"image_height" > "threshold"	[180...1920]
"image_width" > "threshold"	[180...1080]

For the ["face_quality"](#) image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.3.4 Aspect ratio

This estimation determines the proportional ratio of the image width to height. The estimated value is also compared with the specified threshold.

Resources where the estimation is performed:

Aspect ratio estimation is only available using [image checking tool](#):

- ["/detector](#)

Check name — "aspect_ratio".

- ["detect_policy"](#) > ["face_quality"](#) group of checks in the ["/handlers](#) and ["/verifiers](#) resources

Check name — "aspect_ratio".

Acceptable range for passing check:

The image passes the check if it falls within the acceptable range of the corresponding [threshold](#):

Parameter	Acceptable range
"aspect_ratio" > "threshold"	[0.74...0.8]

For the “face_quality” image checking tool, the acceptable value can be set manually, but this will mean a deviation from the standard.

4.3.5 EXIF metadata

When the EXIF estimation is enabled, all the tags of the image are parsed and their names and values are outputted. Please refer to JEITA CP-3451 EXIF specification for details. The following data is returned:

- make
- model
- orientation
- latitude
- longitude
- artist
- software
- dateTime
- digitalZoomRatio
- flash
- uid

Resources where the estimation is performed:

- [“/detector”](#)
Estimation name — “extract_exif”.
- [“/handlers”](#)
Estimation name — “policies” > “detect_policy” > “extract_exif”.
- [“/verifiers”](#)
Estimation name — “policies” > “detect_policy” > “extract_exif”.
- [“/sdk”](#)
Estimation name — “use_exif_info”.

4.4 Body parameters

4.4.1 Gender and age by body image

This estimation determines the basic attributes (gender and age) of a person in the body image.

Estimating gender and age from a body image is less accurate than from a face.

Resources where the estimation is performed:

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “body_attributes” > “estimate_basic_attributes”.

- [“/sdk”](#)

Estimation name — “estimate_body_basic_attributes”.

4.4.2 Upper body

This estimation determines the parameters of the following elements of clothing on the upper body:

- “headwear”. Type: absent, present, undefined; color: white, black, other, undefined.
- “sleeve”. Type: short, long, undefined.
- “upper_clothing”. Color: orange, purple, red, white, yellow, pink, brown, beige, khaki, multicolored, undefined.

Resources where the estimation is performed:

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “body_attributes” > “estimate_upper_body”.

- [“/sdk”](#)

Estimation name — “estimate_upper_body”.

4.4.3 Lower body

This estimation determines the parameters of the following elements of clothing on the lower body:

- “lower_garment”. Type: trousers, shorts, skirt, undefined; color: orange, purple, red, white, yellow, pink, brown, beige, khaki, multicolored, undefined.
- “shoes”. Color: white, black, other, undefined.

Resources where the estimation is performed:

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “body_attributes” > “estimate_lower_body”.

- [“/sdk”](#)

Estimation name — “estimate_lower_body”.

4.4.4 Backpack

This estimation determines the presence of a backpack on the body:

- “0” — There is no backpack on the body image.

- “1” — There is a backpack on the body image.

Resources where the estimation is performed:

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “body_attributes” > “estimate_accessories”.

- [“/sdk”](#)

Estimation name — “estimate_accessories”.

4.5 Liveness

Note: The ability to perform such estimation is adjusted by a special parameter in the [LUNA PLATFORM 5 license key](#).

The Liveness technology enables LUNA PLATFORM to detect presentation attacks. To estimate Liveness in the LUNA PLATFORM, the estimator LUNA SDK OneShotLiveness is used.

As a result of the Liveness estimation, one of the following results may be returned:

- “0” — The person is not real.
- “1” — The person is real.
- “2” — Result of the check is unknown.

Resources where the estimation is performed:

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “estimate_liveness”.

- [“/verifiers”](#)

Estimation name — “policies” > “detect_policy” > “estimate_liveness”.

- [“/sdk”](#)

Estimation name — “estimate_liveness”.

- [“/liveness”](#)

See [“OneShotLiveness description”](#) for more information about Liveness.

4.6 Deepfake

Note: The ability to perform such estimation is adjusted by a special parameter in the [LUNA PLATFORM 5 license key](#).

Note: It is not possible to use a sample as an input image for these estimation.

This estimation enables you to detect facial substitution using DeepFake technology in photo images.

A Deepfake evaluation may return the following results:

- “prediction” = “fake” — The person is not real.
- “prediction” = “real” — The person is real.
- “score” = [0...1] — The degree of reliability of the estimation.

If necessary, you can configure the handler to filter events by the expected result of the Deepfake estimation (“fake” or “real”). To do this, in the request body to create a handler, specify the “deepfake_states” parameter with the value “0” (filter by the value “fake”) or “1” (filter by the value “real”). For example, if the “deepfake_states” parameter is “1” (filter by “real”), and the estimator has determined that the result is “fake”, then an empty “events” field will be returned in the event, and the results of the check will fall into the “filtered_detections” field.

In the requests “[create handler](#)” and “[create verifier](#)”, it is possible to set the “real_threshold” and the operating “mode”. The “[sdk](#)” request will use the default values of these parameters (see below) without the possibility of explicit indication.

Threshold

Using the “real_threshold”, you can set a value in the range [0...1], below which the system will assume that the person is not real.

For example, if the threshold value is “real_threshold” = “0.5”, and the degree of reliability of the estimation is “score” = “0.4”, then the result “prediction” = “fake” will be output in the response body. If the threshold value is “real_threshold” = “0.6”, and the degree of reliability of the estimation is “score” = “0.7”, then the result “prediction” = “real” will be given in the response body.

The default value is “0.5”.

Mode

The operating modes are described in the table below.

Operating mode	Description
“mode” = “1”	Simplified operation mode.
“mode” = “2” (default)	Operation mode using an additional neural network model. When using this mode, a preliminary estimation of the face in the source image will be additionally performed. If the result of the preliminary check determined that the face is fake, then the result “score” = “0” and “prediction” = “fake” will be returned in the response body.

Image requirements

For correct verification results, the following requirements must be met.

The table below shows the requirements for [head pose](#):

Parameter	Required range
“pitch”	[-20...20]
“yaw”	[-30...30]

The table below shows the requirement for [face width](#):

Parameter	Required range
“face_width”	> 150

Resources where the estimation is performed:

- [“/handlers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_deepfake”.
- [“/verifiers”](#)
Estimation name — “policies” > “detect_policy” > “estimate_deepfake”.
- [“/sdk”](#)
Estimation name — “estimate_deepfake”.

4.7 People count

Note: The ability to perform such estimation is adjusted by a special parameter in the [LUNA PLATFORM 5 license key](#).

This estimate determines the number of people in the image.



Figure 28: People count estimation

Resources where the estimation is performed:

- [“/handlers”](#)

Estimation name — “policies” > “detect_policy” > “estimate_people_count”.

- [“/sdk”](#)

Estimation name — “estimate_people_count”.

If necessary, together with estimating the number of people, you can get the X and Y coordinates of people using the “people_count_coordinates” parameter in the [“/handlers”](#) resource and the “people_coordinates” parameter in the [“/sdk”](#) resource.

5 Services interaction

There are separate databases specified for each service in the images below. This is done for illustration purposes.

It is not required to launch several similar databases for each service. You can launch a single database instance (e. g., PostgreSQL) and use it to store data for all the LP services. Each service will have its own table in this database.

See “[General concepts](#)” for details about the databases used by LP services.

API service provides a RESTful interface for faces and bodies detection, extraction and matching procedures.

Requests are sent to LP via HTTP. A common case is when an external service sends requests to LP, receives results and processes the results according to business needs.

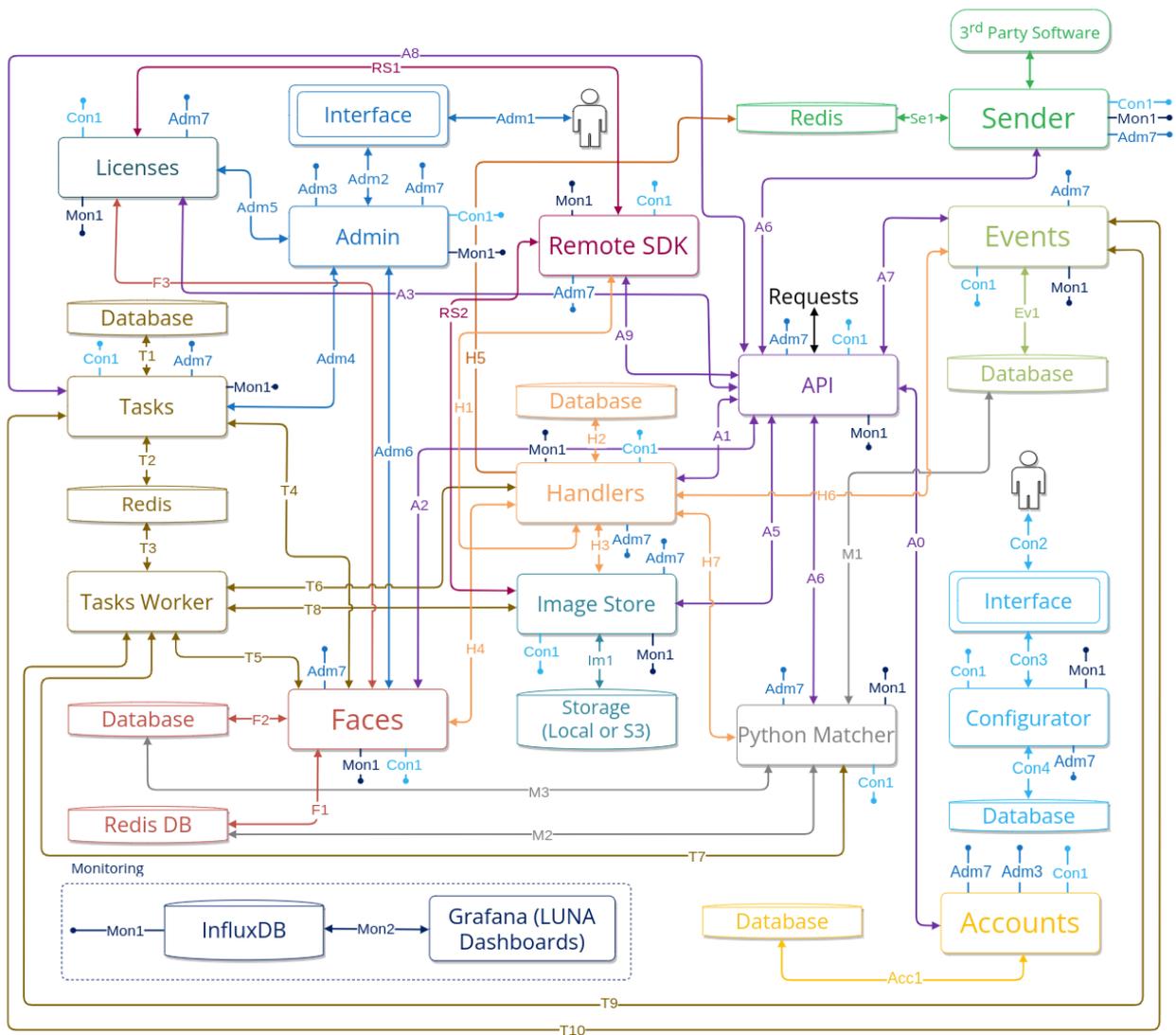


Figure 29: Services interaction

Some services can be disabled (see “General concepts”).

API receives requests, processes them and checks user authorization using a request to the **Accounts** service (**A0**). The Accounts service checks if the user’s credentials exist in the Accounts database (**Acc1**). If the user data is not found in the database, then an authorization error is issued. If the data is found, then the corresponding information is sent to the API service, where it passes the original requests to other services.

The main operations (detection, estimation, extraction, matching) are performed as described below.

Approach “Sequential performing of requests”:

- The “detect faces” request, in which face detection is performed, face parameters are estimated and samples are created, is sent from the API service to the Handlers service (**A1**), and then

redirected to Remote SDK service **(H1)**.

- The “[extract attributes](#)” request, in which temporary attributes are extracted, is sent from the API service to the Handlers service **(A1)**, and then redirected to the Remote SDK service **(H1)**.
- The request “[matching faces](#)” is sent from the API service to the Python Matcher service **(A6)**.

Approach “[Parallel performing of requests](#)”:

- The “[create handler](#)” request with the rules for detecting, estimating, extracting and matching faces and bodies is sent from the API service to the Handlers service **(A1)**.
- The “[generate events](#)” request is sent from the API service to the Handlers service **(A1)**, and then redirected to the Remote SDK service **(H1)**.

The API service sends requests to the Faces service to create/modify new faces (requests from the “[faces](#)” section), as well as create/modify lists (requests from the “[lists](#)”) section **(A2)**.

The API service receives information about the current license terms from the Licenses service **(A3)**.

The API service sends samples from external services to the Image Store service (the “[save face/body sample](#)” request) **(A5)**.

Handlers creates new handlers and stores them in the Handlers database **(H2)**.

The service also redirects requests for detection and estimation to the Remote SDK **(H1)** service.

The Handlers service is enabled/disabled in the “[ADDITIONAL_SERVICE_USAGE](#)” setting.

Remote SDK processes face/body detection and attributes creation requests. It estimates face attributes and face/body parameters.

The Remote SDK service processes the request to detect faces/bodies and estimate parameters from the Handlers **(H1)** service, sends the result back to Handlers **(H1)**, which sends the received samples to the Image Store service **(H3)**.

The Remote SDK service processes the request to create attributes from the Handlers **(H1)** service by requesting existing samples from the Handlers service, which it requests from the Image Store service **(H3)**. Next, the Handlers service sends the created temporary attributes to the Faces **(H4)** service, which saves them in the Redis database **(F1)**.

Requests “[/iso](#)”, “[/sdk](#)”, “[/liveness](#)” are performed directly to the Remote SDK service **(A9)**. In this case, the license is checked by interacting with the Licenses **(RS1)** service. To be able to use a set of samples, the Remote SDK service interacts with the Image Store **(RS2)** service.

Image Store receives samples from the Remote SDK service and stores them on a local storage or in S3-compatible cloud storage and provides access to them **(Im1)**.

The Image Store service is enabled/disabled in the “[ADDITIONAL_SERVICE_USAGE](#)” setting.

Licenses. The Licenses service receives information about the number of created faces with attributes from the Faces database **(Li1)**.

Faces is responsible for interaction with Faces database, which stores: faces, descriptors for faces, and lists (**F2**). It provides access to the stored data for API, Python Matcher, and Tasks services.

Faces also stores the created temporary attributes in the Redis database (**F1**).

Every time a new face is created, the Faces service sends information about the number of created faces with attached attributes to the Licenses service (**F3**) (see the “[Faces limit](#)” section).

Events service stores and provides information about events (*the Events section*). After an event is created, the Events service receives the created event from the Handlers service (**H6**) and stores it in the Events database (**Ev2**).

The Events service is enabled/disabled in the “[ADDITIONAL_SERVICE_USAGE](#)” setting.

Python Matcher can receive matching requests directly from the API service (**A6**). If a matching policy is specified in the handler, then a request to match descriptors will come from the Handlers (**H7**) service. Python Matcher sends matching requests to the Faces database (**M3**) or Events database (**M1**). The databases matches descriptors and sends results back to the API service.

Python Matcher can also receive the temporary attributes required for matching from the Redis database (**M2**).

It is possible to additionally use the Python Matcher Proxy service, which can redirect requests to Python Matcher or matching plugins. Matching plugins can significantly speed up the matching requests execution (see the “[Matching plugins](#)” section). This service is not shown in the diagram, but has the same communication lines as the Python Matcher service.

Admin. A user can manage accounts and perform other actions via the [user interface](#) of the Admin service (**Adm1**). The corresponding requests are sent to the Admin service (**Adm2**).

The Admin service receives information about accounts from the Accounts (**Adm3**) service.

All information about user accounts is stored in the Accounts database (**Acc1**).

The Admin service sends requests to Tasks service (**Adm4**). These tasks are performed for LP databases in general, not for a single account ID.

The Admin service receives information about the current license terms from the Licenses service (**Adm5**).

The Admin service checks the current number of created faces with linked attributes in the Faces database (**Adm6**) and calculates the percentage of the database of fullness.

The Admin service performs requests to all LP services, receiving system information (see “[/luna_sys_info](#)” request) (**Adm7**).

Sender. All generated events are received by the Redis DB from the Handlers service (**H5**). External third-party software subscribes to receive events according to the specified filters (**Se2**). Sender service checks Redis channel, receives the required information and sends it to external software (**Se1**).

The Sender service is enabled/disabled in the [“ADDITIONAL_SERVICE_USAGE”](#) setting.

Configurator. LP services receive configuration parameters from the Configurator service (**Con1**). A user can manage configurations in Configurator by means of the [user interface \(Con2\)](#). The changes are sent to Configurator (**Con3**). All the changes in configuration files are saved in Configurator DB (**Con4**).

Configurator is utilized for each Luna service by default.

Tasks. Tasks service receives requests from API service (**A8**). Then Tasks creates and stores a task in the Tasks database (**T1**).

Next, the Tasks service interacts with its worker via Redis (**T2, T3**), creating subtasks and merging the results. For more information, see [“Task diagrams”](#) section.

The Tasks service receives data from the Faces service for the Clustering, Linker and Additional extraction tasks (**T4**).

The Tasks service receives data from the Events service for the Clustering and Linker tasks (**T9, T10**).

The Tasks service receives data from the Handlers service for the Estimator task (**T6**).

The Tasks service is enabled/disabled in the [“ADDITIONAL_SERVICE_USAGE”](#) setting.

The interaction of Tasks workers with other services depends on the task type.

Tasks workers receive data from Faces service for every processed task (**T5**).

Tasks workers send a request to Python Matcher (**T7**) for clustering, cross-matching and ROC creation tasks.

Tasks workers store all the reports in the storage of Image Store (**T8**). Tasks workers also store and receive clusters from Image Store.

Tasks workers send the request for additional extraction task to the Handlers service (**T6**).

Monitoring. Monitoring system receives requests and errors from each service (**Mon1**). This data stores in the Influx database (**Mon2**). Then the monitoring data is sent to Grafana to visualize the monitoring data (**Mon3**). You can find more information about monitoring in the [“Monitoring”](#) section.

Grafana and InfluxDB services have their own interfaces (see the [“User interfaces”](#) section).

This diagram does not contain the architecture of the Backport 3 and Backport 4 services. See the detailed architecture of the Backport 3 service in [“Backport 3 architecture”](#) section and the Backport 4 service in [“Backport 4 architecture”](#) section.

6 Services description

This section provides more details on functions of the LP services.

Databases can be omitted in the following figures.

See the table with the resource consumption of each of the services listed below in the [“Resource consumption by services”](#) section.

6.1 General information about services

6.1.1 Worker processes

For LUNA PLATFORM services, you can set the number of workers to use additional resources and system memory to process requests to the service. A service will automatically spin up multiple processes and route traffic between the processes.

When starting the service in a Docker container, the number of workers is set using the `WORKER_COUNT` parameter.

For example, if you set the value `WORKER_COUNT=2` for the Faces service, then the service will consume 2 times more resources and memory.

Note the number of available cores on your server when utilizing this feature.

Worker processes utilization is an alternative way for linear service scaling. It is recommended to use additional worker processes when increasing the number of service instances on the same server.

It is not recommended to use additional worker processes for the Remote SDK service when it utilizes GPU. Problems may occur if there is not enough GPU memory, and the workers will interfere with each other.

6.1.2 Automatic configurations reload

LP services support the auto-reload of configurations. When a setting is changed, it is automatically updated for all the instances of the corresponding services. When this feature is enabled, no manual restart of services is required.

This feature is available for all the settings provided for each Python service. You should enable the feature manually upon each service launching. See the [“Enable automatic configuration reload”](#) section.

Starting with version 5.5.0 the configuration reload for Faces and Python Matcher services is done mostly by restarting appropriate processes.

6.1.2.1 Restrictions

Service can work incorrectly while new settings are being applied. It is strongly recommended not to send requests to the service when you change important settings (DB setting, work plugins list, and others).

New settings appliance may lead to service restart and caches resetting (e. g., Python Matcher service cache). For example, the default descriptor version changing will lead to the LP restart. Changing the logging level does not cause service restart (if a valid setting value was provided).

6.1.2.2 Enable automatic configuration reload

You can enable this feature by specifying a `--config-reload` option in the command line. In Docker containers, the feature is enabled using the “RELOAD_CONFIG” option.

You can specify the configurations check period in the `--pulling-time` command line argument. The value is set to 10 seconds by default. In Docker containers, the feature is enabled using the “RELOAD_CONFIG_INTERVAL” option.

6.1.2.3 Configurations update process

LP services periodically receive settings from the Configurator service or configuration files. It depends on the way of configurations receiving for a particular service.

Each service compares its existing settings with the received settings:

- If service settings were changed, they will be pulled and applied.
 - If the configurations pulling has failed, the service will continue working without applying any changes to the existing configurations.
 - If check connections with new settings have failed, the service will retry new configurations pulling after 5 seconds. The service will shut down after 5 failed attempts.
- If current settings and new pulled settings are the same, the Configurator service will not perform any actions.

6.1.3 Database migration execution

You should execute migration scripts to update your database structure when upgrading to new LP builds. By default, migrations are automatically applied when running `db_create` script.

This method may be useful when you need to rollback to the previous LUNA PLATFORM build or upgrade the database structure without changing the stored data. Anyway, it is recommended to create the backup of your database before applying any changes.

You can run migrations from a container or use a single command.

6.1.3.1 Single command

The example is given for the Tasks service.

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/tasks:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.22.1 \  
alembic -x luna-config=http://127.0.0.1:5070/1 upgrade head
```

6.1.3.2 Running from container

To run migrations from a container follow these steps (the example is given for the Configurator service):

- Go to the service docker container. See the “Enter container” section in LP 5 installation manual.
- Run the migrations.

For most of the services, the configuration parameters should be received from the Configurator service and the command is the following:

```
alembic -x luna-config=http://127.0.0.1:5070/1 upgrade head
```

`-x luna-config=http://127.0.0.1:5070/1` — Specifies that the configuration parameters for migrations should be received from the Configurator service.

For the Configurator service the parameters are received from “`srv/luna_configurator/configs/config.conf`” file.

You should use the following command for the Configurator service:

```
alembic upgrade head
```

- Exit the container. The container will be removed after you exit.

```
exit
```

6.2 API service

LUNA API is a facial recognition web service. It provides a RESTful interface for interaction with other LUNA PLATFORM services.

Using the API service you can send requests to other LP services and solve the following problems:

- Images processing and analysis:
 - Face/body detection in photos.
 - Face attributes (age, gender, ethnicity) and face parameters (head pose, emotions, gaze direction, eyes attributes, mouth attributes) estimation.
 - Body parameters (age, gender, accessories, headwear, colors of upper and lower clothing, type of sleeves) estimation.
- Search for similar faces/bodies in the database.
- Storage of the received face attributes in databases.
- Creation of lists to search in.
- Statistics gathering.
- Flexible request management to meet user data processing requirements.

6.3 Remote SDK service

The Remote SDK service is used to:

- Perform face detection and face parameters estimation.
- Perform body detection and body parameters estimation.
- Create samples.
- Perform extraction of basic attributes and descriptor, including aggregated ones.
- Process images using handlers and verifiers policies.

Face, body detection, descriptor extraction, estimation of parameters and attributes are performed using neural networks. The algorithm evolves with time and new neural networks appear. They may differ from each other by performance and precision. You should choose a neural network following the business case of your company.

6.3.1 Remote SDK with GPU

Remote SDK service can utilize GPU instead of CPU for calculations. A single GPU is utilized per Remote SDK service instance.

Attributes extraction on the GPU is engineered for maximum throughput. The input images are processed in batches. This reduces computation cost per image but does not provide the shortest latency per image.

GPU acceleration is designed for high load applications where request counts per second consistently reach thousands. It won't be beneficial to use GPU acceleration in non-extensively loaded scenarios where latency matters.

6.3.2 Aggregation

Based on all images transferred in one request, a single set of basic attributes and an aggregated descriptor can be obtained. In addition, during the creation of the event, the aggregation of the received values of Liveness, emotions, medical mask states for faces and upper/lower body, gender, age and the body accessories for bodies is performed.

The matching results are more precise for aggregated descriptor. It is recommended to use aggregation when several images were received from the same camera. It is not guaranteed that aggregated descriptors provide improvements in other cases.

It is considered that each parameter is aggregated from sample. Use the "aggregate_attributes" parameter of the "extract_attributes" (only for faces) and "sdk" requests to enable attributes aggregation. Aggregation of liveness, emotion, and mask states for faces and upper body, gender, age and the body accessories for bodies is available using the "aggregate_attributes" parameter in the "generate_events", provided that these parameters were estimated earlier in the handler, as well as in the "sdk" request.

An array of “sample_ids” is returned in the response even if there was only a single sample used in the request. In this case, a single sample ID is included in the array.

6.3.3 Descriptor formats

LUNA PLATFORM supports the following descriptor formats:

Descriptor format	File content	Size
SDK +	Set of bytes (descriptor itself). ----- ---+ Set of bytes indicating the version. ----- -----+ Set of signature bytes.	Size depends on neural network version (see “Neural networks” section). ----- -----+ Size is 4 bytes. ----- -----+ Size is 4 bytes.
Raw	Set of bytes (descriptor itself) encoded in Base64.	Size depends on neural network version (see “Neural networks” section).
XPK files	Files that store descriptor in SDK format.	Depends on the number of descriptors inside the file.

SDK and Raw formats can be directly linked to a face or stored in a temporary attribute (see “[Create objects using external data](#)” below).

In most extraction requests, the descriptor is saved to the database as set of bytes, without being returned in the response body.

There are several requests that can be used to get descriptor in SDK format:

- request “[sdk](#)”;
- request “[get temporary attributes](#) and ”[get temporary attribute](#).”

With LUNA PLATFORM, it is not possible to get descriptors in Raw and SDK formats. You can use other VisionLabs software to get these formats (eg LUNA SDK). Descriptors obtained using the above resources or using the VisionLabs software are referred to as **raw** descriptors.

Use raw descriptors for matching

The descriptor formats described above can be used in requests for the use of raw descriptors.

An external raw descriptor can be used as **reference** in the following resources:

- “[/matcher/faces](#)”
- “[/matcher/bodies](#)”

- [“/matcher/raw”](#)
- [“/handlers/{handler_id}/events”](#) when “multipart/form-data” request body schema is set
- [“/verifiers/{verifier_id}/verifications”](#) when “multipart/form-data” request body schema is set
- [“/verifiers/{verifier_id}/raw”](#)

An external raw descriptor can be used as a **candidate** in the following resources:

- [“/matcher/raw”](#)
- [“/verifiers/{verifier_id}/raw”](#)

6.3.4 Create objects using external data

You can create a temporary attribute of face by sending basic attributes and descriptors to LUNA PLATFORM. Thus you can store this data in external storage and send it to LP for the processing of requests only.

You can create an attribute or face using:

- Basic attributes and their samples.
- Descriptors (raw descriptor in Base64 or SDK descriptor in Base64).
- Both basic attributes and descriptors with the corresponding data.

Samples are optional and are not required for an attribute or face creation.

See the [“create temporary attribute” request](#) and [“create face request”](#) for details.

6.3.5 Checking images for compliance with standards

The Remote SDK service enables you to check images according to the [ISO/IEC 19794-5:2011](#) standard or user-specified thresholds using three ways:

- Request [“iso”](#).
- Parameter [“estimate_face_quality”](#) of the request [“detect faces”](#).
- Group of parameters [“face_quality”](#) of the policy [“detect_policy”](#) of the request [“generate events”](#).

For example, it is necessary to check whether the image is of a suitable format, specifying the [“JPEG”](#) and [“JPEG2000”](#) formats as a satisfactory condition. If the image fits this condition, the system will return the value [“1”](#), if the format of the processed image is different from the specified condition, the system will return the value [“0”](#). If the conditions are not set, the system will return the estimated value of the image format.

The list of estimations and checks performed is described in the [“Image check”](#) section.

The ability to perform check and estimation of image parameters is regulated by a special parameter in the license file.

6.3.6 Enable/disable several estimators and detectors

By default, the Remote SDK service is launched with all estimators and detectors enabled. If necessary, you can disable the use of some estimators or detectors when launching the Remote SDK container. Disabling unnecessary estimators enables you to save RAM or GPU memory, since when the Remote SDK service launches, the possibility of performing these estimates is checked and neural networks are loaded into memory.

If you disable the estimator or detector, you can also remove its neural network from the Remote SDK container.

Disabling estimators or detectors is possible by transferring documents with the names of estimators to the launch command of the Remote SDK service. Arguments are passed to the container using the “EXTEND_CMD” variable.

List of available estimators:

Argument	Description
--enable-all-estimators-by-default	enable all estimators by default
--enable-human-detector	simultaneous detector of bodies and bodies
--enable-face-detector	face detector
--enable-body-detector	body detector
--enable-people-count-estimator	people count estimator
--enable-face-landmarks5-estimator	face landmarks5 estimator
--enable-face-landmarks68-estimator	face landmarks68 estimator
--enable-head-pose-estimator	head pose estimator
--enable-deepfake-estimator	Deepfake estimator
--enable-liveness-estimator	Liveness estimator
--enable-fisheye-estimator	FishEye effect estimator
--enable-face-detection-background-estimator	image background estimator
--enable-face-warp-estimator	face sample estimator
--enable-body-warp-estimator	body sample estimator
--enable-quality-estimator	image quality estimator
--enable-image-color-type-estimator	face color type estimator
--enable-face-natural-light-estimator	natural light estimator
--enable-eyes-estimator	eyes estimator

Argument	Description
--enable-gaze-estimator	gaze estimator
--enable-mouth-attributes-estimator	mouth attributes estimator
--enable-emotions-estimator	emotions estimator
--enable-mask-estimator	mask estimator
--enable-glasses-estimator	glasses estimator
--enable-eyebrow-expression-estimator	eyebrow estimator
--enable-red-eyes-estimator	red eyes estimator
--enable-headwear-estimator	headwear estimator
--enable-basic-attributes-estimator	basic attributes estimator
--enable-face-descriptor-estimator	face descriptor extraction estimator
--enable-body-descriptor-estimator	body descriptor extraction estimator
--enable-body-attributes-estimator	body attributes estimator

You can explicitly specify which estimators and detectors are enabled or disabled by passing the appropriate arguments to the “EXTEND_CMD” variable, or you can enable (by default) or disable them all with the `enable-all-estimators-by-default` argument. You can turn off the use of all estimators and detectors, and then turn on specific estimators by passing the appropriate arguments.

Example of a command to start the Remote SDK service using only a face detector and estimators of a face sample and emotions.

```
docker run \
...
--env=EXTEND_CMD="--enable-all-estimators-by-default=0 --enable-face-
  detector=1 --enable-face-warp-estimator=1 --enable-emotions-estimator=1"
\
...
```

6.4 Handlers service

The Handlers service is used to create and store [handlers](#) and [verifiers](#).

The data of handlers and verifiers are stored in the [Handlers database](#).

6.4.1 Send events to third-party service

LUNA PLATFORM provides the ability to send notifications using two policies — “notification_policy” and “callbacks”.

The descriptions of these policies are provided below.

Policy	Description	Advantages
notification_policy	This policy allows redirecting events to the Sender service, which sends event data (notifications) through websockets. Sending notifications through websockets implies using an open bidirectional communication channel between the client and server. See detailed information in the “Sender service” section.	<ul style="list-style-type: none">• Direct, instantaneous data updates via websockets.• Efficient use of an open bidirectional channel.• Low latency in delivering notifications.
callbacks	This policy represents a mechanism for notifications based on the principles of HTTP webhooks. They provide asynchronous interaction between systems, allowing external services to react to the emergence of events. In the policy, you can specify specific parameters such as the protocol, the address of the external system, parameters, and authorization data.	<ul style="list-style-type: none">• More flexible mechanism for configuring notifications.• Easy integration with various external systems.• Uses familiar HTTP protocols and configurations.

6.5 Image Store service

The Image Store service stores the following data:

- Face and body samples. Samples are stored in Image Store by the Remote SDK service or you can save an external sample using the “[samples](#)” > “[detect faces](#)” and “[samples](#)” > “[save face/body sample](#)” requests.
- Reports about tasks. Reports are stored by the Tasks service workers.
- Any objects loaded using the “[create objects](#)” request.
- Clusterization information.

Image Store can save data either on a local storage device or in S3-compatible cloud storage (Amazon S3, etc.).

6.5.1 Buckets description

The data is stored in special directories called *buckets*. Each *bucket* has a unique name. Bucket names should be set in lower case.

The following buckets are used in LP:

- “[visionlabs-samples](#)”. This bucket stores face samples.
- “[visionlabs-bodies-samples](#)”. This bucket stores human bodies samples.
- “[visionlabs-image-origin](#)”. This bucket stores source images.
- “[visionlabs-objects](#)”. This bucket stores objects.
- “[task-result](#)”. This bucket stores the results received after tasks processing using the Tasks service.
- “[portraits](#)”. This bucket stores portraits. The bucket is required for the usage of Backport 3 service.

Buckets creation is described in LP 5 installation manual in the “[Buckets creation](#)” section.

After running the Image Store container and the commands for containers creation, the buckets are saved to local storage or S3.

By default, local files are stored in the “[/var/lib/luna/current/example-docker/image_store](#)” directory on the server. They are saved in the “[/srv/local_storage/](#)” directory in the Image Store container.

Bucket includes directories with samples or other data. The names of the directories correspond to the first four letters of the sample ID. All the samples are distributed to these directories according to their first four ID symbols.

Next to the bucket object is a “[*.meta.json](#)” file containing the “[account_id](#)” used when performing the request. If the bucket object is not a sample (for example, the bucket object is a JSON file in the “[task-result](#)” bucket), then the “[Content-Type](#)” will also be specified in this file.

An example of the folders structure in the “[visionlabs-samples](#)”, “[task-result](#)” and “[visionlabs-bodies-samples](#)” buckets is given below.

```
./local_storage/visionlabs-samples/8f4f/  
    8f4f0070-c464-460b-sf78-fac234df32e9.jpg  
    8f4f0070-c464-460b-sf78-fac234df32e9.meta.json  
    8f4f1253-d542-621b-9rf7-ha52111hm5s0.jpg  
    8f4f1253-d542-621b-9rf7-ha52111hm5s0.meta.json  
./local_storage/task-result/1b03/  
    1b0359af-ecd8-4712-8fc0-08401612d39b  
    1b0359af-ecd8-4712-8fc0-08401612d39b.meta.json  
./local_storage/visionlabs-bodies-samples/6e98/  
    6e987e9c-1c9c-4139-9ef4-4a78b8ab6eb6.jpg  

```

A significant amount of memory may be required when storing a large number of samples. A single sample takes about 30 Kbytes of the disk space.

It is also recommended to create backups of the samples. Samples are utilized when the NN version is changed or when you need to recover your database of faces.

6.5.2 Use S3-compatible storage

To enable the use of S3-compatible storage, you must perform the following steps:

- Make sure that the access key has sufficient authority to access the buckets of the S3-compatible storage.
- Launch the Image Store service (see “Image Store” section in the installation manual).
- Set the “S3” value for the “[storage_type](#)” setting of the Image Store service settings.
- Fill in the settings for connecting to an S3-compatible storage (host, Access Key and Secret Key, etc.) in the “S3” section of the Image Store service settings.
- Run the script for creating buckets `lis_bucket_create.py` (see the “Create buckets” section in the installation manual).

If necessary, you can disable SSL certificate verification using the “`verify_ssl`” setting in the “S3” section of the Image Store service settings. This enables you to use a self-signed SSL certificate.

6.5.3 Object TTL

You can set the object time to live (TTL) in buckets (both local and S3). Objects mean:

- Samples of faces or bodies.
- Images or objects created in the resources “/images” or “/objects”.
- Source images.
- Task results.

TTL for objects is calculated relative to the GMT time format.

TTL for objects is set in days in the following ways:

- During the creation of a bucket for all objects at once (basic TTL bucket policy).
- After creating a bucket for specific objects using requests to the corresponding resources.

The number of days is selected from the list in the corresponding requests (see below).

In addition to the number of days, the parameter can take the value “-1”, meaning that objects should be stored indefinitely.

6.5.3.1 Configuring basic TTL bucket policy

The basic TTL bucket policy can be configured in the following ways:

- Using the `--bucket-ttl` flag for the `lis_bucket_create.py` script. For example, `python3 ./base_scripts/lis_bucket_create.py -ii --bucket-ttl=2`.
- Using a request to the Image Store service. For example, `curl -X POST http://127.0.0.1:5020/1/buckets?bucket=visionlabs-samples?ttd=2`.

6.5.3.2 Configuring TTL for specific objects

TTL for specific objects can be configured using the “ttl” parameter in the following places:

- In the “storage_policy” > “face_sample_policy”, “body_sample_policy” and “image_origin_policy” handler policies.
- In the requests “create object”, “create images” and “save sample”.
- In the requests to create tasks or schedules in the “result_storage_policy” field.

If the “ttl” parameter is not specified, then the basic policy of the bucket in which the object is located (see above) will be applied.

6.5.3.3 Adding TTL to existing objects

You can add a TTL to an existing object using PUT requests to the `/objects`, `/images`, `/samples/{sample_type}` resources of the Image Store service. It is not possible to add TTL of task results to already created and executed tasks. You can add TTL of task results to an already created schedule using the request “[replace tasks schedule](#)”. For tasks created or running at the time of the request, the TTL of task results will not be applied.

You can add a TTL to an existing local bucket using a PUT request to the Image Store resource `/buckets`.

To add a TTL for a bucket located in S3, you need to perform a migration using the “`base_scripts/migrate_ttl_settings`” script from the Image Store service. This is because for TTL objects in S3 is applied via [tag related filters](#). The command to perform S3 bucket migration is given in the installation manual. See “[Migration to apply TTL to objects in S3](#)” for details on S3 bucket migration.

6.5.3.4 Supported cloud providers

Amazon S3 cloud providers, Yandex cloud storage and MinIO are supported.

6.5.3.5 Migration to apply TTL to objects in S3

Lifecycle customization for S3 is applied through filters associated with tags (see [official documentation](#)). This assumes that objects have a tag with a limited set of values, and buckets have a set of rules based on the value of that tag.

To add tags and rules, you must perform a migration. Migration is strictly necessary to fully apply lifecycle customization for the following reasons:

- Buckets without rules will not delete objects, even if the user specifies a lifetime for a specific object.
- Objects without tags will never be deleted, even if the user specifies a lifetime for the bucket.

You need to add the following tags and rules:

- To support TTL for buckets, you need to add a `vl-expire` tag with a default value for all existing objects.
- To support TTL for specific objects, you need to add a set or TTL-related lifecycle rules for existing segments:

```
{
  "ID": "vl-expire-<ttl>",
  "Expiration": {
    "Days": <ttl>,
  },
  "Filter": {"Tag": {"Key": "vl-expire", "Value": <ttl>}},
  "Status": "Enabled",
}
```

A set of specific tag values associated with an object's TTL is supported: 1, 2, 3, 4, 5, 6, 7, 14, 30, 60, 90, 180, 365.

The migration process consists of two stages:

- Configuration of the bucket life cycle, expanded with a set of life cycle rules related to TTL.
- Assigning a `vl-expire` tag to each object in the bucket, if it does not already have one.

Assigning a tag for each object can be skipped if necessary using the `-update-tags=0` argument.

See the upgrade manual for example commands to perform the migration.

Permission issues

By default, all S3 resources, including buckets, objects, and lifecycle configuration, are private. If necessary, default rules and tags can be created manually by the resource owner using one of the applicable methods. See the official S3 documentation for details.

Useful links to official documentation:

- [Creating a lifecycle configuration](#)
- [Put lifecycle configuration](#)
- [Managing object tags](#)
- [Expiring objects](#)

6.5.3.6 Expiration of TTL

When an object's TTL comes to an end, it is marked for deletion. For local buckets, the cleanup task is performed once a day (at 01:00 am). S3 buckets use internal ttl configuration rules. To prevent conflicts or duplication of cleanup tasks when multiple instances or worker processes are involved, a locking mechanism is implemented. This ensures that only one instance or worker process is responsible for performing the local storage cleanup process.

There may be a delay between the expiration date and the date the item is actually deleted. Both for local storage and S3.

6.5.3.7 Search for expiring objects

To find out when an object expires, you can use queries with the HEAD methods on the `/objects` and `/images` resources. These requests return `X-Luna-Expiry-Date` response headers, which indicate the date on which the object is no longer eligible for persistence.

6.5.4 External samples

You can send an external sample to Image Store. The external sample is received using third-party software or the VisionLabs software (e. g., FaceStream).

See the POST request on the `"/samples/{sample_type}"` resource in ["APIReferenceManual.html"](#) for details.

The external sample should correspond to certain standards so that LP could process it. Some of them are listed in the "Sample requirements" section.

The samples received using the VisionLabs software satisfy this requirement.

In case of third-party software, it is not guaranteed that the result of the external sample processing will be the same as for the VisionLabs sample. The *sample* can be of low quality (too dark, blurry and so on). Low quality leads to incorrect image processing results.

Anyway, it is recommended to consult VisionLabs before using external samples.

6.6 Accounts service

The Accounts service is intended for:

- Creation, management and storage of accounts.
- Creation, management and storage of tokens and their permissions.
- Verification of accounts and tokens.

See “[Accounts, tokens and authorization types](#)” section for more information about the authorization system in LUNA PLATFORM 5.

All created accounts, tokens and their permissions are saved in the [Accounts service database](#).

6.6.1 JWT Tokens algorithms

The JWT (JSON Web Tokens) authentication mechanism supports various algorithms for token signing. This section describes the default algorithm used and the necessary steps to use an alternative algorithm.

6.6.1.1 Default algorithm

By default, the service uses the HS256 algorithm to sign JWT tokens. If you want to use asymmetric cryptographic encryption, you can use the ES256 algorithm.

6.6.1.2 Use ES256 algorithm

To use the ES256 algorithm, follow these steps:

1. Generate a private ECDSA key.

First, you need to generate a private ECDSA key using the `prime256v1` curve. This can be done using command-line tools such as OpenSSL.

Example command:

```
openssl ecparam -genkey -name prime256v1 -out ec_private.pem
```

You can also generate a key protected by a password, for example:

```
openssl ecparam -genkey -name prime256v1 | openssl ec -aes256 -out  
ec_private_enc.pem
```

2. Encode the private key in Base64.

After generating the private key, encode it in Base64 format. This can be achieved with tools available in most operating systems.

Example command:

```
base64 -w0 ecdsa_private.pem > ecdsa_private_base64
```

3. Set the environment variable.

The encoded private key must be specified in the `ACCOUNTS_JWT_ECDSA_KEY` environment variable when starting the container. This allows the service to use the key for signing JWT tokens with the ES256 algorithm.

Additionally, if your private key is protected with a password, you can specify the password in the `ACCOUNTS_JWT_ECDSA_KEY_PASSWORD` environment variable.

Example container run command with environment variables:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=ACCOUNTS_JWT_ECDSA_KEY=jwt_ecdsa_key \  
--env=ACCOUNTS_JWT_ECDSA_KEY_PASSWORD=ecdsa_key_password \  
...
```

By following these steps, the service will be able to sign JWT tokens using the ES256 algorithm, providing enhanced security through asymmetric cryptography.

6.6.1.3 Impact of changing algorithm type

Switching the signing algorithm from HS256 to ES256 (or vice versa) has a significant impact on token validation. **All existing tokens signed with the previous algorithm will become invalid** after the changes are made. This happens because the token signature verification mechanism expects the structure and cryptographic base of the token to match the newly specified algorithm.

6.7 Faces service

Faces service is used for:

- Creating temporary attributes.
- Creating faces.
- Creating lists.
- Attaching faces to lists.
- Managing of the general database that stores faces with the attached data and lists.
- Receive information about the existing faces and lists.

6.8 Matching services

Python Matcher has the following features:

- Matching according to the specified filters. This matching is performed directly on the Faces or the Events database. Matching by DB is beneficial when several filters are set.
- Matching by lists. In this case, it is recommended that descriptors are save in the Python Matcher cache.

Python Matcher Proxy is used to route requests to Python Matcher services and matching plugins.

6.8.1 Python Matcher

Python Matcher utilizes Faces DB for filtration and matching when faces are set as candidates for matching and filters for them are specified. This feature is always enabled for Python Matcher.

Python Matcher utilizes Events DB for filtration and matching when events are set as candidates for matching and filters for them are specified. The matching using the Events DB is optional, and it is not used when the Events service is not utilized.

A VLMATCH matching function is required for matching by DB. It should be registered for the Faces DB and the Events DB. The function utilizes a library that should be compiled for your current DB version. You can find information about it in the installation manual in “VLMATCH library compilation”, “Create VLMATCH function for Faces DB”, and “Create VLMATCH function for Events DB” sections.

Python Matcher service additionally uses workers that process requests.

6.8.2 Python Matcher Proxy

The API service sends requests to the Python Matcher Proxy if it is configured in the API configuration. Then the Python Matcher Proxy service redirects requests to the Python Matcher service or to matching plugins (if they are used).

If the matching plugins are not used, then the service route requests only to the Python Matcher service. Thus, you don't need to use Python Matcher Proxy unless you intend to use matching plugins. See the “[Matching plugins](#)” section for a description of how the matching plugins work.

6.8.3 List caching

When faces are specified as candidates for matching and list IDs for them are specified as filters, Python Matcher performs a matching by lists.

By default, when the Python Matcher service is launched, all descriptors in all lists are cached in its memory.

Caching is managed by the “[DESCRIPTORS_CACHE](#)” section.

The Python Matcher service will not start until it loads all available descriptors into the cache.

When executing a list matching request, the Python Matcher service automatically adds it to the queue, from where it is picked up by the worker and sent to the Cached Matcher entity to perform a matching on cached data.

After performing the matching, the worker takes the results and returns them to the Python Matcher service and the user.

This caching enabling you to significantly increase the performance of the matching.

If necessary, you can process only specific lists using the parameter “[cached_data > faces_lists > include](#)” or exclude lists using the parameter “[cached_data > faces_lists > exclude](#)”. The latter is especially useful when working with the LUNA Index Module to implement the logic of processing parts of lists using Python Matcher, and parts using LIM Indexed Matcher.

For more information about LIM, see “[Matching a large set of descriptors](#)”.

6.8.3.1 Workers cache

When multiple workers are launched for the Python Matcher service, each of the workers uses the same descriptors cache.

This change can both speed up and slow down the service. If you need to ensure that the cache is stored in each of the Python Matcher workers, you should run each of the server instances separately.

6.9 Events service

The Events service is used for:

- Storage of all the created events in the Events database.
- Returning all the events that satisfy filters.
- Gathering statistics on all the existing events according to the specified aggregation and frequency/period.
- Storage of descriptors created for events.

As the event is a report, you can't modify already existing events.

The Events service should be enabled in the API service configuration file. Otherwise, events will not be saved to the database.

6.9.1 Database for Events

PostgreSQL is used as a database for the Events service.

The speed of request processing is primarily affected by:

- The number of events in the database.
- Lack of indexes for PostgreSQL.

PostgreSQL shows acceptable requests processing speed with the number of events from 1 000 000 to 10 000 000. If the number of events exceeds 10 000 000, the request to PostgreSQL may fail.

The speed of the statistics requests processing in the PostgreSQL database can be increased by [configuring the database](#) and creating indexes.

6.9.2 Geo position

You can add a geo position during event creation.

The geo position is represented as a JSON with GPS coordinates of the geographical point:

- “longitude” — Geographical longitude in degrees.
- “latitude” — Geographical latitude in degrees.

The geo position is specified in the “location” body parameter of the event creation request. See the [“Create new events”](#) section of the Events service reference manual.

You can use the geo position filter to receive all the events that occurred in the required area.

6.9.2.1 Geo position filter

A geo position filter is a bounding box specified by coordinates of its center (origin) and some delta.

It is specified using the following parameters:

- “origin_longitude”
- “origin_latitude”
- “longitude_delta”
- “latitude_delta”

The geo position filter can be used when you get events, get statistics on events, and perform events matching.

Geo position filter is considered as properly specified if:

- both “origin_longitude” and “origin_latitude” are set.
- neither “origin_longitude”, “origin_latitude”, “longitude_delta”, or “latitude_delta” is set.

If both “origin_longitude” and “origin_latitude” are set and “longitude_delta” is not set — the default value is applied (see the default value in the OpenAPI documentation).

Read the following recommendations before using geo position filters.

The general recommendations and restrictions for geo position filters are:

- Do not create filters with a vertex or a border on the International Date Line (IDL), the North Pole or the South Pole. They are not fully supported due to the features of database spatial index. The filtering result may be **unpredictable**.
- Geo position filters with edges more than 180 degrees long are not allowed.
- It is highly recommended to use the geo position filter citywide only. If a larger area is specified, the filtration results on the borders of the area can be unexpected due to the spatial features.
- Avoid creating a filter that is too extended along longitude or latitude. It is recommended to set the values of deltas close to each other.

The last two recommendations exist due to the spatial features of the filter. According to these features, when one or two deltas are set to large values, the result may differ from the expected though it will be correct. See the “[Filter features](#)” section for details.

6.9.2.2 Filter performance

Geo position filter performance depends on the spatial data type used to store event geo position in the database.

Two spatial data types are supported:

- GEOMETRY. Spatial object with coordinates expressed as (longitude, latitude) pairs, defined in the Cartesian plane. All calculations use Cartesian coordinates.
- GEOGRAPHY. Spatial object with coordinates expressed as (longitude, latitude) pairs, defined as on the surface of a perfect sphere, or a spatial object in the WGS84 coordinate system.

For a detailed description, see [geometry vs geography](#).

Geo position filter is based on the [ST_Covers](#) PostGIS function supported for both Geometry and Geography type.

6.9.2.3 Filter features

Geo position filter has some features caused by PostGIS.

When geography type is used and the geo position filter covers a wide portion of the planet surface, filter result may be unexpected but geographically correct due to some spatial features.

The following example illustrates this case.

An event with the following geo position was added in the database:

```
{
  "longitude": 16.79,
  "latitude": 64.92,
}
```

We apply a geo position filter and try to find the required point on the map. The filter is too extended along the longitude:

```
{
  "origin_longitude": 16.79,
  "origin_latitude": 64.92,
  "longitude_delta": 2,
  "latitude_delta": 0.01,
}
```

This filter **will not** return the expected event. The event will be filtered due to spatial features. Here is the illustration showing that the point is outside the filter.

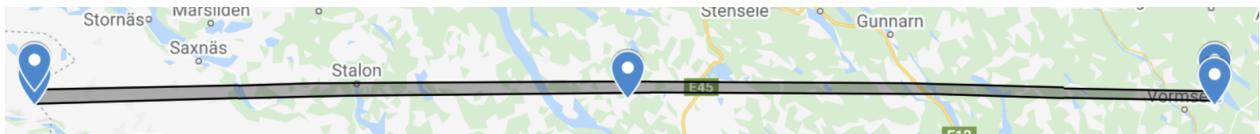


Figure 30: Too wide zone

You should consider this feature to create a correct filter.

For details, see [Geography](#).

6.9.3 Events creation

Events are created using handlers. Handlers are stored in the Handlers database. You should specify the required handler ID in the event creation request. All the data stored in the event will be received according to the handler parameters.

You should perform two separate requests for event creation.

The first request creates a handler. Handler includes policies that describes how the image is processed hence defining the LP services used for the processing.

The second request creates new events using the existing handler. An event is created for each image that has been processed.

You can specify the following additional data for each event creation request:

- external ID (for created faces)
- user data (for created faces)
- source (for created events)
- tags (for created events)

The handler is processed policy after policy. All the data from the request is processed by a policy before going to the next policy. The “detect” policy is performed for all the images from the request, then “multiface” policy is applied, then the “extract” policy is performed for all the received samples, etc. For more information about handlers, see the “[Handlers description](#)” section.

6.9.4 Events meta-information

If any additional data needs to be stored along with the event, the “meta” field should be used. The “meta” field stores data in the JSON format. The total size of the data stored in the “meta” field for one event cannot exceed 2 MB. It is assumed that with the help of this functionality, the user will create his own data model (event structure) and will use it to store the necessary data.

Note that you cannot specify field names with spaces in the “meta” field.

Data in the “meta” field can be set in the following ways:

- In the “[generate events](#)” request body with the content type `application/json` or `multipart/form-data`.
- In the “[save event](#)” request body.
- Using a custom plugin or client application.

In the “[generate events](#)” request body, it is possible to set the “meta” field both for specific images and for all images at once (mutual meta-information). For requests with aggregation enabled, only mutual meta-information will be used for the aggregated event, and meta-information for specific images will be ignored. See the detailed information in the “[generate events](#)” request body in the OpenAPI specification.

Example of recording the “meta” field:

```
{
  "meta": {
    "user_info": {
      "temperature": 36.6
    }
  }
}
```

In order to store multiple structures, it is necessary to explicitly separate them to avoid overlapping fields. For example, as follows:

```
{
  "struct1": {
    ...
  },
  "struct2": {
    ...
  }
}
```

6.9.4.1 Search by “meta” field

You can get the contents of the “meta” field using the appropriate filter in the “[get events](#)” request.

The filter should be entered using a specific syntax — `meta.<path.to.field>__<operator>:<type>`, where:

- `meta.` — An indication that the “meta” field of the Events database is being accessed.
- `<path.to.field>` — Path to the object. A dot (.) is used to navigate nested objects. For example, in the string `{"user_info":{"temperature":"36.6"}}` to refer to the temperature object, use the following filter `meta.user_info.temperature`.
- `__<operator>` — One of the following operators — `eq` (default), `neq`, `like`, `nlike`, `in`, `nin`, `gt`, `gte`, `lt`, `lte`. For example, `meta.user_info.temperature__gte`;
- `:type` — one of the following data types — `string`, `integer`, `numeric`. For example, `meta.user_info.temperature__gte:numeric`.

For each operator, the use of certain data types is available. See the table of operator dependency on data types in the OpenAPI specification.

If necessary, you can build an index to improve the search. See the [Events developer manual](#) for details on building an index.

6.9.4.2 Important notes

When working with the “meta” field, remember the following:

- you need to keep data consistent with given schemes; in case of a mismatch, PostgreSQL will not allow inserting a row with a type that cannot be added to the existing index (if any);
- if necessary, you can migrate data;
- if necessary, you can build an index;
- specify the data type when performing a request (by default, all values are assumed to be strings);
- you need to pay attention to the names of the fields; fields to be filtered by must not contain reserved keywords like `:int`, double underscores, special symbols, and so on.

6.10 Sender service

The Sender service is an additional service that is used to send events via web sockets. This service communicates with the Handlers service (in which events are created) through the [pub/sub](#) mechanism via the Redis DB channel.

If necessary, you can send notifications over the HTTP protocol. See the [“Send events to third-party service”](#) section for more details.

Events are created based on handlers. To receive notifications, the [“notification_policy”](#) must be enabled. This policy has filters that enable you to send notifications only under certain conditions, for example, to send only if the candidate is very similar to the reference (the [“similarity__lte”](#) parameter).

You should configure web sockets connection using special request. It is recommended create web sockets connection using the [“/ws”](#) resource of the API service. You can specify filters (query parameters) in the request, i.e. you can configure the Sender service to receive only certain events. See [OpenAPI specification](#) for detailed information about the configuration of creating a connection to a web socket.

Configuring web sockets directly via Sender is also available (see [“/ws”](#) of the Sender service). It can be used to reduce the load on the API service.

When an event is created it can be:

- Saved to the Events database. The Events service should be enabled to save an event.
- Returned in the response without saving to the database.

In both cases, the event is sent via the Redis DB channel to the Sender service.

In this case, the Redis DB acts as a connection between Sender and Handlers services and does not store transferred events.

The Sender service is independent of the Events service. Events can be sent to Sender even if the Events service is disabled.

Creating handlers and specifying filters for sending notifications

1. The user sends the [“create handler”](#) request to the API service, where it enables the [“notification_policy”](#) and sets filters according to which events will be sent to the Sender service.
2. The API service sends a request to the Handlers service.
3. The Handlers service sends a response to the API service.
4. The API service sends the [“handler_id”](#) to the user.

The user saves the ID [“handler_id”](#), which is necessary for creating events.

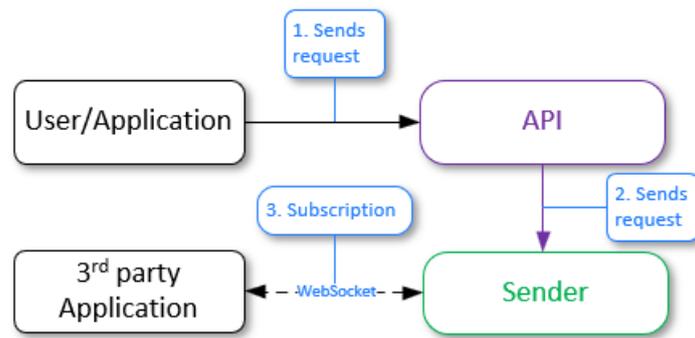


Figure 31: Creating handlers and specifying filters for sending notifications

Activation of subscription to events and filtering of their sending

1. The user or application sends a request “ws handshake” to the API service and sets filters through which it will be possible to filter the received data from the Handlers service.
2. The API service sends a request to the Sender service.
3. The Sender service establishes a connection via web sockets with the user application.

Now, when an event is generated, it will be automatically redirected to the Sender service (see below) in accordance with the specified filters.

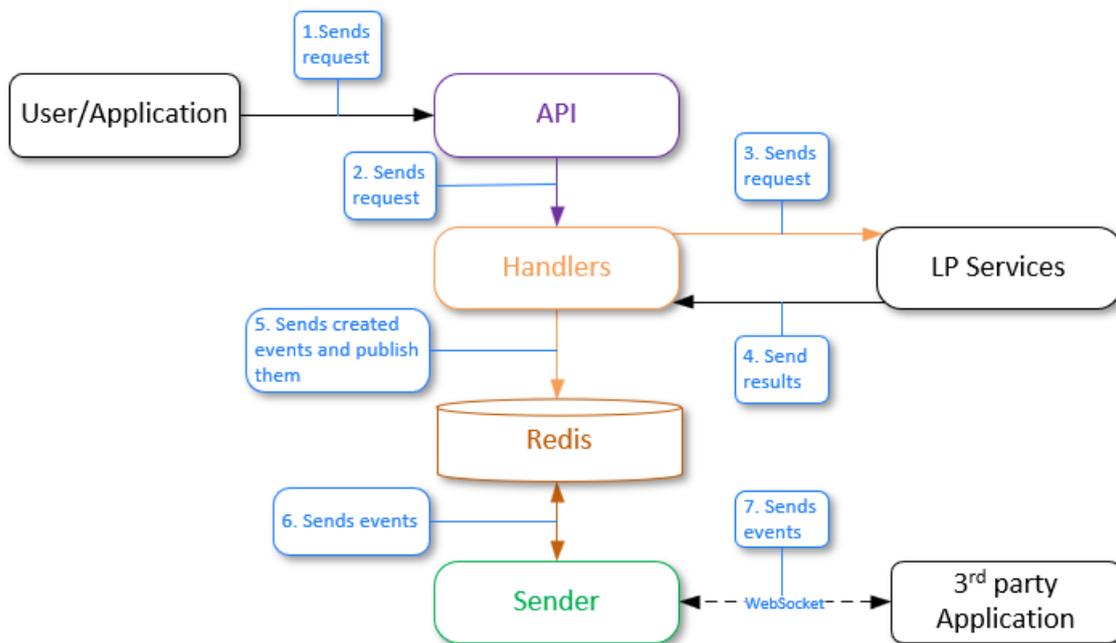


Figure 32: Activating event subscriptions and filtering their sending

Event generation and sending to Sender

The general workflow is as follows:

1. A user or an application sends the “generate events” request to the API service.
2. The API service sends the request to the Handlers service.
3. The Handlers service sends requests to the corresponding LP services.
4. LP services process the requests and send results. New events are created.
5. The Handlers service sends an event to the Redis database using the pub/sub model. Redis has a channel to which the Sender service is subscribed, and it is waiting for messages to be received from this channel.
6. Redis sends the received events to Sender by the channel.
7. Third-party party applications should be subscribed to the Sender service via web sockets to receive events. If there is a subscribed third-party party application, Sender sends events to it according to the specified filters.

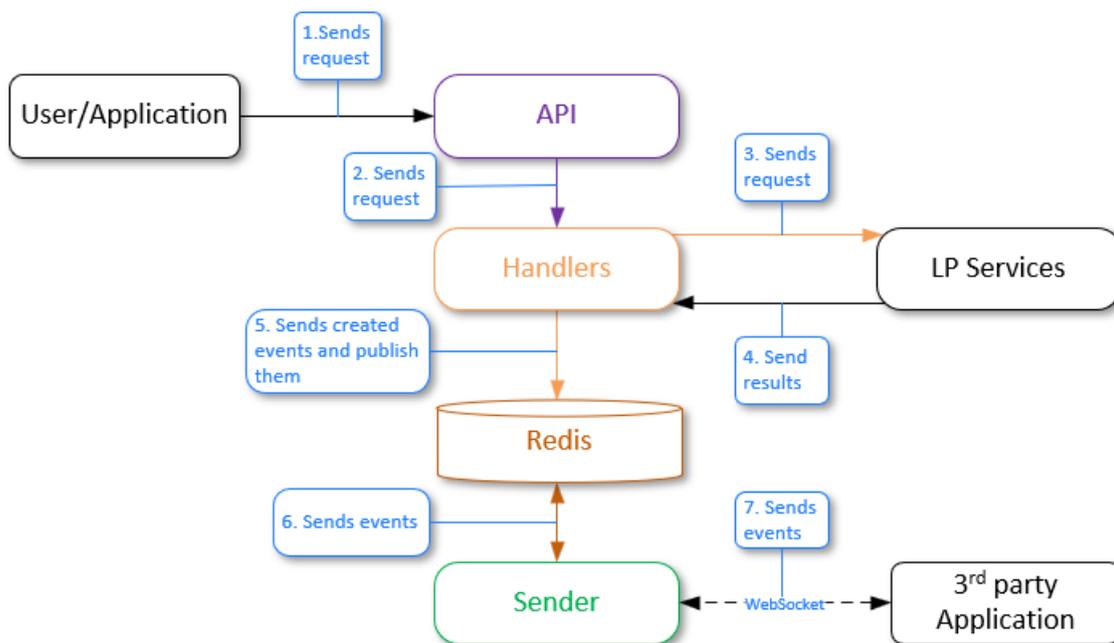


Figure 33: Sender workflow

See the OpenAPI documentation for information about the JSON structure returned by the Sender service.

6.11 Tasks service

The Tasks service is used for long tasks processing.

6.11.1 General information about tasks

As tasks processing takes time, the task ID is returned in the response to the task creation.

After the task processing is finished, you can receive the task results using the “task” > “get task result” request. You should specify the task ID to receive its results.

You can find the examples of tasks processing results in the response section of “task” > “get task result” request. You should select the task type in the **Response samples** section of documentation.

```
{
  - "result": [
    + { - }
  ],
  - "errors": [
    + { - }
  ]
}
```

Figure 34: Select required example

You should make sure that the task was finished before requesting its results:

- You can check the task status by specifying the task ID in the “tasks” > “get task” request. There are the following task statuses:

Tasks status	Value
pending	0
in progress	1
cancelled	2

Tasks status	Value
failed	3
collect results	4
done	5

- You can receive information about all the tasks using the “tasks” > “get tasks” request. You can set filter to receive information about tasks of interest only.

6.11.2 Clustering task

As the result of the task a cluster with objects selected according to the specified filters for faces or events is created. Objects corresponding to all of the filters will be added to the cluster. Available filters depend on the object type: events or faces.

You can receive the task status or result using additional requests (see the “[General information about tasks](#)”).

You can use the reporter task to receive the report about objects added to clusters.

Clustering is performed in several steps:

- Objects with descriptors are collected according to provided filters
- Every object is matched with all the other objects
- Create clusters as groups of “[connected components](#)” from the similarity graph.

Here “connected” means that similarity is greater than provided threshold or default “DEFAULT_CLUSTERING_THRESHOLD” from the config.

- If needed, download existing images corresponding to each object: avatar for a face, first sample for an event.

As a result of the task an array of clusters is returned. A cluster includes IDs of objects (faces or events) whose similarity is greater then the specified threshold. You can use the information for further data analysis.

```
{
  "errors": [],
  "result": {
    "clusters": [
      [
        "6c721b90-f5a0-409a-ab70-bc339a70184c"
      ]
    ]
  }
}
```

```

    [
      "8bc6e8df-410b-4065-b592-abc5f0432a1c"
    ],
    [
      "e4e3fc66-53b4-448c-9c88-f430c00cb7ea"
    ],
    [
      "02a3a1c4-93d7-4b69-99ec-21d5ef23852e",
      "144244cb-e10e-478c-bdac-18cd2eb27ee6",
      "1f4cdbcb-7b1e-40cc-873b-3ff7fa6a6cf0"
    ]
  ],
  "total_objects": 6,
  "total_clusters": 4
}
}

```

The clustering task result can also include information about errors occurred during the objects processing.

For such a task, you can [create a schedule](#).

6.11.3 Reporter task

As a result of the task, the report on the clustering task is created. You can select data that should be added to the report. The report has CSV format.

You can receive the task status or result using additional requests (see the [“General information about tasks”](#)).

You should specify the clustering task ID and the columns that should be added to the report. The selected columns correspond to the general events and faces fields.

Make sure that the selected columns correspond to the objects selected in the clustering task.

You can also receive the images for all the objects in clusters if they are available.

6.11.4 Exporter task

The task enables you to collect event and/or face data and export them from LP to a CSV file. The file rows represent requested objects and corresponding samples (if they were requested).

This task uses memory when collecting data. So, its possible that [Tasks Worker](#) will be killed by OOM (Out-Of-Memory) killer if you request a lot of data.

You can export event or face data using the “/tasks/exporter” request. You should specify what type of object is required by setting `objects_type` parameter when creating a request. You can also narrow your request by providing filters for faces and events objects. See the “[exporter task](#)” request in the API service reference manual.

As a result of the task a zip archive containing a CSV file is returned.

You can receive the task status or result using additional requests (see the “[General information about tasks](#)”).

When executing the Exporter task with a large number of faces in the Faces database (for example, 90,000,000 faces), the execution time of requests to the Faces service can be significantly increased. To speed up request execution, you can set the PostgreSQL setting “[parallel_setup_cost](#)” to 500. However, be aware that changing this setting may have other consequences, so you should be careful when changing the setting.

For such a task, you can [create a schedule](#).

6.11.5 Cross-matching task

When the task is performed, all the references are matched with all the candidates. References and candidates are set using filters for faces and events.

Matching is performed only for objects that contain extracted descriptors.

You can specify the maximum number of matching candidates returned for every match using the `Limit` field.

You can set a `threshold` to specify the minimal acceptable value of similarity. If the similarity of two descriptors is lower than the specified value, the matching result will be ignored and not returned in the response. References without matches with any candidates are also ignored.

Cross-matching is performed in several steps:

- collect objects having descriptors using provided filters
- match every reference object with every candidate object
- match results are sorted (lexicographically) and cropped (limit and threshold are applied)

You can receive the task status or results using additional requests (see the “[General information about tasks](#)”).

As a result an array is returned. Each element of the array includes a reference and top similar candidates for it. Information about errors occurred during the task execution is also returned in the response.

```

{
  "result": [
    {
      "reference_id": "e99d42df-6859-4ab7-98d4-dafd18f47f30",
      "candidates": [
        {
          "candidate_id": "93de0ea1-0d21-4b67-8f3f-d871c159b740",
          "similarity": 0.548252
        },
        {
          "candidate_id": "54860fc6-c726-4521-9c7f-3fa354983e02",
          "similarity": 0.62344
        }
      ]
    },
    {
      "reference_id": "345af6e3-625b-4f09-a54c-3be4c834780d",
      "candidates": [
        {
          "candidate_id": "6ade1494-1138-49ac-bfd3-29e9f5027240",
          "similarity": 0.7123213
        },
        {
          "candidate_id": "e0e3c474-9099-4fad-ac61-d892cd6688bf",
          "similarity": 0.9543
        }
      ]
    }
  ],
  "errors": [
    {
      "error_id": 10,
      "task_id": 123,
      "subtask_id": 5,
      "error_code": 0,
      "description": "Faces not found",
      "detail": "One or more faces not found, including face with id '8f4f0070-c464-460b-bf78-fac225df72e9'",
      "additional_info": "8f4f0070-c464-460b-bf78-fac225df72e9",
      "error_time": "2018-08-11T09:11:41.674Z"
    }
  ]
}

```

For such a task, you can [create a schedule](#).

6.11.6 Linker task

The task enables you to attach faces to lists according to the specified filters.

You can specify creation of a new list or specify the already existing list in the requests.

You can specify filters for faces or events to perform the task. When an event is specified for linking to list a new face is created based on the event.

If the `create_time_lt` filter is not specified, it will be set to the current time.

As the result of the task you receive IDs of faces linked to the list.

You can receive the task status or result using additional requests (see the [“General information about tasks”](#)).

Task execution process for faces:

- A list is created (if `create_list` parameter is set to 1) or the specified `list_id` existence is checked.
- Face ID boundaries are received. Then one or several subtasks are created with about 1000 face ids per each. The number depends on face ID spreading.
- For each subtask:
 - Face IDs are received. They are specified for the current subtask by filters in the subtask content.
 - The request is sent to the Luna Faces to link specified faces to the specified list.
 - The result for each subtask is saved to the Image Store service.
- After the last subtask is finished, the worker collects results of all the subtasks, merges them and puts them to the Image Store service (as task result).

Task execution process for events:

- A list is created (if `create_list` parameter is set to 1) or the specified `list_id` existence is checked.
- Events page numbers are received. Then one or several subtasks are created.
- For each subtask:
 - Event with their descriptors are received from the Events service.
 - Faces are created using the Faces service. Attribute(s) and sample(s) are added to the faces.
 - The request is sent to the Luna Faces to link specified faces to the specified list.
 - The result for each subtask is saved to the Image Store service.
- After the last subtask is finished, the worker collects results of all the subtasks, merges them and puts them to the Image Store service (as task result).

For such a task, you can [create a schedule](#).

6.11.7 Garbage collection task

During the task processing, faces, events or descriptors can be deleted.

- When descriptors are set as a GC target, you should specify the descriptor version. All the descriptors of the specified version will be deleted.
- When events are set as a GC target, you should specify one or several of the following parameters:
 - Account ID.
 - Upper excluded boundary of event creation time.
 - Upper excluded boundary of the event appearance in the video stream.
 - Handler ID used for the event creation.
- When faces are set as a GC target, you should specify one or several of the following parameters:
 - Upper excluded boundary of face creation time.
 - Lower included boundary of face creation time.
 - User data.
 - List ID.

If necessary, you can delete samples along with faces or events. You can also delete image origins for events.

Garbage collection task with faces or events set as the target can be processed using the API service API, while the Admin or Task services API can be used to set faces, events and descriptors as the target. Thus the specified objects will be deleted for all the existing accounts.

You can receive the task status or result using additional requests (see the “[General information about tasks](#)”).

For such a task, you can [create a schedule](#).

6.11.8 Additional extraction task

The Additional extraction task re-extracts descriptors extracted using the previous neural network model using a new version of the neural network. This enables you to save previously used descriptors when updating the neural network model. If there is no need to use the old descriptors, then you can not perform this task and only update the neural network model in the Configurator settings.

This section describes how to work with the Additional extraction task. See detailed information about neural networks, the process of updating a neural network to a new model and relevant examples in the “[Neural networks](#)” section.

Re-extraction can be performed for face and event objects. You can re-extract the descriptors of faces, descriptors of bodies (for events) or basic attributes if they were not extracted earlier.

The samples for descriptors should be stored for the task execution. If any descriptors do not have source samples, they cannot be updated to a new NN version.

The re-extraction tasks are used for the update to a new neural network for descriptors extraction. All the descriptors of the previous version will be re-extracted using a new NN.

It is highly recommended not to perform any requests changing the state of databases during the descriptor version updates. It can lead to data loss.

Create backups of LP databases and the Image Store storage before launching the additional extraction task.

When processing the task, a new neural network descriptor is extracted for each object (face or event) whose descriptor version matches the version specified in the “DEFAULT_FACE_DESCRIPTOR_VERSION” (for faces) or “DEFAULT_HUMAN_DESCRIPTOR_VERSION” (for bodies) settings. Descriptors whose version does not match the version specified in these settings are not re-extracted. They can be removed using the [Garbage collection task](#).

Request to the Admin service

You need to make a request to the “[additional_extract](#)” resource, specifying the following parameters in the request body:

- “content” > “extraction_target” – Face descriptors, body descriptors, basic attributes.
- “content” > “options” > “descriptor_version” – New neural network version (not applicable for basic attributes).
- “content” > “filters” > “object_type” – Faces or events.

If necessary, you can additionally filter the object type by “account_id”, “face_id__lt”, etc.

See the “[create additional extract task](#)” request in the Admin service OpenAPI specification for more information.

You can receive the task status or result using additional requests (see the “[General information about tasks](#)”).

Admin user interface

You need to do the following:

- Go to the Admin user interface: http://<admin_server_ip>:5010/tasks.
- Run the additional extraction task using the corresponding button.
- In the window that appears, set the object type (face or event), the extraction type (face descriptor, body descriptor or basic attributes), new neural network model (not applicable for basic attributes) and click “Start”, confirming the start of the task.

Figure 35: Set required settings

If necessary, you can additionally filter the object type by “account_id”.

See the detailed information about the Admin user interface in the [“Admin user interface”](#) section.

For such a task, you can [create a schedule](#).

6.11.9 ROC-curve calculating task

As a result of the task, the Receiver Operating Characteristic curve with TPR (True Positive Rate) against the FPR (False Positive Rate) is created.

See additional information about ROC-curve creation in [“TasksDevelopmentManual”](#).

ROC calculation task

ROC (or Receiver Operating Characteristic) is a performance measurement for classification tasks at various thresholds settings. The ROC-curve is plotted with TPR (True Positive Rate) against the FPR (False Positive Rate). TPR is a true positive match pair count divided by a count of total expected positive match pairs, and FPR is a false positive match pair count divided by a count of total expected negative

match pairs. Each point (FPR, TPR) of the ROC-cure corresponds to a certain similarity threshold. See more at [wiki](#).

Using ROC the model performance is determined by looking at:

- Area under the ROC-curve (or AUC).
- Type I and type II error rates equal point, i.e. the ROC-curve and the secondary main diagonal intersection point.

The model performance also determined by hit into the top-N probability, i.e. probability of hit a positive match pair into the top-N for any match result group sorted by similarity.

It requires “markup” to make a ROC task. One can optionally specify “threshold_hit_top” (default 0) to calculate hit into the top-N probability, the match “limit” (default 5), “key_FPRs” — list of key FPR values to calculate ROC-curve key points, and “filters” with “account_id”. Also, it needs “account_id” for task creation.

You can receive the task status or result using additional requests (see the [“General information about tasks”](#)).

Markup

Markup is expected in the following format:

```
[{'face_id': <face_id>, 'label': <label>}]
```

Label (or group id) can be a number or any string.

Example:

```
[{'face_id': '94ae2c69-277a-4e46-817d-543f7d3446e2', 'label': 0},  
 {'face_id': 'cd6b52be-cdc1-40a8-938b-a97a1f77d196', 'label': 1},  
 {'face_id': 'cb9bda07-8e95-4d71-98ee-5905a36ec74a', 'label': 2},  
 {'face_id': '4e5e32bb-113d-4c22-ac7f-8f6b48736378', 'label': 3},  
 {'face_id': 'c43c0c0f-1368-41c0-b51c-f78a96672900', 'label': 2}]
```

For such a task, you can [create a schedule](#).

6.11.10 Estimator task

The estimator task enables you to perform batch processing of images using the specified policies.

As a result of the task performing, JSON is returned with data for each of the processed images and information about the errors that have occurred.

In the request body, you can specify the `handler_id` of an already existing static or dynamic handler. For the dynamic `handler_id`, the ability to set the required policies is available. In addition, you can create a static handler specifying policies in the request.

The resource can accept five types of sources with images for processing:

- ZIP archive
- S3-like storage
- Network disk
- FTP server
- Samba network file system

To obtain correct results of image processing using the Estimator task, all processed images should be either in the source format or in the format of samples. The type of transferred images is specified in the request in the “`image_type`” parameter.

For such a task, you can [create a schedule](#). When creating a schedule, it is not possible to specify a ZIP archive as an image source.

ZIP archive as image source of estimator task

The resource accepts for processing a link to a ZIP archive with images. The size of the archive is set using the “`ARCHIVE_MAX_SIZE`” parameter in the “`config.py`” configuration file of the Tasks service. The default size is 100 GB. An external URL or the URL to an archive saved in the Image Store can be used as a link to the archive. In the second case, the archive should first be saved to the LP using a POST request to the “`/objects`” resource.

When using an external URL, the ZIP archive is first downloaded to the [Tasks Worker](#) container storage, where the images are unpacked and processed. After the end of the task, the archive is deleted from the repository along with the unpacked images.

It is necessary to take into account the availability of free space for the above actions.

The archive can be password protected. The password can be passed in the request using the “`authorization`” -> “`password`” parameter.

S3-like storage as image source of estimator task

The following parameters can be set for this type of source:

- “`bucket_name`” — Bucket name/[Access Point ARN](#)/[Outpost ARN](#) (required).
- “`endpoint`” — Storage endpoint (only when specifying the bucket name).
- “`region`” — Bucket region (only when specifying the bucket name).
- “`prefix`” — [File key prefix](#). It can also be used to load images from a specific folder, such as “`2022/January`”.

The following parameters are used to configure authorization:

- Public access key (required)
- Secret access key (required)
- Authorization signature version (“s3v2”/“s3v4”)

It is also possible to recursively download images from nested bucket folders and save original images.

For more information about working with S3-like repositories, see [AWS User Guide](#).

Network disk as image source of estimator task

The following parameters can be set for this type of source:

- “path” — Absolute path to the directory with images in the container (required).
- “follow_links” — Enables/disables symbolic link processing.
- “prefix” — File key prefix.
- “postfix” — File key postfix.

See an example of using prefixes and postfixes in the [“/tasks/estimator”](#) resource description.

When using a network disk as an image source and launching Tasks and [Tasks Worker](#) services through Docker containers, it is necessary to mount the directory with images from the network disk to the local directory and synchronize it with the specified directory in the container. You can mount a directory from a network disk in any convenient way. After that, you can synchronize the mounted directory with the directory in the container using the following command when launching the Tasks and Tasks Worker services:

```
docker run \
...
-v /var/lib/luna/current/images:/srv/images
...
```

Here:

- `/var/lib/luna/current/images` — Path to the previously mounted directory with images from the network disk.
- `/srv/images` - Path to the directory with the images in the container where they will be moved from the network disk. This path should be specified in the request body of the Estimator task (the “path” parameter).

As for S3-like storage, the ability to recursively download images from nested bucket folders is available.

FTP server as image source of estimator task

For this type of source, the following parameters can be set in the request body for connecting to the FTP server:

- “host” — FTP server IP address or hostname (required).
- “port” — FTP server port.

- “max_sessions” — Maximum number of allowed sessions on the FTP server.
- “user”, “password” — Authorization parameters (required).

As in Estimator tasks using S3-like storage or network disk as image sources, it is possible to set the path to the directory with images, recursively receive images from nested directories, select the type of transferred images, and specify the prefix and postfix.

See an example of using prefixes and postfixes in the [“/tasks/estimator”](#) resource description.

Samba as image source of estimator task

For this type of source, the parameters are similar to those of an FTP server, except for the “max_sessions” parameter. Also, if authorization data is not specified, the connection to Samba will be performed as a guest.

6.11.11 Task processing

The Tasks service includes the Tasks service and Tasks workers. Tasks receives requests to the Tasks service, creates tasks in the DB and sends subtasks to Tasks workers. The workers are implemented as a separate Tasks Worker container. Tasks workers receive subtasks and perform all the required requests to other services to solve the subtasks.

The general approach for working with tasks is listed below.

- User sends the request for creation of a new task.
- Tasks service creates a new task and sends subtasks to workers.
- Tasks workers process subtasks and create reports.
- If several workers have processed subtasks and have created several reports, the worker, which finished the last subtask, gathers all the reports and creates a single report.
- When the task is finished, the last worker updated its status in the Tasks database.
- User can send requests to receive information about tasks and subtasks and number of active tasks. The user can cancel or delete tasks.
- User can receive information about errors that occurred during execution of the tasks.
- After the task is finished the user can send a request to receive results of the task.

See the [“Tasks diagrams”](#) section for details about tasks processing.

6.11.12 Running scheduled tasks

In LUNA PLATFORM, it is possible to set a schedule for Garbage collection, Clusterization, Exporter, Linker, Estimator, Additional extract, Cross-matching and Roc-curve calculating tasks.

To use a filter relative to the current time (“now-time”), the current time will be counted not from the creation of the schedule, but from the creation of the task by the schedule in accordance with

the cron expression. See [“Now-time filters”](#) for details.

The schedule is created using the request [“create tasks schedule”](#) to the API service, which specifies the contents of the task being created and the time interval for its launch. To specify the time interval, [Cron expressions](#) are used.

Cron expressions are used to determine the task execution schedule. They consist of five fields separated by spaces. Each field defines a specific time interval in which the task should be completed. The week number starts from Sunday.

For tasks that can only be performed using the Admin service (for example, the task of removing some objects using the GC task), you can assign a schedule only in the Admin service.

In response to the request, a `“schedule_id”` is issued, which can be used to get information about the status of the task, the time of the next task, etc. (requests [“get tasks schedule”](#) and [“get tasks schedules”](#)). The id and all additional information are stored in the `“schedule”` table of the Tasks database.

If necessary, you can create a delayed schedule, and then activate it using the `“action” = “start”` parameter of the [“patch tasks schedule”](#) request. Similarly, you can stop the scheduled task using `“action” = “stop”`. To delete a schedule, you can use the [“delete tasks schedule”](#) request.

Permissions to work with schedules are specified in the token with the `“task”` permission. This means that if the user has permission to work with tasks, then he will also be able to use the schedule.

The possibility of schedule creation is also available for [Lambda tasks](#).

6.11.12.1 Examples of Cron expressions

This section describes various examples of Cron expressions.

1. Run the task every day at 3 a.m.:

```
0 3 * * *
```

2. Run the task every Friday at 18:30:

```
30 18 * * 5
```

3. Run the task every first day of the month at noon:

```
0 12 1 * *
```

4. Run the task every 15 minutes:

```
*/15 * * * *
```

5. Run the task every morning at 8:00, except weekends (Saturday and Sunday):

```
0 8 * * 1-5
```

6. Run the task at 9:00 am on the first and 15th day of each month, but only if it is Monday:

```
0 9 1,15 * 1
```

6.11.13 Send notification about task and subtask status changes

If necessary, you can send notifications about changes in task and subtask status using the callback mechanism. Callbacks allow you to send data to a third-party system at a specified URL or to Telegram. To configure notifications, you need to configure the “notification_policy” in the request parameters of the corresponding task.

You can also configure sending notifications for tasks and subtask in the schedule settings.

If necessary, you can obtain information about the current state of the notification policy or change some policy data using “[get task notification policy](#)” and “[replace task notification policy](#)” requests.

6.11.14 Additional protection for passwords and tokens

Passwords and tokens passed in [Estimator task](#) and in “notification_policy” can be additionally encrypted. To do this, pass custom values to the FERNET_PASSPHRASE and SALT environment variables when starting the Tasks service container.

FERNET_PASSPHRASE is the password or key used to encrypt data using the Fernet algorithm.

SALT is a random string added to the password before it is hashed.

Fernet is a symmetric encryption algorithm that provides authentication and data integrity as well as confidentiality. With this algorithm, the same key is used to encrypt and decrypt data.

Salt is added to make it more difficult to crack the password by brute force. Each time a password is hashed, a unique string is used, making identical passwords hashed differently. This increases the security of the system, especially if users have the same passwords.

Example of a container startup command passing environment variables:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=FERNET_PASSPHRASE=security_passphrase.  
--env=SALT=salt_for_passwords_and_tokens.  
...
```

Important: When the container is started with the above environment variables specified, the old passwords and tokens will no longer work. Additional migration steps must be performed (see section below).

[6.11.14.1 Add encryption when updating](#)

In order to add additional protection for already existing passwords and tokens, you need to specify environment variables in the Tasks database migration command (see above). After that, you should start a new Tasks container with environment variables specified to enable the use of encryption when creating new objects.

6.12 Admin service

The Admin service is used to perform general administrative routines:

- Manage user accounts.
- Receive information about objects belonging to different accounts.
- Create garbage collection tasks.
- Create tasks to extract descriptors with a new neural network version.
- Receive reports and errors on processed tasks.
- Cancel and delete existing tasks.

Admin service has access to all the data attached to different accounts.

Three [types](#) of accounts can be created in the Admin service — “user”, “advanced_user” and “admin”. The first two types are created using an account creation request to the API service, but the third type can only be created using the Admin service.

Using the “admin” account type, you can log in to the interface and perform the above tasks. An account with the “admin” type can be created either in the user interface (see above) or by requesting the [“/4/accounts”](#) resource of the Admin service. To create an account in the last way, you need to specify a username and password.

If you are creating an account for the first time, you must use the default login and password.

Example of CURL request to the [“/4/accounts”](#) resource of the Admin service:

```
curl --location --request POST 'http://127.0.0.1:5010/4/accounts' \  
--header 'Authorization: Basic cm9vdEB2aXNpb25sYWJzLmFpOnJvb3Q=' \  
--header 'Content-Type: application/json' \  
--data '{  
  "login": "mylogin@gmail.com",  
  "password": "password",  
  "account_type": "admin",  
  "description": "description"  
}'
```

All the requests for Admin service are described in [Admin service reference manual](#).

6.12.1 Admin user interface

The user interface of the Admin service is designed to simplify the work with administrative tasks.

The interface can be opened in a browser by specifying the address and port of the Admin service: `<Admin_server_address>:<Admin_server_port>`.

The default Admin service port is 5010.

The default login and password to access the interface are `root@visionlabs.ai/root`. You can also use default login and password in Base64 format — `cm9vdEB2aXNpb25sYWJzLmFpOnJvb3Q=`.

You can change the default password for the Admin service using the [“Change authorization”](#) request.

There are three tabs on the page:

- **Accounts.** The tab is designed to provide information about all created accounts and to create new accounts.
- **Tasks** — The tab is designed for working with Garbage collection and Additional extraction tasks.
- **Info** — The tab contains information about the user interface and the LUNA PLATFORM license.

6.12.1.1 Accounts tab

This tab displays all existing accounts.

Login	Account type	Account ID	Description
root@visionlabs.ai	Admin	00000000-0000-4000-b000-000000000146	
advanced_user@visionlabs.ai	Advanced user	29400d4d-e731-4dc2-81cb-3327dba0753e	
user@visionlabs.ai	User	291873d1-8fd9-4eda-8f59-197cdc6d9f7d	Simple user

Figure 36: Accounts tab

You can manage existing accounts using the following buttons:

 – View account information.

 – Delete account.

Clicking the view info button opens a page containing general information about the account, lists created with that account, and faces.

When you click the “Create account” button, an account creation window opens, containing the standard account creation settings — login, password, account type, description and the desired “account_id”.

See [“Account”](#) for details on accounts and their types.

6.12.1.2 Tasks tab

This tab displays running/completed Garbage collection and Additional extraction tasks.

Task ID	Type	Created	Updated	Finished	Parts done	Parts total	Description	Status
6	Garbage collection	03.02.2023, 14:28:24	03.02.2023, 14:28:24		0	1		⚙️
5	Additional extraction	03.02.2023, 14:28:09	03.02.2023, 14:28:09	03.02.2023, 14:28:09	1	1		✅
4	Additional extraction	03.02.2023, 14:21:27	03.02.2023, 14:21:28	03.02.2023, 14:21:28	1	1		✅
3	Additional extraction	01.02.2023, 18:12:14	01.02.2023, 18:12:14	01.02.2023, 18:12:14	1	1	clustering usefull faces	✅
2	Additional extraction	01.02.2023, 16:53:35	01.02.2023, 16:53:35	01.02.2023, 16:53:35	1	1		✅
1	Garbage collection	01.02.2023, 12:32:01	01.02.2023, 12:32:03	01.02.2023, 12:32:03	1	1		✅

Tasks are displayed in a table whose columns can be sorted and also filtered by the date the tasks were completed.

When you press the “Start Garbage collection” and “Start Additional extraction” buttons, windows for creating the corresponding tasks open.

The “Garbage collection” window contains the following settings, similar to the parameters of the [“garbage collecting task”](#) request body to the Tasks service:

- Description — “description” parameter
- Target — “content > target” parameter
- Account ID — “content > filters > account_id” parameter
- Remove sample — “content > remove_samples” parameter
- Remove image origins — “content > remove_image_origins” parameter
- Delete data before — “content > create_time__lt” parameter

See [“Garbage collection task”](#) for details.

The “Additional extraction” window contains the following settings, similar to the parameters of the [“additional extract task”](#) request body to the Tasks service:

- Objects type — “content > filters > object_type” parameter
- Extraction type — “content > extraction_target” parameter
- Descriptor version — “content > options > descriptor_version” parameter
- Description — “description” parameter
- Account ID — “content > filters > account_id” parameter

See [“Additional extraction task”](#) for details.

After creating a task, its execution begins. The progress of the task is displayed by the icon . The task is considered completed when the “Parts done” value matches the “Parts total” value and the icon changes to . If necessary, you can stop the task execution using the icon .

The following buttons are available for each task:

-  – download the task result as a JSON file.
-  – go to the page with a detailed description of the task and errors received during its execution.
-  – delete task.

Tasks are executed by the Tasks service after receiving a request from the Admin service.

6.12.1.3 Schedules tab

This tab is intended for working with [task scheduling](#).

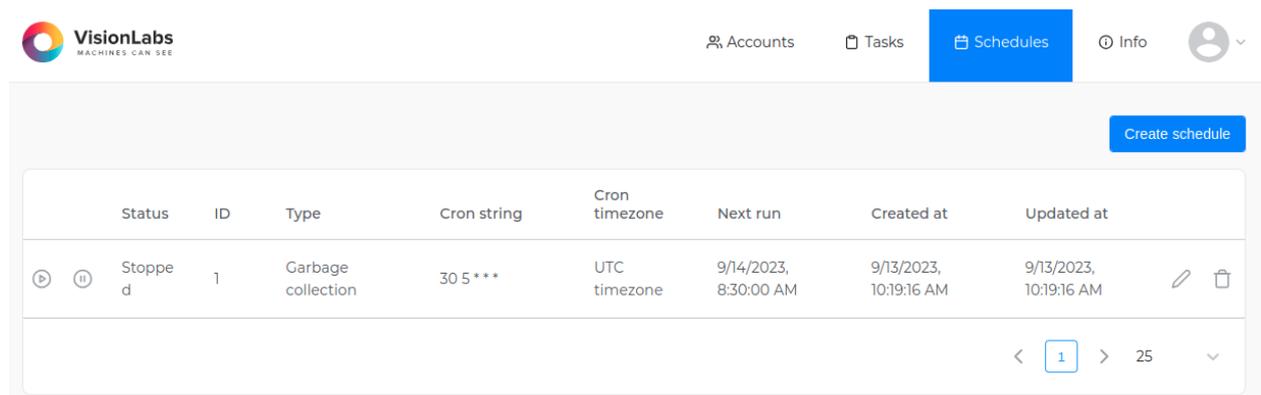


Figure 37: Schedules tab

The tab displays all created task schedules and all relevant information (status, ID, Cron string, etc.).

When you click on the “Create schedule” button, the schedule creation window opens.

Schedule

Task type

Garbage collection

Target

Events

Store results

Remove samples

Remove image origins

Account ID

example: a9709ed9-a836-47e2-befe-f3b588e7c0c3 (optional)

Delete data before

 02.10.2023 14:14

Cron string

0 0 ***

Cron timezone

Local timezone

Start immediately

Create Stopped

Create schedule

Figure 38: Schedule creation window

In the window you can specify schedule settings for the [Garbage collection](#) task. The parameters in this window correspond to the parameters of the “[create tasks schedule](#)” request.

After filling in the parameters and clicking the “Create schedule” button, the schedule will appear in the Schedules tab.

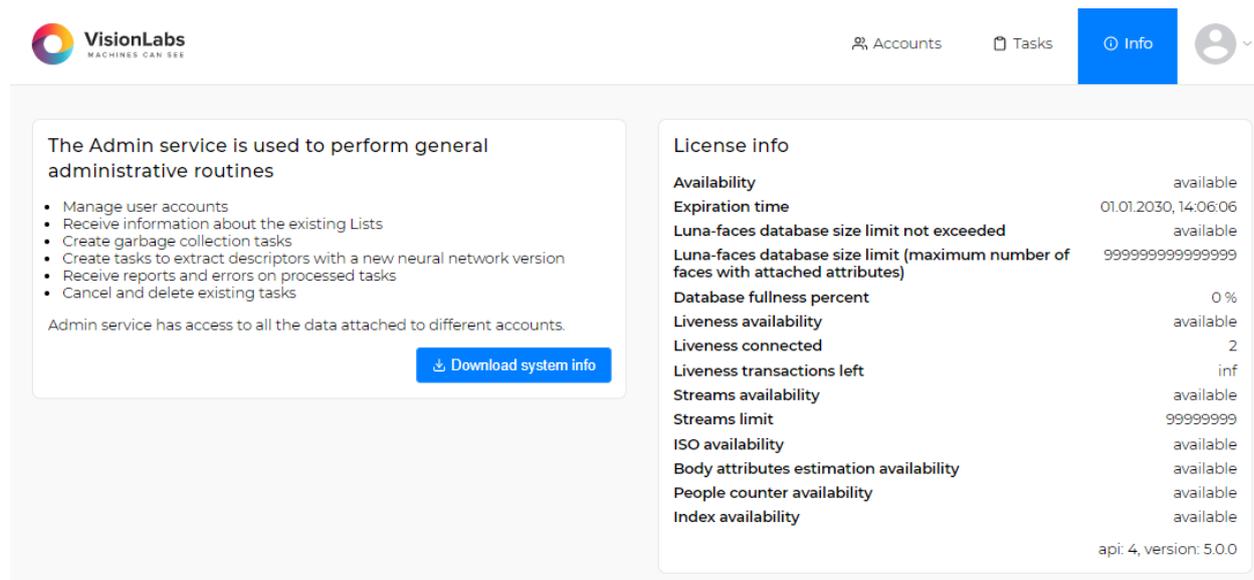
You can control delayed start using the following buttons:

-  – start the schedule.
-  – pause the schedule.

Using the  button, you can edit the schedule. Using the  button you can delete a schedule.

6.12.1.4 Info tab

This tab displays complete license information and features that can be performed using the Admin UI.



The Admin service is used to perform general administrative routines

- Manage user accounts
- Receive information about the existing Lists
- Create garbage collection tasks
- Create tasks to extract descriptors with a new neural network version
- Receive reports and errors on processed tasks
- Cancel and delete existing tasks

Admin service has access to all the data attached to different accounts.

[Download system info](#)

License info

Availability	available
Expiration time	01.01.2030, 14:06:06
Luna-faces database size limit not exceeded	available
Luna-faces database size limit (maximum number of faces with attached attributes)	9999999999999999
Database fullness percent	0 %
Liveness availability	available
Liveness connected	2
Liveness transactions left	inf
Streams availability	available
Streams limit	99999999
ISO availability	available
Body attributes estimation availability	available
People counter availability	available
Index availability	available

api: 4, version: 5.0.0

See the detailed license description in the [“License information”](#) section.

By clicking on the “Download system info” button, you can also get the following technical information about the LP:

- LUNA PLATFORM version
- Versions of the services
- Number of descriptors
- Configuration files values
- License information
- Requests and estimations statistics (see section [“Requests and estimations statistics gathering”](#))

You can also get the above system information using the [“get system info”](#) request to the Admin service.

6.13 Configurator service

The Configurator service simplifies the configuration of LP services.

The service stores all the required configurations for all the LP services in a single place. You can edit configurations through the user interface or special limitation files.

You can also store configurations for any third-party party software in Configurator.

The general workflow is as follows:

- User edits configurations in the UI.
- Configurator stores all changed configurations and other data in the database.
- LP services request Configurator service during startup and receive all required configurations. All the services should be configured to use the Configurator service.

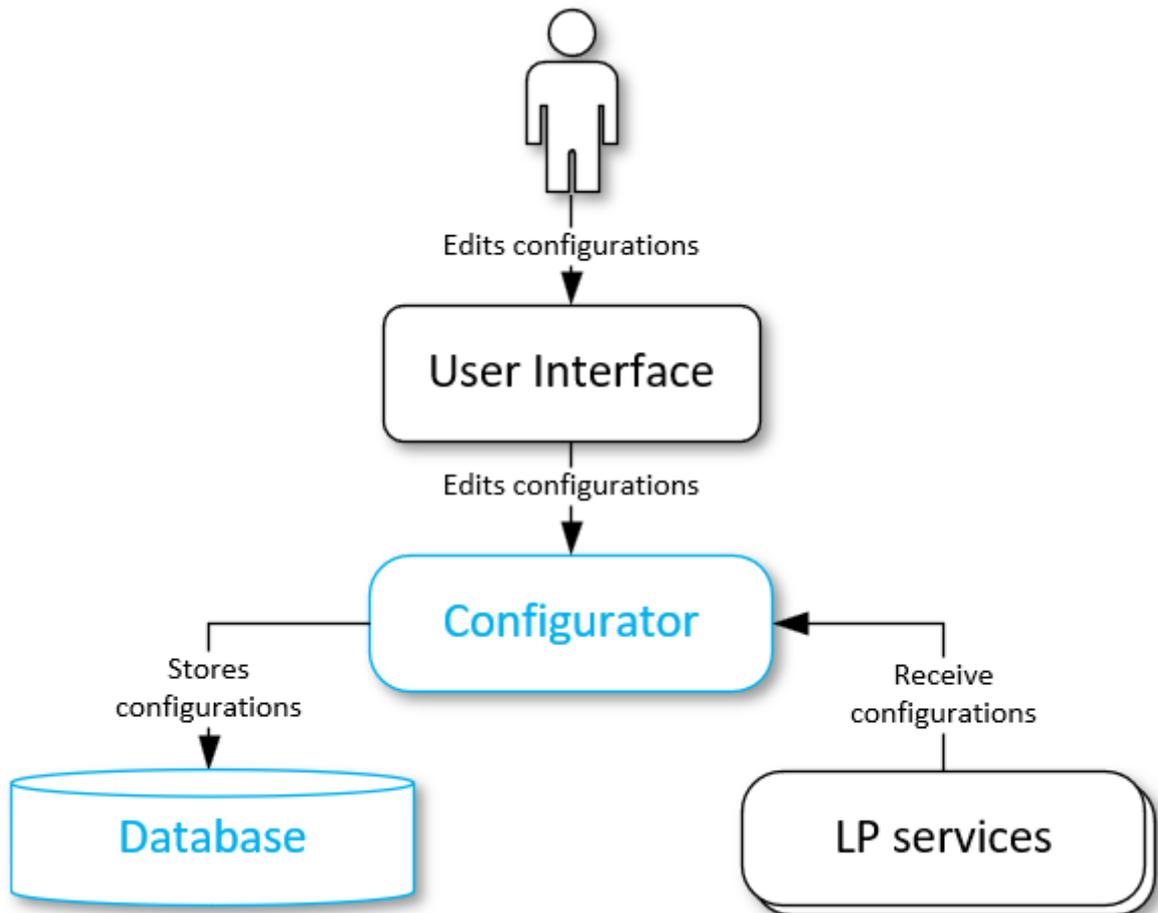


Figure 39: Configurator workflow

During Configurator installation, you can also use your limitation file with all the required fields to create limitations and fill in the Configurator database. You can find more details about this process in the [Configurator development manual](#) documentation.

Settings used by several services are updated for each of the services. For example, if you edit the “LUNA_FACES_ADDRESS” section for the Handlers service in the Configurator user interface, the setting will be also updated for API, Admin and Python Matcher services.

6.13.1 Configurator UI

Open the Configurator interface in your browser: <Configurator_server_address> :5070

This URL may differ. In this example, the Configurator service interface is opened on the Configurator service server.

LP includes the beta version of the Configurator UI. The UI was tested on Chrome and Yandex browser. The recommended screen resolution for working with the UI is 1920 x 1080.

The following tabs are available in the UI of Configurator:

- **Settings.** All the data in the Configurator service is stored on the **Settings** tab. The tab displays all the existing settings. It also allows to manage and filter them;
- **Limitations.** The tab is used to create new limitations for settings. The limitations are templates for JSON files that contain available data type and other rules for the definition of the parameters;
- **Groups.** The tab allows to group all the required settings. When you select a group on the Settings tab, only the settings corresponding to the group will be displayed. It is possible to get settings by filters and/or tags for a single specific service. For this purpose, the **Groups** tab is used.
- **About.** The tab includes information about the Configurator service interface.

6.13.1.1 Settings

Each of the Configurator settings contain the following fields:

- “Name” — Name for the setting.
- “Description” — Setting description.
- “ID and Times” — Unique setting ID.
- “Create time” — Setting create time.
- “Last update time” — Setting last update time.
- “Value” — Body of the setting.
- “Schema” — Verification template for the schema body.
- “Tag” — Tags for the setting used to filter settings for the services.

Name	Value	Schema
DETECTOR_QUEUE	<pre>{ "routing_key": "detector", "rabbit_exchange": "luna.detect" }</pre>	<pre>{ "type": "object", "properties": { "routing_key": { "type": "string" }, "rabbit_exchange": { "type": "string" } }, "required": ["routing_key"], "additionalProperties": false }</pre>
Description default Detector queue settings.		
Id and Times <input checked="" type="checkbox"/>		
63		
Create time Mon Sep 02 2019 14:49:39 GMT		
Last update time Mon Sep 02 2019 14:49:39 GMT		

Figure 40: Configurator interface

The “Tags” field is not available for the default settings. You should press the **Duplicate** button and create a new setting on the basis of the existing one.

The following options for the settings are available:

- Create a new setting — press the **Create new** button, enter required values and press **Create**. You should also select an already existing limitation for the setting. The Configurator will try to check the value of a setting if the **Check on save** flag is enabled and there is a limitation selected for the setting;
- Duplicate existing setting — press the **Duplicate** button on the right side of the setting, change required values and press **Create**. The Configurator will try to check the setting value if the **Check on save** flag is enabled on the lower left side of the screen and there is such a possibility;

Create new setting

Limitation

LUNA_TASKS_LOGGER
▼

Description (str <= 128 chars)

Max Luna Tasks logger settings

Value (depends on schema)

{"log_level":"INFO","folder_with_logs":"./","max_log_file_size":1024,"log_time":"LOCAL"}

Tags (str, separate by ',')

Max_Val

Cancel
Create

Figure 41: Duplicate setting window

- Delete existing setting — press the **Delete** button on the right side of the setting.
- Update existing setting — change name, description, tags, value and press **Save** button on the right side of the setting.
- Filter existing settings by name, description, tags, service names, groups — use the filters on the left side of the screen and press **Enter** or click on **Search** button;

Show limitations. The flags are used to enable displaying of limitations for each of the settings.

JSON editors. The flag enables you to switch the mode of the value field representation. If the flag is disabled, the name of the parameter and a field for its value are displayed. If the flag is enabled, the Value field is displayed as a JSON.

The **Filters** section on the left side of the window enables you to display all the required settings according to the specified values. You may enter the required name manually or select it from the list:

- **Setting.** The filter enables you to display the setting with the specified name.
- **Description.** The filter enables you to display all settings with the specified description or part of description.
- **Tags.** The filter enables you to display all settings with the specified tag.
- **Service filter.** The filter enables you to display all settings that belong to the selected service.
- **Group.** The filter enables you to display all settings that belong to the specified group. For example, you can select to display all the services belonging to LP.

6.13.1.2 Use tagged settings

Using tagged settings, you can run several identical services that will use different settings from the Configurator.

To do this, follow these steps:

1. Duplicate or create a new setting by specifying a tag for it. For example, you can duplicate the “LUNA_EVENTS_DB” setting and assign it the “EVENTS_DB_TAG” tag.
2. Pass the following arguments to the command “run.py” the corresponding container:
 - `--luna-config` — flag containing the address of the Configurator service
 - `--<configuration_name>` — flag containing the configuration and tag

See the “Service arguments” section of the installation guide for more information on the arguments.

For example, to configure “LUNA_EVENTS_DB” with the tag “EVENTS_DB_TAG”, the container launch command will look like this:

```
docker run \  
...  
dockerhub.visionlabs.ru/luna/luna-events:v.4.12.9  
python3 /srv/luna_events/run.py --luna-config http://127.0.0.1:5070/1 --  
LUNA_EVENTS_DB EVENTS_DB_TAG
```

6.13.1.3 Limitations

Limitations are used as service settings validation schema.

Settings and limitations have the same names. A new setting is created upon limitation creation.

The limitations are set by default for each of the LP services. You cannot change them.

Each of the limitations includes the following fields:

- **Name** is the name of the limitation.
- **Description** is the description of the limitation.
- **Service list** is the list of services that can use settings of this limitation.
- **Schema** is the object with JSON schema to validate settings
- **Default value** is the default value created with the limitation.

The following actions are available for managing limitations:

- Create a new limitation — Press the **Create new** button, enter required values and press “Create”. Also, the setting with default value will be created.
- Duplicate existing limitation — Press the **Duplicate** button on the right side of the limitation, change required values and press **Create**. Also, the setting with default value will be created.

- Update limitation values — Change name/description/service list/validation schema/default values and press the **Save** button on the right side of the limitation.
- Filter existing limitations by names, descriptions, and groups.
- Delete existing limitation — Press the **Delete** button on the right side of the limitation.

6.13.1.4 Groups

Group has a name and a description.

It is possible to:

- Create a new group — Press the **Create new** button, enter the group name and optionally description and press **Create**.
- Filter existing groups by group names and/or limitation names — Use filters on the left side and press “RETURN” or click on **Search** button.
- Update group description — Update the existing description and press the **Save** button on the right side of the group.
- Update linked limitation list — to unlink limitation, press “-” button on the right side of the limitation name, to link limitation, enter its name in the field at the bottom of the limitation list and press the “+” button. To accept changes, press the **Save** button.
- Delete group — Press the **Delete** button on the right side of the group.

6.13.2 Settings dump

The dump file includes all the settings of all the LP services.

6.13.2.1 Receive settings dump

You can fetch the existing service settings from the Configurator by creating a dump file. This may be useful for saving the current service settings.

To receive a dump file use the following options:

- wget: `wget -O settings_dump.json 127.0.0.1:5070/1/dump`
- curl: `curl 127.0.0.1:5070/1/dump > settings_dump.json`
- text editor

The current values, specified in the Configurator service, are received.

6.13.2.2 Apply settings dump

To apply the dumped settings use the `db_create.py` script with the `--dump-file` command line argument (followed with the created dump file name): `base_scripts/db_create.py --dump-file settings_dump.json`.

You can apply full settings dump on an empty database only.

If the settings update is required, you should delete the whole “limitations” group from the dump file before applying it.

```
"limitations": [  
    ...  
],
```

Follow these steps to apply the dump file:

1. Enter the Configurator container.
2. Run `python3 base_scripts/db_create.py --dump-file settings_dump.json`.

Limitations from the existing limitations files are replaced with limitations from the dump file, if limitations names are the same.

6.13.3 Limitations file

6.13.3.1 Receive limitation file

Limitation file includes limitations of the specified service. It does not include existing settings and their values.

To download a limitations file for one or more services, use the following commands:

1. Enter the Configurator container.
2. Create the output “base_scripts/results” directory: `mkdir base_scripts/results`
3. Run the “base_scripts/get_limitation.py” script: `python3 base_scripts/get_limitation.py --service luna-image-store luna-handlers --output base_scripts/results/my_limitations.json`.

Note the “base_scripts/get_limitation.py” script parameters:

- `--service` for specifying one or more service names (required)
- `--output` for specifying the directory or a file where to save the output. The default value: “current_dir/{timestamp}_limitation.json” (optional)

6.14 Licenses service

6.14.1 General information

The Licenses service stores information about the available licensed features and their limits.

There are three ways to get license information:

- Using the “[get license](#)” request to the Licenses service.
- Using the “[get system info](#)” request to the Admin service.
- Using the user interface of the “[Admin service](#)”.

You can also use the “[get platform features](#)” request to the API service, in the response to which you can get information about the license status, the license functions enabled (“[face_quality](#)”, “[body_attributes](#)” and “[liveness](#)”) and the status of optional services (Image Store, Events, Tasks and Sender) from the “[ADDITIONAL_SERVICES_USAGE](#)” configuration of the Configurator service.

If you disable some license feature and try to use a request that requires this function, error 33002 will be returned with the description “License problem Failed to get value of License feature {value}”.

6.14.2 License information

LP license includes the following features:

- License expiration date.
- Maximum number of faces with linked descriptors or basic attributes.
- OneShotLiveness estimation availability.
- OneShotLiveness current balance.
- Deepfake estimation availability.
- Image check according to ISO/IEC 19794-5:2011 standard availability.
- Body parameters estimation availability.
- People count estimation availability.
- Using Lambda service availability.
- Possibility of using the Index Matcher service in the LUNA Index Module.
- Maximum number of streams created by the LUNA Streams service.

When ordering the license, you need to inform technical support about the need to use any of the above features.

The features “Possibility of using the Index Matcher service in the LUNA Index Module” and “Maximum number of streams created by the LUNA Streams service” are described in the LUNA Index Module and FaceStream documentation, respectively.

Notifications are available for some features when approaching the limit. Notifications work in the form of sending messages to the logs of the corresponding service. For example, when approaching the allowable number of created faces with descriptors, the following message will be displayed in the

Faces service logs: “License limit exceeded: 8% of the available license limit is used. Please contact VisionLabs for license upgrade or delete redundant faces”. Notifications work due to constant monitoring implemented using the Influx database. Monitoring data is stored in the corresponding fields of the Influx database.

See the detailed information in the section [“Monitoring”](#).

6.14.2.1 Expiration date

When the license expires, you cannot use LUNA PLATFORM.

By default, the notification about the end of the license is sent two weeks before the expiration date.

When the license ends, the following message is returned “License has expired. Please contact VisionLabs for a license extension.”.

The Licenses service writes data about the license expiration date to the logs and the Influx database in the “license_period_rest” field.

6.14.2.2 Faces limit

The Faces service checks the number of faces left according to the maximum available number of faces received from the Licenses service. The faces with linked descriptors or basic attributes are counted only.

The percentage of the used limit for faces is written in the Faces log and displayed in the Admin GUI.

The Faces service writes data about the created faces to the logs and the Influx database in the “license_faces_limit_rate” field.

The created faces are written in the Faces log and displayed in the Admin GUI as a percentage of the database fullness. You should calculate the number of faces with descriptors left using the current percentage.

You start receiving notifications when there are 15% of available faces left. When you exceed the number of available faces, the message “License limit exceeded. Please contact VisionLabs for license upgrade or delete redundant faces” appears in logs. You cannot attach attributes to faces if the number of faces exceeds 110%.

Consequences of missing a feature

If this feature is disabled, it will be impossible to perform the following requests:

- “[create face](#)”
- “[generate events](#)” specifying a handler with “policies” > “storage_policy” > “face_policy” > “store_face”

6.14.2.3 OneShotLiveness

An unlimited license or a license with a limited number of transactions is available to estimate Liveness using the OneShotLiveness estimator.

Each use of Liveness in requests reduces the transaction count. It is impossible to use the Liveness score in requests after the transaction limit is exhausted. Requests that do not use Liveness and requests where the Liveness estimation is disabled are not affected by the exhaustion of the limit. They continue to work as usual.

The Licenses service stores information about the liveness transactions left. The number of transactions left is returned in the response from the “/license” resource.

The Remote SDK service writes data on the number of available Liveness transactions to the logs and the Influx database in the “liveness_balance” field.

A warning about the exhaustion of the number of available transactions is sent to the monitoring and logs of the Remote SDK service when the remaining 2000 transactions of Liveness are reached (this threshold is set in the system).

See the “[OneShotLiveness description](#)” section for more information on how Liveness works.

Consequences of missing a feature

If this feature is disabled, it will be impossible to estimate Liveness (the “estimate_liveness” parameter) in the following requests:

- “sdk”
- “detect faces”
- “generate events” specifying a handler with “policies” > “detect_policy” > “estimate_liveness”
- “perform verification” specifying a handler with “policies” > “detect_policy” > “estimate_liveness”
- “estimator task” specifying a handler with “policies” > “detect_policy” > “estimate_liveness”
- “predict liveness”

6.14.2.4 Body parameters estimation

This feature enables you to estimate [body parameters](#). Two values can be set in the license — 0 or 1. Monitoring is not intended for this parameter.

Consequences of missing a feature

If this feature is disabled, it will be impossible to estimate the body parameters (parameters “estimate_upper_body”, “estimate_lower_body”, “estimate_body_basic_attributes”, “estimate_accessories”) in the following requests:

- “sdk”
- “generate events” specifying a handler with “policies” > “detect_policy” > “body_attributes”

6.14.2.5 People count estimation

This feature enables you to estimate [the number of people](#). Two values can be set in the license — 0 or 1. Monitoring is not intended for this parameter.

Consequences of missing a feature

If this feature is disabled, it will be impossible to estimate the number of people (the “estimate_people_count” parameter) in the following requests:

- “sdk”
- “generate events” specifying a handler with “policies” > “detect_policy” > “estimate_people_count”

6.14.2.6 Image check by ISO/IEC 19794-5:2011 standard

This feature enables you to perform various [image checks by ISO/IEC 19794-5:2011 standard](#). Two values can be set in the license — 0 or 1. Monitoring is not intended for this parameter.

Consequences of missing a feature

If this feature is disabled, it will be impossible to perform the following requests:

- “iso”
- “detect faces” with the parameter “estimate_face_quality”
- “generate events” specifying a handler with “policies” > “detect_policy” > “face_quality”
- “estimator task” specifying a handler with “policies” > “detect_policy” > “face_quality”
- “perform verification” specifying a handler with “policies” > “detect_policy” > “face_quality”

6.15 Lambda service

The Lambda service is intended to work with user modules that mimic the functionality of a separate service. The service enables you to write and use your own handler or write an external service that will closely interact with the LUNA PLATFORM and immediately have several functions typical of LP services (such as [logging](#), [automatic configuration reload](#), etc.).

The Lambda service creates a Docker image and then runs it in a Kubernetes cluster. It is impossible to manage a custom module without Kubernetes. **Full-fledged work with the Lambda service is possible when deploying LUNA PLATFORM services in Kubernetes.** To use it, you must independently deploy LUNA PLATFORM services in Kubernetes or consult VisionLabs specialists. If necessary, you can use Minikube for local development and testing, thus providing a Kubernetes-like environment without the need to manage a full production Kubernetes cluster.

This functionality should not be confused with the [plugin mechanism](#). Plugins are designed to implement narrow targeted functionality, while the Lambda service enables you to implement the functionality of full-fledged services.

It is strongly recommended to learn as much as possible about the objects and mechanisms of the LUNA PLATFORM (especially about [handlers](#)) before starting to work with this service.

A custom module running in a Kubernetes cluster is called **lambda**. Information about the created lambda is stored in the [Lambda database](#).

The number of lambda created is unlimited. Each lambda has the option to add its own OpenAPI specification.

To work with the Lambda service, you need **a special license feature**. If the feature is not available, the corresponding error will be returned when requesting the creation of lambda.

Note: The description given below is intended for general acquaintance with the functionality of the Lambda service. See [developer manual](#) for more details. The “Quick start guide” section is available in the developer manual, which enables you to start working with the service.

6.15.1 Before start

Before you start working with the Lambda service, you need to familiarize yourself with all the requirements and set the service settings correctly.

6.15.1.1 Code and archive requirements

The module is written in Python and must be transferred to the Lambda service in a ZIP archive.

The code and archive must meet certain requirements, the main of which are listed below:

- Python version 3.11 or higher must be used.
- Development requires the “luna-lambda-tools” library, available in VisionLabs PyPI.
- Archive should not be password-protected.

Also, the files in the archive must have a certain structure. See the detailed information in the section “[Requirements](#)” of the developer manual.

6.15.1.2 Environment requirements

To work with the Lambda service, the following environment requirements are required:

- Availability of running Licenses and Configurator services*.
- Availability of S3 bucket for storing archives.
- Availability of Docker registry for storing images.
- Availability of Kubernetes cluster.

* during its operation, lambda will additionally interact with some LUNA PLATFORM services. The list of services depends on the lambda type (see “[Lambda types](#)”).

Write/read access to the S3 storage bucket must be provided and certain access rights in the Kubernetes cluster must be configured. You need to transfer the basic Lambda images to your Docker registry. The commands for transferring images are given in the LUNA PLATFORM installation manual.

When the Lambda service starts, the Docker registry and base images are checked.

For more information, see the “[Requirements](#)” section of the developer manuals.

6.15.1.3 Lambda service configuration

In the Lambda service settings, you must specify the following data:

- Location of the Kubernetes cluster (see setting “[CLUSTER_LOCATION](#)”):
 - “internal” — Lambda service works in a Kubernetes cluster and does not require other additional settings.
 - “remote” — Lambda service works with a remote Kubernetes cluster and correctly defined settings “[CLUSTER_CREDENTIALS](#)” (host, token and certificate).
 - “local” — Lambda service works in the same place where the Kubernetes cluster is running.

In the classic version of working with the Lambda service, it is assumed to use the “internal” parameter.

- S3 bucket settings (see setting “[S3](#)”).
- Registry address for storing the Docker image (see setting “[LAMBDA_REGISTRY](#)”).
- Addresses of insecure registries that may need access during lambda creation (see setting “[LAMBDA_INSECURE_REGISTRIES](#)”).

For more information, see the “[Configuration requirements](#)” section of the developer manual.

6.15.1.4 Configuring lambda entities

A specific set of settings is available for running lambda entities. They can be set in the Configurator by filtering the settings by the name “luna-lambda-unit”.

The settings for the Lambda service and the settings for lambda entities are different.

The settings contain service addresses, connection timeouts, logging settings, etc., allowing lambda entities to effectively interact with LUNA PLATFORM services. See all available settings in the section “[Lambda configuration](#)”.

The settings apply to all lambda at the same time.

6.15.2 Lambda types

Lambda can be of three types:

- **Handlers-lambda**, intended to replace the functionality of the classic handler.
- **Standalone-lambda**, intended to implement independent functionality to perform close integration with the LUNA PLATFORM.
- **Tasks-lambda**, intended to implement additional custom long task types.

Each type has certain requirements for LUNA PLATFORM services, the actual settings of which will be automatically used to process requests. Before starting work, the user must decide which lambda he needs.

6.15.2.1 Handlers-lambda

Examples of possible functionality:

- Performing verification with the possibility of saving an event.
- Matching of two images without performing the rest of the functionality of the classic handler.
- Adding your own filtering logic to the matching functionality;
- Circumventing certain limitations of the LUNA PLATFORM (for example, specify the maximum number of candidates greater than 100).
- Embedding the neural network SDK bypassing the LUNA PLATFORM.

During its operation, Handlers-lambda will interact with the following LUNA PLATFORM services:

- Configurator — To get the settings.
- Faces — For working with faces and lists.
- Remote SDK — For performing detections, estimations and extractions.
- Events* — For working with events.
- Python Matcher/Python Matcher Proxy** — For classical/cross-matching of faces or bodies.
- Image Store* — For storing samples and source images of faces or bodies.

The Lambda service will not check the connection to the disabled services and will give an error if the user tries to make a request to the disabled service.

To run the Lambda service, only the presence of the Configurator and Licenses services is required.

* the service can be disabled in the [“ADDITIONAL_SERVICES_USAGE”](#) setting.

** the service is disabled by default. To enable the service, see the setting [“ADDITIONAL_SERVICES_USAGE”](#).

The Lambda-handler can be used in two cases:

- As a custom handler that has its own response scheme, which may differ from the response of classic handlers and cannot be properly used in other LUNA PLATFORM services.
- As a custom handler that mimics the response of a classic handler. There are some requirements for such a case:

- The response must match the response scheme of the [event generation](#) request.
- The handler must process incoming data correctly so that other services can use it, otherwise there is no guarantee of compatibility with other services. That is, if such a handler implies face recognition, the module should return information about face recognition in response, if the handler implies body detection, the module should return body detection in response, etc.

For example, if a Lambda handler satisfies the above conditions, then it can be used in [Estimator task](#) as a classic handler.

For more information and code examples for Handlers-lambda, see the “[Handlers lambda development](#)” section of the developer manual.

6.15.2.2 Standalone-lambda

Examples of possible functionality:

- Filtering incoming images by format for subsequent sending to the Remote SDK service.
- Creation of a service for sending notifications by analogy with the Sender service.
- Creation of a service for recording a video stream and saving it as a video file to the Image Store service for subsequent processing by the FaceStream application.

During its operation, Standalone-lambda will interact at least with the Configurator service, which enables lambda to receive its settings (for example, logging settings).

To run the Lambda service, only the presence of the Configurator and Licenses services is required.

For more information and sample code for Standalone-lambda, see the “[Standalone lambda development](#)” section of the developer manual.

6.15.2.3 Tasks-lambda

Examples of possible functionality:

- Unlink faces from lists if faces do not similar to the specified face.
- Remove duplicates from list.
- Recursively find events similar to the specified photo.

During its operation, Tasks-lambda will interact with the following LUNA PLATFORM services:

- Configurator
- Faces
- Python Matcher
- Remote SDK
- Tasks
- Events*

- Image Store*
- Handlers*

* the service can be disabled in the [“ADDITIONAL_SERVICES_USAGE”](#) setting

After [creating lambda](#), the [“lambda task”](#) requests must be executed to create a Lambda task. The results of the task/subtask can be obtained using standard [Tasks service](#) requests.

The Lambda task is created according to the [general task creation process](#), **except** that the Lambda service is used instead of the Tasks worker.

When writing code for Tasks-lambda, it is recommended to break each task into subtasks for the convenience of representing the process and parallelizing the execution of tasks.

You can create [schedule](#) for Tasks-lambda.

For more information and code examples for Tasks-lambda, see [“Tasks lambda development”](#) in the developer manual.

6.15.3 Create lambda

To create a lambda, you need to do the following:

1. Write Python code in accordance with [the type of future lambda](#) and [code requirements](#).
2. Move files to the archive in accordance with [archive requirements](#).
3. Perform the [“create lambda”](#) request, specifying the following mandatory data:
 - “archive” — The address to the archive with the user module.
 - “credentials” > “lambda_name” — The name for the lambda being created.
 - “parameters” > “lambda_type” — The type of lambda being created ([“handlers”](#), [“standalone”](#) or [“tasks”](#)).

Also optionally in the [“deploy_parameters”](#) section you can set the number of pods, allocate resources to pods, set namespaces, as well as [enable GPU use](#).

It is also possible to specify labels for running lambda using specific Kubernetes nodes using the [“deploy_parameters”](#) > [“selector”](#) parameter. This enables for more granular control over lambda deployment, allowing administrators to specify the nodes on which lambda should be deployed based on their resource needs. By applying labels to Kubernetes nodes and using the [“selector”](#) parameter, administrators can manage resource allocation not only for individual lambdas, but also for lambda sections with similar resource requirements.

If necessary, you can specify a list of additional Docker commands to create a lambda container. See the [“Lambda — Archive requirements”](#) section of the developer manual.

In response to a successful request, the [“lambda_id”](#) will be issued.

The creation of lambda consists of several stages, namely:

- Creating a Docker image:
 - Getting the provided ZIP archive.
 - Addition of the archive with the necessary files.
 - Saving the archive in S3 storage.
 - Publishing the image to the registry.
- Creating a service in the Kubernetes cluster.

During lambda creation, you can run the following requests:

- [“get lambda image creation status”](#) to get the Docker image creation status (“in_progress”, “error”, “completed”, “not_found”)
- [“get lambda image creation logs”](#) to get Docker image creation logs
- [“get lambda status”](#) to get the lambda creation status (“running”, “waiting”, “terminated”, “not_found”)

See the lambda creation sequence diagram in [“Lambda creation diagram”](#).

See the detailed description of the lambda creation process in the [“Creation pipeline”](#) section of the developer manual.

6.15.4 Create handler for Handlers-lambda

If lambda is supposed to be used as a custom handler simulating the response of a classic handler, then it is necessary to [create a handler](#), specifying “handler_type” = “2” and the resulting “lambda_id”.

During the creation of the handler, the Handlers service will perform a health check to the Kubernetes cluster.

The resulting “handler_id” can be used in requests [“generate events”](#) or [“estimator task”](#).

6.15.5 Use lambda

The table below shows resources for working with lambda, depending on its type.

Resource	Lambda type	Request and response body format
“/lambdas/{lambda_id}/proxy”	Standalone-lambda, Handlers-lambda with own response scheme	Own
“/handlers/{handler_id}/events”	Handlers-lambda	Corresponding OpenAPI specification

Resource	Lambda type	Request and response body format
"/tasks/estimator"	Handlers-lambda	Corresponding OpenAPI specification

See the lambda processing sequence diagram in [“Lambda processing diagram”](#).

Each lambda has its own API, the description of which is also available using the [“get lambda open api documentation”](#) request.

Each lambda response will contain multiple headers, including:

- “Luna-Request-Id” — Classic external ID of the LUNA PLATFORM request.
- “Lambda-Version” — Contains the current lambda version.

Useful requests when working with lambda:

- [“get lambda status”](#) to get lambda creation status (“running”, “waiting”, “terminated”, “not_found”)
- [“get lambda”](#) to get complete information about the created lambda (creation time, name, status, etc.)
- [“get lambda logs”](#) to get lambda creation logs

6.15.6 Create GPU-enabled lambda

It is possible to create a lambda that can leverage graphics processing unit (GPU) resources. To enable GPU usage, the “deploy_parameters” > “enable_gpu” parameter must be included in the lambda creation request.

Lambda supports only NVIDIA graphics processors. For additional information on GPU usage, refer to the [Kubernetes documentation](#).

If there is only one GPU available but more are required, you can enable shared GPU access (refer to the [official documentation](#)).

The Lambda service does not manage cluster resources, including GPU allocation. Resource management is handled by the Kubernetes administrator.

Additionally, when deploying LUNA PLATFORM to Kubernetes, it is recommended to configure the manifest in a specific way for all containers that use the GPU (including the Lambda service container) to avoid problems with device visibility. For detailed information, refer to the “Configuring manifest GPU support” section in the installation manual.

6.15.7 Update lambda

The base image for lambda containers is updated periodically. This image contains the necessary modules to interact with LUNA PLATFORM services. Applying updates requires that the lambda be recreated so that the container can be rebuilt based on the new image. After updating the base image and the “luna-lambda-tools” library, lambda functionality may be broken.

You can update the lambda using the request [“update lambda”](#) using the latest base image. It is recommended to have a backup copy of the archive on S3.

See the [“Lambda updates”](#) section of the developer manual for detailed information about the update mechanism, backup strategies, and restoring from a backup.

6.16 Backport 3

The Backport 3 service is used to process the requests for LUNA PLATFORM 3 using LUNA PLATFORM 5.

Although most of the requests are performed in the same way as in LUNA PLATFORM 3, there are still some restrictions. See [“Backport 3 features and restrictions”](#) for details.

See [Backport 3 OpenAPI specification](#) for details about the Backport 3 API.

6.16.1 Backport 3 new resources

6.16.1.1 Liveness estimation

Backport 3 provides Liveness estimation in addition to the LUNA PLATFORM 3 features. See the [“liveness > predict liveness”](#) section in the Backport 3 OpenAPI specification.

6.16.1.2 Handlers

The Backport 3 service provides several handlers: “extractor”, “identify”, “verify”. The handlers enable to perform several actions in a single request:

- [“handlers” > “face extractor”](#) — Enables you to extract a descriptor from an image, create a person with this descriptor, attach the person to the predefined list.
- [“handlers” > “identify face”](#) — Enables you to extract a descriptor from an image and match the descriptor with the predefined list of candidates.
- [“handlers” > “verify face”](#) — Enables you to extract a descriptor from an image and match the descriptor with the person’s descriptor.

The description of the handlers and all the parameters of the handlers can be found in the following sections:

- [“handlers” > “patch extractor handler”](#)
- [“handlers” > “patch verify handler”](#)
- [“handlers” > “patch identify handler”](#)

The requests are based on handlers and unlike the standard [“descriptors” > “extract descriptors”](#), [“matching” > “identification”](#), and [“matching” > verification](#) requests the listed above request are more flexible.

You can patch the already existing handlers thus applying additional estimation to the requests. E. g. you can specify head angles thresholds or enable/disable basic attributes estimation.

The Handlers are created for every new account at the moment the account is created. The created handlers include default parameters.

Each of the handlers has the corresponding handler in the Handlers service. The parameters of the handlers are stored in the luna_backport3 database.

Each handler supports *GET* and *PATCH* requests thus it is possible to get and update parameters of each handler.

Each handler has its version. The version is incremented with every *PATCH* request. If the current handler is removed, the version will be reset to 1:

- For the requests with *POST* and *GET* methods:

If the Handlers and/or Backport 3 service has no handler for the specified action, it will be created with default parameters.

- For requests with *PATCH* methods:

If Handlers and/or Backport 3 service has no handler for the specified action, a new handler with a mix of default policies and policies from the request will be created.

6.16.2 Backport 3 architecture

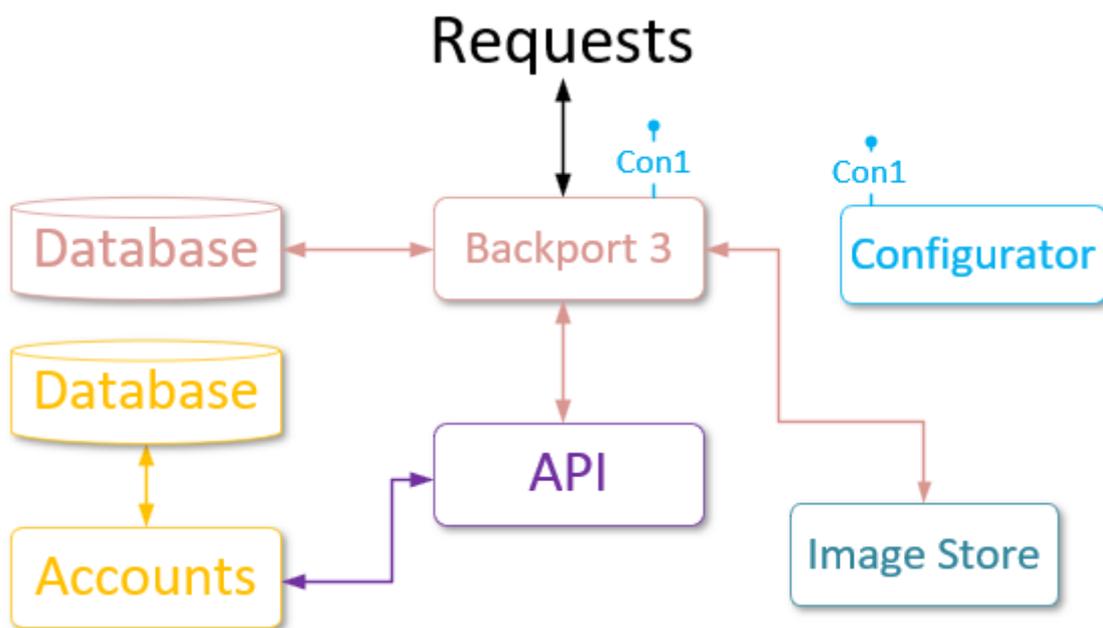


Figure 42: Interaction of Backport 3 and LP 5 services

Backport 3 interacts with the API service and sends requests to LUNA PLATFORM 5 using it. In turn, the API service interacts with the Accounts service to check the authentication data.

Backport 3 has its own database (see “[Backport 3 database](#)”). Some of its tables are similar to the tables of the Faces database of LP 3. It enables you to create and use the same entities (persons, account tokens and accounts) as in LP 3.

The backport service uses Image Store to store portraits.

You can configure Backport 3 using the Configurator service.

6.16.3 Backport 3 features and restrictions

The following features have core differences:

For the following resources on method POST default descriptor version to extract from image is 56:

- `/storage/descriptors`
- `/handlers/extractor`
- `/handlers/verify`
- `/handlers/identify`
- `/matching/search`

You can still upload the existing descriptors of versions 52, 54, 56. The older descriptor versions are no longer supported.

- For resource `/storage/descriptors` on method POST, estimation of “saturation” property is no longer supported, and the value is always set to 1.
- For resource `/storage/descriptors` on method POST, estimation of “eyeglasses” attribute is no longer supported. The *attributes* structure in the response will lack the “eyeglasses” member.
- For resource `/storage/descriptors` on method *POST*, head position angle thresholds can still be sent as float values in range [0, 180], but they will be internally rounded to integer values. As before, thresholds outside the range [0, 180] are not taken into account.

6.16.4 Garbage collection (GC) module

According to LUNA PLATFORM 3 logic, garbage is the descriptors that are linked neither to a person nor to a list.

For normal system operation, one needs to regularly delete garbage from databases. For this, run the system cleaning script `remove_not_linked_descriptors.py` from `./base_scripts/gc/` folder.

According to Backport 3 architecture, this script removes faces, which do not have links with any lists or persons from the Backport 3 database, from the Faces service.

6.16.4.1 Script execution pipeline

The script execution pipeline consists of several stages:

1. A temporary table is created in the Faces database. See more info about temporary tables for [oracle](#) or [postgres](#).

2. IDs of faces that are not linked to lists are obtained. The IDs are stored in the temporary table.
3. While the temporary table is not empty, the following operations are performed:
 - The batch of IDs from the temporary table is obtained. First 10k (or less) face ids are received.
 - Filtered IDs are obtained. Filtered IDs are ids that do not exist in the “person_face” table of the Backport 3 database.
 - Filtered IDs are removed from the Faces database. If some of the faces cannot be removed, the script stops.
 - Filtered IDs are removed from the Backport 3 database (foolcheck). A warning will be printed.
 - IDs are removed from the temporary table.

6.16.4.2 Script launching

```
docker run --rm -t --network=host --entrypoint bash dockerhub.visionlabs.ru/  
luna/luna-backport-3:v.0.11.9 -c "python3 ./base_scripts/gc/  
remove_not_linked_descriptors.py"
```

The output will include information about the number of removed faces and the number of persons with faces.

6.17 Backport 4

The Backport 4 service is used to process the requests for LUNA PLATFORM 4 using LUNA PLATFORM 5.

Although most of the requests are performed in the same way as in LUNA PLATFORM 4, there are still some restrictions. See [“Backport 4 features and restrictions”](#) for details.

See [Backport 4 OpenAPI specification](#) for details about the Backport 4 API.

6.17.1 Backport 4 architecture

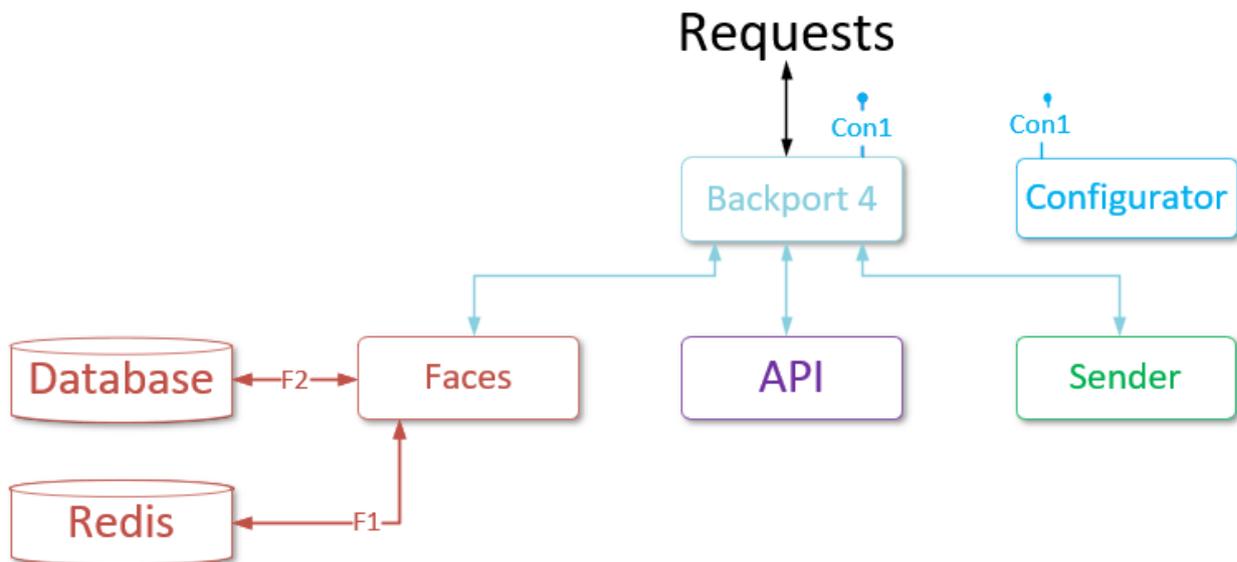


Figure 43: Interaction of Backport 4 and LP 5 services

Backport 4 interacts with the API service and sends requests to LUNA PLATFORM 5 using it.

Backport 4 directly interacts with the Faces service to receive the number of existing attributes.

Backport 4 directly interacts with the Sender service. All the requests to Sender are sent using the Backport 4 service. See the [“ws” > “ws handshake”](#) request in the [Backport 4 OpenAPI specification](#).

You can configure Backport 4 using the Configurator service.

6.17.2 Backport 4 features and restrictions

The following features have core differences:

The current versions for LUNA PLATFORM services are returned on the request to the `/version` resource. For example, the versions of the following services are returned:

- “luna-faces”

- “luna-events”
- “luna-image-store”
- “luna-python-matcher” or “luna-matcher-proxy”
- “luna-tasks”
- “luna-handlers”
- “luna-api”
- “LUNA PLATFORM”
- “luna-backport4” — Current service

Resources changelog:

- Resource `/attributes/count` is available without any query parameters and does not support accounting. The resource works with temporary attributes.
- Resource `/attributes` on method GET: “attribute_ids” query parameter is allowed instead of “page”, “page_size”, “time__lt” and “time__gte” query parameters. Thus you can get attributes by their IDs not by filters. The resource works with temporary attributes.
- Resource `/attributes/<attribute_id>` on methods GET, HEAD, DELETE and resource `/attributes/<attribute_id>/samples` on method GET interact with temporary attributes and return attribute data if the attribute TTL has not expired. Otherwise, the “Not found” error is returned.
- If you already used the attribute to create a face, use the “face_id” to receive the attribute data. In this case, the “attribute_id” from the request is equal to “face_id”.
- Resource `/faces` enables you to create more than one face with the same “attribute_id”.
- Resource `/faces/<face_id>` on method DELETE enables you to remove face without removing its attribute.
- Resource `/faces/<face_id>` on method PATCH enables you to patch attribute of the face making the first request to patch “event_id”, “external_id”, “user_data”, “avatar” (if required) and the second request to patch attribute (if required).
- If face attribute_id is to be changed, the service will try to patch it with temporary attribute data if the temporary attribute exists. Otherwise, the service tries to patch it with attribute data from the face with “face_id” = “attribute_id”.
- The match policy of resource `/handlers` now has the default match limit that is configured using the “MATCH_LIMIT” setting from the Backport 4 “config.py” file.
- Resource `/events/stats` on method POST: “attribute_id” usage in “filters” object was prohibited as this field is no longer stored in the database. The response with the 403 status code will be returned.
- The “attribute_id” in events is not null and is equal to “face_id” for back compatibility. GC task is

unavailable because all the attributes are temporary and will be removed automatically. Status code 400 is returned on a request to the `/tasks/gc` resource.

- The column “`attribute_id`” is not added to the report of the Reporter task and this column is ignored if specified in the request. Columns “`top_similar_face_id`”, “`top_similar_face_list`”, “`top_similar_face_similarity`” are replaced by the “`top_match`” column in the report if any of these columns is passed in the reporter task request.
- Linker task always creates new faces from events and ignores faces created during the event processing request.
- Resource `/matcher` does not check the presence of provided faces thus error “`FacesNotFound`” is never returned. If the user has specified a non-existent candidate of type “`faces`”, no error will be reported, and no actual matching against that face will be made.
- Resource `/matcher` checks whether reference with type attribute has the ID of face attribute or the ID of temporary attribute and performs type substitution. Hence it provides sending references for matching in the way it was done in the previous version.
- Resource `/matcher` takes matching limits into account. By default, the maximum number of references or candidates is limited to 30. If you need to overcome these limits, configure “`reference_limit`” and “`candidate_limit`”.
- Resource `/ws` has been added. There was no `/ws` resource in the LUNA PLATFORM 4 API as it was a separate resource of the Sender service. This added resource is similar to the Sender service resource, except that “`attribute_id`” of candidates faces is equal to “`face_id`”.
- Resource `/handlers` returns the error “`Invalid handler with id {handler_id}`”, if the handler was created in the LUNA PLATFORM 5 API and is not supported in LUNA Backport 4.

6.17.3 Backport 4 User Interface

The User Interface service is used for the visual representation of LP features. It does not include all the functionality available in LP. User Interface enables you to:

- Download photos and create faces using them.
- Create lists.
- Match existing faces.
- Show existing events.
- Show existing handlers.

All the information in User Interface is displayed according to the account data, specified in the configuration file of the User Interface service (`./luna-ui/browser/env.js`).

User Interface works with only one account at a time, which must be of “`user`” type.

You should open your browser and enter the User Interface address. The default value is: \<server_url \>:4200.

You can select a page on the left side of the window.

6.17.3.1 Lists/faces page

The starting page of User Interface is **Lists/Faces**. It includes all the faces and lists created using the account specified in the configuration.

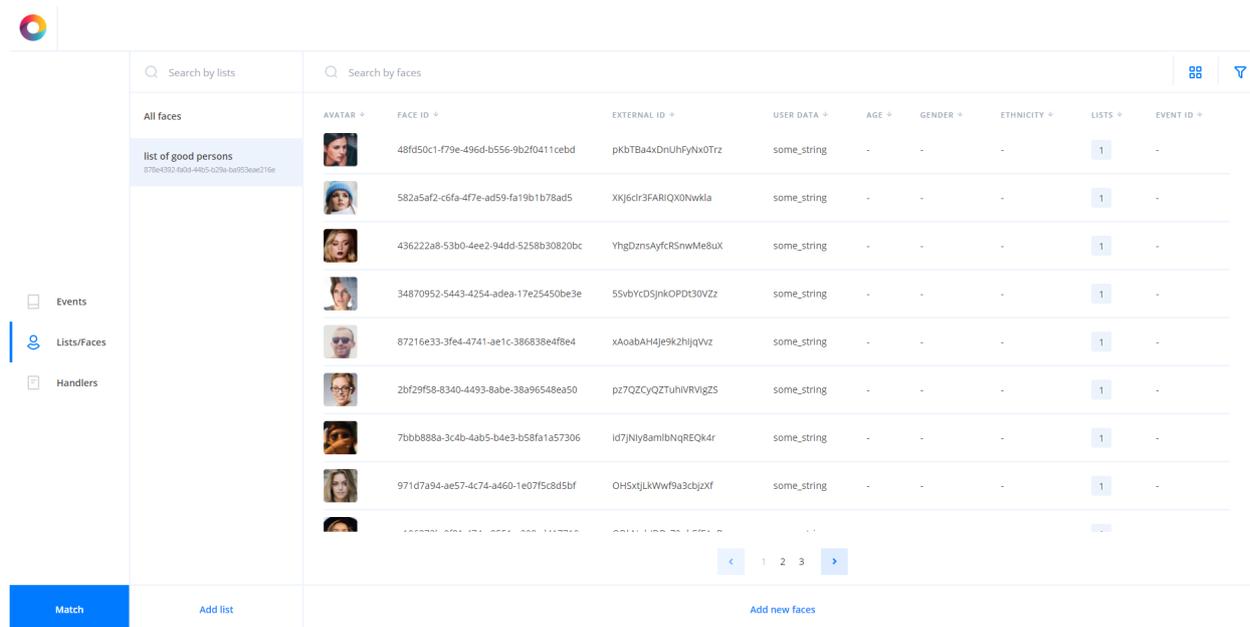


Figure 44: Lists/Faces Page

The left column of the workspace displays existing lists. You can create a new list by pressing the **Add list** button. In the appeared window you can specify the user data for the list.

The right column shows all the created faces with pagination.

Use the **Add new faces** button to create new faces.

On the first step, you should select photos to create faces from. You can select one or several images with one or several faces in them.

After you select images, all the found faces will be shown in a new dialog window.

All the correctly preprocessed images will be marked as “Done”. If the image does not correspond to any of the requirements, an error will be displayed for it.

Press the **Next step** button.

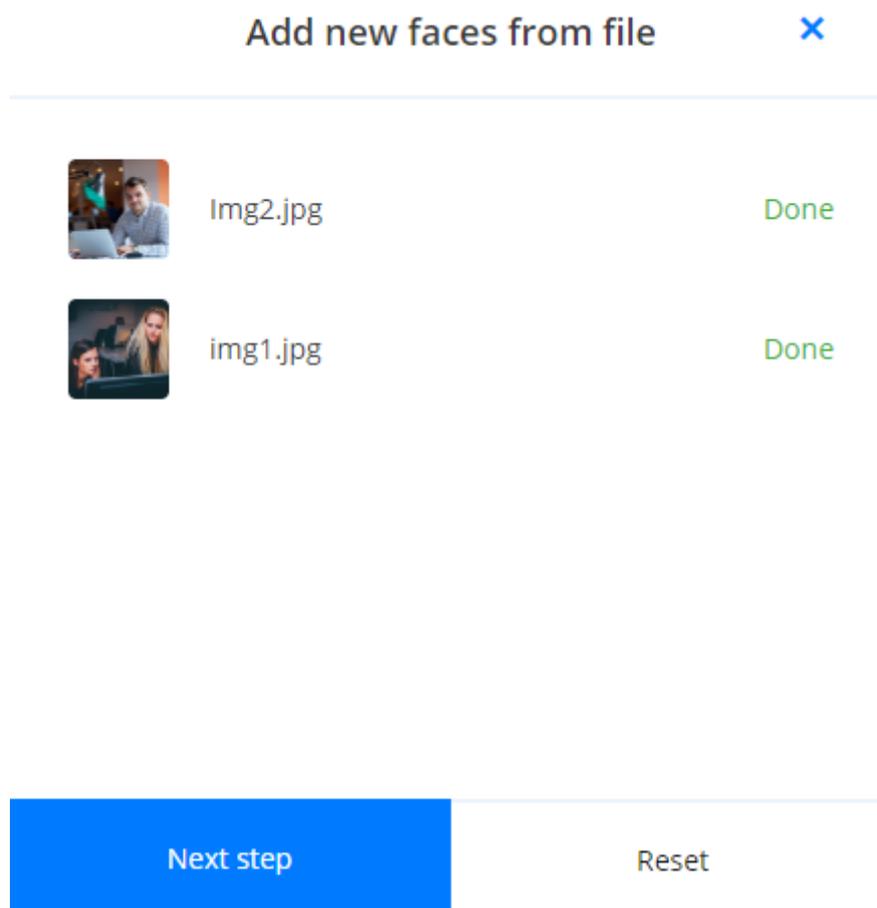


Figure 45: Select images

On the next step, you should select the attributes to extract for the faces.

Press the **Next step** button.

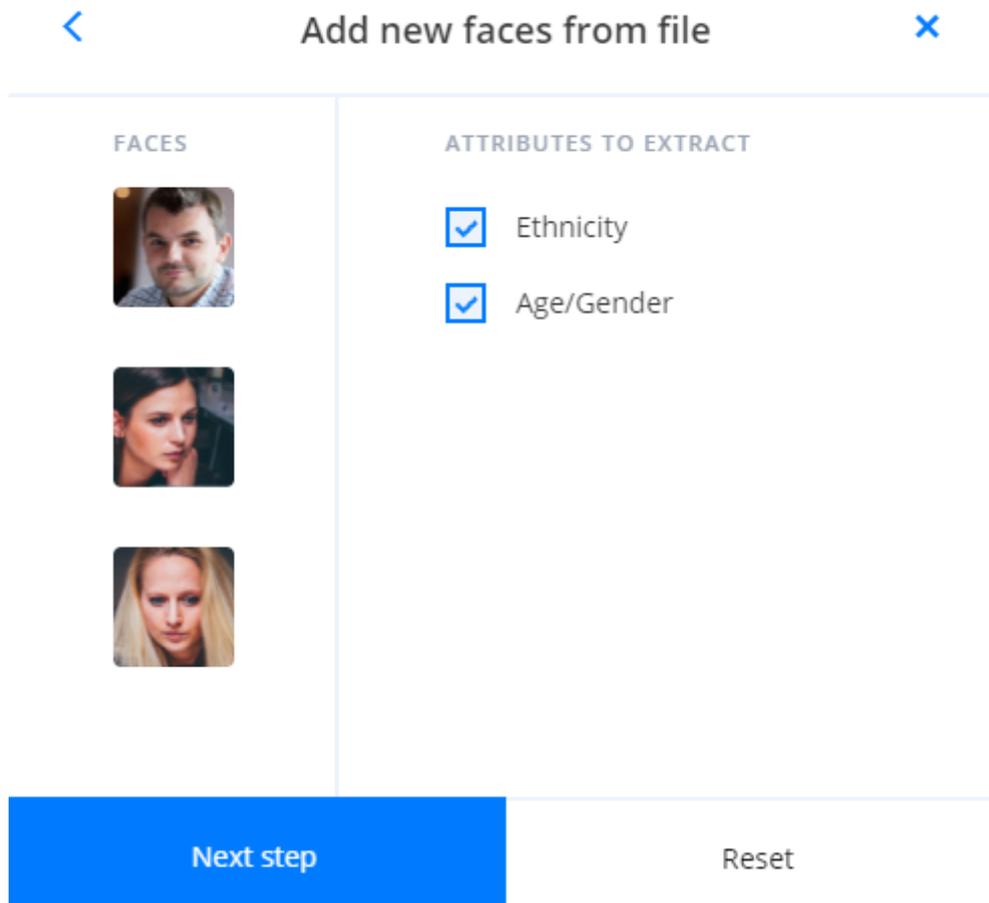


Figure 46: Select attributes

On the next step, you can specify user data and external ID for each of the faces. You can also select lists to which each of the faces will be added. Press **Add Faces** to create faces.

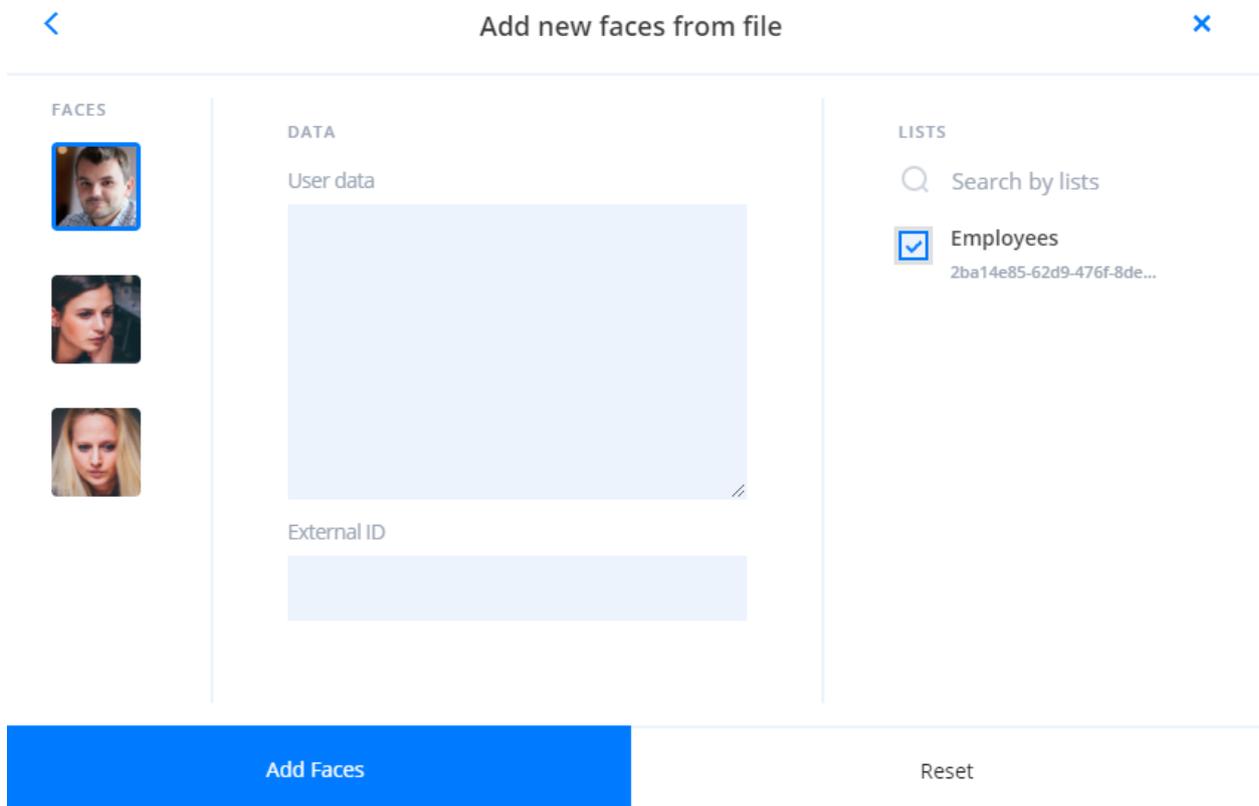


Figure 47: Add user data, external ID and specify lists

You can change pages using arrow buttons.



You can change the display of faces and filter them using buttons in the top right corner.

 Filter faces. You can filter faces by ID, external ID or list ID.

 /  Change view of the existing faces.

6.17.3.2 Handlers page

Handlers page displays all handlers created using the account specified in the configuration.

All the information about specified handler policies is displayed when you select a handler.

You can edit or delete a handler using edit  and delete  icons.

Search by handlers

test1
8144162b-77c7-4838-80df-41108803e300

test2
49f1923a-414d-44e5-93a6-4446f53c2b0c

Handler_1
6807474a-27d3-4356-80d-6228a1670b05

DETECT POLICY

detect_landmarks68	Yes
estimate_emotions	Yes
estimate_eyes_attributes	Yes
estimate_gaze	Yes
estimate_head_pose	No
estimate_mouth_attributes	Yes
estimate_quality	Yes
extract_exif	No
pitch_threshold	180
roll_threshold	180
yaw_threshold	180

EXTRACT POLICY

extract_basic_attributes	Yes
extract_descriptor	Yes
score_threshold	No

MATCH POLICY

list_id	053aa67a-f7eb-4136-ab88-46ec6c089f80
age_gte	30
age_lt	30
ethnicities	1
gender	1

Match
Add handler

Figure 48: Handlers page

6.17.3.3 Events page

The events page displays all the events created using the account specified in the configuration.

Search 🔍

AVATAR	FACE ID	EVENT ID	TIME	SOURCE	AGE	GENDER	HANDLER
	2cda9bd5-4b5f-414d-bb47-4447c48d553e	51054e68-30a3-47b9-a342-982cc107f7ca	03.19.20 16:24	Office_camera	23	Female	def83252-80ca-4473-992c-e914bec2083a
	79d695fb-b779-44e2-86da-d5d53f1fb9df	7f7d1ccb-deed-4f38-ae3f-5f9ae427d4c7	03.19.20 16:24	Office_camera	27	Female	def83252-80ca-4473-992c-e914bec2083a
	254e4a53-3a84-45ba-8fe7-cd742bb78acd	edfce94a-9144-4392-bace-740219b62ff9	03.19.20 16:24	Office_camera	26	Female	def83252-80ca-4473-992c-e914bec2083a
	ba180da8-bdb1-4362-adbd-8b787ce8f621	2223cdf9-0820-46fc-8793-eb962a0383f7	03.19.20 16:24	Office_camera	40	Male	def83252-80ca-4473-992c-e914bec2083a
	d442a48b-465b-4038-96c4-8c908175e396	51137d7f-9bc5-4cd1-a46a-39152f4b59d5	03.19.20 16:24	Office_camera	24	Female	def83252-80ca-4473-992c-e914bec2083a
	22ccb06-9b9a-4773-b11f-afda9db89ee6	5fccb4e8-e322-463c-8a4a-d40d66e45ffb	03.19.20 16:24	Office_camera	21	Male	def83252-80ca-4473-992c-e914bec2083a
	01ac2ab2-2320-4c1a-b4a0-46f2df1a6dba	4615e5c8-fcc2-4c29-8ea5-c8becc11eb44	03.19.20 16:24	Office_camera	29	Female	def83252-80ca-4473-992c-e914bec2083a
	81e35c7b-ce26-4a5b-9781-ae405428b5b9	51c1fd0b-652f-4f24-ab59-ad0b1e6d5591	03.19.20 16:24	Office_camera	27	Female	def83252-80ca-4473-992c-e914bec2083a
	84913a73-3e3a-4d6d-aeb0-9ee9a38e21c3	3a5ea6ca-71af-4286-84f2-2e7c5e9f8ae1	03.19.20 16:24	Office_camera	29	Male	def83252-80ca-4473-992c-e914bec2083a

Match

< 1 2 ... 8 >

Figure 49: Events Page

It also includes filters for displaying of events .

6.17.4 Common information

You can edit  or delete  an item (face, list or handler) using special icons. The icons appear when you hover the cursor on an item.

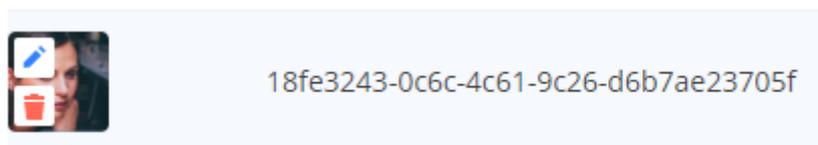


Figure 50: Icons for element

6.17.5 Matching dialog

The **Matching** button in the bottom left corner of the window enables you to perform matching.

After pressing the button you can select the number of the received results for each of the references.

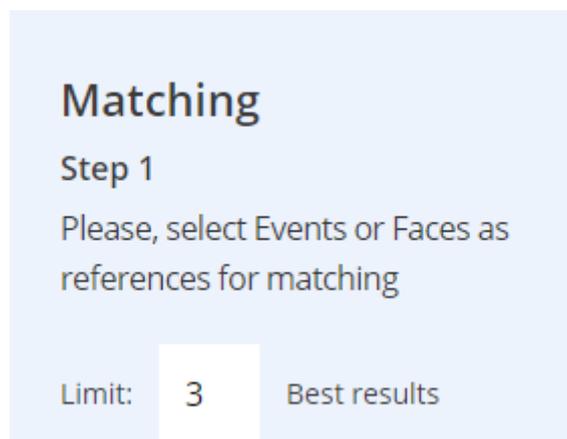


Figure 51: Select number of results

On the first step, you should select references for matching. You can select faces and/or events as references.

-  Events
-  Lists/Faces
-  Handlers

AVATAR ↓	FACE ID ↓	EVENT ID ↓
<input type="checkbox"/> 	167d3213-80a5-42e3-a097-204a18c95d97	ab83858b-2136-46e8-80c9-8e2f6b94d132
<input checked="" type="checkbox"/> 	879f9d91-9d7d-4940-a12a-58d93c3dabe3	6284ea46-f641-4f93-9c40-f77e06e7e444
<input type="checkbox"/> 	1335c09c-6abc-45d1-ae08-212f746d48d0	f4810fa5-2700-4905-9f85-ac5d268f6dfd
<input checked="" type="checkbox"/> 	1896f7ac-56d6-47d0-867c-7489560ca1ea	49fd6b5b-5569-475c-a954-238708646404
<input type="checkbox"/> 	46ac174d-5416-443b-89a8-863d2616bdb5	247cfa26-5e8e-4243-b8a1-573592eae844

Matching

Step 1

Please, select Events or Faces as references for matching

Proceed to candidates

References




Figure 52: Select references

On the second step, you should select candidates for matching. You can select faces or lists as candidates.

Search by lists

All faces

Employees
fac27c06-6db8-40f4-ae11-01594f06...

Events

Lists/Faces

Handlers

Search by faces

AVATAR	FACE ID	EXTERNAL ID	USER DATA	AGE
	78b4eca1-83de-4d66-a029-01ac264755de	-	-	27
	61d3229a-562b-4da7-87a1-e894bd92c119	-	-	20
	140a93dd-040c-4832-beba-2a2220b311ea	-	-	26
	e18717d8-800f-44fa-a9e3-c704af9427ea	-	-	29
	ee5fbaa7-cd92-408e-a5b4-5c654cc7f745	-	-	40
	80394d15-4768-47e8-879e-328ef6239eb7	-	-	44

< 1 2 ... 4 >

Matching

Step 2

Please, select Faces or Lists as candidates for matching

Proceed to matching

References

+

Candidates

Employees
fac27c06-6db8-40...
×

Figure 53: Select candidates

On the last step, you should press the **Start matching** button to receive results.

Matching

Step 3

Now you can start matching or edit references/candidates

Start matching

Figure 54: Start matching

6.18 Resource consumption by services

Below is a table describing the most significant resource consumption by services. The Remote SDK and Python Matcher services perform the most resource-intensive operations.

Service	Most consumed resources
Remote SDK	<p>CPU, RAM, GPU</p> <p>Remote SDK service performs mathematical transformation of images and extraction of descriptors. These operations require significant computing power. Both CPU and GPU can be used for calculations. Using GPU is preferable, because request processing is more efficient. However, not all types of video cards are supported.</p>
Python Matcher	<p>CPU, RAM</p> <p>Python Matcher performs a list matching. Matching requires CPU resources, but you should also allocate the maximum possible amount of RAM for each instance of Python Matcher. RAM is used to store descriptors obtained from the database. Thus, the Python Matcher service does not need to request each descriptor from the database.</p> <p>When distributing instances on multiple servers, the performance of each server should be taken into account. For example, if a large task is performed by several instances of Python Matcher, and one of them is on a server with low performance, the execution of the entire task as a whole may slow down.</p>
Postgres	SSD, CPU, RAM
Image Store	SSD, CPU, RAM
Handlers	CPU
Tasks	RAM
API	CPU
Faces	CPU
Events	CPU
Backport 3	CPU
Backport 4	CPU
Lambda	<p>RAM, CPU</p> <p>The amount of RAM depends on the size of the archive being transferred. See “Lambda service” section.</p>
Sender, Admin, Accounts, Licenses	In normal scenarios should not consume a lot of resources.

7 Additional Information

7.1 OneShotLiveness description

The Liveness technology enables to detect presentation attacks. A presentation attack is a situation when an imposter tries to use a video or photos of another person to circumvent the recognition system and gain access to the person's private data.

To estimate Liveness in the LUNA PLATFORM, the estimator LUNA SDK OneShotLiveness is used.

There are the following general types of presentation attacks:

- **Printed Photo Attack.** One or several photos of another person are used.
- **Video Replay Attack.** A video of another person is used.
- **Printed Mask Attack.** An imposter cuts out a face from a photo and covers his face with it.
- **3D Mask Attack.** An imposter puts on a 3D mask depicting the face of another person.

Liveness has the ability to license the number of performed transactions. You can also choose between an unlimited license and a license with a limited number of transactions (a maximum of 16 777 215 transactions can be specified in the key). After the transaction limit is exhausted, it will be impossible to use the Liveness estimation in requests. Requests that do not use Liveness or with Liveness estimation disabled are not affected by the exhaustion of the limit, they continue to work as usual.

7.1.1 Liveness check results

The Liveness algorithm uses a single image for processing and returns the following data:

- Liveness probability [0..1]. Here 1 means real person, 0 means spoof. The parameter shows the probability of a live person being present in the image, i.e. it is not a presentation attack. In general, the estimated probability must exceed the [Liveness threshold](#).
- Image quality [0..1]. Here 1 means good quality, 0 means bad quality. The parameter describes the integral value of image, facial, and environmental characteristics. In general, the estimated quality must exceed the [Quality threshold](#).
- Prediction. Based on the above data, LUNA PLATFORM issues the following prediction:
 - 0 (spoof). Check revealed that the person is not real.
 - 1 (real). Check revealed that the person is real.
 - 2 (unknown). Result of the check is unknown. Such a prediction may return if the quality of the image being checked is below the [Quality threshold](#), which determines the quality of the processed image.

7.1.2 Requests for estimating Liveness

Liveness is used in the following resources:

- [“/Liveness”](#)
- [“/sdk”](#)
- [“/handlers”](#)
- [“/verifiers”](#)

You can filter events by Liveness in the [“handlers/{handler_id}/events”](#) and [“/verifiers/{verifier_id}/verifications”](#) resources, i.e. you can exclude “spoof”, “real” or “unknown” results from image processing.

Filtering by Liveness is available for the following scenarios:

- When performing matching operations.
- When performing [tasks](#).
- When [sending data via web sockets](#).

You can also specify the Liveness estimation parameter when manually creating and saving events in the [“handlers/{handler_id}/events/raw”](#) resource.

For multiple uploaded images, you can aggregate the Liveness results to obtain more accurate data.

7.1.2.1 Liveness requirements

The requirements for the processed image and the face in the image are listed below.

Parameters	Requirements
Minimum resolution for mobile devices	720x960 pixels
Maximum resolution for mobile devices	1080x1920 pixels
Minimum resolution for webcams	1280x720 pixels
Maximum resolution for webcams	1920x1080 pixels
Compression	No
Image warping	No
Image cropping	No
Effects overlay	No
Mask	No
Number of faces in the frame	1

Parameters	Requirements
Face detection bounding box width	More than 200 pixels
Frame edges offset	More than 10 pixels
Head pose	-20 to +20 degrees for head pitch, yaw, and roll
Image quality	The face in the frame should not be overexposed, underexposed, or blurred.

See image quality thresholds in the [“Image quality”](#) section.

7.1.3 Liveness thresholds

By default, two thresholds are used for checking Liveness — **Quality threshold** and **Liveness threshold**.

7.1.3.1 Quality threshold

Quality threshold — the threshold of the processed image quality, lower which no check will be performed and the result **“unknown”** will be returned. This threshold is set in the **“quality_threshold”** setting from the **“LUNA_REMOTE_SDK_LIVENESS_ESTIMATOR_SETTINGS”** section of the Configurator service. The default threshold value is 0.5 (50%).

If the quality of the image being checked is lower than the Quality threshold value, then it is recommended to re-take a photo and send it to the system to perform the Liveness check. If this is not possible and you want to perform the Liveness check on the previously sent image, then you can set the Quality threshold to **“0”**, or interpret the **“unknown”** value as **“spoof”**.

When re-taking a photo, the average accuracy of the Liveness check increases. The table below shows the dependence of the FRN increase on the Quality threshold, where:

- FNR (False Negative Rate) — The percentage of real images rated as fake.
- Drop rate — The approximate percentage of images clipped by the Quality threshold (estimated based on a sample of 170,000 images).

Quality threshold	FNR	Drop rate
0	11%	0%
0.25	10%	0.5%
0.5	7%	1.4%
0.75	0.75%	2.3%

7.1.3.2 Liveness threshold

Liveness threshold — the threshold lower which the system will consider the result as a presentation attack (“spoof”). This threshold is set in the “[real_threshold](#)” setting from the “LUNA_REMOTE_SDK_LIVENESS_ESTIMATOR_SETTINGS” section of the Configurator service. The default threshold value is 0.5 (50%).

For images received from mobile devices, it is recommended to set a threshold **0.5** . For images obtained from a webcam, it is recommended to set a threshold **0.364** . At these threshold values, approximately equal results are obtained in terms of the accuracy of the algorithm.

If the quality of the image being checked is lower than the Quality threshold value, then setting the Liveness threshold will not make sense, since the result of the Liveness check will be unknown (“unknown”).

7.1.3.3 Changing thresholds on “/handlers” and “/verifiers” resources

In the “/handlers” and “/verifiers” there are two additional parameters:

- “quality_threshold” — Sets the [Quality threshold](#).
- “liveness_threshold” — Sets the [Liveness threshold](#).

Setting these parameters redefines the values of the “quality_threshold” and “real_threshold” settings from the “LUNA_REMOTE_SDK_LIVENESS_ESTIMATOR_SETTINGS” section of the Configurator service.

7.1.3.4 Changing thresholds on «/Liveness» and «/sdk» resources

There are no additional parameters for overriding thresholds for the “/Liveness” and “/sdk” resources. Thresholds are set in the “quality_threshold” and “real_threshold” settings from the “LUNA_REMOTE_SDK_LIVENESS_ESTIMATOR_SETTINGS” section of the Configurator service.

7.2 Video analytics

LUNA PLATFORM provides the ability to perform video analytics using the resource [“videosdk”](#).

Important: Currently, the resource is in beta testing status. Input and output schemes may be changed in future releases without backward compatibility support.

Video analytics is a set of functions that process frame by frame and assess valuable data.

To perform video analytics, an external link to a video file not exceeding 1 GB in size must be specified. If necessary, you can rotate the video before processing it at an angle of 90, 180 or 270 degrees (parameter “video” > “orientation” > “angle”).

During video analytics, a certain number of events are generated according to some rules, where each event has a start and end. Events are recorded in the response body and contain specific information about the particular video analytics. The response body also contains basic metadata about the video (number of frames, frame rate, video duration). The obtained information is only available in the response body and is not saved anywhere.

Important: An event in video analytics is in no way related to events generated by handlers.

In the [“LUNA_REMOTE_SDK_VIDEO_SETTINGS”](#) section of the Remote SDK service, you can configure video processing settings, such as the number of decoder working processes, processor type temporary directory for storing videos, etc.

7.2.1 People counting video analytics

The event of people counting video analytics starts when, on the last frame of the specified consecutive frames (parameter “probe_count”), the specified number of people (parameter “people_count_threshold”) appears. For example, if the minimum number of people is 10, and the number of consecutive frames is 3, the event will start if there are 10 people in 3 frames, starting from the 3rd frame. If there are 10 people in 2 frames, and 9 people in the third frame, the event will not start.

By default, every 10 frames are processed. If necessary, you can adjust the frame processing frequency using the “rate” parameter (for example, process every 3 seconds or every 3 frames).

In the request body, you can specify the “targets” parameter, which takes the following values:

- “coordinates” — Calculate coordinates of people.
- “overview” — Get an image with the maximum number of people.

You can set both values or none of them. If no values are set, a basic analysis will be performed, containing only information about the video recording, the number of people and the frames associated with them.

For such video analytics it is also possible to configure [ROI area of interest](#) on a frame (x, y coordinates, width, height, units - percent or coordinates). The principle of ROI is similar to FaceStream.

For this type of video analytics, you can also configure a rectangular [region of interest](#) (parameter “roi”) on the frame (coordinates “x”, “y”, width, height, units - per cent or coordinates). The principle of ROI region of interest is similar to FaceStream.

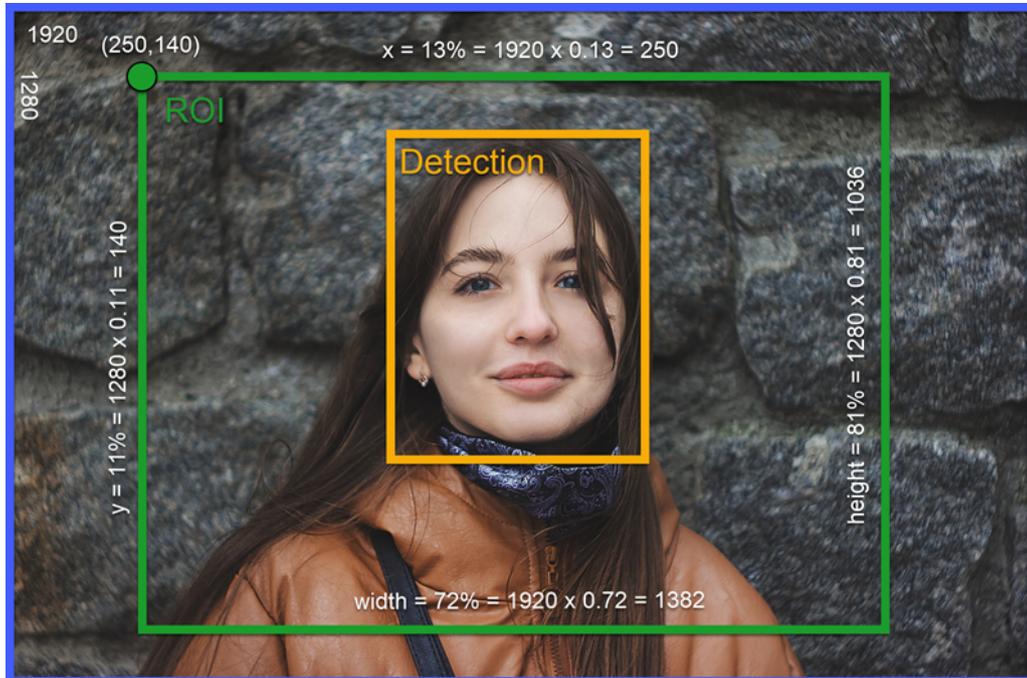


Figure 55: Faces database schema

Each event in people counting video analytics contains the following information:

- “event_id” — Event ID.
- “max_people_count” — Maximum number of people detected in the processed video segment.
- “video_segment” — Information about the video segment (start time of the event, end time of the event, first frame number, last frame number).
- “frames_estimations” — Array with information about estimates on each frame, containing the following information:
 - Frame number.
 - Time corresponding to the frame.
 - Number of people.

7.2.2 Human tracking video analytics

For people tracking video analytics, the key concept is a track. A track is an object containing information about the position of one person in a sequence of frames. A track is created when the detection of a face, body or body with a face (depends on the detector type in the “detector_type” parameter) appears on the last frame of the specified number of consecutive frames (“probe_count” parameter). There can be as

many tracks as the number of people detected on the frame. Along with a track, a video analytics event is started.

Note: Currently there can be only one event for one track. In future releases, it will be possible to have multiple events for a track, which can be sent without waiting for the track to end.

For example, if “probe_count” = “3” and there are detections on two consecutive frames and no detections on the third frame, the track and event will not start. If at any moment of tracking a person there were detections on two consecutive frames and no detections on the third frame, the track and event will end.

During tracking of a person it is possible to estimate attributes of faces, bodies, images (list of estimations is set in the “targets” parameter), and also to get the image with the best detection. The estimation can be performed by one best image from the track or by several (parameter “face_samples”/“body_samples” > “count”). If necessary, you can filter the best shots by “score”, “head_pitch”, “head_roll”, “head_yaw” parameters. All scores set in the “targets” field depend on the type of detector used.

If necessary, you can enable Re-identification of the body (ReID). The ReID feature is used to improve tracking accuracy by combining different tracks of the same person.

For this type of video analytics, you can also configure a rectangular [region of interest](#) (parameter “roi”) on the frame (coordinates “x”, “y”, width, height, units - per cent or coordinates). The principle of ROI region of interest is similar to FaceStream.

Each people count video analytics event contains the following information:

- “event_id” — Event ID.
- “track_id” — Track ID.
- “video_segment” — Information about the video segment (start time of the event, end time of the event, first frame number, last frame number).
- “frames_estimations” — Array with information about estimates on each frame, containing the following information:
 - Frame number.
 - Time corresponding to the frame.
 - Bbox coordinates and score.
- “aggregated_estimations” — Result of the score specified in the “targets” parameter in the request body.
- “track_count” - Number of tracks.

7.2.2.1 roi

ROI (Region of Interest) specifies the region of interest in which the estimation is performed.

The specified rectangular area is cut out from the frame and LUNA PLATFORM performs further processing using this image.

Correct exploitation of the “roi” parameter significantly improves the performance of video analysis.

ROI on the source frame is specified by the “x”, “y”, “width”, “height” and “mode” parameters, where:

- “x”, “y” – Coordinates of the upper left point of the ROI area of interest.
- “width” and “height” – Width and height of the processed area of the frame.
- “mode” – Mode for specifying “x”, “y”, “width” and “height”. Two modes are available:
 - “abs” – Parameters “x”, “y”, “width” and “height” are set in pixels.
 - “percent” – Parameters “x”, “y”, “width” and “height” are set as percentages of the current frame size.

If the “mode” field is not specified in the request body, then the value “abs” will be used.

With width and height values of “0”, the entire frame is considered the region of interest.

The coordinate system on the image is set similarly to the figure below.

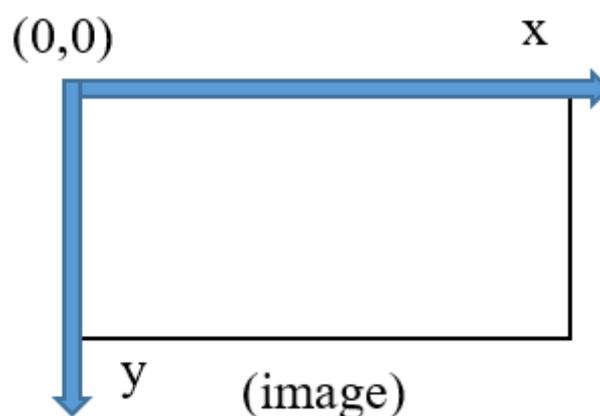


Figure 56: ROI coordinate system

When the values of width and height are set to “0”, the entire frame will be the region of interest.

Below is an example of calculating ROI as a percentage:

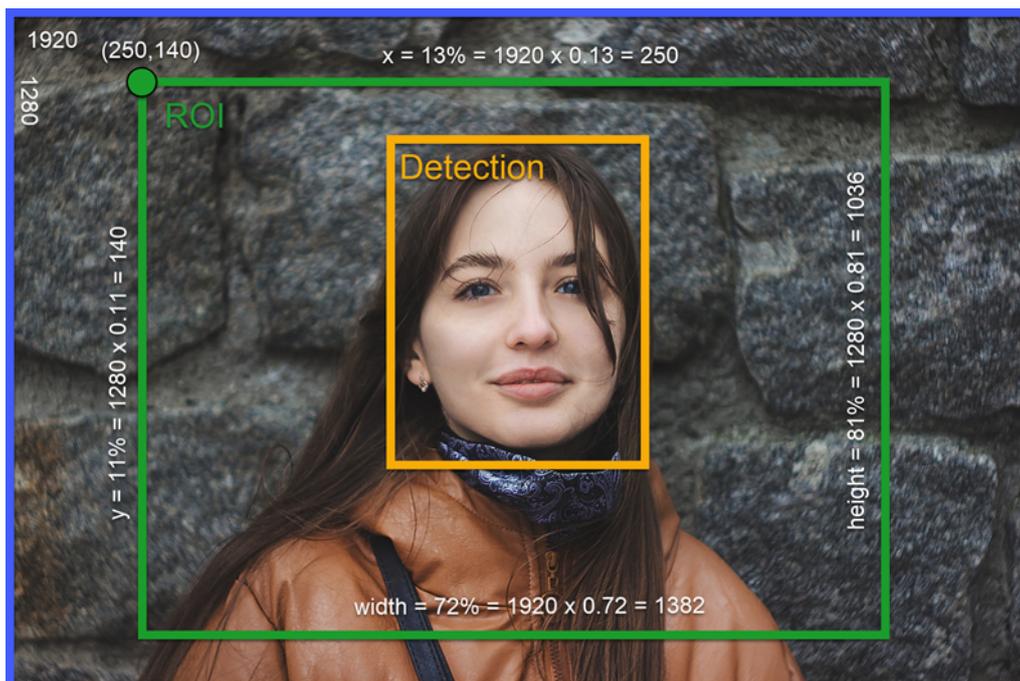


Figure 57: Example of calculating ROI as a percentage

7.2.2.2 droi

Note: DROI is only available for [people counting video analytics](#).

DROI specifies a region of interest relative to the source frame. DROI, unlike ROI, is not an estimation parameter, but works as a filter after estimation has been performed in accordance with the “targets” field. In other words, ROI is for estimation and optimization, DROI is for implementing business logic. You can use DROI together with ROI or separately. For example, if after execution the estimate taking into account the ROI of the number of people in the frame was equal to M, then after performing an additional filter on the DROI area, the number of people in the frame can be reduced to N.

The DROI on the source frame is specified by the following parameters:

- “area” — Geometry of the region of interest. The parameter is represented as an array of polygons. Each polygon is represented by an array of objects, where each object contains the x and y coordinates of the polygon’s vertex. For example, you can define a triangular area of interest.
- “mode” – mode of specifying “x”, “y”. Two modes are available:
 - “abs” — Parameters “x”, “y”, “width” and “height” are set in pixels.
 - “percent” — Parameters “x”, “y”, “width” and “height” are set as percentages of the current frame size.
- “form” — Format of the region of interest. In the current implementation there is only one possible value, “common”. Future implementations will support other formats in addition to polygons.

7.3 Filters

The LUNA PLATFORM has the ability to filter objects according to certain rules. For example, you can match faces only from a certain list or get events for a certain period of time.

As a rule, filters are located in the “filters” field of various requests.

7.3.1 Filters by comparison operators

For some filters, it is possible to specify comparison operators using the suffix “__operator”. The operator is used both for time filters (from a certain date, for some time) and for ordinary filters (for some age of a person, for a list of IDs in lexicographic order).

__lt

The “less than” filter is used to search for values less than the specified one. For example, when using the filter “create_time__lt:”2022-08-11T09:11:41.674Z” in the “get events” request events with the creation time preceding the specified date and time will be returned, i.e. events created before August 11, 2022, 09:11:41.674 UTC.

__lte

The “less than or equal to” filter is used to search for values less than or equal to the specified one. For example, when using the filter “face_id__lte:”2046d39b-410d-481e-b9de-ed6a0c7367f” in the filters of the request “matching faces” for candidates, only those candidates whose IDs begin will be returned with “2046d39b-410d-481e-b9de-ed6a0c7367f” or less than this value in lexicographic order.

__gte

The “greater than or equal to” filter is used to search for values greater than or equal to the specified one. For example, when using the filter “similarity__gte:”0.9” in the “storage_policy” > “face_sample_policy” > “filters” > “match” of the “create handler” request, you can configure a rule for saving a face sample in the Faces database only if the “matching_policy” has determined that the similarity of the candidate with the reference is greater than or equal to “0.9”.

7.3.2 Now-time filters

For the time filters “create_time”, “insert_time” and “end_time”, the format for setting the time relative to the current time (“now-time” format) is available. For example, when creating a [schedule](#) for a task [Garbage collection](#), you can specify the filter “create_time__lt:”now-30d”, which will delete all objects except those created in the last 30 days.

It must be remembered that:

- When creating a [schedule](#), the current time will be counted not from the creation of the schedule, but from the creation of the task by the schedule in accordance with the cron expression.

- When an event is generated by a handler that specifies the “now-time” filter (for example, in the “matching_policy” > “candidates” policy), the current time will be counted from the moment the event is generated.

To use the time relative to event generation, you can set policies directly in the request for event generation using the “[multipart/form-data](#)” scheme or using the [Estimator task](#) with the creation of a new handler.

In this format, the time is set according to the following pattern `now-(\d+)[smhdwMy]`, where “[d+]” is a number, “[smhdwMy]” is the required period: m (minutes), h (hours), d (days), w (weeks), M (months), y (years).

7.4 User interfaces

7.4.1 LUNA CLEMENTINE 2.0 Web Service

The LUNA CLEMENTINE 2.0 web service can be used to perform basic LP requests to create objects and perform matching operations.

This service is not supplied with LUNA PLATFORM 5, it must be downloaded and configured separately.

7.4.2 Backport 3 and Backport 4 service interfaces

Interfaces for interacting with LP 4 and LP 3 APIs are available in LUNA PLATFORM 5. The interfaces are intended only for those users who chose not to upgrade to the new version of the LUNA PLATFORM 5 API. They don't include most of the features available in LP 5 and will never include.

Service	Description	Default login data	Default port
Backport 4	Interface for working with LP API 4	-	4200
Backport 3	Interface for working with LP API 3	To log in to the interface, you need to create an account on the Sign Up tab.	4100

7.4.3 Other interfaces

These interfaces are needed to simplify working with LUNA PLATFORM services.

Service	Description	Default login data	Default port
Configurator	Interface for working with LUNA PLATFORM settings. It enables you to configure the parameters of all services in one place. It is especially convenient if auto-reloading of services is enabled after updating the settings.	-	5070

Service	Description	Default login data	Default port
Admin	Interface for performing administrative tasks of the LUNA PLATFORM: creating accounts using the GUI, launching and monitoring the execution of long tasks (Garbage collection task , Additional extraction task).	root@visionlabs.ai/root	5010
InfluxDB	The interface allows you to view the monitoring data of LUNA PLATFORM services.	luna/password	8086
Grafana (LUNA Dashboards)	Interface for visualizing LUNA PLATFORM monitoring data stored in InfluxDB. Dashboards have been written for it to visualize and filter information. You can also use the log aggregation system Grafana Loki .	admin/admin	3000

7.5 Disableable services

Some unused [general](#) services can be disabled. When a service is disabled, its main functions will not work. For example, if the Tasks service is disabled, attempting to execute a task will result in an error. Similarly, if the Sender service is disabled and an event handler with the “notification_policy” is used, it will also result in an error.

As a result, all resources that require a disabled service will return errors like `<Service_name> service is disabled` with a 403 response code.

If any service fails during the operation of the LUNA PLATFORM or it is manually stopped, the operation of dependent services will fail. For example, when the Licenses service is stopped, the following error will be displayed in the Faces service logs:

```
[0000001 2023-08-11 13:00:51.042000] ERROR: luna-faces
1690887528,00000000-0000-4000-a000-000000000001: Check connection to Luna
Licenses : GET:http://127.0.0.1:5120/version:Cannot connect to host
127.0.0.1:5120 ssl:default [Connection refused]
```

The corresponding error will be returned when trying to make a request that depends on the problematic service.

Then the work of services dependent on the Faces service will also be terminated with a similar error.

You can check the dependence of services on each other using the [interaction diagram](#) or you can find a setting like “service_name_ADDRESS” in the settings of the service for which you want to define dependent services. For example, in the settings of the Faces service there is a setting “LUNA_LICENSES_ADDRESS” responsible for connecting to the Licenses service, and in the settings of the Admin service there is a setting “LUNA_FACES_ADDRESS” responsible for connecting to the Faces service.

7.5.1 Disabling services process

- Open the Configurator user interface `http://<configurator_server_ip>:5070`.
- Enter the name of the setting “ADDITIONAL_SERVICE_USAGE” in the “Setting name” field and click “Apply Filters”.
- Set the value for the required service to “false”.
- Save the changes by clicking the “Save” button.

7.5.2 Image Store service disabling consequences

When the Image Store service is disabled, there are some specific features to note:

- Objects of the type `images`, `objects`, `samples`, and `sample save policies` in handlers/verifiers will be unavailable.

- All tasks, except Garbage Collection, Linker, and Estimator, will become unavailable. However, there are some limitations for these tasks:
 - Garbage Collection, Estimator, Linker: task/subtask results will not be saved
 - Garbage Collection, Estimator, Linker: after the subtask completes, the task status will be updated to Done, and the task result ID will be None.
 - Garbage Collection: deleting samples will become unavailable.

If the Image Store service is disabled after events with the `image_origin_policy` are generated, when using the Garbage Collection task and the `remove_image_origins` parameter, the Tasks service will still attempt to delete the source images with an external URL.

7.5.2.1 Image Store service disabling consequences for Backport 3

In the Backport 3 service, the “BACKPORT3_ENABLE_PORTRAITS” setting is available, which enables you to disable the ability to use portraits, but leave the ability to use the rest of the functionality of the Image Store service. If the use of the Image Store service is disabled in the “ADDITIONAL_SERVICES_USAGE” setting, then the above setting must also be disabled.

When the Image Store service is disabled, samples and portraits, as well as the “get portrait” and “get portrait thumbnail” resources, become unavailable.

7.5.3 Handlers service disabling consequences

When disabling the Handlers service:

- Launching the API service will lead to the unavailability of using the following requests:
 - “detect faces”
 - “extract attributes”
 - “estimator task”
 - All resource requests “/handlers”
 - All resource requests “/verifiers”
- Launching the Tasks service will lead to the unavailability of performing the tasks “Additional extraction” and “Estimator”.
- Launching the Admin service will lead to the unavailability of performing the task “Additional extraction”.

7.5.4 Events service disabling consequences

When disabling the Events service:

- It will be possible to create a handler with an event saving policy, but the event itself will not be possible to generate.
- All requests to the “/events” resource will not work.

- Requests to resources `"/handlers/{handler_id}/events"`, `"/handlers/{handler_id}/events/raw"` not will work.
- The `"event_id"` parameter in the `"create face"` request will not work.
- Matching by candidate events will not be available.
- Filters by events in tasks will not be available.
- Using the `"ws handshake"` request will not make sense.

7.5.5 Tasks service disabling consequences

When disabling the Tasks service:

- All requests to the `"/tasks"` resource (sections `"task processing"`, `"task info"`, `"task errors"`) will not work.
- The use of tasks in all services will be unavailable.

Tasks performed in the Admin service user interface are also unavailable when the Tasks service is disabled.

7.5.6 Sender service disabling consequences

When disabling the Sender service:

- The `"ws handshake"` request will not work.
- The `"notification_policy"` policy in the handler will not work.

7.5.7 Disabling additional services

Additional services can also be disabled if they were previously enabled.

If the Python Matcher Proxy service was previously used, then disabling it will make it impossible to use [matching plugins](#) or [LIM module](#).

If the Lambda service was previously used, then disabling it will make it impossible to send requests to the `"/lambdas"` resources and work with the created lambda.

If Backport 3 or Backport 4 services were previously used, then disabling them will make it impossible to process LUNA PLATFORM 3/4 requests using LUNA PLATFORM 5.

Enabling Backport 3, Backport 4, User Interface 3 and User Interface 4 services is regulated only by launching the corresponding containers. There are no parameters in the `"ADDITIONAL_SERVICE_USAGE"` settings that enable these services.

7.6 Nuances of working with services

When working with different services, it is necessary to take into account some nuances that will be described in this section.

7.6.1 Auto-orientation of rotated image

It is not recommended to send rotated images to LP as they are not processed properly and should be rotated. There are two methods to auto-orient a rotated image — based on EXIF image data (query parameters) and using LP algorithms (Configurator setting). Both methods for automatic image orientation can be used together.

If auto-orientation is not used, the sample creation mechanism will rotate the image and produce an image with a random rotation angle.

7.6.1.1 Auto-orientation based on EXIF data

This method of image orientation is performed in the query parameters using the “use_exif_info” parameter. This parameter can enable or disable auto-orientation of the image based on EXIF data.

This parameter is available and enabled by default in the following resources:

- “/detector”
- “/verifiers/{verifier_id}/verifications”
- “/sdk”
- “/handlers/{handler_id}/events”

The “use_exif_info” parameter cannot be used with samples. When the “warped_image” or “image_type” query parameter is set to the appropriate value, the parameter is ignored.

7.6.1.2 Auto-orientation based on Configurator setting

This method of image orientation is performed in the Configurator using the “LUNA_REMOTE_SDK_USE_AUTO_ROTATION” setting. If this setting is enabled and the input image is rotated by 90, 180 or 270 degrees, then LP rotates the image to the correct angle. If this setting is enabled, but the input image is not rotated, then LP does not rotate the image.

Performing auto-orientation consumes a significant amount of server resources.

The “LUNA_HANDLERS_USE_AUTO_ROTATION” setting cannot be used with samples. If the “warped_image” or “image_type” query parameter is set to the appropriate value and the input image is a sample and rotated, then the “LUNA_HANDLERS_USE_AUTO_ROTATION” setting will be ignored.

7.6.2 Saving source images

The URL to the source image can be saved in the “image_origin” field of the created events when processing the [“/handlers/{handler_id}/events”](#) request.

To do this, you should specify the “store_image” parameter in the “image_origin_policy” when creating handler.

Then you should set an image for processing in the [“generate events”](#) request.

If “use_external_references=0” and the URL to an external image was transferred in the “generate events” request, then this image will be saved to the Image Store storage, and the ID of the saved image will be added in the “image_origin” field of the generated event.

The “use_external_references” parameter enables you to save an external link instead of saving the image itself:

- If “use_external_references” = “1” and the URL to an external image was transferred in the “generate events” request, then that URL will be added in the “image_origin” field. The image itself will not be saved to the Image Store.
- If “use_external_references” = “1”, the sample was provided in the “generate events” request and “face_sample_policy > store_sample” is enabled, the URL to the sample in the Image Store will be saved in the “image_origin” field. The duplication of the sample image will be avoided.

If an external URL is too long (more than 256 symbols), the service will store the image to the Image Store.

You can also provide the URL to the source image directly using the [“/handlers/{handler_id}/events”](#) resource. To do this, you should use the “application/json” or “multipart/form-data” body schema in the request. The URL should be specified in the “image_origin” field of the request.

If the “image_origin” is not empty, the provided URL will be used in the created event regardless of the “image_origin_policy” policy.

The image provided in the “image_origin” field will not be processed in the request. It is used as a source image only.

7.7 Neural networks

Using neural networks, the parameters of faces or bodies are estimated and their descriptors are extracted.

There are two types of neural networks — for performing estimation and for extracting descriptors.

Neural networks for performing detection and estimation are located in the Remote SDK service container in the format `<estimation_name>_<architecture>.plan`, where `<estimation_name>` is the name of the neural network, `<architecture>` the architecture used (`cpu-avx2` or `gpu`). Such neural networks are called **detectors** and **estimators**, respectively.

Neural networks for extracting descriptors are located in the Remote SDK service container in the format `cnn<model>_<architecture>.plan`, where `<model>` is the neural network model, `<architecture>` the architecture used (`cpu-avx2` or `gpu`).

To work with a neural network, a configuration file of the `cnndescriptor_<model>.conf` format is used, where `<model>` is a neural network model. Configuration files for all neural networks are also located in the Remote SDK service container.

Next, information only for neural networks for extracting descriptors will be described.

It is possible to remove any neural network from the Remote SDK service container if [use of some estimators or detectors is disabled](#).

The current build of LUNA PLATFORM supports neural network models for extracting descriptors:

Object from which descriptor is extracted	Neural network models	Default model
Face	59, 60, 62	62
Body	107, 110	110

The sizes of all neural network models for extracting face descriptors are given below:

Model	Data size in Raw format (bytes)	Metadata size (bytes)	Data size in SDK format (Raw + metadata)
54	512	8	520
56	512	8	520
57	512	8	520
58	512	8	520
59	512	8	520

Model	Data size in Raw format (bytes)	Metadata size (bytes)	Data size in SDK format (Raw + metadata)
60	512	8	520
62	512	8	520

The sizes of all neural network models for extracting body descriptors are given below:

Model	Data size in Raw format	Metadata size (bytes)	Data size in SDK format (Raw + metadata)
102	2048	8	2056
103	2048	8	2056
104	2048	8	2056
105	512	8	520
106	512	8	520
107	512	8	520

See the detailed information about descriptor formats in the section [“Descriptor formats”](#).

Neural network 105 is newer version of neural network 102.

Descriptors received using different models of the neural networks are not comparable with each other. That is why you should re-extract all the descriptors from the existing samples if you are going to use a new NN model (see [“Change neural network used”](#)).

You can store several descriptors for the same image with different NN linked to a single face.

See the `“DEFAULT_FACE_DESCRIPTOR_VERSION”` and `“DEFAULT_HUMAN_DESCRIPTOR_VERSION”` parameters in the Configurator service to check the current extraction neural network.

Before any actions with neural networks, you should consult with VisionLabs specialists.

7.7.1 Change neural network used

Changing the used neural network model is required when it is necessary to improve the quality of face/body recognition or when old models are declared obsolete and removed from the Remote SDK service container.

When changing the neural network model used, one should:

- **to preserve the possibility of using old descriptors:** perform the long additional extraction task so the already existing descriptors can be extracted using the new neural network model (see [“Launch Additional extraction task”](#));
- set the new neural network model in LP configurations (see [“Change neural network model in settings”](#)).

You should not change the default neural network, before finishing the additional extraction task.

You can both increase the model of the neural network, and lower it. To downgrade the neural network model, you need to perform similar steps to upgrading.

7.7.1.1 Launch Additional extraction task

The additional extraction task can be launched using one of the following ways:

- Using the [“/additional_extract”](#) request to the Admin API.
- Using the Admin service user interface, by default located at `http://<admin_server_ip>:5010`.

Depending on the way, you need to specify the following information:

- Object type: faces or events.
- Extraction target: face descriptor, body descriptor or basic attributes.
- New neural network version (not applicable for basic attributes).

For more information, see [“Additional extraction task”](#) section.

7.7.1.2 Change neural network model in settings

You should set the new neural network model in the configurations of services. Use the Configurator GUI for this purpose:

- Go to to the `http://<configurator_server_ip>:5070`.
- Set the required neural network in the `“DEFAULT_FACE_DESCRIPTOR_VERSION”` parameter (for faces) or the `“DEFAULT_HUMAN_DESCRIPTOR_VERSION”` (for bodies).
- Save changes using the `“Save”` button.
- Wait until the setting is applied to all the LP services.

7.7.2 Use non-delivery neural network model

This section describes the process of moving a non-delivery neural network to the Remote SDK container. This is necessary if the user is using the old neural network from a previous version of LP and does not want to change it when upgrading to a new version of LP.

It is necessary to request an archive with neural network files from a VisionLabs representative. The archive contains the following files:

- Neural network file(s) `cnn<model>_<architecture>.plan`, where `<model>` — neural network model, `<architecture>` architecture used (cpu-avx2 and/or gpu).
- Configuration file `cnndescriptor_<model>.conf`, where `<model>` — neural network model.

After downloading the archive with the neural network and the archive with its configuration, you should perform the following steps:

- Unzip the archive.
- Assign rights to neural networks.
- Copy the neural networks and their configuration files to the launched Remote SDK container.
- Make sure that the required model of the neural network is used in the service configurations (see [“Change neural network model in settings”](#) section).

Below is an example of commands for transferring neural networks to the Remote SDK container.

7.7.2.1 Unzip neural networks

Go to the directory with the archives and unzip them.

```
unzip <archive_name>.zip
```

7.7.2.2 Assign rights to neural networks

```
chown -R 1001:0 <archive_name>/cnn<model>_<architecture>.plan
```

7.7.2.3 Copy neural network and configuration file to Remote SDK container

Copy the neural network and its configuration file to the Remote SDK container using the following commands.

```
docker cp <archive_name>/cnn<model>_<architecture>.plan luna-remote-sdk:/srv/  
fsdk/data/
```

```
docker cp <archive_name>/cnndescriptor_<model>.conf luna-remote-sdk:/srv/  
fsdk/data/
```

`luna-remote-sdk` — Name of the launched Remote SDK container. This name may differ in your installation.

Check that the required model for the required device (CPU and/or GPU) was successfully loaded:

```
docker exec -t luna-remote-sdk ls /srv/fsdk/data/
```

7.8 Logging

There are two ways to output logs in LUNA PLATFORM:

- Standard log output (stdout).
- Log output to a file.

Log output settings are set in the settings of each service in the `<SERVICE_NAME>_LOGGER` section.

By default, logs are output only to standard output.

You can view the service logs via standard output using the `docker logs <service_name>` command.

It is recommended to set up an external system for collecting and storing logs. This manual does not provide an example of configuring an external system to configure logs.

If necessary, you can use both methods of displaying logs.

When enabling logging to a file in a container, the default path is `srv/logs` for each container. Logs are saved in the format `<service_name>_<type_of_logs>.txt`, where:

- `<service_name>` — Name of the service, for example, `luna-faces`.
- `<types_of_logs>` — Type of output logs — “ERROR”, “INFO”, “WARNING”, “DEBUG”.

It is possible to create up to six files of each type. For each type of logs output in the Configurator service settings, you can set the maximum size (the “`max_log_file_size`” parameter in the `<SERVICE_NAME>_LOGGER` section). So, for example, six 1024 MB files can be created for the “INFO” type. For other types, the principle of operation is similar. Thus, if the maximum value of “`max_log_file_size`” is 1024 MB, then the total amount of memory occupied by logs in the container cannot exceed $6 * 4 * 1024 = 24576$ MB. After the remaining space in the last file is used up, the first file will be overwritten. If it is necessary to reduce the amount of memory occupied by logs, then it is necessary to reduce the value of the parameter “`max_log_file_size`”.

You can synchronize the service logs folder with the local folder on the server using the command `-v /tmp/logs/<service_name>:/srv/logs\` when starting the container. To do this, you first need to create the corresponding directory on the server and assign it the necessary rights.

Without assigning rights to the mounted folder, the service will issue the corresponding error.

When you enable saving logs to a file, you should remember that logs occupy a certain place in the storage (see above), and the process of logging to a file negatively affects system performance.

7.9 Image check

LUNA PLATFORM enables you to perform various front-type image checks. Check can be done either with thresholds conforming to [ISO/IEC 19794-5:2011](#), and by manually entering thresholds and selecting the necessary checks.

The results of the checks are not stored in the database, they are returned only in the response.

ISO/IEC 19794-5:2011 checks are performed using the `"/iso"` resource (see detailed description in the ["Image check according to ISO/IEC 19794-5:2011"](#) section below).

Checks with manually specified thresholds using the `"face_quality"` group of checks of `"detect_policy"` policy of `"/handlers"` and `"/verifiers"` resources (see detailed description in the ["Image check according to specified conditions"](#) section below).

The possibility of performing such checks is regulated by a special parameter in the license file.

The result of all checks is determined by the `"status"` parameter, where:

- `"0"` — Any of the checks were not passed.
- `"1"` — All checks have been passed.

The result of each check is also displayed next to it (the `"result"` parameter).

You can enable checking for multiple faces in a photo using the `"multiface_policy"` parameter. Checks are performed for each face detection found in the photo. Check results are not aggregated.

For some checks, certain requirements should be met. For example, in order to get the correct results of the status of eyebrows, it is necessary that the head angles are in a certain range and the face width is at least 80 pixels. The requirements for the checks are listed in the section ["Estimated data"](#). If any requirements are not met when check a certain parameter, the results of checking this parameter may be incorrect.

For the following checks, it is **not available to use a sample** as a source image:

- [Vertical/horizontal face position checks](#) (parameters `head_horizontal_center`, `head_vertical_center`)
- [Vertical/horizontal head sizes check](#) (parameters `head_width`, `head_height`)
- [Distance between eyes check](#) (parameter `eye_distance`)
- [Face width/height check](#) (parameters `indent_upper`, `indent_lower`, `indent_right`, `indent_left`)
- [Indents from image edges checks](#) (parameters `indent_upper`, `indent_lower`, `indent_right`, `indent_left`)
- [Illumination uniformity check according to ICAO standard](#) (parameter `illumination_uniformity`)
- [Dynamic range check](#) (parameter `dynamic_range`)
- [Background check](#) (parameters `background_lightness`, `background_uniformity`)

The set of checks for “face_quality” and “/iso” resource is different (see the difference between checks in “[Comparison table of available checks](#)” section).

7.9.1 Image check according to ISO/IEC 19794-5:2011

This check is performed using the “/iso” resource.

By default, images with one face present are checked. For each of the found faces, the estimates and coordinates of the found face will be returned. It should be noted that many ISO checks assume the presence of one face in the frame, so not all checks for multiple faces will be performed successfully.

The order of the returned responses after processing corresponds to the order of the transferred images.

You can additionally enable the extraction of EXIF data of the image in the request.

For each check, thresholds are set that comply with ISO requirements. The value of the thresholds for each check is given in the sample response to the “/iso” request in the OpenAPI documentation.

Some checks are united under one ISO requirement. For example, to successfully pass the eye status check, the statuses of the left and right eyes should take the value “open”.

The following information is returned in the response body:

- Verdict on passing checks, which is 1 if all checks are successful.
- Results of each of the tests. This enables you to determine which particular test was not passed. The following values are returned:
 - The name of the check.
 - The value obtained after performing the check.
 - The default threshold. The thresholds are set in the system by the requirements of the ISO/IEC 19794-5:2011 standard and cannot be changed by the user.
 - The result of this check. When passing the thresholds, 1 is returned.
- The coordinates of the face.

If an error occurs for one of the processed images, for example, if the image is damaged, an error will be displayed in the response. Processing of the rest of the images will continue as usual.

In addition to the “/iso” resource, the ability to check for compliance with ISO/IEC 19794-5:2011 and ICAO standards is available in the “/detector” (parameter “estimate_face_quality”) resource. The principle of performing checks is similar to the one described above, however, additional checks from the “face_quality” checks group are available in this resource.

7.9.2 Image check according to specified conditions

The principle of operation is similar to check according the ISO standard, but the user has the right to decide for himself which checks need to be performed and which thresholds to set.

To enable checks, you should specify the value “1” in the “estimate” field for “face_quality”. Image check is disabled by default. To disable a certain check, you need to set “0” in the “estimate” field for this check. By default, all checks are enabled and will be performed when “face_quality” is enabled.

Depending on the type of check, you can specify the minimum and maximum values of the threshold, or allowable values for this check. For this, the “threshold” field is used. If the minimum or maximum threshold is not set, the minimum or maximum allowable value will be automatically selected as the unset threshold. If the maximum value is unlimited (for example, “>=0”), then the value “null” will be returned in the “max” field in the event response body. If both thresholds are not specified, the check will be performed using the standard thresholds set in the [“FACE_QUALITY_SETTINGS”](#) section in the Configurator service or in the “face_quality” request body. If the threshold is specified in the “face_quality” request body, then this overrides the standard thresholds specified in the “FACE_QUALITY_SETTINGS” section.

Default thresholds are selected by VisionLabs specialists to obtain optimal results. These thresholds may vary depending on shooting conditions, equipment, etc.

When setting thresholds for the checks [“Image quality”](#) and [“Head pose”](#), it is recommended to take into account the standard thresholds preset in the system settings. For example, to check the image for blurriness (the “blurriness_quality” parameter), it is recommended to set the threshold in the range [0.57...0.65]. When setting a threshold outside this range, the results may be unpredictable. When choosing the angles of the head position, you need to pay attention to the recommended maximum thresholds for estimation in cooperative and non-cooperative modes. Information about these thresholds is provided in the relevant sections of the administrator manual.

It is recommended to consider the results of mouth state checks (“mouth_smiling”, “mouth_occluded”, “mouth_open”, “smile_properties”) together. So, for example, if the check revealed that the face is occluded with something, then the rest of the results of the mouth check will not be useful.

It is possible to enable filtering based on check results (“filter” parameter). If one of the “face_quality” checks for the detection fails, then the results and the reason for filtering will be returned. No further policies will be performed for this detection.

In addition, some checks are available in the “face_quality” group of checks, which are not available in the images check according the standard (see below).

7.9.3 Comparison table of available checks

The following checks are available for [“/iso”](#) resource and “face_quality” group of checks of the [“/handlers”](#) and [“/verifiers”](#) resources:

Checks description	Checks name	Resource	
		“/iso”	“face_quality”
Image quality check	illumination_quality, specularity_quality, blurriness_quality, dark_quality, light_quality	+	+
Background check	background_- lightness, background_- uniformity	+	+
Illumination uniformity check according to ICAO standard	illumination_- uniformity	-	+
Head pose check	head_yaw, head_pitch, head_roll	+	+
Gaze check	gaze_yaw, gaze_pitch	+	+
Mouth attributes check	mouth_smiling, mouth_occluded, mouth_open	+	+
Smile state check	smile_properties (none, smile_lips, smile_teeth)	+	+*
Glasses state check	glasses	+	+
Eyes attributes check	left_eye (open, occluded, closed), right_eye (open, occluded, closed)	+	+
Distance between eyes check	eye_distance	+	+
Natural light check	natural_light (0, 1)	+	+
Radial distortion check (Fisheye effect)	radial_distortion (0, 1)	+	+
Red eyes effect check	red_eyes (0, 1)	+	+
Eyebrows state check	eyebrows_state (neutral, raised, squinting, frowning)	+	+*

Checks description	Checks name	Resource	
		“/iso”	“face_quality”
Headwear type check	headwear_type (none, baseball_cap, beanie, peaked_cap, shawl, hat_with_ear_flaps, helmet, hood, hat, other)	+	+*
Vertical/horizontal face position checks	head_horizontal_center, head_vertical_center	+	+
Vertical/horizontal head sizes check	head_width, head_height	+	+
Image format check	image_format	+	+
Face color type check	face_color_type (color, grayscale, infrared)	+	+
Shoulders position check	shoulders_position	+	+
Image size check	image_size	-	+
Indents from image edges checks	indent_upper, indent_lower, indent_right, indent_left	-	+
Image width/height checks	image_width, image_height	-	+
Image aspect ratio check	aspect_ratio	-	+
Face width/height check	face_width, face_height	-	+
Dynamic range check	dynamic_range	-	+

* Several parameters can be specified for these checks.

7.10 Services health checks

A “/healthcheck” resource is intended for service health checks. The resource can be used to actively check the health of a service, namely, whether the service can perform its functions in full or not. The ability to connect this service to the LP and DB services on which it depends is checked.

It is possible to set up a periodic resource check using HAProxy, NGINX or another system. This will enable you to determine that the service is unavailable and decide whether to disconnect the service from the contour or restart it.

Using the “include_luna_services” option, you can enable and disable health check for the LUNA PLATFORM services on which this service depends. If this option is enabled, additional requests are sent to the “/healthcheck” resources of these services. The “include_luna_services” option is disabled in order not to perform recursive checking of the same services. For example, when several services on which this service depends at once will send requests to the Faces service and thereby increase the load on it.

If the health check is successful, only the connection execution time in the “execution_time” field is returned.

If one or more services are unavailable, an error code 502 “Unhealthy” is returned. The response body lists the components, check statuses, and errors that have occurred. The error code 500 in the response body does not necessarily mean a problem with the service. A long request may fail due to exceeded timeouts, increased server load, network problems or other reasons.

When performing a request to the “/healthcheck” resource, it is recommended to set a timeout of several seconds. If the request does not have time to be processed, this is a sign that problems have arisen during the operation of the system.

To check the health of services, a “get health (redirect)” request is also available, which enables you to not specify the API version in the request.

7.11 Upload images from folder

The “folder_uploader.py” script uploads images from the specified folder and processes uploaded images according to the preassigned parameters.

7.11.1 General information about the script

The “folder_uploader.py” script can be utilized for downloading images using the API service only.

You cannot specify the Backport 4 address and port for utilizing this script. You can use the data downloaded to the LP 5 API in Backport 4 requests.

You cannot use the “folder_uploader.py” script to download data to Backport 3 service as the created objects for Backport 3 differs (e. g. “person” object is not created by the script).

7.11.2 Script usage

Script pipeline:

1. Search images of the allowed type (formats: “.png”, “.jpg”, “.jpeg”, “.bmp”, “.ppm”, “.tif”, “.tiff”, color model: RGB, CMYK) in the specified folder (source).
2. Start asynchronous image processing according to the specified parameters (see section “[Script launching](#)”).

Image processing pipeline:

1. Detect faces and create samples.
2. Extract attributes.
3. Create faces and link them to a list.
4. Add record to the log file.

If an image was loaded successfully, the record is added to the {start_upload_time}_success_log.txt: success load logfile. The record has the following structure:

```
{
  "image name": ...,
  "face id": [...]
}
```

If errors occur at any step of the script processing, the image processing routine is terminated and a record is added to the error log file {start_upload_time}_error_log.txt: error. The record has the following structure:

```
{  
  "image name": ...,  
  "error": ...  
}
```

7.11.3 Install dependencies

Before the script launching you must install all the required dependencies to launch the script.

It is strongly recommended to create a virtual environment for python dependencies installation.

Install Python packages (version 3.7 and higher) before launching installation. The packages are not provided in the distribution package and their installation is not described in this manual:

- python3.7
- python3.7-devel

Install gcc.

```
yum -y install gcc
```

Go to the directory with the script

```
cd /var/lib/luna/current/extras/utils/folder_uploader
```

Create a virtual environment

```
python3.7 -m venv venv
```

Activate the virtual environment

```
source venv/bin/activate
```

Install the tqdm library.

```
pip3.7 install tqdm
```

Install luna3 libraries.

```
pip3.7 install ../luna3*.whl
```

Deactivate virtual environment

```
deactivate
```

7.11.4 GC script launching

Use the command to run the script (the virtual environment must be activated):

```
python3.7 folder_uploader.py --account_id 6d071cca-fda5-4a03-84d5-5
  bea65904480 --source "Images/" --warped 0 --descriptor 1 --origin http
  ://127.0.0.1:5000 --avatar 1 --list_id 0dde5158-e643-45a6-8a4d-
  ad42448a913b --name_as_userdata 1
```

Make sure that the `--descriptor` parameter is set to `1` so descriptors are created.

The API version of the service is set to `6` by default in the script, and it cannot be changed using arguments.

`--source "Images/"` — “Images/” is the folder with images located near the “folder_uploader.py” script. Or you can specify the full path to the directory.

`--list_id 0dde5158-e643-45a6-8a4d-ad42448a913b` — Specify your existing list here.

`--account_id 6d071cca-fda5-4a03-84d5-5bea65904480` — Specify the required account ID.

`--origin http://127.0.0.1:5000` — Specify your current API service address and port here.

See help for more information about available script arguments:

```
python3.7 folder_uploader.py --help
```

Command line arguments:

- “account_id” — Account ID used in requests to LUNA PLATFORM (**required**).
- “source” — Directory with images to load (**required**).
- “warped” — Whether the images warped or not (0,1) (**required**).
- “descriptor” — Whether to extract descriptor (0,1). Default — 0.
- “origin” — Origin. Default — “http://127.0.0.1:5000”.
- “avatar”. Whether to set sample as avatar (0,1). Default — 0.
- “list_id” — List ID to link faces with (a new LUNA list will be created if list_id is not set and list_linked=1). Default — None.

- “list_linked” — Whether to link faces with list (0,1). Default — 1.
- “list_userdata” — User data for list to link faces with (for newly created list). Default — None.
- “pitch_threshold” — Maximum deviation pitch angle [0..180].
- “roll_threshold” — Maximum deviation roll angle [0..180].
- “yaw_threshold” — Maximum deviation yaw angle [0..180].
- “multi_face_allowed” — Whether to allow several face detection from single image (0,1). Default — 0.
- “get_major_detection” — Whether to choose major face detection by sample Manhattan distance from single image (0,1). Default — 0.
- “basic_attr” — Whether to extract basic attributes (0,1). Default — 1.
- “score_threshold” — Descriptor quality score threshold (0..1). Default — 0.
- “name_as_userdata” — Whether to use image name as user data (0,1). Default — 0.
- “concurrency” — Parallel processing image count. Default — 10.

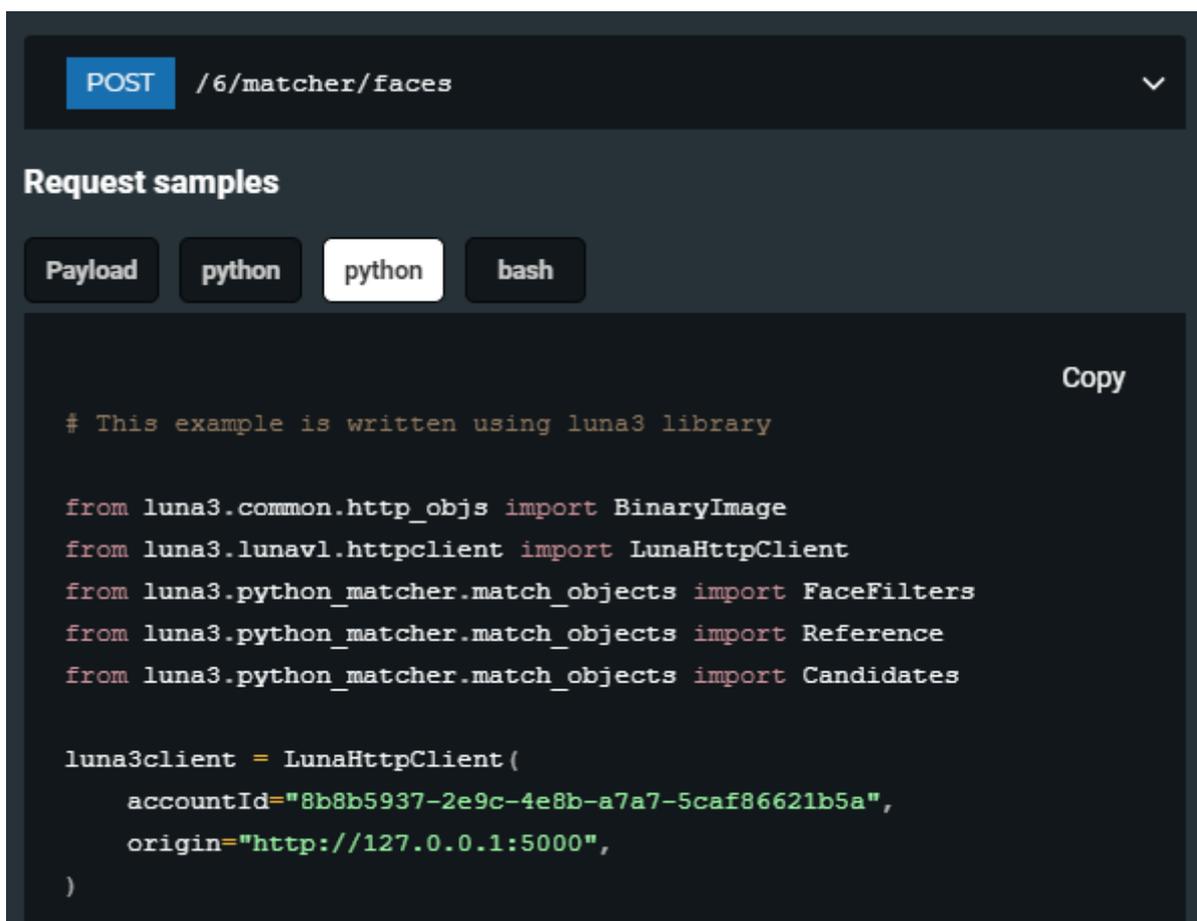
7.12 Client library

7.12.1 General information

The archive with the client library for LUNA PLATFORM 5 is provided in the distribution package: `/var/lib/luna/current/extras/utils/luna3-*.whl`

This Python library is an HTTP client for all LUNA PLATFORM services.

You can find the examples of the library utilization in the `/var/lib/luna/current/docs/ReferenceManuals/APIReferenceManual.html` document.



```
POST /6/matcher/faces
```

Request samples

Payload python python bash

Copy

```
# This example is written using luna3 library

from luna3.common.http_objs import BinaryImage
from luna3.lunavl.httpclient import LunaHttpClient
from luna3.python_matcher.match_objects import FaceFilters
from luna3.python_matcher.match_objects import Reference
from luna3.python_matcher.match_objects import Candidates

luna3client = LunaHttpClient(
    accountId="8b8b5937-2e9c-4e8b-a7a7-5caf86621b5a",
    origin="http://127.0.0.1:5000",
)
```

Figure 58: Luna3 library usage example

The example shows the request for faces matching. The luna3 library is utilized for the request creation. See “matcher” > “matching faces” in “APIReferenceManual.html”:

```
# This example is written using luna3 library {#this-example-is-written-using-luna3-library}
```

```

from luna3.common.http_objs import BinaryImage
from luna3.lunavl.httpclient import LunaHttpClient
from luna3.python_matcher.match_objects import FaceFilters
from luna3.python_matcher.match_objects import Reference
from luna3.python_matcher.match_objects import Candidates

luna3client = LunaHttpClient(
    accountId="8b8b5937-2e9c-4e8b-a7a7-5caf86621b5a",
    origin="http://127.0.0.1:5000",
)

# create sample {#create-sample}
sampleId = luna3client.saveSample(
    image=BinaryImage("image.jpg"),
    raiseError=True,
).json["sample_id"]

attributeId = luna3client.extractAttrFromSample(
    sampleIds=[
        sampleId,
    ],
    raiseError=True,
).json[0]["attribute_id"]

# create face {#create-face}
faceId = luna3client.createFace(
    attributeId=attributeId,
    raiseError=True,
).json["face_id"]

# match {#match}
candidates = Candidates(
    FaceFilters(
        faceIds=[
            faceId,
        ]
    ),
    limit=3,
    threshold=0.5,
)
reference = Reference("face", faceId)

response = luna3client.matchFaces(
    candidates=[candidates], references=[reference],
    raiseError=True,

```

```
)  
  
print(response.statusCode)  
print(response.json)
```

7.12.2 Library installation example

In this example a virtual environment is created for luna3 installation.

You can use this Python library on Windows, Linux, MacOS.

Install Python packages (version 3.7 and later) before launching installation. The packages are not provided in the distribution package and their installation is not described in this manual:

- python3.7
- python3.7-devel

Install gcc.

```
yum -y install gcc
```

Go to the directory with any script, for example, `folder_uploader.py`

```
cd /var/lib/luna/current/extras/utils/folder_uploader
```

Create a virtual environment.

```
python3.7 -m venv venv
```

Activate the virtual environment.

```
source venv/bin/activate
```

Install luna3 libraries.

```
pip3.7 install ../luna3*.whl
```

Deactivate virtual environment.

```
deactivate
```

7.13 Plugins

Plugins are used to perform secondary actions for the user's needs. For example, you can create your own resource based on the abstract class, or you can describe what needs to be done in some resource in addition to the standard functionality.

Files with base abstract classes are located in the `.plugins/plugins_meta` folder of specific service.

Plugins should be written in the Python programming language.

There are two sorts of plugins:

- Event plugin
- Background plugin
- Matching plugin

7.13.1 Event plugins

The first sort is triggered when an event occurs. The plugin should implement a callback function. This function is called on each event of the corresponding type. The set of event types is defined by the service developers. There are two types of event plugins available for the Remote SDK service:

- Monitoring event
- Sending event

For other services, only monitoring event type is available.

For examples of monitoring and sending plugins, see the [development manual](#).

7.13.2 Background plugins

The second sort of plugin is intended for background work. The background plugin can implement:

- Custom request for a specific resource (route).
- Background monitoring of service resources.
- Collaboration of an event plugin and a background plugin (batching monitoring points).
- Connection to other data sources (Redis, RabbitMQ) and their data processing.

For examples of background plugins, see the [development manual](#).

7.13.3 Matching plugins

The third type of plugins enables you to significantly speed up the processing of matching requests.

Plugins are not provided as a ready-made solution for matching. It is required to implement the logic required for solving particular business tasks.

Matching plugins are enabled in Python Matcher Proxy service. This service is not installed by default. You should run it to work with plugins. See the Python Matcher Proxy launching command in the “Use Python Matcher with Python Matcher Proxy” section of the LUNA PLATFORM installation manual.

In the default LUNA PLATFORM operation processing using Python Matcher Proxy, all matching requests are processed by Python Matcher Proxy service by redirecting requests to Python Matcher service. It is possible that matching requests processing is slower than it needs for several reasons, including:

- Large amount of data and inability to speed up requests by any database configuration changes, e.g. create an index in a database that speeds up request.
- Way of data storage — descriptor and entity id (face_id/event_id) are kept in different database tables. Filters, which specified in matching request also can be presented in a separate database table, which slows down the request processing speed.
- Internal database specific restrictions.

It is possible to separate some groups of requests and improve their processing speed by utilizing matching plugins, including by transferring data to another storage with a specific way of data storage which makes possible the fastest matching in comparison to the default way (see “[Plugin data source](#)”).

Examples:

- Matching requests where all faces (let’s say that all matching candidates are faces) are linked to one list and any other filters do not specify in the request.

In this case, it is possible to duplicate data for these matching candidates to other data storage than the default data storage and create a matching plugin, which will only match specified references with these candidates, but not with any other entities.

The matching request processing will be faster in comparison to the default way because the plugin will not spend time to separate faces, which are linked to list from all faces, which are store in the database.

- Matching requests where all candidates are events and specify only one filter — “event_ids” and it is required to match only by bodies.

In this case, it is possible to duplicate all event_id and its body descriptors to other data storage than the default data storage and create a matching plugin, that will match specified references with these candidates, but not with any other entities.

The matching request processing will be faster in comparison to the default way because the plugin will not spend time to separate events with bodies from all events and overlook filters.

You can use **built-in matching plugins** or create **user matching plugins**.

There are three examples of built-in matching plugins available:

- The “Thin event” plugin, which is used for fast matching the faces with the simplified events.
- The “Thin face” plugin, which is used for fast matching the faces with the simplified faces.
- The “Cached Matcher” plugin, which is used for fast matching the faces from large lists.

Thin event

The plugin uses its own table in the “luna_events” database.

Below are a few features that speed up the matching using the plugin compared to the default method:

- The “Thin event” database contains fewer data fields.
- The “Thin event” database stores “event_id” and face “descriptor” in one table.
- The “Thin event” database stores “age”, “gender” and some other filters in one table.

Thin face

The plugin uses its own table in the database “luna_faces” with three mandatory columns (“face_id”, “descriptor”, “descriptor_version”) and a number of additional ones that can be configured: “account_id”, “lists”, “create_time”, “external_id”, “user_data”, “event_id”, “avatar”.

Cached Matcher

Matching faces by list is a time-consuming process, so the following methods are implemented using the plugin to improve performance:

- Separate service “LUNA-CACHED-MATCHER” is used to match candidates and references.
- Data for the candidate (“list_id”, “face_id”, “descriptor”) is stored in the memory cache. This provides quick access to data.
- Data is divided horizontally into segments (the “LUNA-CACHED-MATCHER” service is used as a segment), which provides a quick search for matching results.

See the detailed description of the built-in plugins and instructions for writing user plugins in the [developer manual](#).

7.13.3.1 General plugin processing pipeline

Each matching request is presented in the form of all possible combinations of candidates and references, then each such combination is processed as a separate sub-request as follows (further sub-request means combination of reference and candidates):

- Get the sub-request matching cost (see [“Matching cost”](#)).
- Choose the way for the sub-request processing using the lowest estimated matching cost: matching plugin or Python Matcher service.

- If in the previous step Python Matcher service was selected, it will process sub-request, returns the response to the Python Matcher Proxy service.
- If in the previous step matching plugin was selected, it will process sub-request. If sub-request was successfully processed, the response returns to the Python Matcher Proxy service. If a sub-request was not successfully processed, it will try to process by Python Matcher service.
- If the request was successfully processed by matching plugin and plugin does not have access to all [matching targets](#) which specified in sub-request, then Python Matcher Proxy service will enrich data before next step, see matching targets for details.
- The Python Matcher Proxy service collects results from all sub-requests, sorts them in the right order, and replies to the user.

7.13.3.2 Matching cost

Matching cost is a float numeric expression of matching request process complexity using a plugin. Matching cost is necessary to choose the best way to process a matching request: Python Matcher service or one or more plugins.

The matching cost value for the Python-Matcher service is 100. If there are several plugins, then the matching cost value will be calculated for each plugin. The matching plugin with the lowest matching cost will be used if its matching cost is lower than the Python Matcher matching cost. All requests with matching costs greater than 100 will be processed in the Python Matcher service. If there are no plugins, Python Matcher will be used for the request processing.

7.13.3.3 Matching targets

The Python Matcher service has access to all data of matching entities, so it can process matching requests with all targets. Matching plugins may not have access to data, which is specified in request targets. In this case, Python Matcher Proxy service will enrich response of plugin with missing targets data, e.g.:

- Matching response contains next targets: `face_id`, `user_data` and `similarity` and the chosen matching plugin does not have access to `user_data` field:
 - Matching plugin match reference with specified `face_ids` and return the matching response to the Python Matcher Proxy, which contains only pairs of `face_id` and `similarity`.
 - For every match candidate in result, Python Matcher Proxy service will get `user_data` from the main database by `face_id` and merge `face_id` and `similarity` with `user_data`.
 - Return a prepared response with specified targets and `face_id` as target to the user. This mechanics requires that plugin must supports corresponding entity ID as target. If plugin does not support the entity ID as target such request will not sent to this plugin.

- Matching response contains next targets: age, gender (all candidates are events' faces) and the chosen matching plugin have access only to event_id, descriptor, and age fields:
 - Matching plugin match reference and return the matching response to the Python Matcher Proxy, which contains only pairs of event_id, age and similarity.
 - For every match candidate in result, Python Matcher Proxy service will get gender from the main database by event_id and merge event_id with gender, also after that it drops non-required event_id and similarity from the response.
 - Return a prepared response with specified targets and event_id as target to the user. This mechanics requires that plugin must supports corresponding entity ID as target. If plugin does not support the entity ID as target such request will not sent to this plugin.

The workflow of matching plugins is shown below:

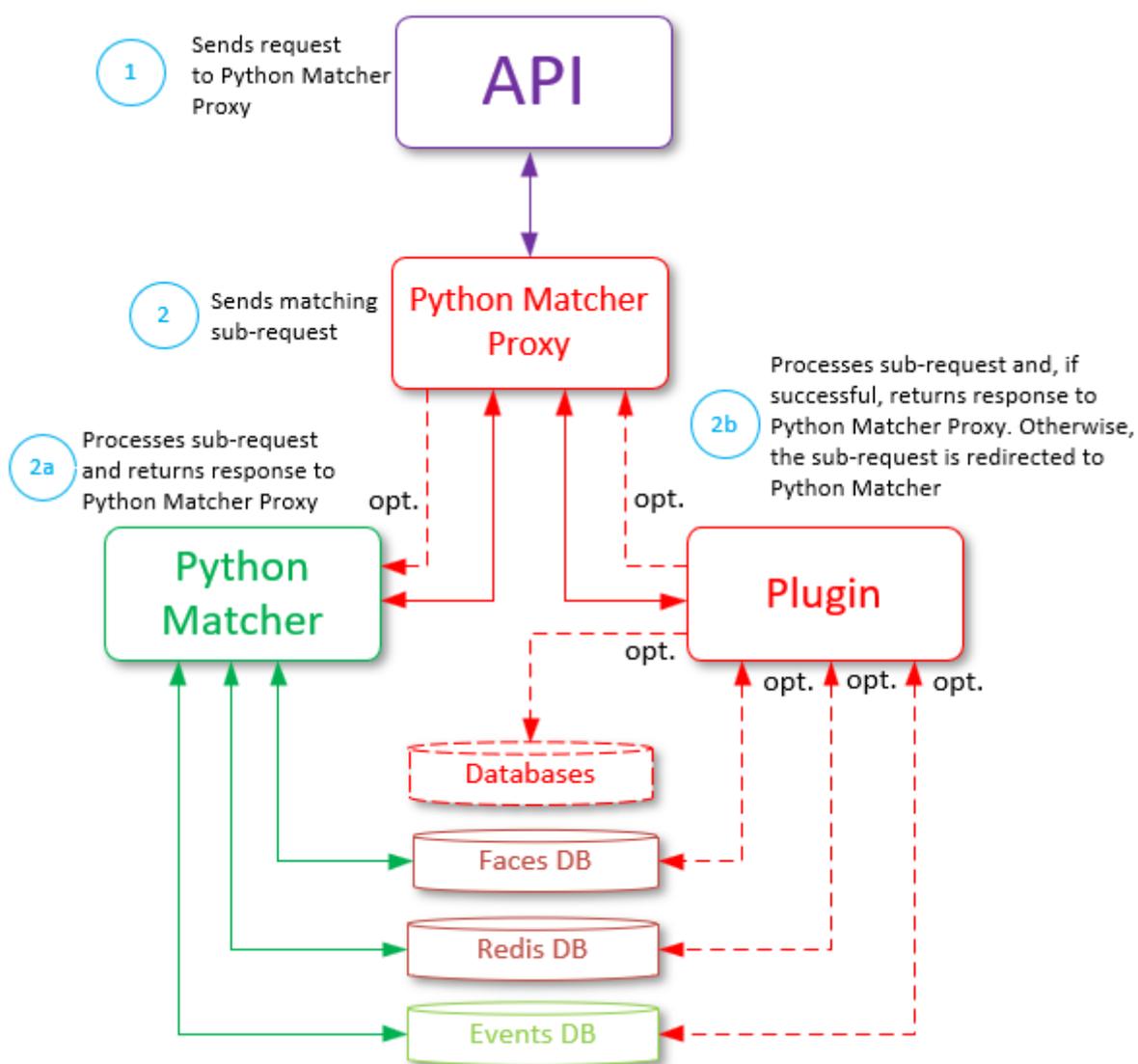


Figure 59: Matching plugin workflow

7.13.3.4 Plugin data source

To speed up request processing, each matching plugin may use a separated data source instead of the default one (Events DB, Faces DB, or Attributes DB (see the “Database description” section)), for example, use a separate database, a new table in the existing database, in-memory cache, etc.

For more information about matching plugins, see the [developer manual](#).

7.13.4 Plugins usage

7.13.4.1 Adding plugins to the directory manually

This way can be used when the plugin does not require any additional dependencies that are not provided in the service Docker container.

There are two steps required to use a plugin with the service in Docker container:

- Add the plugin file to the container.
- Specify the plugin usage in the container configurations.

When starting the container, you need to forward the plugin file to the folder with plugins of specific service. For example, for the Remote SDK service it will be the `/srv/luna_remote_sdk/plugins` folder.

This can be done in any convenient way. For example, you can mount the folder with plugins to the required service directory during service launching (see service launching commands in the installation manual):

You should add the following volume if all the required plugins for the service are stored in the `"/var/lib/luna/remote_sdk/plugins"` directory:

```
-v /var/lib/luna/remote_sdk/plugins:/srv/luna_remote_sdk/plugins/ \
```

The command is given for the case of manual service launching.

Next, you should add the filename(s) to the `"LUNA_REMOTE_SDK_ACTIVE_PLUGINS"` configuration in the Configurator service.

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain filenames without extension (.py).

After completing these steps, LP will automatically use the plugin(s).

More information about plugins for a specific service can be found in the API development manual.

7.13.4.2 Building new Docker container with plugin

This way can be used when additional dependencies are required for the plugin utilization or when the container with the plugin is required for the production usage.

You should create your docker container based on the basic service container.

Add “Dockerfile” with the following structure to your CI:

```
FROM dockerhub.visionlabs.ru/luna/luna-remote-sdk:v.0.6.0
USER root
...
USER luna
```

FROM should include the address to the basic service container that will be used. USER root — change the privileges to the root user to perform the following actions. Then the commands for adding the plugin and its dependencies should be listed. They are not given in this manual. Check the Docker documentation. USER luna — after all the commands are executed, the user should be changed back to “luna”.

Add the plugin filename to the “LUNA_REMOTE_SDK_ACTIVE_PLUGINS” configuration in the Configurator service.

You can:

- Update settings manually in the Configurator service as described above.
- Create a dump file with LP plugin settings and add them to the Configurator service after its launch.

An example of the dump file with the Remote SDK plugin settings is given below.

```
{
  "settings": [
    {
      "value": [
        "plugin_1",
        "plugin_2",
        "plugin_3"
      ],
      "description": "list active plugins",
      "name": "LUNA_REMOTE_SDK_ACTIVE_PLUGINS",
      "tags": []
    },
  ]
}
```

Then the file is applied using the following command. For example the file is stored in “/var/lib/luna/”. The dump filename is “luna_remote_sdk_plugin.json”.

```
docker run \
-v /var/lib/luna/luna_remote_sdk_plugin.json:/srv/luna_configurator/
used_limitations/luna_remote_sdk_plugin.json \
```

```
--network=host \  
--rm \  
--entrypoint=python3 \  
dockerhub.visionlabs.ru/luna/luna-configurator:v.2.2.18 \  
./base_scripts/db_create.py --dump-file /srv/luna_configurator/  
used_limitations/luna_remote_sdk_plugin.json
```

7.14 Monitoring

There are several monitoring methods in the LUNA PLATFORM:

- [Send data to InfluxDB](#) (enabled by default)
- [Export of metrics in Prometheus format](#) using the `/metrics` resource (disabled by default)

7.14.1 Send data to InfluxDB

Starting with version 5.5.0, LUNA PLATFORM provides a possibility to use InfluxDB of version 2.

If necessary, you can migrate from version 1 to version 2 using the built-in tools. See the [Influx documentation](#).

To work with InfluxDB, you need to register with a username and password and specify the bucket name, organization name and token. All this data is set when starting the InfluxDB container using environment variables.

To use monitoring, it is necessary to set exactly the same data for the fields `bucket`, `organization`, `token` in the settings of each service that was specified when launching the InfluxDB container. So, for example, if the following settings were used when starting the InfluxDB container...:

```
-e DOCKER_INFLUXDB_INIT_BUCKET=luna_monitoring \  
-e DOCKER_INFLUXDB_INIT_USERNAME=luna \  
-e DOCKER_INFLUXDB_INIT_PASSWORD=password \  
-e DOCKER_INFLUXDB_INIT_ORG=luna \  
-e DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=kofqt4Pfqn6o \  

```

... then the following parameters should be specified in the settings of each service:

```
"influxdb": {  
  "organization": "luna",  
  "token": "kofqt4Pfqn6o",  
  "bucket": "luna_monitoring",  
}
```

By default, the values for the container environment variables and the values of the service settings are the same. This means that if the LUNA PLATFORM is launched using the installation manual or the Docker Compose script, monitoring will be automatically enabled.

Login and password are used to access the InfluxDB user interface.

It is also possible to launch InfluxDB on a separate server. The address of the server with InfluxDB must be specified in the `host` and `port` parameters in the service settings.

See other settings for InfluxDB on the example of the API service in the section [“INFLUX_-MONITORING”](#).

7.14.1.1 Data being sent

There are two types of events that are monitored: “request” (all requests) and “error” (failed requests only).

Every event is a point in the time series. For the API service, the point is represented using the following data:

- Series name (requests or errors)
- Timestamp of the request start
- Tags
- Fields

For other services, the set of event types may differ. For example, the Remote SDK service also collects data on SDK usage, estimations, and licensing.

The tag is an indexed data in storage. It is represented as a dictionary, where:

- Keys — String tag names.
- Values — String, integer or float.

The field is a non-indexed data in storage. It is represented as a dictionary, where:

- Keys — String field names.
- Values — String, integer or float.

Below is an example of tags and fields for the API service. These tags are unique for each service. You can find information about monitoring a specific service in the relevant documentation:

- [“API”](#)
- [“Licenses”](#)
- [“Configurator”](#)
- [“Image Store”](#)
- [“Faces”](#)
- [“Admin”](#)
- [“Backport 3”](#)
- [“Backport 4”](#)
- [“Events”](#)
- [“Remote SDK”](#)
- [“Handlers”](#)
- [“Python Matcher”](#)
- [“Sender”](#)
- [“Tasks”](#)

- “Lambda”

Saving data for requests is triggered on every request. Each point contains data about the corresponding request (execution time and etc.).

- Tags

Tag name	Description
service	Always “luna-api”.
account_id	Account ID or none.
route	Concatenation of a request method and a request resource (POST:/extractor).
status_code	HTTP status code of response .

- Fields

Fields	Description
request_id	Request ID.
execution_time	Request execution time.

Saving data for errors is triggered when a request fails. Each point contains “error_code” of LUNA error.

- Tags

Tag name	Description
request_id	Always “luna-api”.
account_id	Account ID or none.
route	Concatenation of a request method and a request resource (POST:/extractor).
status_code	HTTP status code of response .
error_code	LUNA PLATFORM error code.

- Fields

Fields	Description
request_id	Request ID.

Every handler can add additional tags or fields. For example, handler of `"/handlers/{handler_id}/events"` resource adds tag `handler_id`.

7.14.1.2 View monitoring data

You can use the InfluxDB GUI to view monitoring data.

- Go to the InfluxDB GUI `<server_ip>:<influx_port>`. The default port is 8086. The default login data is `luna/password`.
- Select the “Explore” tab.
- Select a way to display information in the drop-down list (graph, histogram, table, etc.).
- Select a bucket at the bottom of the page. By default — `luna_monitoring`.
- Filter the necessary data.
- Click “Submit”.

InfluxDB version 2 also enables you to visualize monitoring data using the [“LUNA Dashboards \(Grafana\)”](#) tool.

7.14.1.3 Requests and estimations statistics gathering

LUNA PLATFORM gathers the number of completed requests and estimates per month based on monitoring data if statistics gathering is enabled. Statistics gathering works only when monitoring is enabled and InfluxDB of version 2.0.8 and later is installed.

To get statistics, use the `"/luna_sys_info"` resource or go to the Admin service GUI to the “help” tab and click “Get LUNA PLATFORM system info”. The necessary information is contained in the “stats” section.

This section contains two fields — “`estimators_stats`” and “`routes_stats`”.

The first field contains a list of performed estimations. Three fields are displayed for each estimation:

- “name” — Name of estimation performed (for example, `estimate_emotions`).
- “count” — Total number of completed estimations of the same type.
- “month” — Month for which gathering was made (for example, `2021-09`).

The second field contains a list of services to which requests were made. Five fields are displayed for each service:

- “service” — Name of service (for example, `luna-api`).
- “route” — Method and request (for example, `GET:/version`).
- “month” — Month for which gathering was made.
- “errors” — Number of requests performed with a specific error (for example, `[{ "count": 1, "error_code": "12012" }]`).

- “request_stats” — Number of successful requests (for example, [{ "count": 1, "status_code": "200"}]).

The information is impersonal and contains only quantitative data.

Statistics are gathered in InfluxDB based on data from the “luna_monitoring” bucket and stored in the “luna_monitoring_aggregated” bucket. The buckets are created in InfluxDB. Do not delete data from this bucket, otherwise, it will be impossible to get statistics.

Statistics are gathered once a day, so they are not displayed immediately after the LP is launched.

Tasks for gathering statistics can be found in the InfluxDB GUI on the “Tasks” tab. There you can manually start their performing.

To enable this functionality, run the `python3 ./base_scripts/influx2_cli.py create_usage_task --luna-config http://127.0.0.1:5070/1` command after starting the Admin service (see the installation manual). The command automatically creates the necessary bucket “luna_monitoring_aggregated”. If this command is not performed, the response “/luna_sys_info” will not display statistics.

If necessary, you can disable statistics gathering by deleting or disabling the corresponding tasks on the “Tasks” tab in the InfluxDB GUI.

7.14.2 Export metrics in Prometheus format

Each LUNA PLATFORM service can collect and save metrics in Prometheus format in the form of time series data that can be used to track the behavior of the service. Metrics can be integrated into the Prometheus monitoring system to track performance. See [Prometheus official documentation](#) for more information.

By default, the collection of metrics is disabled. The collection of metrics is enabled in the “LUNA_SERVICE_METRICS” section.

Note that all metric data is reset when the service is shut down.

7.14.2.1 Type of metrics

Two types of metrics are available:

- **Counters**, which increase with each event.
- **Cumulative histograms**, which are used to measure the distribution of duration or size of events.

A cumulative histogram is a mapping that counts the cumulative number of observations in all of the bins up to the specified bin. See description in [Wikipedia](#).

The following metrics of type **counters** are available:

- `request_count_total` — Total number of requests
- `errors_count_total` — Total number of errors

Each of them has at least two labels for sorting:

- `status_code` (or `error_code` for error metrics)
- `path` — Path consisting of a request method and an endpoint route.

Labels are key pairs consisting of a name and a value that are assigned to metrics.

If necessary, you can add custom label types by specifying the pair `tag_name=tag_value` in the “extra_labels” parameter.

Note that the pair `tag_name=tag_value` will be added to each metric of the LUNA PLATFORM service.

A special manager distributes all requests passing through the service among the counters using these tags. This ensures that two successful requests sent to different endpoints or to the same endpoint, but with different status codes, will be delivered to different metrics.

Unsuccessful requests are distributed according to the metrics `request_count_total` and `request_errors_total`.

The requests metric of **cumulative histogram** type tracks the duration of requests to the service. The following intervals (bucket) are defined for the histogram, in which the measurements fall:

- 0.0001
- 0.00025
- 0.0005
- 0.001
- 0.0025
- 0.005
- 0.01
- 0.025
- 0.05
- 0.075
- 0.1
- 0.25
- 0.5
- 0.75
- 1.0
- 2.5
- 5.0
- 7.5
- 10.0
- Inf

In this way the range of request times can be broken down into several intervals, ranging from very fast

requests (0.0001 seconds) to very long requests (Inf - infinity). Histograms also have labels to categorize the data, such as `status_code` for the status of a request or `route` to indicate the route of a request.

Examples

If you send one request to the `/healthcheck` resource, followed by three requests to the `/docs/spec` resource, one of which will be redirected (response status code 301), then when executing the request to the `/metrics` resource, the following result will be displayed in the response body:

```
# HELP request_count_total Counter of requests
# TYPE request_count_total counter
request_count_total{path="GET:/docs/spec",status_code="200"} 2.0
request_count_total{path="GET:/docs/spec",status_code="301"} 1.0
request_count_total{path="GET:/healthcheck",status_code="200"} 1.0
```

If you send one invalid POST request to the `/handlers` resource, then when executing the request to the `/metrics` resource, the following result will be displayed in the response body:

```
# HELP request_count_total Counter of requests
# TYPE request_count_total counter
request_count_total{path="POST:/handlers",status_code="401"} 1.0
# HELP request_errors_total Counter of request errors
# TYPE request_errors_total counter
request_errors_total{error_code="12010",path="POST:/handlers"} 1.0
# HELP requests Histogram of request time metrics
# TYPE requests histogram
requests_sum{route="GET:/docs/spec",status_code="200"} 0.003174567842297907
requests_bucket{le="0.0001",route="GET:/docs/spec",status_code="200"} 0.0
requests_bucket{le="0.00025",route="GET:/docs/spec",status_code="200"} 0.0
requests_bucket{le="0.0005",route="GET:/docs/spec",status_code="200"} 0.0
requests_bucket{le="0.001",route="GET:/docs/spec",status_code="200"} 1.0
...
requests_count{route="GET:/docs/spec",status_code="200"} 2.0
requests_sum{route="GET:/docs/spec",status_code="301"} 0.002381476051209132
```

7.14.3 Configuring metrics collection for Prometheus

Prometheus must be configured to collect LUNA PLATFORM metrics.

Example [Prometheus configuration](#) for collecting LP service metrics:

```
- job_name: "luna-api"
  static_configs:
```

```

- targets: ["127.0.0.1:5000"]
...
- job_name: "luna-configurator"
  static_configs:
    - targets: ["127.0.0.1:5070"]

```

See the [official documentation](#) for an example of running Prometheus.

Prometheus dashboards have already been created in the [LUNA Dashboards](#) service.

7.14.4 LUNA Dashboards

The LUNA Dashboards tool is intended to visualize monitoring data. LUNA Dashboards based on the Grafana web application create a set of dashboards for analyzing the state of individual services, as well as two summarised dashboards that can be used to evaluate the state of the system.

Use “http://IP_ADDRESS:3000” to go to the Grafana web interface.

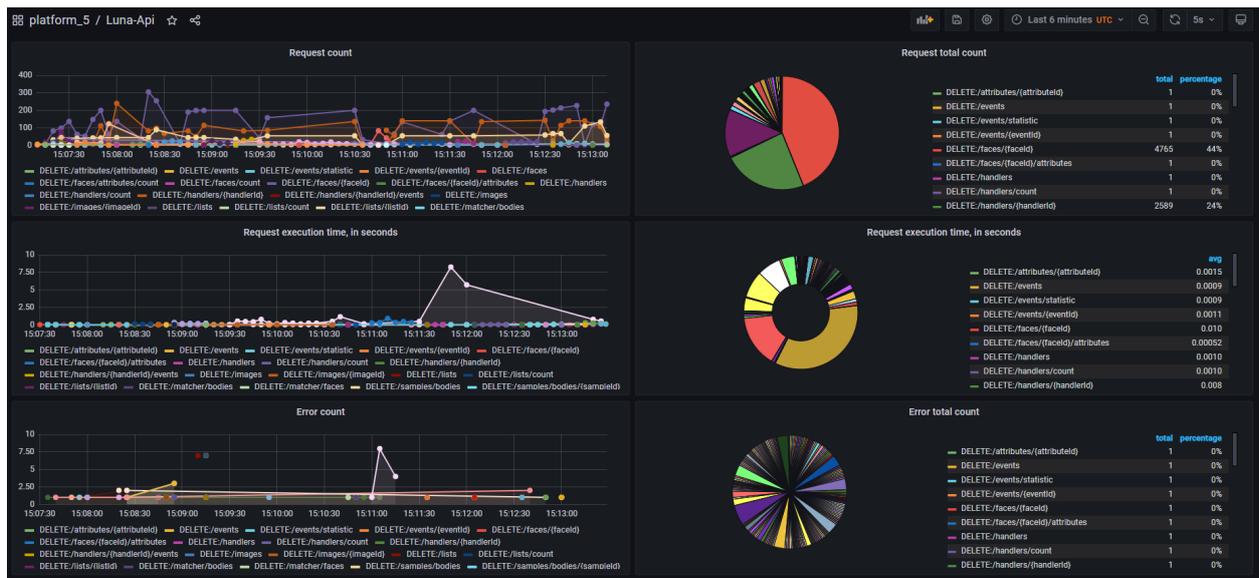


Figure 60: API service dashboard when starting testing

The data source is configured in Grafana, with the help of which it can communicate with the [InfluxDB](#) from which monitoring data is received during the operation of all LP services.



Figure 61: LUNA Dashboards workflow

LUNA Dashboard can be useful:

- To monitor the state of the system.
- For error analysis.
- To get statistics on errors.
- To analyze the load on individual services and on the platform as a whole, load by days of the week, by time of day, etc..
- To analyze statistics of request execution, i.e. what resources account for what proportion of requests for the entire platform.
- To analyze the dynamics of request execution time.
- To evaluate the average value of the execution time of requests for a specific resource.
- To analyze Prometheus metrics.
- To analyze changes in the indicator over time.

After installing the dashboards (see below), the “luna_platform_5” directory becomes available in the Grafana web interface, which contains the following dashboards:

- Luna Platform Heatmap.
- Luna Platform Summary.
- Dashboards for individual services.

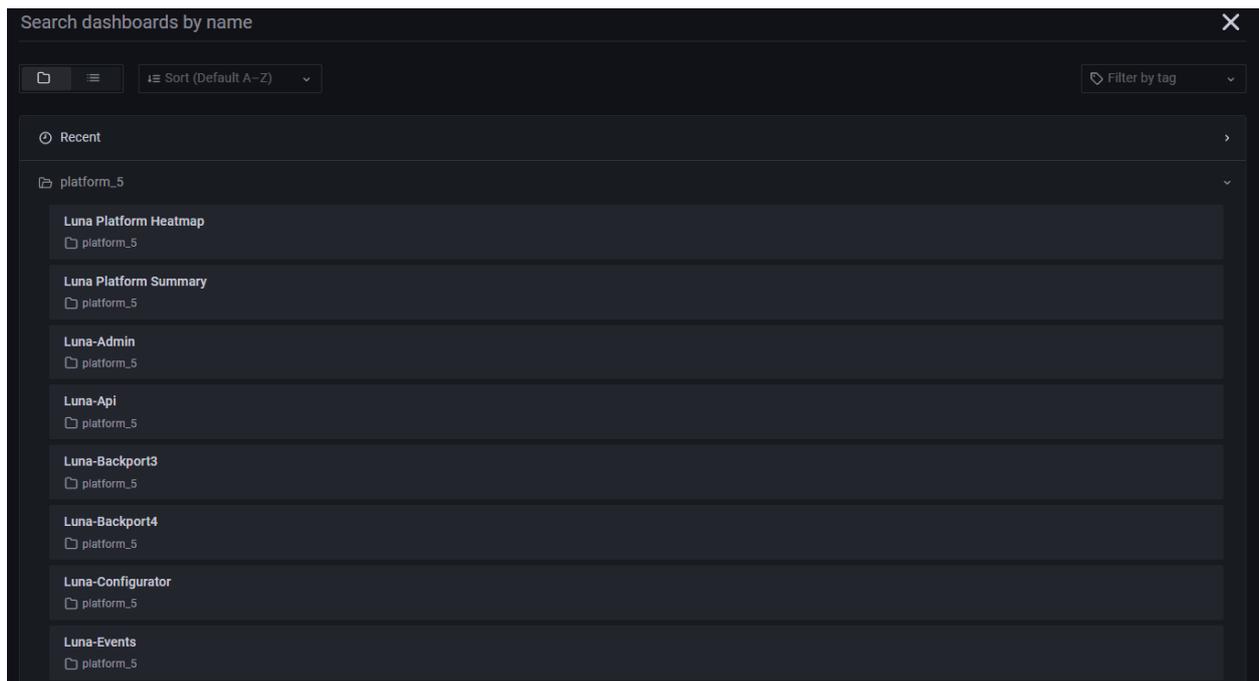


Figure 62: LUNA Dashboards structure

Luna Platform Heatmap enables you to evaluate the load on the system without a specific resource. In statistics, you can evaluate the time of activity for the system at a certain time.

Luna Platform Summary enables you to get statistics on requests for all services in one place, as well as evaluate graphs by RPS (Request Per Seconds).

Dashboards for individual services enables you to get information about requests for individual resources, errors and status codes for each service. In such a dashboard, not load data will be displayed, but artificially generated data at a selected time interval.

The following software is required for Grafana dashboard utilization:

- InfluxDB 2.0 (the currently used version is 2.0.8-alpine)
- Grafana (the currently used version is 8.5.20)

InfluxDB and Grafana are already included in the package. You can use your own Grafana installation or install it manually.

[7.14.4.1 Manual installation of LUNA dashboards](#)

Note: Below are the steps to manually install Grafana dashboards. Dashboards can also be automatically launched using a special command in the section “Monitoring and logs visualization using Grafana” of the installation manual.

Plugin installation

In addition to build-in Grafana plugins, dashboards also use a [piechart plugin](#). Use the new grafana-cli tool to install piechart-panel from the command line:

```
grafana-cli plugins install grafana-piechart-panel
```

If necessary, you can use the archive “grafana-piechart-panel.zip” in “/var/lib/luna/current/extras/utils/”.

A restart is required to apply the plugin:

```
sudo service grafana-server restart
```

Launch dashboards

Dashboards can be launched manually or using a special Grafana container with dashboards called luna-dashboards:

- Launching using a special Grafana container with dashboards is described in the installation manual.
- Manual launch is described further in this document.

Install Grafana. An example of the command is given below:

```
docker run \  
--restart=always \  
--detach=true \  
--network=host \  
--name=grafana \  
-v /etc/localtime:/etc/localtime:ro \  
-e "GF_INSTALL_PLUGINS=grafana-piechart-panel" \  
dockerhub.visionlabs.ru/luna/grafana:8.5.20
```

If necessary, in the environment variable “GF_INSTALL_PLUGINS” you can specify the path to the archive “/var/lib/luna/current/extras/utils/grafana-piechart-panel.zip”.

The scripts for Grafana plugins installation can be found in “/var/lib/luna/current/extras/utils/”.

Install Python version 3.7 or later before launching the following script. The packages are not provided in the distribution package and their installation is not described in this manual.

Go to the luna dashboards directory.

```
cd /var/lib/luna/current/extras/utils/luna-dashboards_linux_rel_v.*
```

Create a virtual environment.

```
python3.7 -m venv venv
```

Activate the virtual environment.

```
source venv/bin/activate
```

Install luna dashboards file.

```
pip install luna_dashboards-*-py3-none-any.whl
```

Go to the following directory.

```
cd luna_dashboards
```

The “luna_dashboards” folder contains the configuration file “config.conf”, which includes the settings for Grafana, InfluxDB and monitoring periods. By default, the file already includes the default settings, but you can change the settings use “vi config.conf”.

Run the following script to create dashboards.

```
python create_dashboards.py
```

Deactivate virtual environment.

```
deactivate
```

Use “http://IP_ADDRESS:3000” to go to the Grafana web interface when the Grafana and InfluxDB containers are running.

In the upper left corner, select the “General” button, then expand the “luna_platform_5” folder and select the necessary dashboard.

7.14.5 Grafana Loki

Grafana Loki is a log aggregation system that enables you to flexibly work with LUNA PLATFORM logs in Grafana.

With Grafana Loki, you can perform the following tasks:

- Collecting LUNA PLATFORM logs.
- Search by LUNA PLATFORM logs.
- Visualization of LUNA PLATFORM logs.
- Extraction of numeric metrics from LUNA PLATFORM logs.
- Other.

See the detailed information about the capabilities of Grafana Loki in the official documentation: <https://grafana.com/oss/loki/>.

Grafana Loki is included in the LUNA PLATFORM distribution.

To launch Grafana Loki, the following software is required:

- Launched Grafana with configured Loki data source in Grafana (see “LUNA Dashboards” section).
- Launched Promtail log delivery agent (see “Promtail” section below).

Thus, the following chain of actions is performed in Grafana to work with LUNA PLATFORM logs:

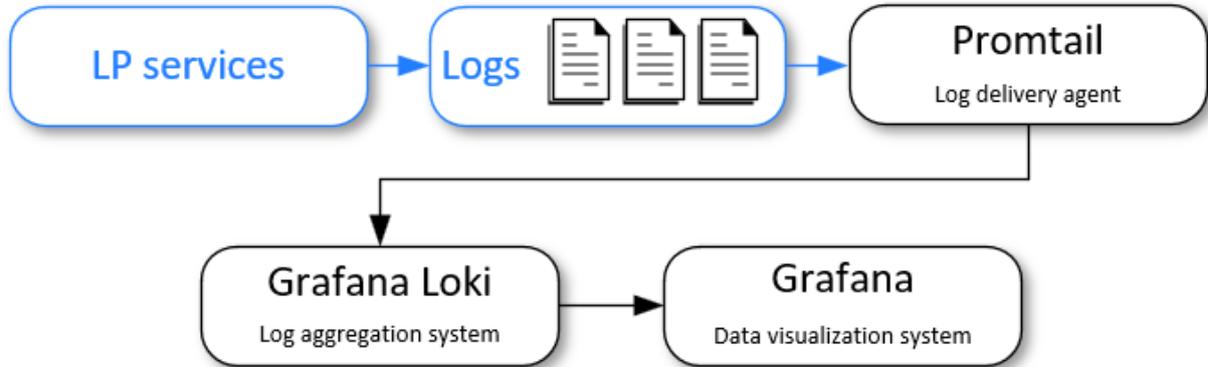


Figure 63: Grafana Loki workflow

Commands for launching Grafana and Promtail are given in the installation manual. The Grafana container from the LUNA PLATFORM distribution already has a Loki data source configured.

You can also launch Grafana and Promtail by executing an additional Docker Compose script after the main Docker Compose script (see “Deployment using Docker Compose” document).

7.14.5.1 Promtail

A specially configured Promtail agent is used to deliver LUNA PLATFORM logs to Grafana Loki. Like Grafana Loki, the Promtail agent is included in the LUNA PLATFORM distribution.

A configured Promtail settings file is available in the LUNA PLATFORM distribution, which provides the ability to filter by the following tags:

- LP logging level
- LP services
- URI
- LP status codes

Some additional labels (for example, LP service version) can also be specified using the `client.external-labels` argument in the Promtail launch command.

A [derived field](#) is also configured in Grafana Loki to search for a specific request ID in the logs.

7.15 Databases

7.15.1 Manual creation of services databases in PostgreSQL

This section describes commands required for configuring external PostgreSQL for working with LP services. External means that you already have working DB and want to use it with LP.

Commands for Oracle are not listed in this documentation.

It is necessary to specify an external database in LP service configurations.

For Faces and Events services it is necessary to add additional VLMatch functions to the used database. The VLMatch library must first be compiled and migrated to PostgreSQL, and then the VLMatch function must be added to the Events and Faces databases. See [“Build VLMatch for PostgreSQL”](#), [“Add VLMatch function for Faces DB in PostgreSQL”](#) and [“Add VLMatch function for Events DB in PostgreSQL”](#) for details.

7.15.1.1 PostgreSQL user creation

Create a database user.

```
runuser -u postgres -- psql -c 'create role luna;'
```

Assign a password to the user.

```
runuser -u postgres -- psql -c "ALTER USER luna WITH PASSWORD 'luna';"
```

7.15.1.2 Configurator DB creation

It is assumed that the DB user is already created.

Create the database for the Configurator service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_configurator;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_configurator TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.1.3 Accounts DB creation

It is assumed that the DB user is already created.

Create the database for the Accounts service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_accounts;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_accounts TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.1.4 Handlers DB creation

It is assumed that the DB user is already created.

Create the database for the Handlers service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_handlers;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_handlers TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.1.5 Backport 3 DB creation

It is assumed that the DB user is already created.

Create the database for the Backport 3 service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_backport3;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_backport3 TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.1.6 Faces DB creation

It is assumed that the DB user is already created.

Create the database for the Faces service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_faces;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE luna_faces  
TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.1.7 Events DB creation

It is assumed that the DB user is already created.

Create the database for the Events service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_events;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE luna_events  
TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.1.8 Tasks DB creation

It is assumed that the DB user is already created.

Create the database for the Tasks service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_tasks;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE luna_tasks  
TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.1.9 Lambda DB creation

It is assumed that the DB user is already created.

Create the database for the Lambda service.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_lambda;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE luna_lambda  
TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

7.15.2 Build VLMatch for PostgreSQL

Note: The following instruction describes installation for PostgreSQL 16.

You can find all the required files for the VLMatch user-defined extension (UDx) compilation in the following directory:

```
/var/lib/luna/current/extras/VLMatch/postgres/
```

The following instruction describes installation for PostgreSQL 16.

For VLMatch UDx function compilation one needs to:

- Install RPM repository:

```
dnf install -y https://download.postgresql.org/pub/repos/yum/repopms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

- Install PostgreSQL:

```
dnf install postgresql16-server
```

- Install the development environment:

```
dnf install postgresql16-devel
```

- Install the gcc package:

```
dnf install gcc-c++
```

- Install CMAKE. The version 3.5 or higher is required.
- Open the make.sh script using a text editor. It includes paths to the currently used PostgreSQL version. Change the following values (if necessary):

SDK_HOME specifies the path to PostgreSQL home directory. Default value: /usr/include/postgresql/16/server;

LIB_ROOT specifies the path to PostgreSQL library root directory. Default value: /usr/lib/postgresql/16/lib.

- Go to the make.sh script directory and run it:

```
cd /var/lib/luna/current/extras/VLMatch/postgres/
```

```
chmod +x make.sh
```

```
./make.sh
```

7.15.3 Add VLMatch function for Faces DB in PostgreSQL

The Faces service requires an additional VLMatch function to be added to the database used. LUNA PLATFORM cannot perform descriptor matching calculations without this function.

The VLMatch library is compiled for a specific version of the database.

Do not use a library created for a different version of the database. For example, a library created for PostgreSQL version 12 cannot be used for PostgreSQL version 16.

This section describes how to create a function for PostgreSQL. The VLMatch library must be compiled and ported to PostgreSQL. See [“Build VLMatch for PostgreSQL”](#).

7.15.3.1 Add VLMatch function to Faces database

The VLMatch function should be applied to the PostgreSQL DB.

- Define the function inside the Faces database.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_faces -c "CREATE FUNCTION  
VLMatch(bytea, bytea, int) RETURNS float8 AS 'VLMatchSource.so', 'VLMatch'  
' LANGUAGE C PARALLEL SAFE;"
```

- Test function by sending re following request to the service database.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_faces -c "SELECT VLMatch('\  
x1234567890123456789012345678901234567890123456789012345678901234'  
::bytea, '\x012345678901234567890123456789012345678901234567890123'  
::bytea, 32);" 
```

The result returned by the database must be “0.4765625”.

7.15.4 Add VLMatch function for Events DB in PostgreSQL

The Events service requires an additional VLMatch function to be added to the database used. LUNA PLATFORM cannot perform descriptor matching calculations without this function.

The VLMatch library is compiled for a specific version of the database.

Do not use a library created for a different version of the database. For example, a library created for PostgreSQL version 12 cannot be used for PostgreSQL version 16.

This section describes how to create a function for PostgreSQL. The VLMatch library must be compiled and ported to PostgreSQL. See [“Build VLMatch for PostgreSQL”](#).

7.15.4.1 Add VLMatch function to Events database

The VLMatch function should be applied to the PostgreSQL DB.

Define the function inside the Events database.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_events -c "CREATE FUNCTION
  VLMatch(bytea, bytea, int) RETURNS float8 AS 'VLMatchSource.so', 'VLMatch
  ' LANGUAGE C PARALLEL SAFE;"
```

Test function within call.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_events -c "SELECT VLMatch('\
  x1234567890123456789012345678901234567890123456789012345678901234'::bytea
  , '\x0123456789012345678901234567890123456789012345678901234567890123'::
  bytea, 32);" 
```

The result returned by the database must be “0.4765625”.

7.15.5 VLMatch for Oracle

Note: The following instruction describes installation for Oracle 21c.

You can find all the required files for the VLMatch user-defined extension (UDx) compilation in the following directory:

```
/var/lib/luna/current/extras/VLMatch/oracle
```

For VLMatch UDx function compilation one needs to:

- Install required environment, see [requirements](#):

```
sudo yum install gcc g++
```

- Change SDK_HOME variable — oracle sdk root (default is \$ORACLE_HOME/bin, check \$ORACLE_HOME environment variable is set) in the makefile:

```
vi /var/lib/luna/current/extras/VLMatch/oracle/make.sh
```

- Open the directory and run the file “make.sh”.

```
cd /var/lib/luna/current/extras/VLMatch/oracle
```

```
chmod +x make.sh
```

```
./make.sh
```

- Define the library and the function inside the database (from database console):

```
CREATE OR REPLACE LIBRARY VLMatchSource AS '$ORACLE_HOME/bin/VLMatchSource.  
so';  
CREATE OR REPLACE FUNCTION VLMatch(descriptorFst IN RAW, descriptorSnd IN  
RAW, length IN BINARY_INTEGER)  
RETURN BINARY_FLOAT  
AS  
LANGUAGE C  
LIBRARY VLMatchSource  
NAME "VLMatch"  
PARAMETERS (descriptorFst BY REFERENCE, descriptorSnd BY REFERENCE,  
length UNSIGNED SHORT, RETURN FLOAT);
```

- Test function within call (from database console):

```
SELECT VLMatch(HEXTORAW('1234567890123456789012345678901234567890123456789012345678901234'),  
HEXTORAW('0123456789012345678901234567890123456789012345678901234567890123'), 32)  
FROM DUAL;
```

The result returned by the database must be “0.4765625”.

7.16 Collect information for technical support

For efficient and prompt problem solving, VisionLabs technical support needs to provide LUNA PLATFORM service logs and additional information about the status of third-party services, license status, LUNA PLATFORM settings, etc.

Collect the data described below and send it to VisionLabs specialists.

7.16.1 Collect services logs

There are two ways to output logs in LUNA PLATFORM:

- Standard log output (stdout).
- Log output to a file.

The log output settings are specified in the settings of each service in the <SERVICE_NAME>_LOGGER section.

By default, logs are output only to standard output.

For more information about the LUNA PLATFORM logging system, see “Logging information” in the administrator manual.

Collect logs for all services. For example, you can collect logs for the last 10 minutes for all services using the commands below.

```
docker logs --since 10m luna-licenses > luna-licenses_log.txt
docker logs --since 10m luna-faces > luna-faces_log.txt
docker logs --since 10m luna-image-store > luna-image-store_log.txt
docker logs --since 10m luna-accounts > luna-accounts_log.txt
docker logs --since 10m luna-tasks > luna-tasks_log.txt
docker logs --since 10m luna-events > luna-events_log.txt
docker logs --since 10m luna-sender > luna-sender_log.txt
docker logs --since 10m luna-admin > luna-admin_log.txt
docker logs --since 10m luna-remote-sdk > luna-remote-sdk_log.txt
docker logs --since 10m luna-handlers > luna-handlers_log.txt
docker logs --since 10m luna-lambda > luna-lambda_log.txt
docker logs --since 10m luna-python-matcher > luna-python-matcher_log.txt
docker logs --since 10m luna-backport3 > luna-backport3_log.txt
docker logs --since 10m luna-backport4 > luna-backport4_log.txt
```

7.16.2 Collect additional information

The following additional information must be collected:

- LUNA PLATFORM version.

The LUNA PLATFORM version can be found in the name of the archive with the delivery set. You can also find out the current version by going to the `http://your_server_ip_address:5000/version` page in your browser.

- License status depending on the vendor:

- HASP — Information from `http://your_server_ip_address:1947/_int/features.html` and `http://your_server_ip_address:1947/_int/devices.html` pages
- Guardant — Information from pages `http://your_server_ip_address:3189/#/dongles/list` and `http://your_server_ip_address:3189/#/sessions`.

- Actual settings of LUNA PLATFORM.

Up-to-date settings can be obtained by going to the page `http://your_server_ip_address:5010/4/luna_sys_info`, specifying the login and password from the account. The default login and password is `root@visionlabs.ai/root`. After entering the password, a file in json format will be downloaded and should be submitted to technical support.

- Status of third-party services:

- Docker: `systemctl status docker`
- aksusbd: `systemctl status aksusbd`
- grdcontrol: `systemctl status grdcontrol`

- Status of LUNA PLATFORM containers.

You can get a list of all containers using the `docker ps -a` command.

- List of open ports.

You can get a list of open ports using the `ss -ltn` command.

- A list of registries in the Docker configuration that can be connected to without using a secure connection.

You can get a list of registries using the `cat /etc/docker/*` command.

- Firewall rules.

Firewall rules can be obtained using the `iptables-save` command.

- General system information:

- CPU information: `lscpu`
- Memory usage: `free -h`; `lsmem`
- Disk space usage: `df -h`

- Environment and server type.

Specify the environment in which the system is running (test, production) and whether the server is virtual or physical.

8 Recommendations

This section provides recommendations for optimal work with the LUNA PLATFORM.

8.1 Resource optimization

This section provides tips for optimizing system resources when working with the LUNA PLATFORM.

1. Remove unnecessary [policies](#) in handlers enabled by default. For example, saving samples and events is enabled in handlers by default.

If saving is not required, then you need to disable them. Otherwise, it is necessary to monitor and periodically delete outdated data so that the server memory does not fill up.

If estimates that are not required are enabled, then their execution will increase the execution time of each request.

2. [Disable the use of unnecessary services](#).
3. Instead of increasing the number of service instances, increase the number of [workers](#) (with the exception of the Remote SDK service on GPU and Python Matcher).
4. Adjust the number of instances/workers of the Remote SDK and the “num_threads” parameter in the settings of the Remote SDK service when running on the CPU.

Typically, the value of the parameter “num_threads” and the number of instances/workers of the Remote SDK is a multiple of the number of physical cores. I.e. if there are 8 physical cores, it is recommended to use 2 instances of the Remote SDK and “num_threads” = 4 for each instance (2x4=8). Similarly, if there are 24 cores, then you need 4 Remote SDK instances and “num_threads” = 6 for each instance (4x6=24).

It should be remembered that too many instances/workers can negatively affect performance, so you should not run 8 instances/workers with “num_threads” = 1 with 8 cores available. You can divide the number of cores by 6 or 8 to determine the number of instances/workers.

The “num_threads” parameter does not affect the operation on the GPU.

5. When performing [matching](#) explicitly specify the required “target” fields.
For example, if nothing is required from the matching result except “face_id”, then it makes no sense to waste resources on using the remaining fields (default behavior if you do not set the fields in “target”).
Within the handlers, the “match_policy” policy also specifies “targets”, which also need to be configured.
6. With small lists, you can run more instances of Python Matcher by reducing the “thread_count” parameter.

7. With large lists of more than 1-2M, do not do more than one Python Matcher service on one NUMA-node.

8. Correctly specify the limit of candidates and references and do not set more than necessary (see the “PLATFORM_LIMITS” setting in the Python Matcher service settings).

For example, if you need to get only one candidate with the highest similarity in the response, then you do not need to specify that the top-3 candidates should be returned.

9. Set the logging level to “WARNING”.

Note that reducing the logging level may reduce resource consumption, however, in case of problems, this level of logs may not be enough.

10. Use [task schedule](#) to control the launch of the [Garbage collection](#) task.

11. Use “Accept-Encoding” headers with the value “gzip, deflate” to optimize network traffic during API requests when receiving data in JSON format.

12. Decrease the value of the “optimal_batch_size” parameter in the settings of the Remote SDK service when working on the CPU (the values are different for each estimator).

When running on CPU, the “optimal_batch_size” parameter should be less than the “num_threads” value. For example, if “num_threads” = 4, it is recommended to set “optimal_batch_size” <= 4.

As in the case of “num_threads”, the optimal value of “optimal_batch_size” depends on the specific system and the characteristics of the task, so it should be adjusted experimentally.

13. Do not forget to specify “image_type” equal to “1” or “2” when sending samples of the face or body. By default, the value is “0” (raw images).

If you leave the value “0”, then repeated detection of the face in the image is performed, which increases the processing time of the request and may lead to unforeseen problems. For example, when one face was found in the frontend, and two faces were found in the backend, because different versions of detectors or different settings for detecting faces are used.

14. [Run Remote SDK only with certain estimators](#).

If some estimators are not required for the implementation of business logic, then they can be disabled and removed from the container to reduce memory consumption.

15. Read the section “[Resource consumption by services](#)” in order to select optimal servers for services.

8.2 Advanced PostgreSQL setting

PostgreSQL can be configured to interact effectively with LUNA PLATFORM 5. To do this, you need to set certain values for the PostgreSQL settings in the `postgresql.conf` file.

This section does not provide a complete list of all settings with a detailed description. See [official PostgreSQL website](#) for a complete list of settings and their descriptions.

Useful tips for calculating PostgreSQL configuration are described here: https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server.

It is possible to calculate configuration for PostgreSQL based on the maximum performance for a given hardware configuration (see <https://pgtune.leopard.in.ua/>).

8.2.1 Recommended values for settings

Note: The following settings should be changed with caution as manually changing PostgreSQL settings requires experience.

The recommended values of the settings and their description are given below.

`max_connections` = **200** — determines the maximum number of concurrent connections to the database server. The default value is 100.

The default value may be enough for test demonstrations of the LUNA PLATFORM, but for real purposes, the standard value may not be enough and it will need to be calculated.

In the Configurator service, you can set the number of DB connections using the `connection_pool_size` setting located in the `LUNA_<SERVICE_NAME>_DB` sections, where `<SERVICE_NAME>` is the name of the service that has the database. The actual number of connections may be greater than the value of this setting by 1.

If there are too many connections, but not enough active ones, you can use third-party load balancing services, for example, `haproxy` or `pgbouncer`. When using balancing services, it is necessary to take into account some nuances described here: <https://magicstack.github.io/asyncpg/current/faq.html#why-am-i-getting-prepared-statement-errors>.

`maintenance_work_mem` = **2GB** — specifies the maximum amount of memory to be used by maintenance operations.

`shared_buffers` = **0.25...0.5 * RAM (MB)** — determines how much memory Postgres will allocate for caching data. It depends on how often the [matching by database](#) is performed, which indexes, etc.

`effective_io_concurrency` = **100** — sets the number of concurrent disk I/O operations that PostgreSQL expects can be executed simultaneously. Raising this value will increase the number of I/O operations that any individual PostgreSQL session attempts to initiate in parallel.

`max_worker_processes` = **CPU_COUNT** — sets the maximum number of worker processes that the system can support.

`max_parallel_maintenance_workers` = **4** — sets the maximum number of parallel worker processes performing the index creation command (`CREATE INDEX`).

`max_parallel_workers_per_gather` = **4** — sets the maximum number of workers that a query or subquery can be parallelized to.

`max_parallel_workers` = **CPU_COUNT** — sets the maximum number of workers that the system can support for parallel operations.

The following values of settings are related to the function of [matching by database](#) for large tables.

`enable_bitmapscan` = **off** — enables or disables the query planner's use of bitmap-scan plan types. Sometimes it may be necessary when PostgreSQL erroneously determines that bitmapscan is better than the index. It is recommended to change it only if necessary, when it is assumed that the query will use the index, but for unknown reasons does not use it.

`seq_page_cost` = **1** — sets the planner's estimate of the cost of a disk page fetch that is part of a series of sequential fetches.

`random_page_cost` = **1.5** — sets the planner's estimate of the cost of a non-sequentially-fetched disk page.

`parallel_tuple_cost` = **0.1** — sets the approximate cost of transferring one tuple (row) from a parallel worker to another worker.

`parallel_setup_cost` = **5000.0** — sets the approximate cost of running parallel workers.

`max_parallel_workers_per_gather` = **CPU_COUNT/2** * sets the maximum number of workers that a query or subquery can be parallelized to.

`min_parallel_table_scan_size` = **1MB** — sets the minimum amount of table data that should be scanned in order for a parallel scan to be considered.

`min_parallel_index_scan_size` = **8k** — sets the minimum amount of index data for parallel scanning.

9 Sequence diagrams

This chapter describes the general requests to LP and shows interactions between services during the request processing.

More information about the requests can be found in the OpenAPI specification.

9.1 Samples creation diagrams

9.1.1 Sample creation diagram

The request enables detecting the faces in input photos. The photos can be sent in standard image formats (JPEG, PNG, etc.) or in the BASE64 format.

See the “detect faces” request in [OpenAPI specification](#) for details.

Request	Description	Type
detect faces	The request enables to detect faces in incoming images, estimate face parameters, extract samples and save them in Image Store.	POST

The request result contains the parameters of the detected faces and the IDs for all the created *samples*.

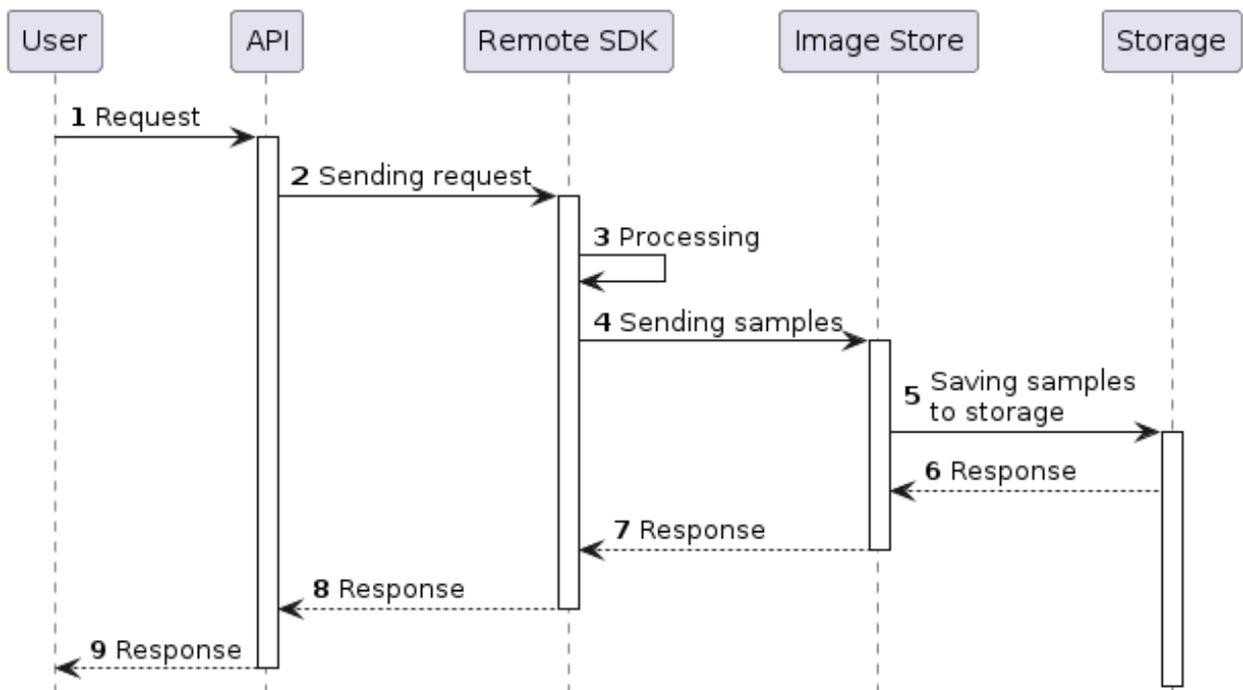


Figure 64: Detect faces request

The general request processing scheme:

1. The face detection request is sent to API.
2. API receives the request, processes it and sends the task to the Remote SDK service.
3. The Remote SDK processes the task according to the specified parameters.
4. The Remote SDK sends received samples to Image Store.

5. Image Store saves the samples to the storage.
6. The storage returns the result of sample saving.
7. Image Store returns the samples IDs and URLs.
8. The Remote SDK sends the samples IDs and the received face parameters to the API service.
9. API generates and sends the response.

9.1.2 Get information about samples and save samples

The following request adds external samples to Image Store.

See the “[save samples](#)” request in [OpenAPI specification](#) for details.

Request	Description	Type
save sample	The external sample is saved in Image Store.	POST

The following requests enables to manage the already existing samples.

See the “[samples](#)” section in [OpenAPI specification](#) for details.

Request	Description	Type
get sample	Receive a sample by its ID.	GET
remove sample	Delete a sample by its ID.	DEL
check to exist sample	Check existence of the sample with the specified ID.	HEAD

All the requests above are processed by the Image Store service.

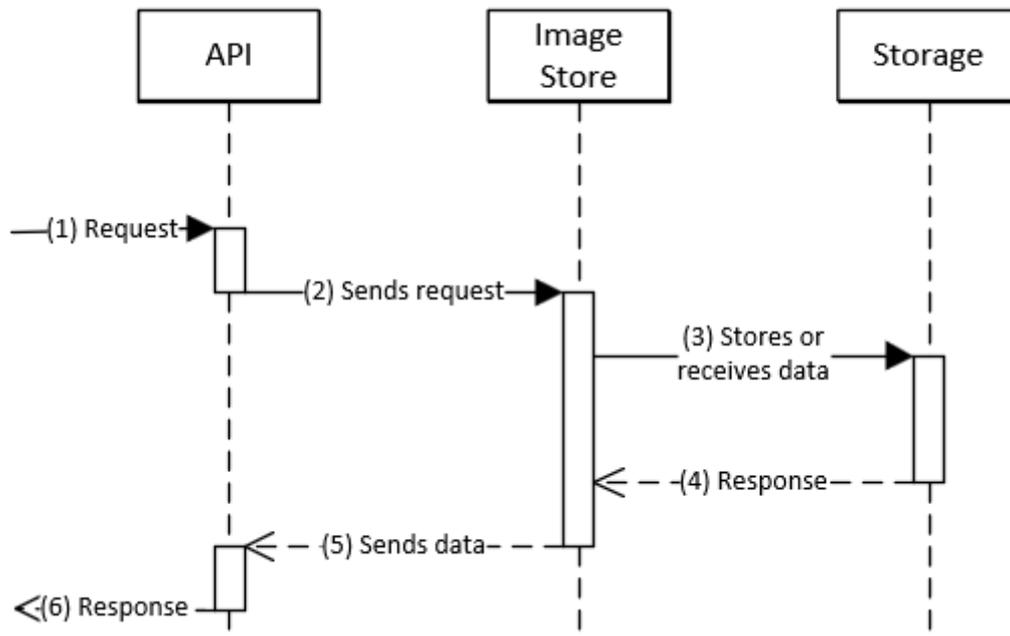


Figure 65: Image Store request

The requests processing scheme:

1. A request is sent to the API service.
2. API sends the request to Image Store.
3. Image Store performs the required actions with the storage.
4. The storage returns the required data.
5. Image Store returns the result to API.
6. API generates and sends the response.

More information about the requests can be found in the OpenAPI specification document in the “Samples” section.

9.2 Attributes diagrams

9.2.1 Temporary attributes extraction diagram

Handlers receives samples from Image Store and extracts information from them.

See the “[extract attributes](#)” request in [OpenAPI specification](#) for details.

You should specify the array of sample IDs.

Request	Description	Type
extract attributes	Extracts the following attributes: descriptors and basic attributes (age, gender, ethnicity).	POST

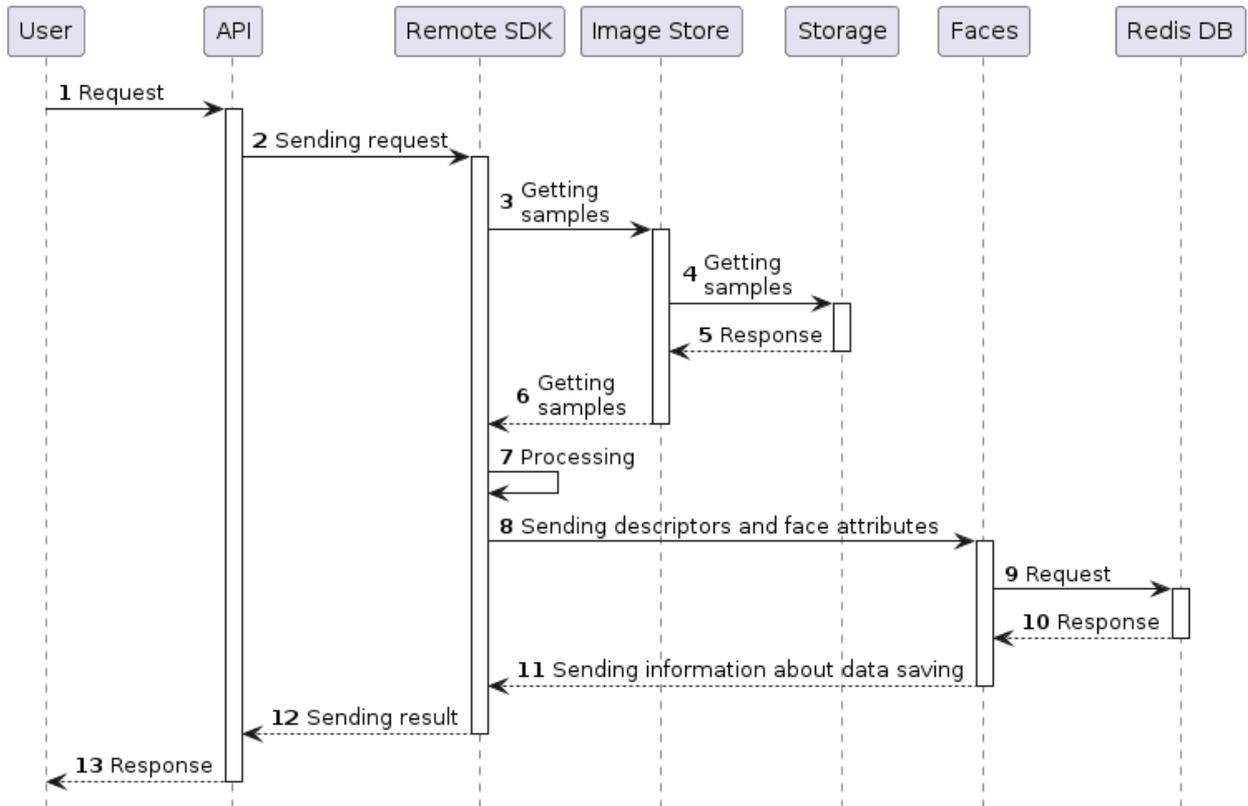


Figure 66: Temporary attributes extraction sequence diagram

The general request processing scheme:

1. The extraction request is sent to API.
2. API receives the request, processes it and sends the request to the Remote SDK service.
3. The Remote SDK service requests samples from Image Store.
4. Image Store requests the samples from the storage.
5. The storage sends samples.
6. Image Store sends the samples to the Remote SDK service.
7. The Remote SDK service processes the task according to the specified parameters.
8. The Remote SDK service sends descriptors and basic attributes to the Faces service.

9. The Faces service sends requests to store temporary attributes in the Redis database.
10. Redis database sends the response to Faces.
11. Faces sends the response to the Remote SDK service.
12. The Remote SDK service sends received attribute IDs, basic attributes, attribute URLs, filtering results and GS score to the API service.
13. API sends the response.

9.2.2 Diagram of attribute creation using external data

The diagram shows attributes creation using data from an external database.

See the [“attributes” section](#) in [OpenAPI specification](#) for details.

Request	Description	Type
create temporary attribute	New temporary attributes creation.	POST

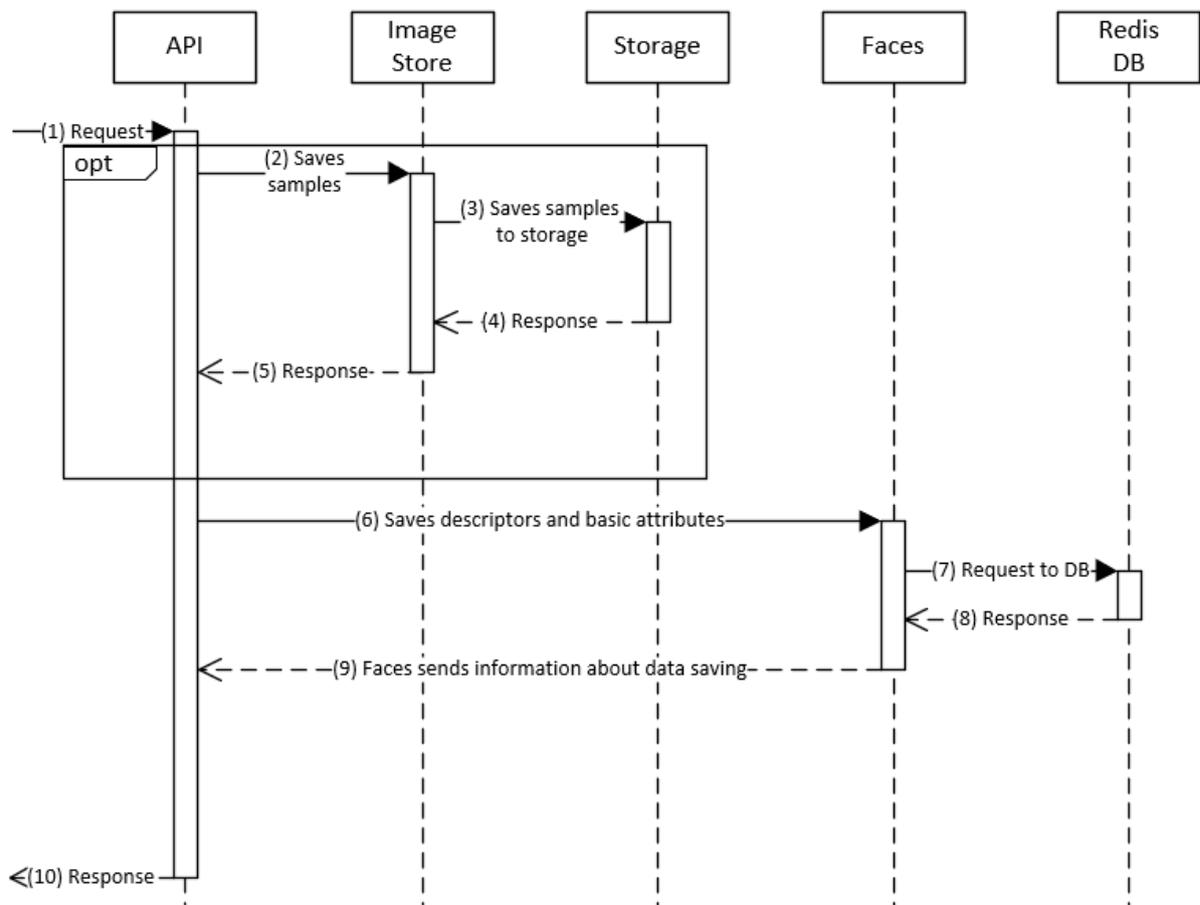


Figure 67: External temporary attributes saving sequence diagram

1. The request for adding new attributes (descriptors and/or basic attributes) is sent to the API service. All the required data for attributes creation is sent with the request. **Optional.** The request to the Image Store service is sent when samples were provided with descriptors and/or attributes.
2. The API service sends the request to the Image Store service to save the received samples.
3. Image Store requests the samples from the storage.
4. The storage sends samples.
5. Image Store sends the samples to the API service.
6. The API service sends descriptors and basic attributes to the Faces service.
7. The Faces service sends requests to store temporary attributes in the Redis database.
8. Redis database sends the response to Faces.
9. Faces sends the response to the API service.
10. The API service sends response.

9.2.3 Attributes information diagrams

The following requests enable you to receive data about the already existing attributes or delete them.

Request	Description	Type
get temporary attributes	Receive all attribute IDs and their creation time according to the targets.	GET
get temporary attribute	Receive the temporary attribute information by ID.	GET
check temporary attribute	Check existence of an attribute by its ID.	HEAD
delete attributes	Delete the attribute with the specified ID.	DEL
get temporary attribute samples	Get all the temporary attribute samples by the attribute ID.	GET

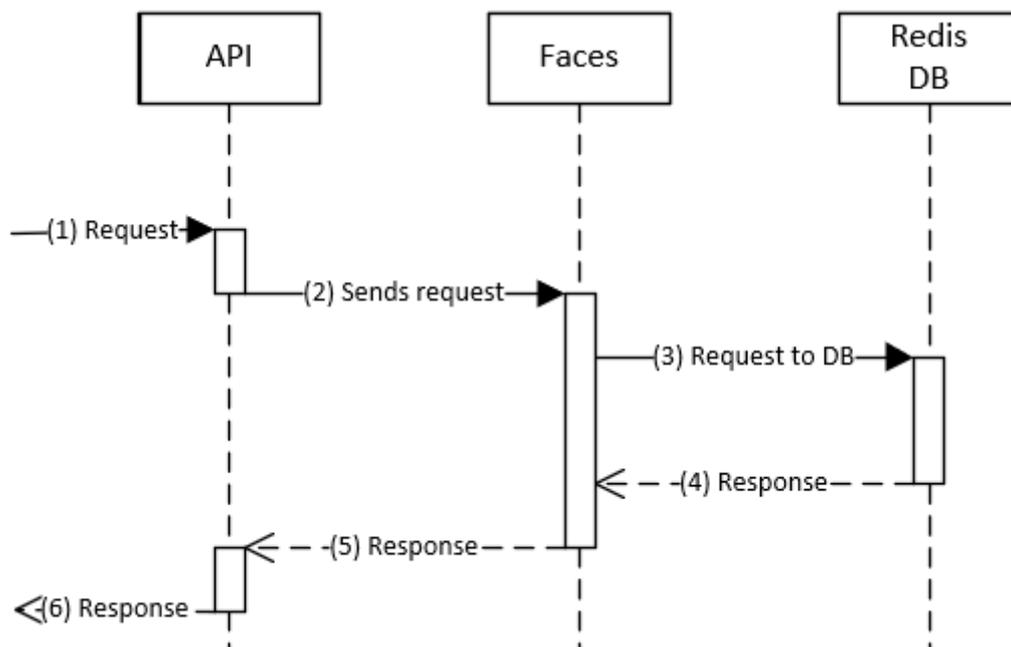


Figure 68: Get information about temporary attributes

The general request processing scheme:

1. A request is sent to the API service.
2. API sends the request to the Faces service.
3. Faces service sends the request to the Redis database to receive information about temporary attributes.

4. The Redis database returns requested information.
5. The Faces service returns the result.
6. The API service returns the information to the user. If the TTL of the attribute has expired, an error is returned.

9.3 Faces and lists diagrams

All the requests in this section are processed by the Faces service.

See the “faces” section in [OpenAPI specification](#) for details.

9.3.1 Face creation diagram

Request	Description	Type
create face	Create a new face with the specified attribute ID, user’s data, avatar and lists.	POST

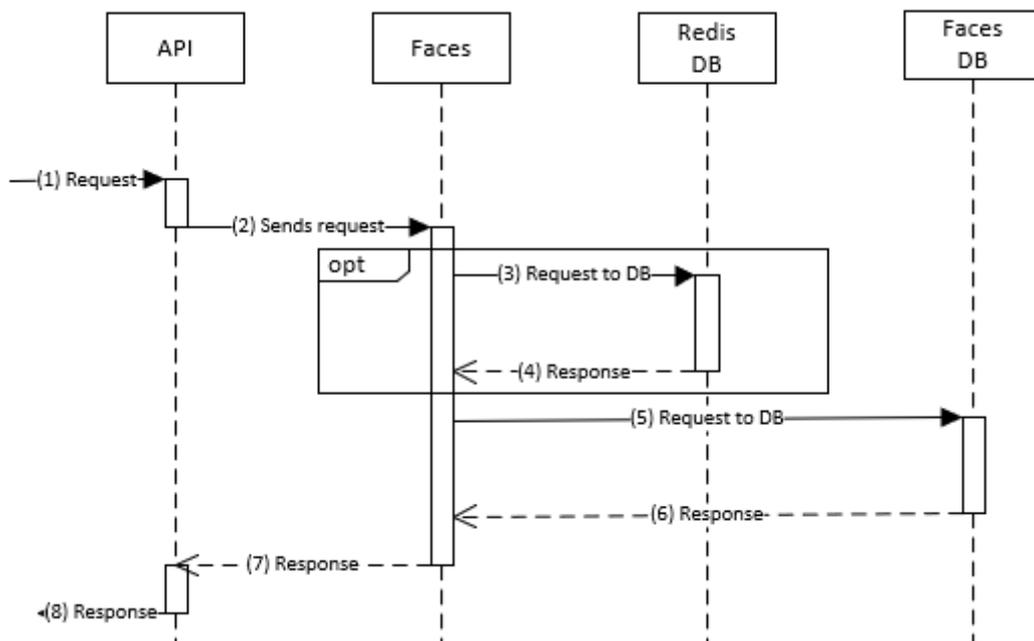


Figure 69: Create new face

The general request processing scheme:

1. A request is sent to the API service.

2. API sends the request to the Faces service.
3. Faces service sends the request to the Redis database to receive temporary attributes. **Optional.** The request to the Redis DB is not sent if there are external attributes specified for the face or when there are no attributes set for the face.
4. The Redis database returns requested information.
5. The Faces service returns the result.
6. The Faces service sends the request to the Faces database to create a new face using the specified data.
7. The Faces database saves the data.
8. The Faces service returns the information about the created face.
9. The API service returns the information to the user.

9.3.2 Faces and Lists information and management

All the following requests have similar sequence diagrams.

The following requests enable you to create faces, receive data about the already existing faces or delete them.

Request	Description	Type
get faces	Get the array of all the existing faces and their data: face ID, external ID, user's data, create time, avatar, account ID, and lists to which the Face is attached.	GET
delete faces	Delete several faces by IDs.	DEL
get face count	Receive the number of existing faces according to the specified filter.	GET
get count of faces with attributes	Count of faces with attributes.	GET
get face	Receive data of the face (face ID, external ID, user's data, create time, avatar, account ID, and lists to which the face is attached) that corresponds to the specified ID.	GET
patch face	Update a Face using the specified data: user's data, event ID, external ID, avatar.	PATCH
remove face	Delete the specified face.	DEL
check to exist a face	Check existence of the face by its ID.	HEAD

Request	Description	Type
put face attribute	Put face attribute, changing all the attribute data corresponding to the specified face.	PUT
get face attribute	Receive attributes of the specified face: gender, age, ethnicity, create time.	GET
delete face attribute	Remove face attribute by ID.	DEL
get face attribute samples	Receive information about samples (sample ID) that correspond to the specified face.	GET

The following requests enable you to create lists and receive data about the already existing lists or delete them.

Request	Description	Type
create list	Create a new empty list. You can specify user data for it.	POST
get lists	Receive the array of all existing lists with the following data: list ID, user data, account ID, creation time, last update time.	GET
delete lists	Delete several lists by IDs.	DEL
get list count	Get number of the existing lists.	GET
get list	Receive information (list ID, user data, account ID, creation time, last update time) about the list by the specified ID.	GET
check list existence	Check existence of the list with the specified ID.	HEAD
update list	Update the user_data field of the list.	PATCH
delete list	Delete the list with the specified ID.	DEL
attach/detach faces to the list	Update the list by attaching or detaching the specified faces from it.	PATCH

The following diagram represents workflow for all the listed above requests to Faces.

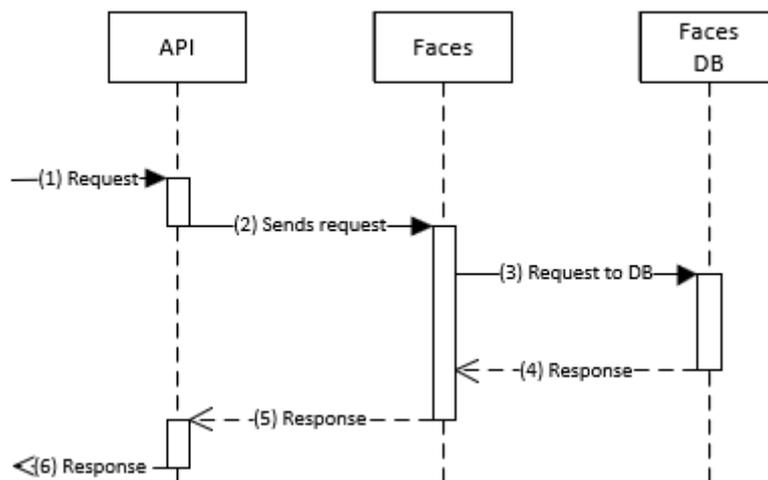


Figure 70: Faces request processing diagram

The general request processing scheme:

1. A request is sent to the API service.
2. API sends the request to the Faces service.
3. The Faces service sends the request to the Faces database to receive information or manage the existing data.
4. The Faces database returns requested information or information about database changes.
5. The Faces service returns the result.
6. The API service returns the result to the user.

9.4 Matching diagrams

You should specify references and candidates for matching. You can limit the number of candidates with the highest similarity value.

Request	Description	Type
matcher/faces	Matcher compares given references with the given candidates. As a result, matcher returns similarity level for each of the candidates and additional information about candidates.	POST
human body matching	Allows to submit tasks to a service that searches for human bodies similar to a given reference(s) by matching them.	POST
raw matching	Allows to do similarity calculations for input descriptors.	POST

See the “[matching faces](#)”, “[human body matching](#)” and “[raw matching](#)” sections in [OpenAPI specification](#) for details.

9.4.1 Matching using Python Matcher

9.4.1.1 Matching by DB

The example of matching events (references) with faces (candidates) is given below.

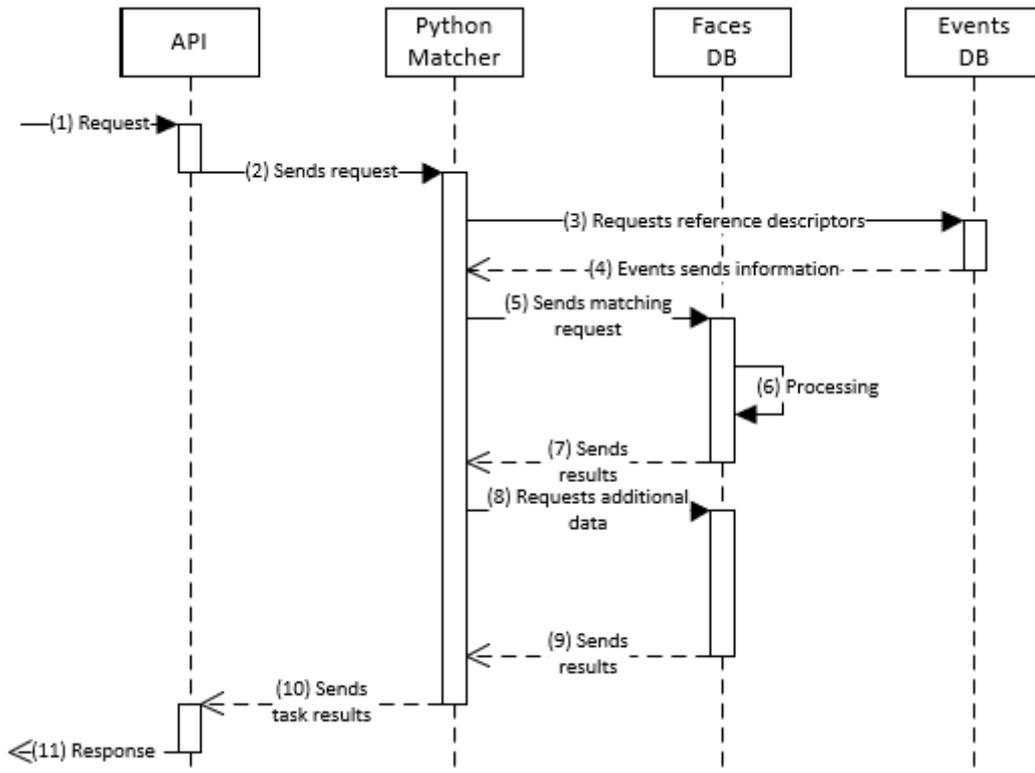


Figure 71: Matching of events and faces

1. The matching request is sent to the API service.
2. The API service sends the request to the Python Matcher service.
3. The Python Matcher service requests references from the Events database.
4. The Events database returns the data.
5. The Python Matcher service sends the requests for matching by the Faces database.
6. The matching is performed.
7. The Faces database returns the matching results.
8. The Python Matcher service requests additional data for candidates.
9. The Faces database returns the data.

10. The Python Matcher service returns results to the API service.
11. The API service sends the response.

9.4.1.2 Matching by list

The example of matching faces (references) with the list of faces (candidate) is given below.

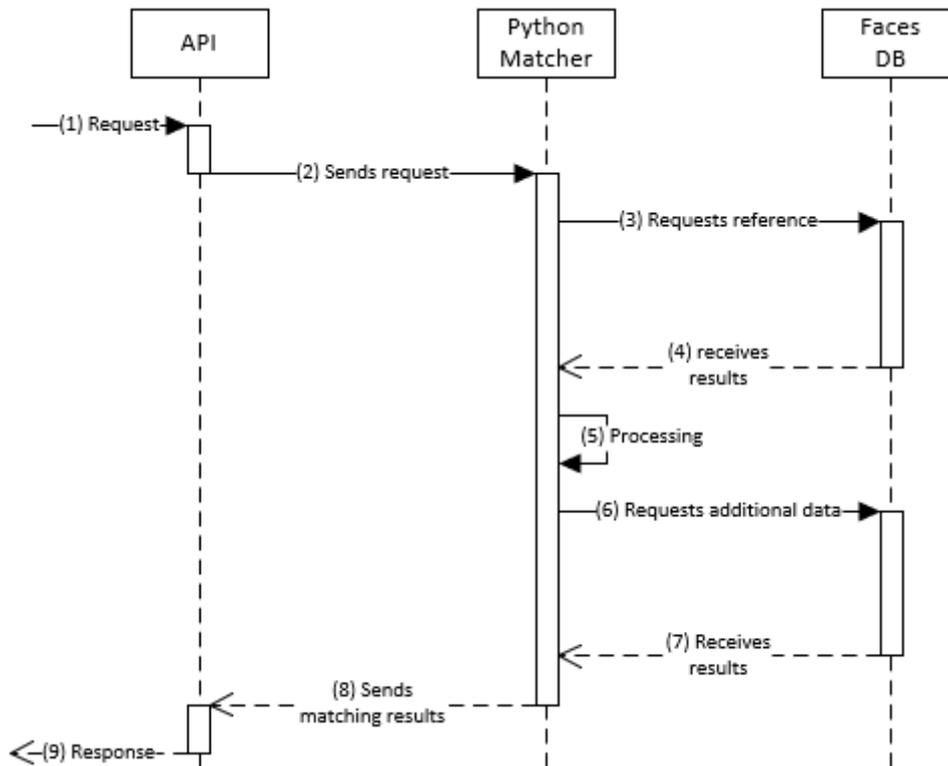


Figure 72: Matching faces by list

1. The matching request is sent to the API service.
2. The API service sends the request to the Python Matcher service.
3. The Python Matcher service requests references from the Faces database.
4. The Faces database returns the data.
5. The matching is performed in The Python Matcher service. The cached descriptors are used.
6. The Python Matcher service requests additional data for candidates.
7. The Faces database returns the data.
8. The Python Matcher service returns results to the API service.
9. The API service sends the response.

9.5 Handlers diagrams

9.5.1 Handlers management requests

A handler defines the logic for input image processing. You should specify a handler when a new event is created.

The following requests are related to the handler.

Request	Description	Type
create handler	Create handler.	POST
get handlers	Get handlers according to the specified filters.	GET
get handler count	Receive number of existing handlers.	GET
get handler	Receive handler policies by the handler ID.	GET
replace handler	Update fields of a handler. You should specify the handler ID. You should fill in all the required policies in the request body. Update of individual parameters of a handler is not allowed.	PUT
check to exist a handler	Check if the handler with the specified ID exists.	HEAD
remove handler	Remove a handler by ID.	DEL

The general scheme of handler creation is shown in the figure below. All the handler requests have similar sequence diagrams.

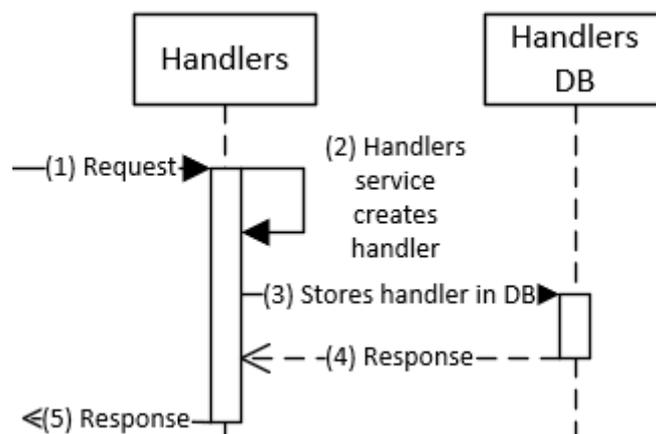


Figure 73: Handler creation diagram

1. A request for a new handler creation is sent from the API service to the Handlers service.
2. The Handlers service processes the request and creates a handler.

3. The Handlers service stores the handler to API database.
4. The Handlers service database returns result.
5. The Handlers service service returns the ID of the created handler.

The handler is used upon Event creation and its usage example is described in the “Events” section. All the results are stored in the Events database.

9.6 Events diagrams

9.6.1 Event creation general diagram

Event is created after an image is processed according to a handler.

Request	Description	Service
generate events	Create an event. You should specify the required handler ID or set dynamic handler policies and specify images for processing. You can set additional parameters and data for the created event.	POST

The sequence diagram for a new event creation will differ depending on the specified handler policies.

The sequence diagram below shows the general process of new event creation and includes only general entry points for the execution of policies.

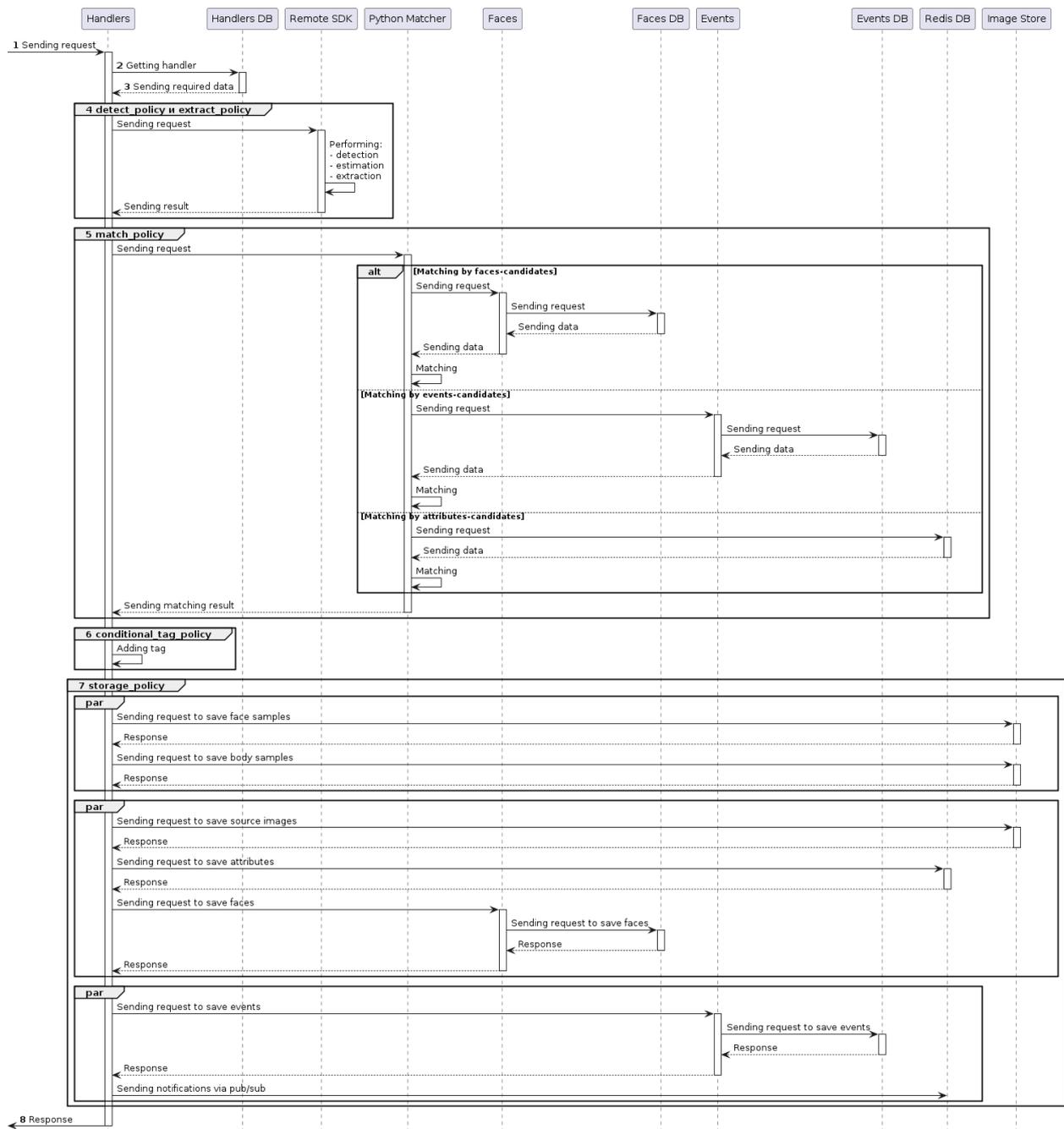


Figure 74: Event creation diagram

1. The API service sends the request for new Event creation to the Handlers service.
2. The Handlers service receives the corresponding handler from the Handlers database.
3. The Handlers database sends the handler to the Handlers service.
4. The “detect_policy” and “extract_policy” are processed by the Remote SDK service. The received samples and attributes are stored in RAM.

5. The “match_policy” is processed according to the provided filters. Descriptors received during the “extract_policy” execution are provided as references for matching. There can be several ways for matching execution:
 - Faces are set as matching candidates. The matching will be performed using the Faces DB. You can specify the necessary lists with faces using filters.
 - Events are set as matching candidates. The matching will be performed using the Events DB.
 - Attributes are set as matching candidates. The matching will be performed using the Redis DB.
6. The “conditional_tags_policy” is processed by the Handlers service.
7. The “storage_policy” is processed according to the specified data:
 - Face samples, body samples and source images are saved to the Image Store service;
 - Attributes are created and saved in the Redis database by the Faces service;
 - Faces are created and saved in the Faces database by the Faces service. They also can be linked to lists using “link_to_lists_policy”;
 - Events are saved to the Events database by the Events service;
 - Notifications are sent via pub/sub to Redis (see the “[Sender service](#)” section).
8. The Handlers service returns results to the API service.

9.6.2 Get statistics on events and events information

Policy	Description	Service
get statistics on events	Receive statistics on Events. The target group enables to aggregate events according to the specified aggregators: number of items, max or min value, average value, grouping. You can also apply filters and periods to select events that should be aggregated. Grouping by frequency or grouping by time intervals is available.	POST
get events	Receive all events that satisfy the specified filters. In the filters you can set values of one or several event fields. The target parameter enables to select fields that you want to receive in response. You can left the target field of the request empty. In this case all data about the events will be shown. You may also set sorting parameters, number of events on page and number of pages. If the create_time__gte is not set, it is set to one month by default.	GET
get event	Receive all fields for the event. You should specify “event_id” for the request.	GET

Policy	Description	Service
check event existence	Check existence of the specified event. You should specify “event_id” for the request.	HEAD

The sequence diagram describes the request processing.

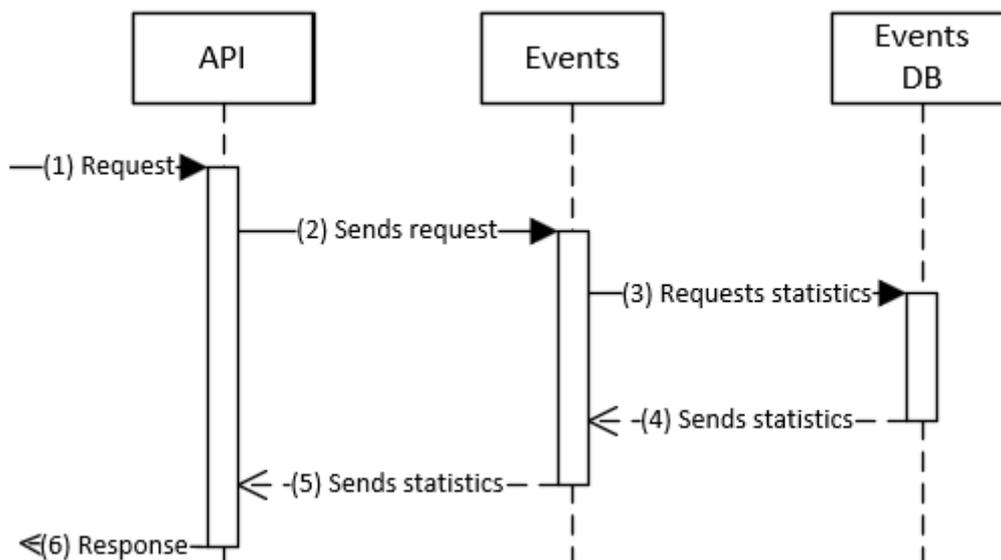


Figure 75: Events information

1. API receives request to Events.
2. API sends the request to Events.
3. Events receives the required data from the database.
4. The database returns result.
5. Events returns results.
6. API returns results.

9.7 Tasks diagrams

This section provides diagrams of the sequence of tasks.

The section “[General diagram of task creation and execution](#)” shows the general process of task processing. For some tasks, the process may be different. For example, for a Clustering task, only one subtask is created and only one worker is used. Separate comments are provided for such tasks.

9.7.1 General diagram of task creation and execution

A diagram of the general process of creating and executing a task is presented below.

Note that the Tasks worker is constantly waiting for any data to be received. The service has a certain priority for the execution of work, namely:

1. Checking whether the current task/subtask needs to be canceled (not reflected in the diagram).
2. Requests for splitting into subtasks.
3. Requests for merging results.
4. Requests for processing subtasks.

This means that the Tasks worker will not process a subtask request until it performs a cancellation check on the current task.

9.7.1.1 Starting task creation

This section describes the process of starting to create a task.

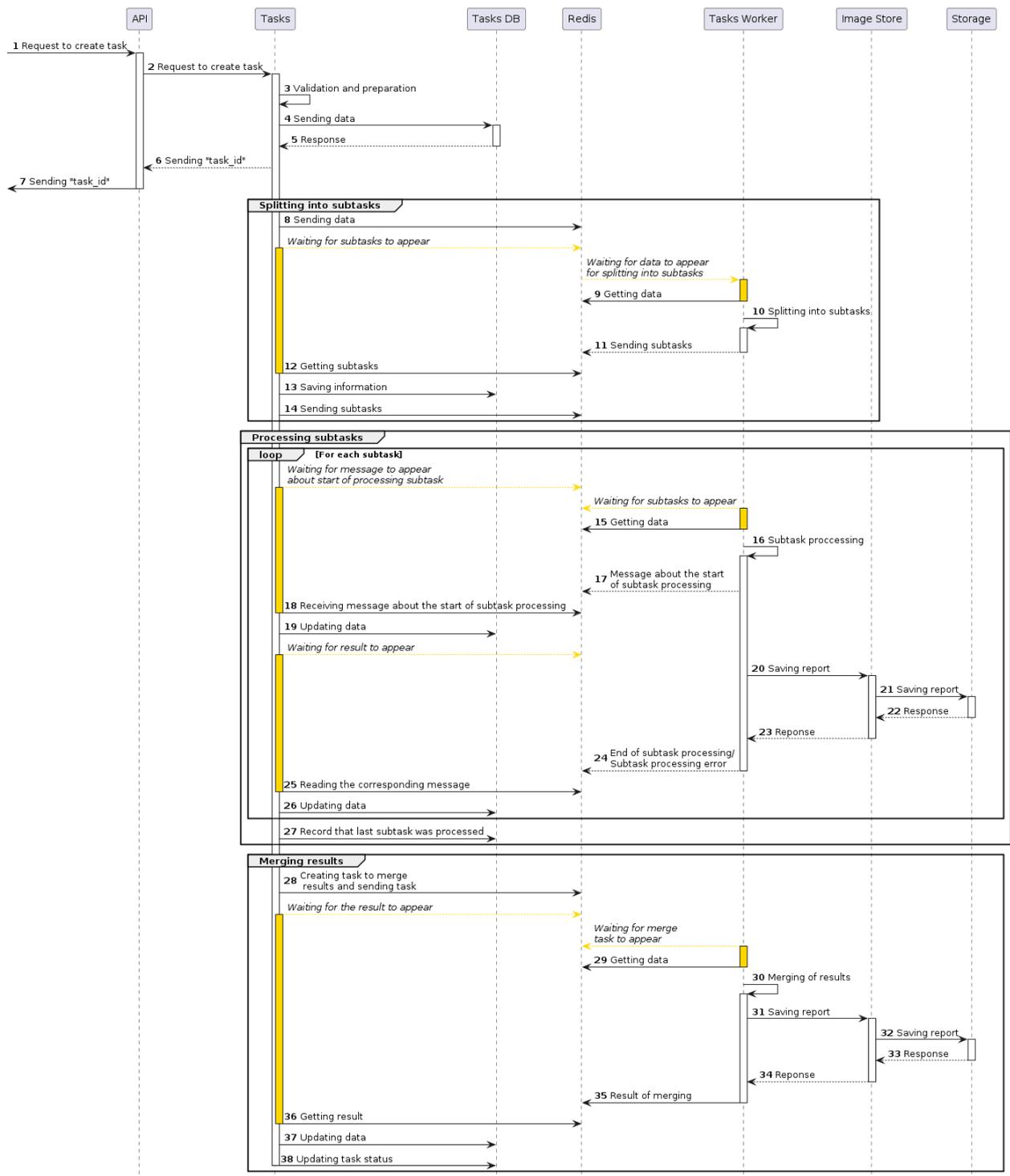


Figure 76: Task creation diagram

1. The API service receives a request to create a task.
2. The API service sends a request to the Tasks service.

3. The Tasks service performs task validation, creates it and prepares for its processing. At this stage, the Tasks service can perform additional actions, for example, get the “descriptor_id” to perform the task “[Additional extraction](#)”.
4. The Tasks service sends data about the task being created to the Tasks database.
5. The Tasks service receives a response.
6. The Tasks service sends the task ID “task_id” to the API service.
7. The user receives the task ID, i.e. the API service issues the ID of the created task and does not wait for the completion of the task. By this identifier, the user can evaluate the status of the task.

To get a report with the status of task completion and perform other actions with the task, you should use the task ID. The relevant requests are listed in the “[General information about tasks](#)” section.

9.7.1.2 Splitting tasks into subtasks

This section describes the process of splitting a task into subtasks. If only one subtask is assumed for some task (for example, Clustering task), then the task will be split into one subtask according to the process described below.

8. The Tasks service sends data to Redis.

The Tasks service starts waiting for subtasks to appear in Redis.

The Tasks Worker service is waiting for data to appear in Redis to split them into subtasks.

9. The Tasks Worker service receives data from Redis.
10. The Tasks Worker service performs the process of splitting into subtasks.
11. The Tasks Worker service sends subtasks back to Redis.
12. The Tasks service receives subtasks.
13. The Tasks service stores information about subtasks in the Tasks database.
14. The Tasks service sends subtasks to Redis.

9.7.1.3 Processing each subtask

This section describes the general process of processing a subtask. Each task has its own processing process, described in the corresponding section below.

The Tasks service starts waiting for a message to appear about the start of processing a subtask in Redis.

The Tasks Worker service is waiting for data to appear in Redis to process a subtask.

15. The Tasks Worker service receives data from Redis.
16. The Tasks Worker service starts processing the subtask.
17. The Tasks Worker service sends a message to Redis that processing has started.
18. The Tasks service receives a message about the start of processing.
19. The Tasks service updates information about the Tasks database task.

The Tasks service starts waiting for the result to appear.

20. The Tasks Worker service saves the task report to the Image Store service.
21. The Image Store service sends a request to the Storage to save the report.
22. The Image Store service receives a response from the user.
23. The Image Store service returns a response to the Tasks Worker service.
24. The Tasks Worker service sends a message to Redis that the subtask has been processed or that some error occurred during processing.
25. The Tasks service reads the corresponding message from Redis.
26. The Tasks service updates information about the Tasks database task.
27. After processing the last subtask, the Tasks service updates the information about the Tasks database task.

9.7.1.4 Merging results and completing processing

This section describes the general process of merging the results and completing the processing. If only one subtask is assumed for some task (for example, Clustering task), then merging the results will not be performed. See the “Task processing completion” section for a specific task.

28. The Tasks service creates an internal task to combine all the results and sends this task to Redis.

The Tasks service starts waiting for the result to appear.

The Tasks Worker service is waiting for a merge task to appear.

29. The Tasks Worker service receives the task data for merging.
30. The Tasks Worker service merges the results of all subtasks.
31. The Tasks Worker service saves the task report to the Image Store service.
32. The Image Store service sends a request to the Storage to save the report.
33. The Image Store service receives a response from the user.

34. The Image Store service returns a response to the Tasks Worker service.
35. The Tasks Worker service sends the result of the merge to Redis.
36. The Tasks service reads the result of the merge from Redis.
37. The Tasks service updates the task data in the Tasks database.
38. The Tasks service updates the status of the task in the Tasks database.

9.7.2 General diagram of task cancellation

A diagram of the general task cancellation process is presented below.

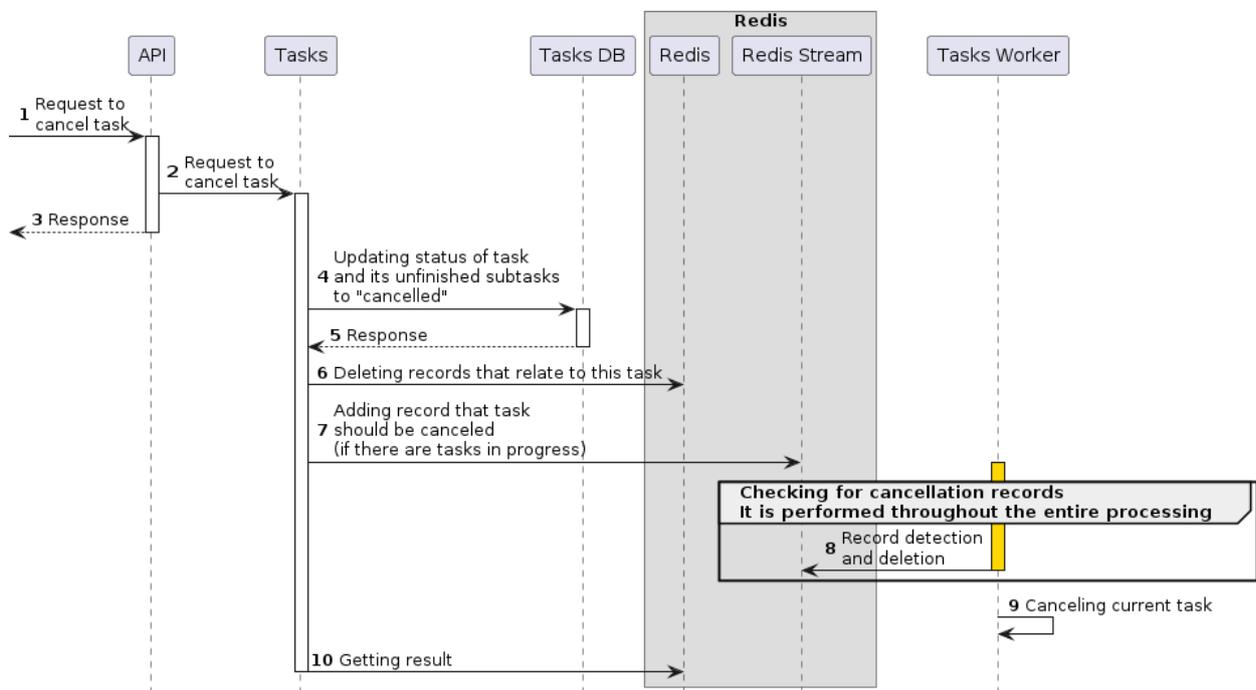


Figure 77: Task cancellation diagram

1. The user sends a request to cancel the task to the API service.
2. The API service sends a request to cancel the task to the Tasks service.
3. The API service returns a response that the task cancellation process has been activated.
4. The Tasks service updates the status of the task and its unfinished subtasks to “cancelled” in the Tasks database.
5. The Tasks service receives a response.
6. The Tasks service deletes entries in Redis that relate to the task being canceled.

7. The Tasks service adds an entry to the Redis Stream stating that the task should be canceled if there are tasks whose status is in “in_progress”.
8. The Tasks worker detects a cancellation entry in Redis Streams and deletes it.

The Tasks worker performs a check for the presence of records throughout the processing.

9. The Tasks worker Tasks cancels task processing.
10. The Tasks service receives the cancellation result from Redis.

Redis is used to store data, including records, which are represented as event streams in Redis Streams.

At this stage, the task is considered completely canceled, i.e. the status “2” (“cancelled”) will be returned in responses to requests like “[get task](#)”.

9.7.3 Clustering task diagrams

Request	Description	Method
clustering task	Creates the task for clustering Faces or Events according to the specified filters.	POST

9.7.3.1 Clustering task creating

Creating a task is performed in the standard way described in the section “[Starting task creation](#)” in the section with the general diagram of the work of the Tasks service.

9.7.3.2 Splitting Clustering task into subtask

A single subtask is created for this task type. Subtask contains filters according to which faces or events are selected. The subtask is processed by a single worker.

See “[Splitting tasks into subtasks](#)” in the section with the general diagram of the Tasks service.

9.7.3.3 Clustering subtask processing

Clustering task processing depends on the objects (faces or events) specified in the request.

The general workflow for processing a subtask is shown below.

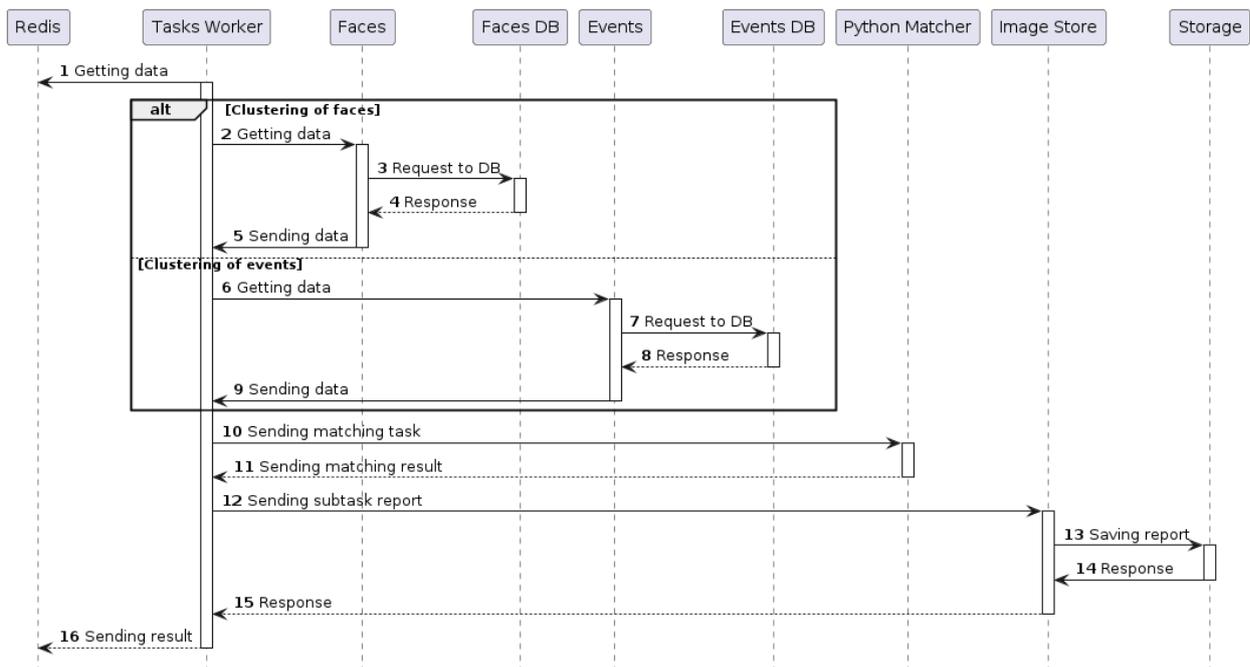


Figure 78: Clustering task processing diagram

1. The Tasks worker receives data from Redis.
2. **Faces:** The Tasks worker sends the request to Faces service. The Tasks worker requests all the required attribute IDs according to the filters specified in the subtask.
3. **Faces:** The Faces service sends the request to the Faces database.
4. **Faces:** The Faces database returns IDs.
5. **Faces:** The Faces service sends IDs to the Tasks worker.
6. **Events:** The Tasks worker sends the request to Events service. The Tasks worker requests all the required attribute IDs according to the filters specified in the subtask.
7. **Events:** The Events service sends a request to the database to receive data.
8. **Events:** The Events database sends the required data.
9. **Events:** The Events sends data to the Tasks worker.
10. The Tasks worker matching request. The requests is processed according to one of the schemes described in section [“Matching diagrams”](#).
11. The Python Matcher service sends the results.
12. The Tasks service saves the task report to the Image Store service.
13. The Image Store service sends a request to the Storage to save the report.
14. The Image Store service receives a response from the user.

15. The Image Store service returns a response to the Tasks service.
16. The Tasks worker sends the result to Redis.

9.7.3.4 Clustering task processing completion

Next, the Tasks service receives the result from Redis and updates the status of the task in the Tasks database.

9.7.4 Linker task diagrams

Request	Description	Method
linker task	The request enables you to create a linker task.	POST

9.7.4.1 Linker task creation

The linker task can be created for faces and events objects. Linker task creation process depends on the object type.

Attach faces to list

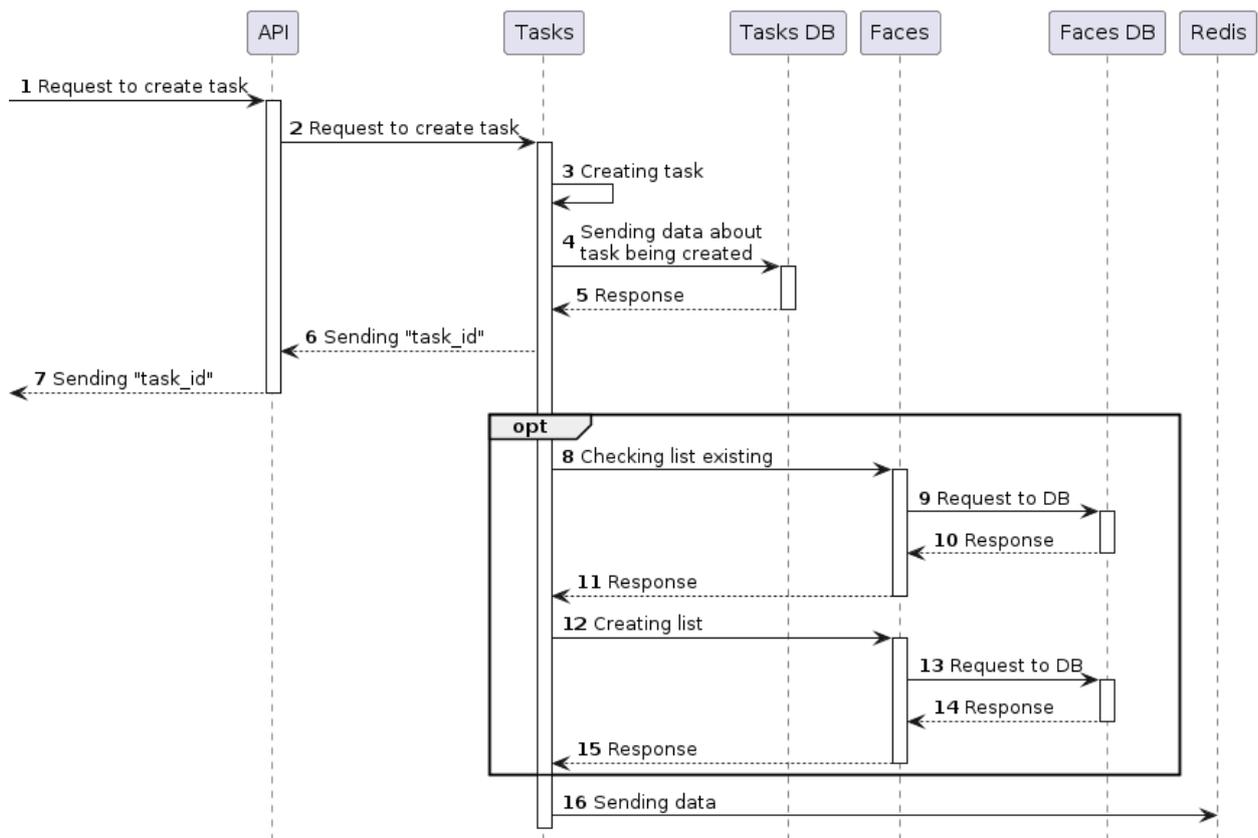


Figure 79: Creation of linking task for faces

1. The API service receives a request to create a task.
2. The API service sends a request to the Tasks service.
3. The Tasks service creates a new linker task.
4. The Tasks service sends information to the Tasks database.
5. The task ID is returned from the Tasks database.
6. The task ID is sent to the API service.
7. The API service sends the task ID as a response.
8. **Optional.** If you have specified a list ID in the request, Tasks service checks the existence of the list.
9. **Optional.** The Faces service checks the list existence in the Faces database.
10. **Optional.** The Faces database sends response.
11. **Optional.** The Faces service sends response to the Tasks service.
12. **Optional.** If the specified list does not exist or you have specified new list creation in the request, the Tasks service sends request for new list creation.

13. **Optional.** The Faces service creates the list in the Faces database.
14. **Optional.** The Faces database sends response.
15. **Optional.** The Faces service sends the response to the Tasks service.
16. The Tasks service sends data to Redis.

Attach faces created from events to list

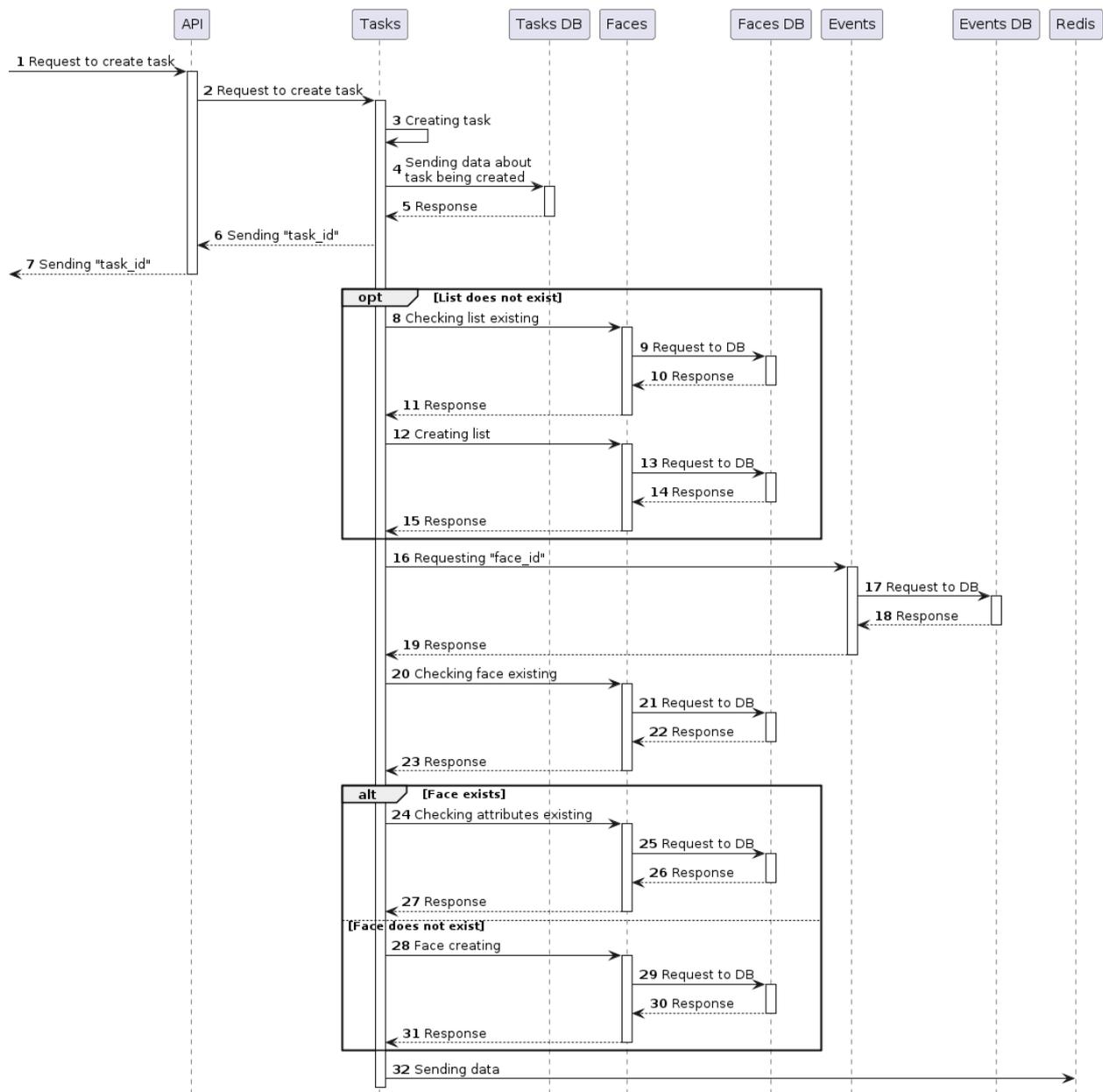


Figure 80: Creation of linking task for events

1. The API service receives a request to create a task.

2. The API service sends a request to the Tasks service.
3. The Tasks service creates a new linker task.
4. The Tasks service sends information to the Tasks database.
5. The task ID is returned from the Tasks database.
6. The task ID is sent to the API service.
7. API service sends the task ID as a response.
8. **Optional.** If you specified a list ID in the request, Tasks service service checks the existence of the list.
9. **Optional.** The Faces service checks the list existence in the Faces database.
10. **Optional.** The Faces database sends response.
11. **Optional.** The Faces service sends the response to the Tasks service.
12. **Optional.** If the list does not exist or you have specified to create a new list in the request, the Tasks service sends the request to create a new list.
13. **Optional.** The Faces service creates the list in the Faces database.
14. **Optional.** The Faces database sends response.
15. **Optional.** The Faces service sends the response to the Tasks.
16. The Tasks service receives face IDs and attribute IDs from the Events service.
17. The Events service receives the data from the Events database.
18. The Events database sends response.
19. The Faces service sends the IDs to the Tasks service.
20. The Tasks service checks the existence of faces in Faces.
21. The Faces service sends the request to the Faces database.
22. The Faces database sends information about faces existence.
23. The Faces service sends the data to Tasks service.
24. **Faces exist in Faces database** If the face for an event exists, The Tasks service checks that the attribute ID of the face is similar to the attribute ID of the event.
25. The Faces service sends a request to the Faces database.
26. The Faces database send attribute IDs.
27. The Faces service sends attribute IDs to the Tasks service. If the attribute IDs for the existing face and event are not equal, the “28009: Attribute is not equal” error will be returned.

28. **Faces do not exist in Faces database** If there are no faces with the specified IDs in the Faces database, the Tasks service sends the request to create new faces in Faces database.
29. The Faces service sends the request to the Faces database.
30. The Faces database sends results of face creation.
31. The Faces service sends the response to the Tasks service.
32. The Tasks service sends data to Redis.

9.7.4.2 Splitting Linker task into subtasks

There can be several workers used for task execution.

See “[Splitting tasks into subtasks](#)” in the section with the general diagram of the Tasks service.

9.7.4.3 Linker subtasks processing

The general workflow for processing each subtask is shown below.

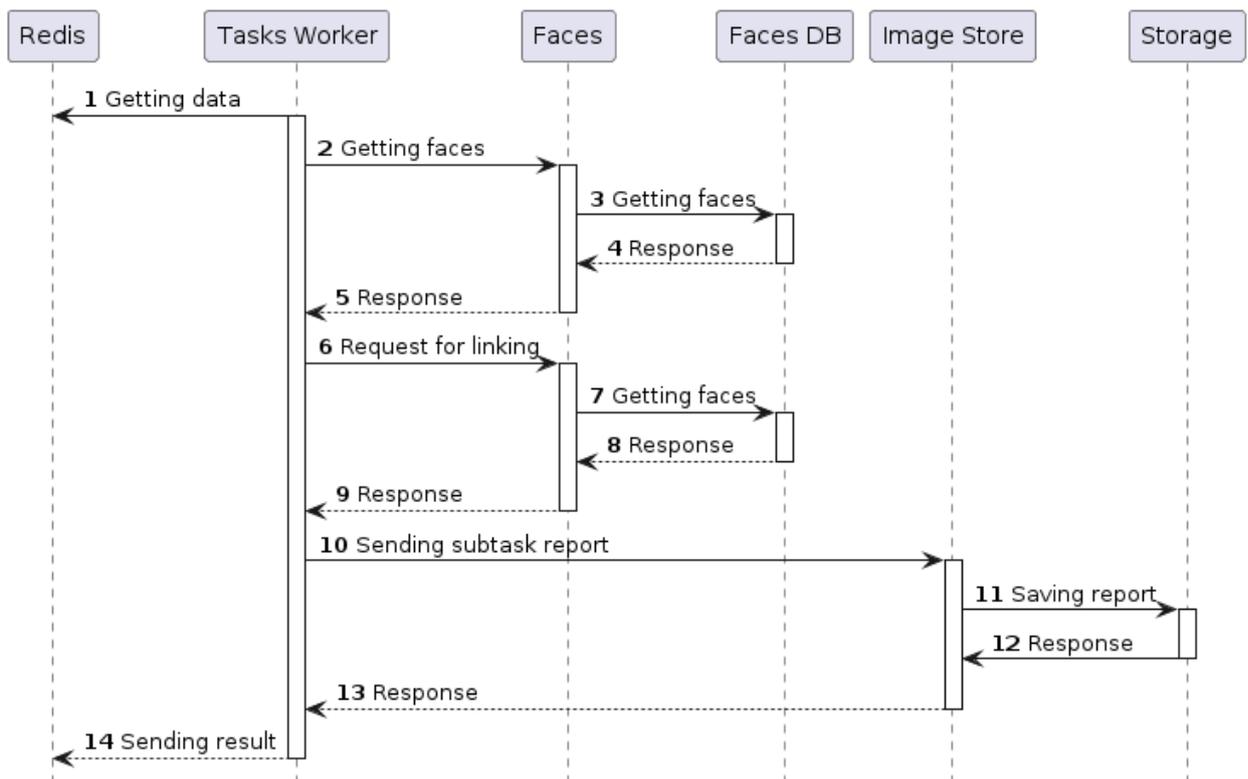


Figure 81: Linker task processing

1. The Tasks worker receives data from Redis.
2. The Tasks worker requests faces in Faces service according to the received face IDs.

3. The Faces service requests faces in the Faces database.
4. The database sends faces to Faces.
5. The Faces service sends faces to the Tasks worker.
6. The Tasks worker sends requests for attaching faces to the specified list.
7. The Faces service sends requests for attaching to the database.
8. The database sends the result of attaching to Faces.
9. The Faces service sends the result to the Tasks worker.
10. The Tasks worker generates a report on the basis of the subtask results and stores it in the Image Store.
11. Image Store stores report in the storage.
12. The storage sends result of storage. The result has its result ID.
13. The Image Store service returns the result to the Tasks worker.
14. The Tasks worker sends the result to Redis.

9.7.4.4 Linker task completing processing

Next, the results of the subtasks are merged and a general report is sent to the Image Store. See “[Merging results and completing processing](#)” in the section with the general diagram of the Tasks service.

9.7.5 Garbage collection task diagram

Request	Description	Method
garbage collection task	A task for deleting <i>attributes</i> according to the specified creation time filter. Only attributes that are not attached to any face are deleted. The task allows to delete unused data from Faces database.	POST

9.7.5.1 Garbage collection task creating

Creating a task is performed in the standard way described in the section “[Starting task creation](#)” in the section with the general diagram of the work of the Tasks service. The only difference is that the Garbage collection task can be created not only through the API service, but also through the Admin or Tasks service.

9.7.5.2 Splitting Garbage collection task into subtasks

Several subtasks are created for this task type. Each of the subtasks contains the range of attribute IDs. See “Splitting tasks into subtasks” in the section with the general diagram of the Tasks service.

9.7.5.3 Garbage collection task processing

The general workflow for processing each subtask is shown below.

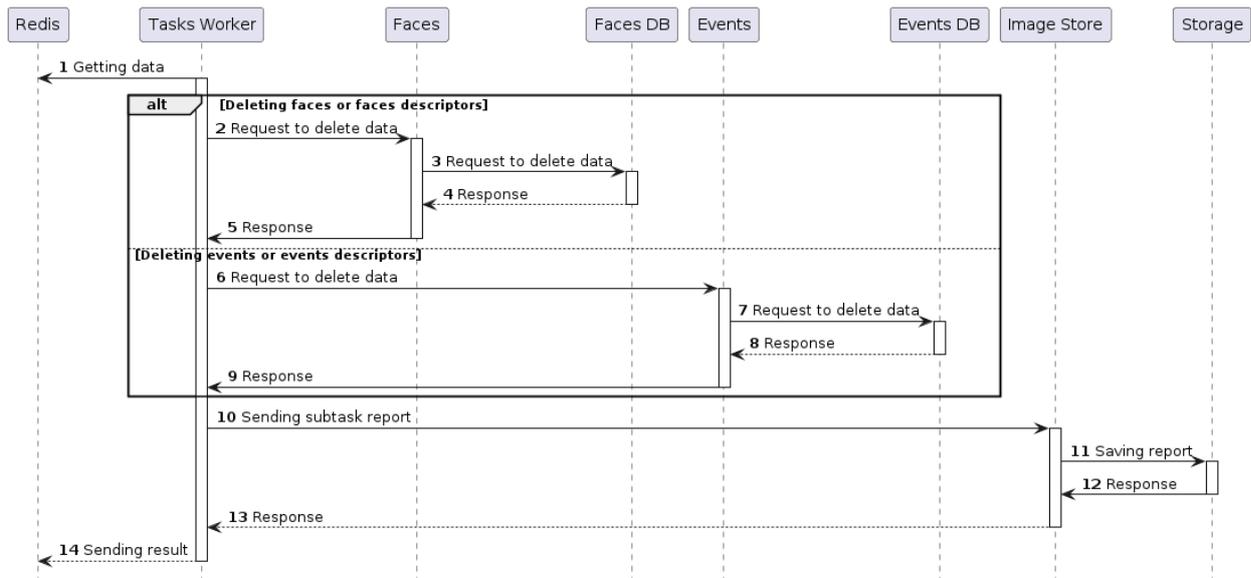


Figure 82: Garbage collection task processing

1. The Tasks worker receives data from Redis.
2. **Removing faces or face descriptors.** The Tasks worker has a range of the attribute IDs that should be deleted. Tasks worker requests deletion of temporary attributes and corresponding descriptors in Faces service.
3. The Faces service sends the request to Faces database to delete attributes with the specified IDs.
4. The database sends results of deletion.
5. The Faces service sends the response to the Tasks worker.
6. **Removing events or event descriptors.** The Tasks worker has a range of the attribute IDs that should be deleted. The Tasks worker requests deletion of temporary attributes and corresponding descriptors in Events service.
7. The Events service sends the request to Events database to delete attributes with the specified IDs.
8. The database sends results of deletion.
9. The Events sends the response to the Tasks worker.

10. The Tasks worker stores report in Image Store.
11. Image Store sends report to the storage.
12. The storage sends information about storage results to Image Store.
13. Image Store sends the result to the Tasks worker.
14. The Tasks worker sends the result to Redis.

9.7.5.4 Garbage collection task completing processing

Next, the results of the subtasks are merged and a general report is sent to the Image Store. See [“Merging results and completing processing”](#) in the section with the general diagram of the Tasks service.

9.7.6 Reporter task diagram

Request	Description	Method
reporter task	Create a report for a complete clustering task. The report is in the CSV format. You can specify the columns that should be added to the report. The report is in a zip archive and includes avatar images for each of the objects in a cluster.	POST

9.7.6.1 Reporter task creating

Creating a task is performed in the standard way described in the section [“Starting task creation”](#) in the section with the general diagram of the work of the Tasks service.

9.7.6.2 Splitting Reporter task into subtasks

A single subtask is created for this task type. The subtask includes filters for the data that should be obtained for inclusion into the report. See [“Splitting tasks into subtasks”](#) in the section with the general diagram of the Tasks service.

9.7.6.3 Reporter subtask processing

Request processing depends on the data, stored in the clustering report. If the report was created for faces, face data will be requested from the Faces service and added to the report. If the report was created for events, event data will be requested from the Events service and added to the report.

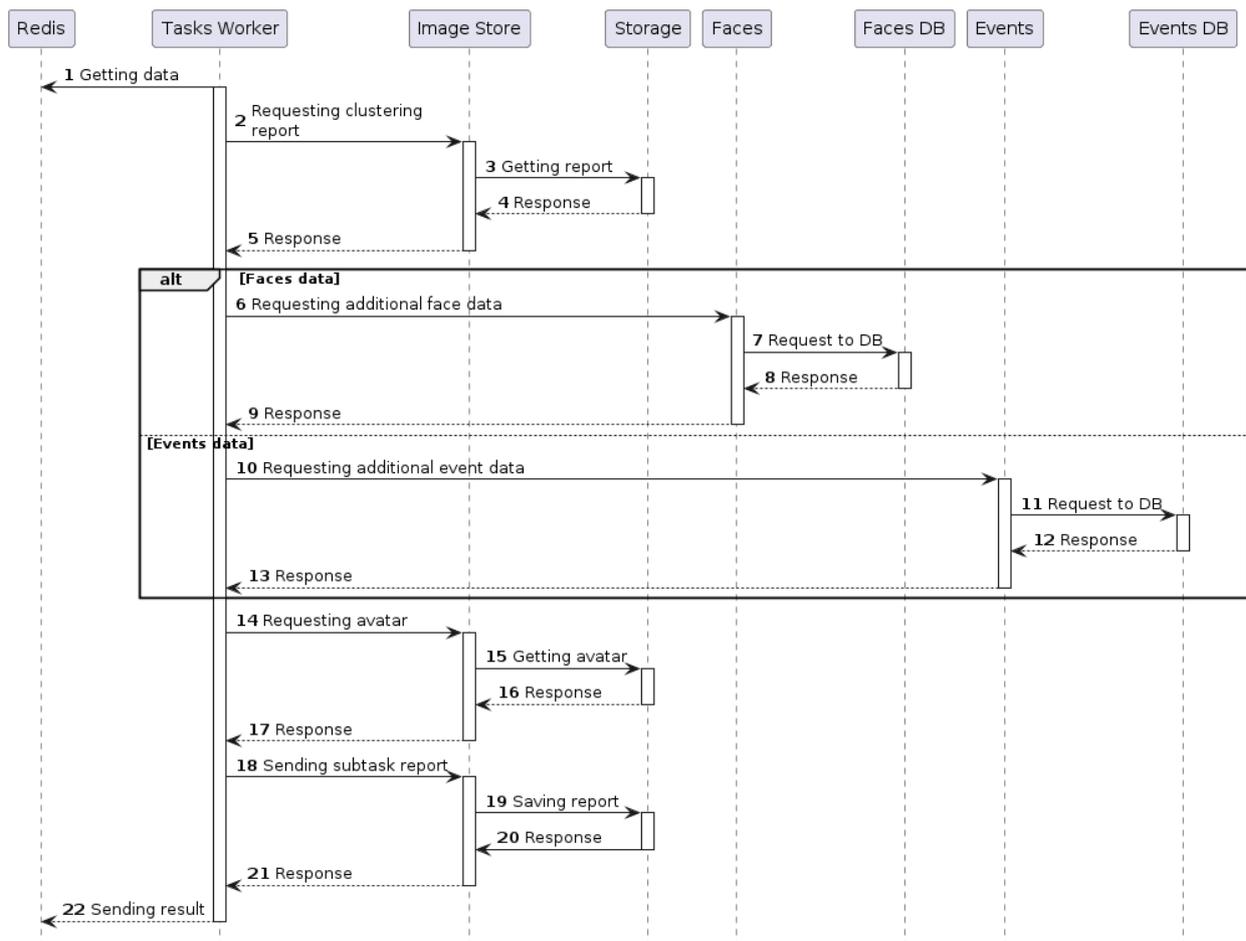


Figure 83: Reporter task processing

1. The Tasks worker receives data from Redis.
2. The Tasks worker requests a clustering report in Image Store.
3. The Image Store service requests the report from the storage.
4. The storage sends the report.
5. The Image Store service sends the report.
6. **Report for faces.** The Tasks worker receives all the face IDs from the cluster and requests additional face data from the Faces service. The requested data depends on the columns specified in the request.
7. The Faces service requests data from the Faces database.
8. The Faces database sends the required data.
9. The Faces service sends the data to the Tasks worker.

10. **Report for events.** The Tasks worker requests additional event data from the Events service. The requested data depends on the columns specified in the request.
11. The Events service requests data from the Events database.
12. The Events database sends the required data.
13. The Events service sends the data to the Tasks worker.
14. The Tasks worker requests the avatar image for each of the faces or events. The image for a face is specified in the “avatar” field of the Faces database. It can be stored in the Image Store storage or any other storage. The image for the event is the corresponding sample from the Image Store storage.
15. The Image Store service requests the image from the storage.
16. The storage sends the image.
17. The Image Store service sends the image.
18. The Tasks worker sends report to Image Store.
19. The Image Store service saves the report in the storage.
20. Storage sends response about saving.
21. The Image Store service sends response to the Tasks worker.
22. The Tasks worker sends the result to Redis.

9.7.6.4 Reporter task completing processing

Next, the Tasks service receives the result from Redis and updates the status of the task in the Tasks database.

9.7.7 Exporter task diagram

Request	Description	Method
exporter task	Collects event/face data and exports them to a CSV file. The exported CSV file is in a zip archive and includes avatar images for each of the objects.	POST

9.7.7.1 Exporter task creating

Creating a task is performed in the standard way described in the section “[Starting task creation](#)” in the section with the general diagram of the work of the Tasks service.

9.7.7.2 Splitting Exporter task into subtask

A single subtask is created for this task type. See “Splitting tasks into subtasks” in the section with the general diagram of the Tasks service.

9.7.7.3 Exporter subtask processing

The general workflow for processing each subtask is shown below.

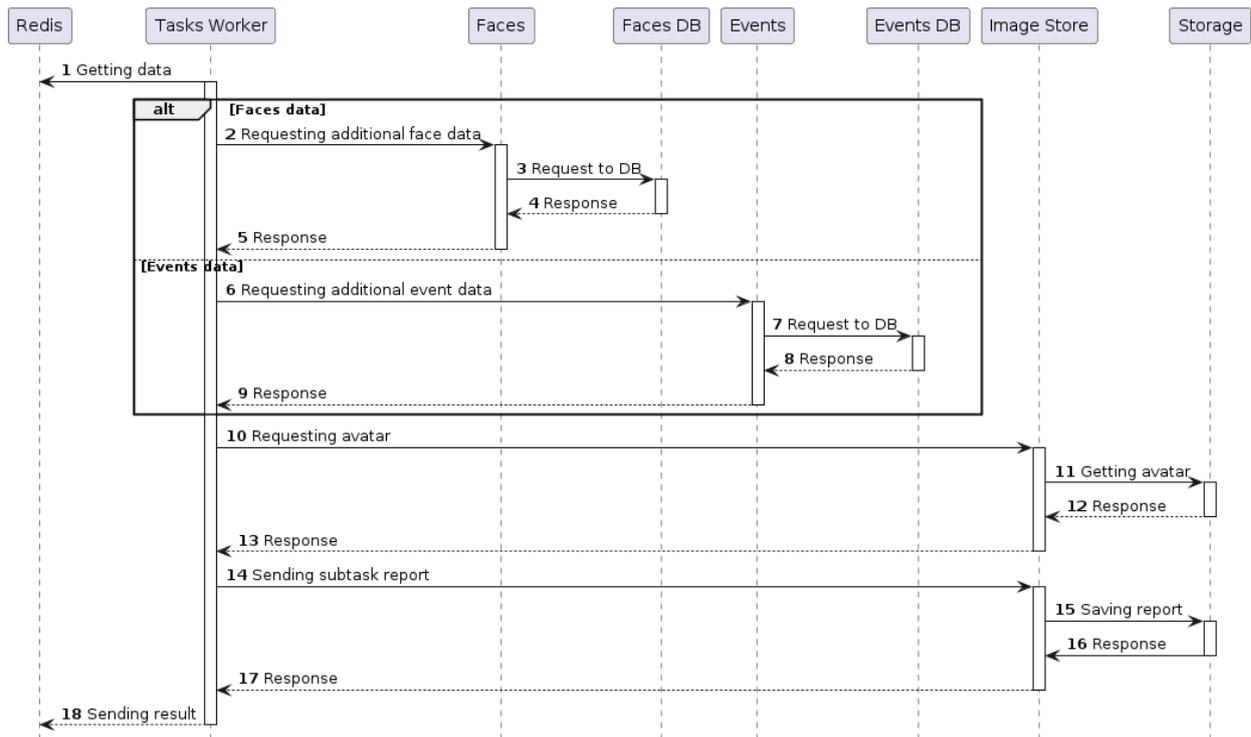


Figure 84: Exporter task processing

1. The Tasks worker receives data from Redis.
2. **Export data for faces.** The Tasks worker receives all the face IDs and requests additional face data from the Faces service. The requested data depends on the columns specified in the request.
3. The Faces service requests data from the Faces database.
4. The Faces database sends the required data.
5. The Faces service sends the data to the Tasks worker.
6. **Export data for events.** The Tasks worker requests additional event data from the Events service. The requested data depends on the columns specified in the request.
7. The Events service requests data from the Events database.
8. The Events database sends the required data.

9. The Events service sends the data to the Tasks worker.
10. The Tasks worker requests the avatar image for each of the faces or events. The image for a face is specified in the “avatar” field of the Faces database. It can be stored in the Image Store storage or any other storage. The image for the event is the corresponding sample from the Image Store storage.
11. The Image Store service requests the image from the storage.
12. The storage sends the image.
13. The Image Store service sends the image.
14. The Tasks worker sends exported data to Image Store.
15. The Image Store service saves the exported data in the storage.
16. The Storage sends response about saving.
17. The Image Store service sends response to the Tasks worker.
18. The Tasks worker sends the result to Redis.

9.7.7.4 Exporter task completing processing

Next, the Tasks service receives the result from Redis and updates the status of the task in the Tasks database.

9.7.8 Cross-matching task diagrams

Request	Description	Method
cross-matching task	The task enables to match events and faces according to the specified filters. Faces and events can be set as references and as candidates. Additional filters can be specified for both references and candidates.	POST

9.7.8.1 Cross-matching task creating

Creating a task is performed in the standard way described in the section “[Starting task creation](#)” in the section with the general diagram of the work of the Tasks service.

9.7.8.2 Splitting Cross-matching task into subtasks

A single subtask is created for this task type. The subtask includes all the required reference and candidate filters. See “[Splitting tasks into subtasks](#)” in the section with the general diagram of the Tasks service.

9.7.8.3 Cross-matching subtasks processing

Cross-matching subtask execution may vary depending on the specified references and candidates.

They can be specified using faces and/or events. In the diagrams below, requests to Faces and Events services are marked as alternative. The requests to Faces service are used when faces are set as references or candidates. The requests to Events service are used when events are set as references or candidates.

Cross-matching task processing depends on the objects selected (events or faces).

The general workflow for processing each subtask is shown below.

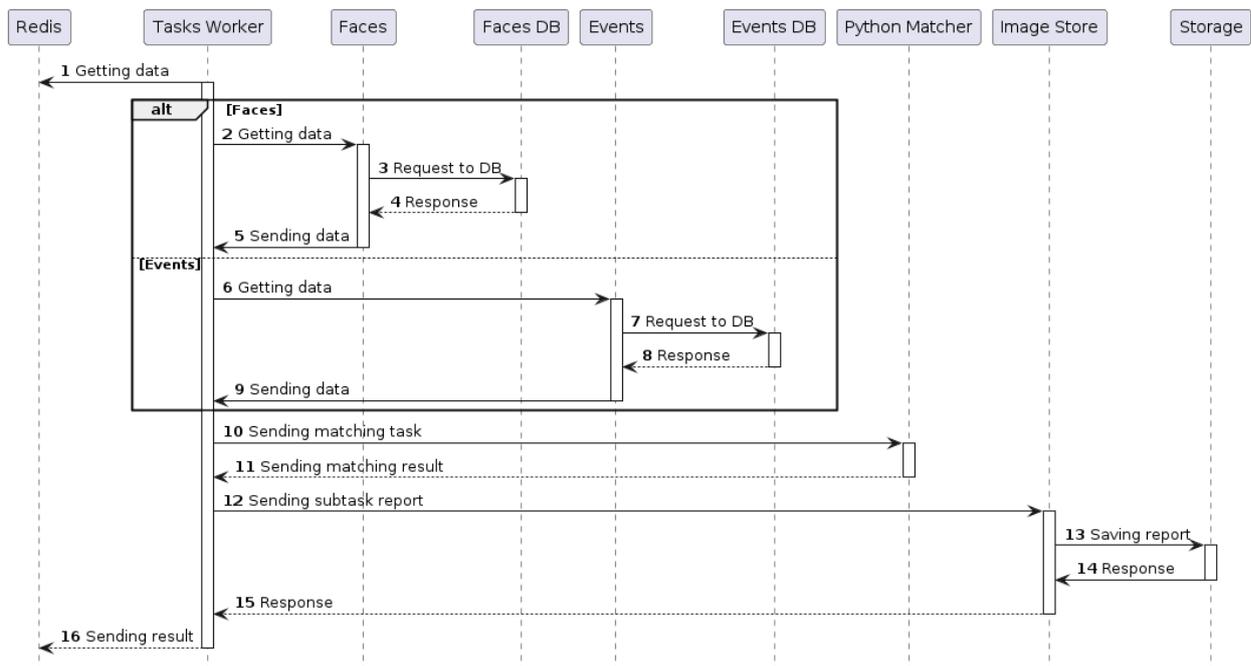


Figure 85: Cross-matching task processing

1. The Tasks worker receives data from Redis.
2. **Faces are set as references or candidates.** The Tasks worker sends the request to the Faces service to receive attribute IDs for faces. Faces are selected according to the specified filters.
3. The Faces service requests IDs in the Faces database.
4. The Faces database sends attribute IDs.
5. The Faces service sends the attribute IDs to the Tasks worker.
6. **Events are set as references or candidates.** The Tasks worker sends the request to the Events service to receive attribute IDs for events. Events are selected according to the specified filters.
7. The Events service requests event IDs in the Events database.
8. The Events database sends the IDs.

9. The Events service sends the IDs to the Tasks worker.
10. The Tasks worker matching request. The requests is processed according to one of the schemes described in section [“Matching diagrams”](#).
11. The Python Matcher service sends the results.
12. A report is sent from the Tasks worker to the Image Store.
13. Using the Image Store, the report is saved to the storage.
14. The repository sends a response about saving.
15. The Image Store service sends a response to the Tasks worker.
16. The Tasks worker sends the result to Redis.

9.7.8.4 Cross-matching task completing processing

Next, the Tasks service receives the result from Redis and updates the status of the task in the Tasks database.

9.7.9 Estimator task diagram

Request	Description	Method
estimator task	Create an Estimator task. With this task, you can perform batch image processing with the specified policies.	POST

9.7.9.1 Estimator task creating

Creating a task is performed in the standard way described in the section [“Starting task creation”](#) in the section with the general diagram of the work of the Tasks service.

9.7.9.2 Splitting Estimator task into subtask

A single subtask is created for this task type. See [“Splitting tasks into subtasks”](#) in the section with the general diagram of the Tasks service.

9.7.9.3 Estimator task processing

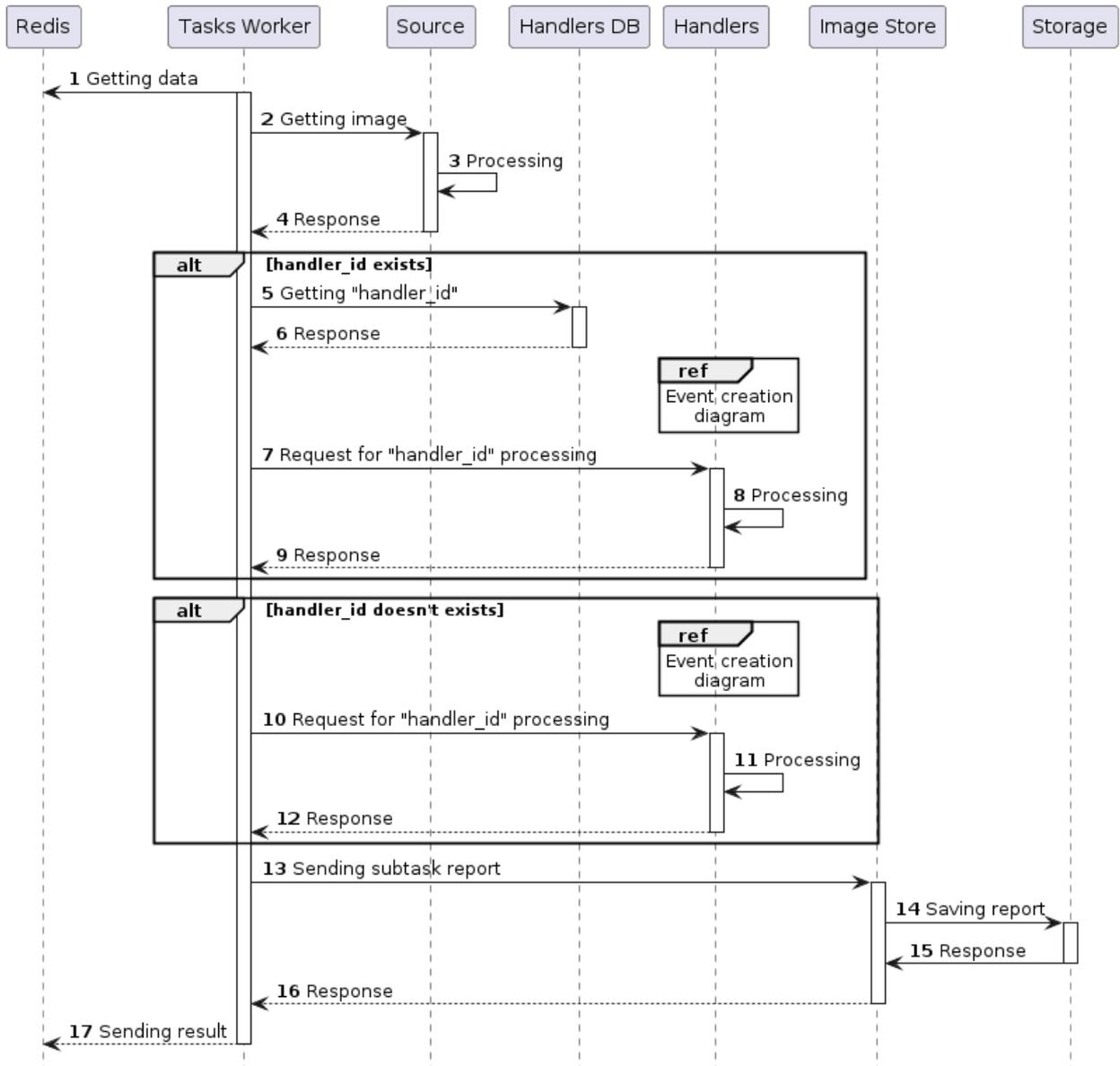


Figure 86: Estimator task diagram

1. The Tasks worker receives data from Redis.
2. The Tasks worker requests a batch of images from the following sources:
 - From the Image Store service in a ZIP archive if the “zip” type is selected as the image source.
 - From the S3 storage if the “s3” type is selected as the image source.
 - From the FTP server if the “ftp” type is selected as the image source.
 - From the Samba if the “samba” type is selected as the image source.
 - From a directory on a network disk that is mounted on a local drive and synchronized with the

directory in the containers of the Tasks and Tasks Worker service (see the detailed description in the “[Estimator task](#)” section).

3. Processing the parameters specified in the request — authorization, use of prefixes, etc.
4. Source (Image Store/S3/Network disk/FTP server/Samba) sends an images to the Tasks worker.
5. **handler_id exists:** The Tasks worker sends a request to get the existing handler ID in the Handlers database.
6. **handler_id exists:**From the Handlers database, a response comes with handler_id to the Tasks worker.
7. **handler_id exists:**The Tasks worker sends a request to the Handlers service to process handler_id.
8. **handler_id exists:**The Handlers service processes the batch of images by the received handler_id in accordance with the specified policies (see the [event creation diagram](#)).
9. **handler_id exists:**The received response is returned to the Tasks worker.
10. **handler_id doesn't exist.** The Tasks worker sends a request to process the policies specified directly in the request to the Handlers service.
11. **handler_id doesn't exist.** The Handlers service processes the batch of images in accordance with the specified policies (see the [event creation diagram](#)).
12. **handler_id doesn't exist.** The received response is returned to the Tasks worker.
13. A report is sent from the Tasks worker to the Image Store.
14. Using the Image Store, the report is saved to the storage.
15. The repository sends a response about saving.
16. The Image Store service sends a response to the Tasks worker.
17. The Tasks worker sends the result to Redis.

9.7.9.4 Estimator task completing processing

Next, the Tasks service receives the result from Redis and updates the status of the task in the Tasks database.

9.7.10 Additional extraction task diagram

Request	Description	Method
create additional extraction task	Extract all the existing descriptors using the new NN version.	POST

9.7.10.1 Additional extraction task creation

The request is sent using Admin service. Task creation is shown in the diagram below.

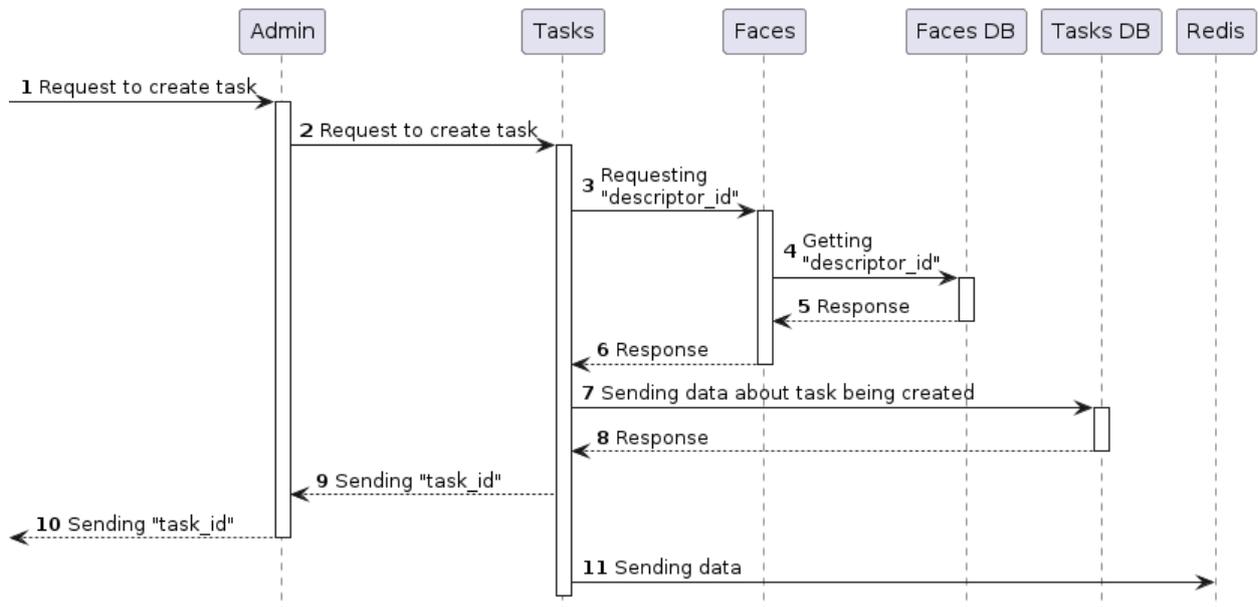


Figure 87: Additional extraction task creation

1. The administrator sends a request to the Admin service using the Admin UI or any other application.
2. The Admin service sends a request to Tasks.
3. The Tasks service sends a request to Faces to receive descriptor IDs that were not extracted using the new NN.
4. The Faces service sends the request to the database.
5. The Faces database returns IDs.
6. The Faces service sends IDs to Tasks.
7. The Tasks service sends data about the created task to the Tasks database.
8. The Tasks service gets a "task_id".
9. The Tasks service sends the task ID to the Admin service.
10. The Admin service sends the created task ID to the user interface or application used by the administrator.
11. The Tasks service sends data to Redis, from where the Tasks worker will take the task for further processing.

9.7.10.2 Splitting Additional extraction task into subtasks

Several subtasks are created for this task type. Each of the subtasks contains the range of attribute IDs. See “Splitting tasks into subtasks” in the section with the general diagram of the Tasks service.

9.7.10.3 Additional extraction subtask processing

The general workflow for each subtask processing is shown below.

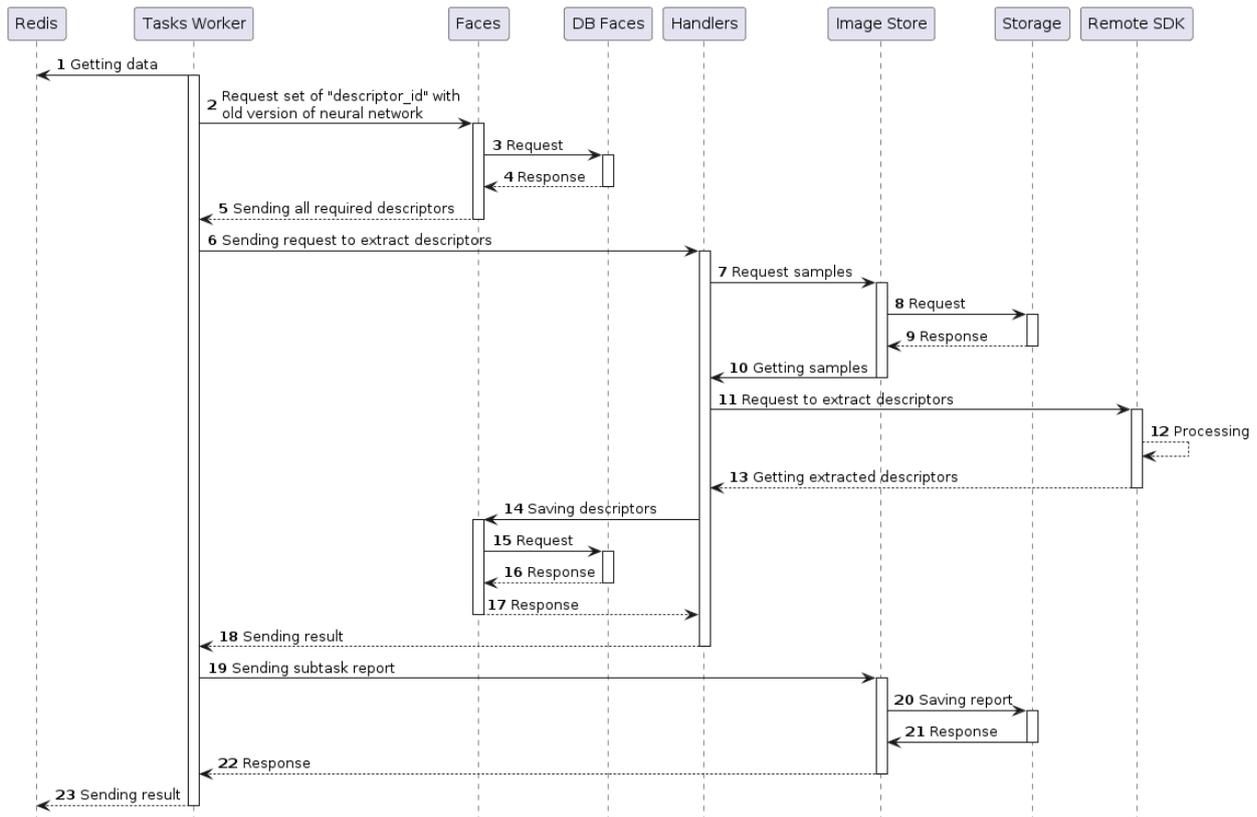


Figure 88: Additional extraction task processing

1. The Tasks worker receives data from Redis.
2. The Tasks Worker sends the request to Faces to receive all the required descriptors according to their IDs.
3. The Faces sends the request to the DB.
4. The DB sends back the required descriptors.
5. The Faces sends the descriptors to the Tasks worker.
6. The Tasks Worker sends the extraction request to the Handlers service.
7. The Handlers service requests samples for extraction from Image Store.

8. The Image Store requests samples from the storage.
9. The Image Store receives samples from the storage.
10. The Image Store sends samples to the Handlers service.
11. The Handlers service sends request for extracting descriptors to the Remote SDK service.
12. The Remote SDK extracts descriptors.
13. The Remote SDK service sends extracted descriptors to Handlers service.
14. The Handlers service sends descriptors and basic attributes to Faces.
15. Faces sends the request to store data in the database.
16. The database returns the result of data storage.
17. The Faces service returns the result of data storage.
18. The Handlers service sends the result of the task execution to the Tasks service.
19. A report is sent from the Tasks worker to the Image Store.
20. Using the Image Store, the report is saved to the storage.
21. The repository sends a response about saving.
22. The Image Store service sends a response to the Tasks worker.
23. The Tasks worker sends the result to Redis.

9.7.10.4 Additional extraction task completing processing

Next, the results of the subtasks are merged and a general report is sent to the Image Store. See [“Merging results and completing processing”](#) in the section with the general diagram of the Tasks service.

9.7.11 Tasks information diagrams

The following requests provide information on the status of existing tasks.

Request	Description	Method
cancel task	Cancel the tasks execution by its ID.	PATCH
get tasks	Receive tasks information. You can set filters for the tasks.	GET
get tasks count	Receive number of tasks according to the specified filters.	GET
get task	Receive information about a task by its ID.	GET
get subtasks	Receive information about all subtasks of the specified task.	GET

The following requests provide information about errors occurred while processing tasks.

Request	Description	Method
get errors of task	Receive all the errors of the specified task by the task ID.	GET
get errors	Receive all the errors according to the specified filters.	GET
get errors count	Receive the number of errors according to the specified filter.	GET
get error	Receive information about an error by the error ID.	GET

The corresponding diagram is shown below.

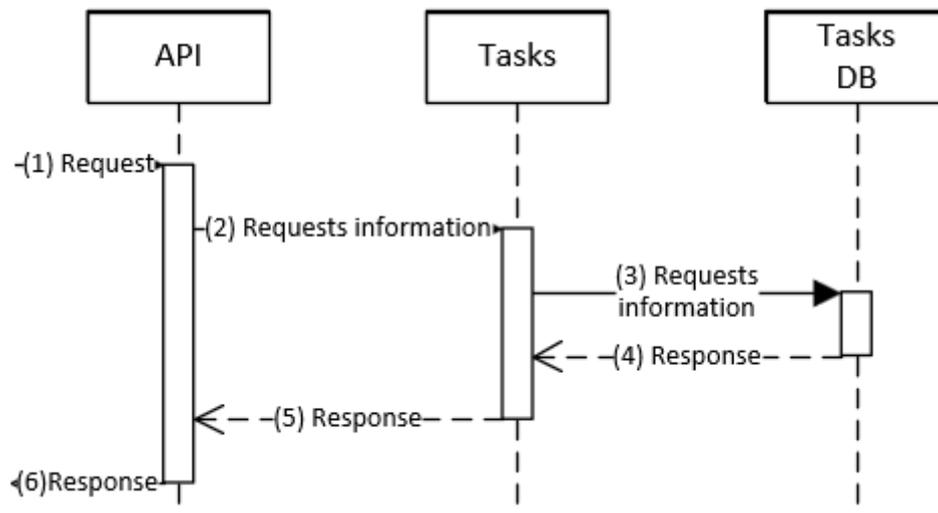


Figure 89: Receive task information or cancel task

1. A request is sent to API.
2. API service sends the request to Tasks to receive the required data or perform required actions.
3. Tasks sends the request to the Tasks database to receive the required data or perform required actions.
4. The database sends the response to Tasks.
5. Tasks sends the response to API service.
6. API service returns information.

Request	Description	Method
delete task	Delete the specified task and its results.	DEL

Request	Description	Method
get task result	Receive results of the specified task.	GET

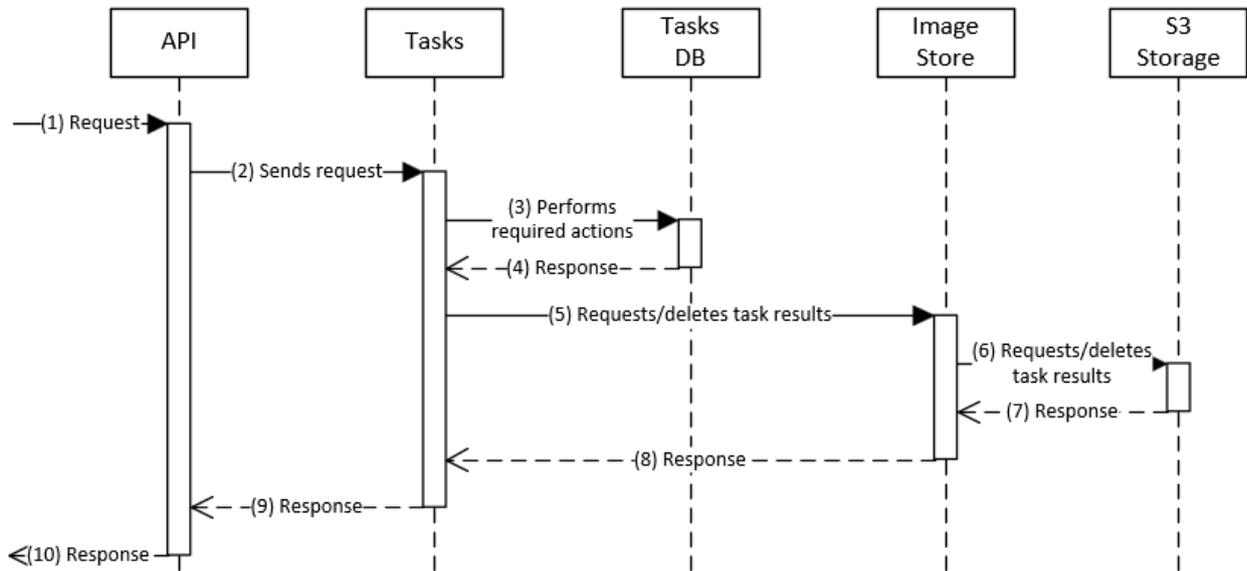


Figure 90: Delete task/get task result

1. The request is sent to API.
2. API sends the request to Tasks to receive the required task data or delete a task.
3. Tasks sends the request to the Tasks database to receive the required data or delete a task.
4. The database sends the response to Tasks.
5. Tasks sends the request to Image Store to receive or delete the results of the specified task.
6. Image Store sends the request to the storage.
7. The storage sends the response to Image Store.
8. Image Store returns the result to Tasks.
9. Tasks sends the response to API service.
10. API service sends the response.

9.8 Lambda diagrams

9.8.1 Lambda creation diagram

This diagram describes the general process of creating lambda.

Request	Response	Method
create lambda	Create lambda	POST

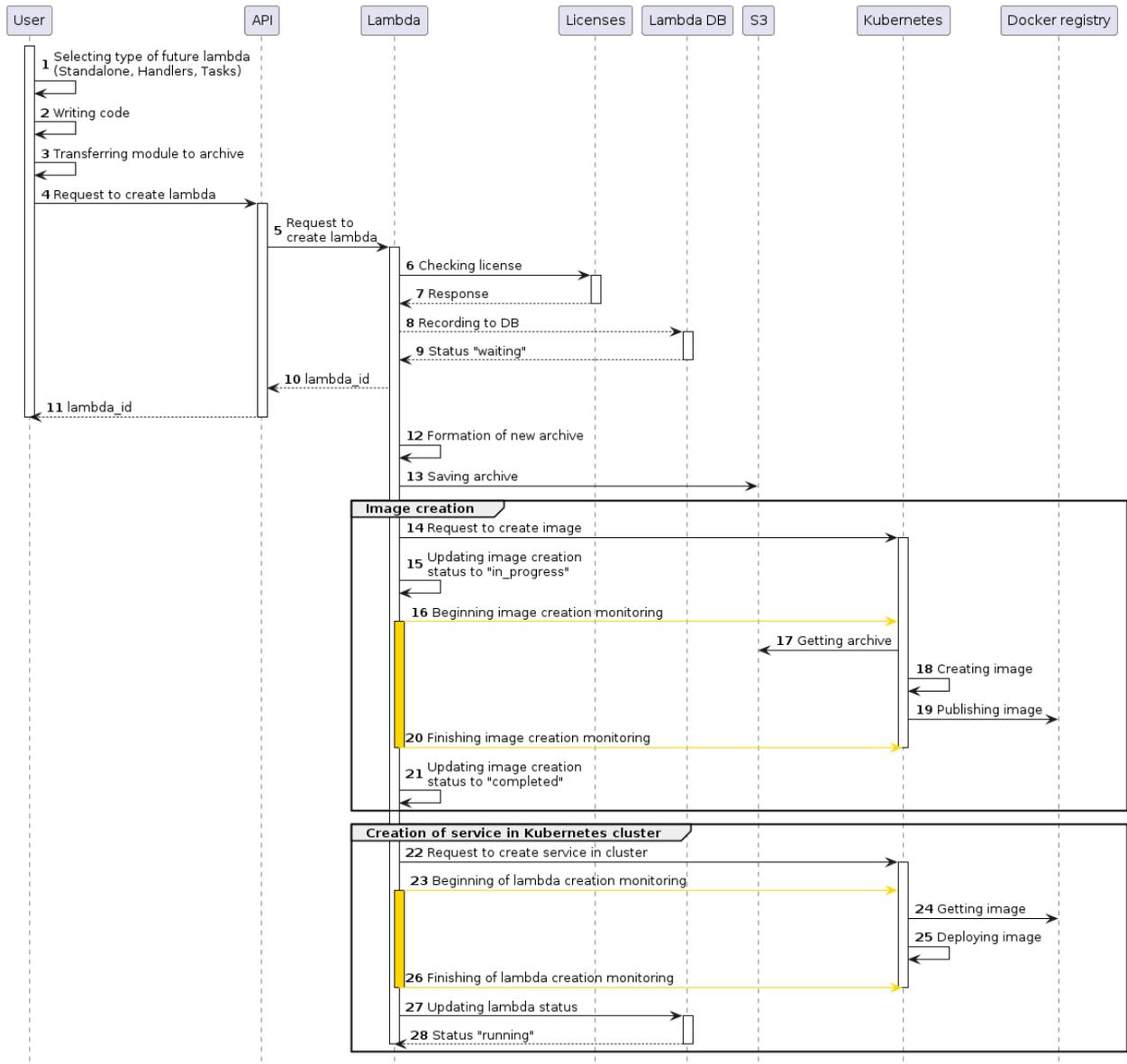


Figure 91: Lambda creation diagram

General scheme of request processing:

1. The user determines which type of lambda he will need — [Handlers-lambda](#), [Standalone-lambda](#), [Tasks-lambda](#).
2. The user writes Python code according to the type of future lambda and [code requirements](#).

3. The user transfers the module to the archive in accordance with [archive requirements](#).
4. The user performs a request “[create lambda](#)” to the API service.
5. The API service sends a request to the Lambda service.
6. The Lambda service sends a request for license check to the Licenses service.
7. The Lambda service receives a response.
8. The Lambda service writes data to the Lambda database, updating the lambda creation status to “waiting”.
9. The Lambda service receives a response.

During lambda creation, it is possible to get the status of its creation using the request “[get lambda status](#)”.

10. The Lambda service returns the “lambda_id” to the API service.
11. The API service returns the “lambda_id” to the user.
12. The Lambda service adds additional files and creates a new archive.
13. The Lambda service saves a new archive in S3 storage.

Image creation

14. The Lambda service sends an image creation request to Kubernetes.
15. The Lambda service updates the image creation status to “in_progress”.
16. The Lambda service starts monitoring the status of image creation in Kubernetes.

The user can get the status and logs of image creation using the requests “[get lambda image creation status](#)” and “[get lambda image creation logs](#)”.

17. Kubernetes gets the archive from S3 storage.
18. Kubernetes creates an image.
19. Kubernetes publishes the image in Docker registry.

S3, Docker registry and Kubernetes settings are set in the Lambda service settings.

20. The Lambda service records the publication of the image.
21. The Lambda service updates the image creation status to “completed”.

Creation of service in Kubernetes cluster

22. The Lambda service sends a request to create a lambda in the Kubernetes cluster.
23. The Lambda service starts monitoring the status of lambda creation in Kubernetes.

During lambda creation, it is not possible to get the status and logs of image creation. The lambda creation status and logs can be obtained using the requests [“get lambda status”](#) and [“get lambda logs”](#).

24. Kubernetes gets the image from Docker registry.
25. Kubernetes expands the resulting image.
26. The Lambda service captures the state of the deployed image.
27. The Lambda service updates the lambda status to “running” in the Lambda database.
28. The Lambda service receives a response.

Then you can start sending lambda requests.

9.8.2 Lambda processing diagram

This diagram describes the general lambda processing process. The lambda processing process depends on its type — Standalone-lambda, Handlers-lambda or Tasks-lambda.

Processing of the Tasks-lambda type is not reflected in this diagram. It is performed according to the [general task creation process](#), **except** that the Lambda service is used instead of the Tasks worker.

For Handlers-lambda and Standalone-lambda types, you can run [“proxy post request to lambda”](#) requests for direct interaction with Kubernetes. If Handlers-lambda can mimic the response of a classic handler, then you can perform the [“generate events”](#) request.

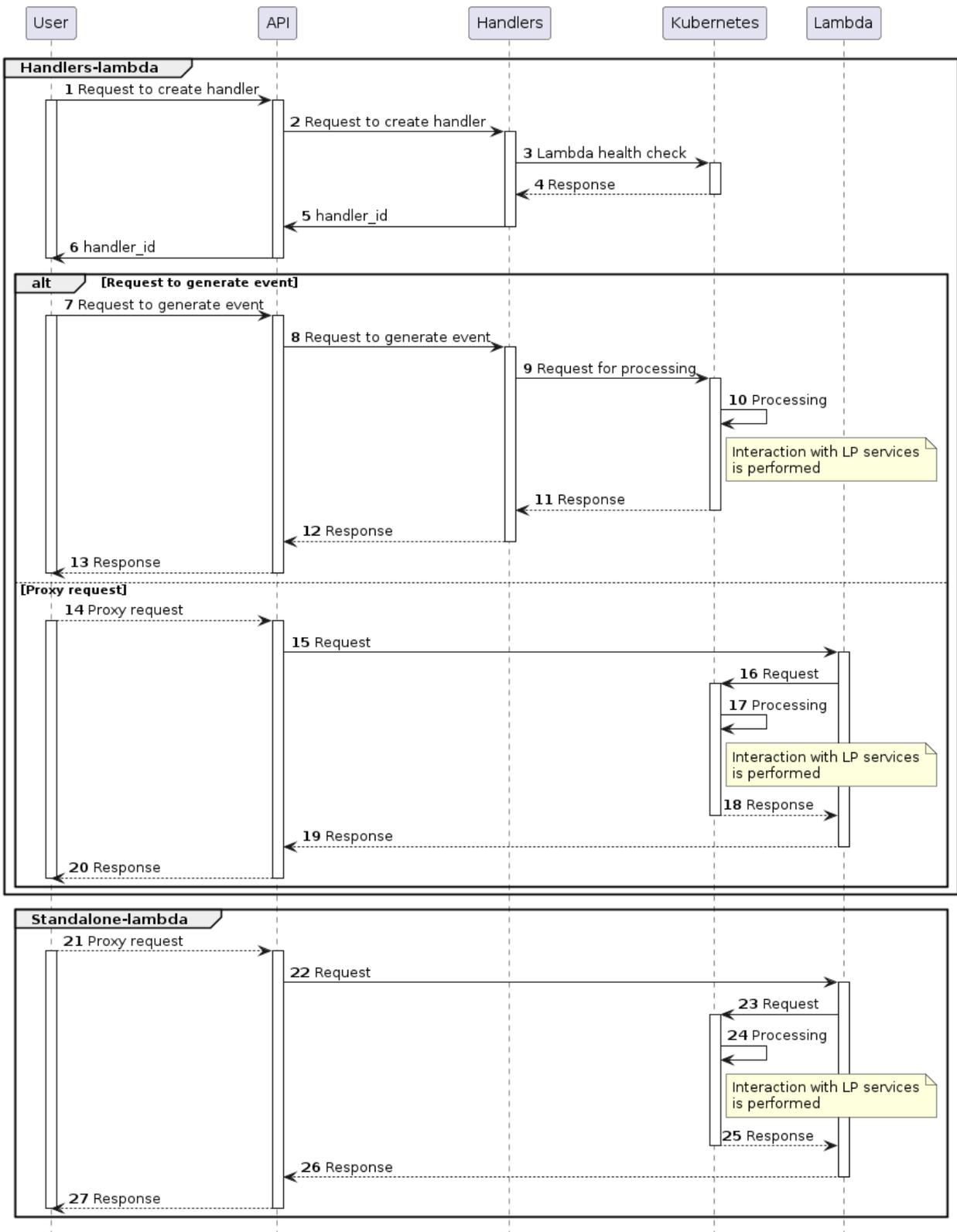


Figure 92: Lambda processing diagram

General request processing diagram:

Handlers-lambda

1. The user performs a “[create handler](#)” request to the API service, specifying “handler_type” = “2” and the resulting “lambda_id” (see “[Lambda creation diagram](#)”).
2. The API service sends a request to the Handlers service.
3. The Handlers service performs a healthcheck of the lambda deployed in Kubernetes and creates a handler.
4. The Handlers service receives a response.
5. The API service receives the “handler_id”.
6. The user receives the “handler_id”.

Request to generate an event

7. The user performs a “[generate events](#)” request to the API service, specifying the “handler_id” received in the previous step.
8. The API service sends a request to the Handlers service.
9. The Handlers service sends a request for processing to Kubernetes.
10. The Kubernetes cluster processes user code.

During the operation of Handlers-lambda, interaction with some LUNA PLATFORM services will be performed. See the section “[Handlers-lambda](#)” for more information.

11. The Handlers service receives a response.
12. The API service receives a response.
13. The user receives a response.

Proxy request

14. The user performs a “[proxy pass request to lambda](#)” request to the API service.
15. The API service sends a request to the Lambda service.
16. The Lambda service sends a request for processing to Kubernetes.
17. The Kubernetes cluster processes user code.

During the operation of Handlers-lambda, interaction with some LUNA PLATFORM services will be performed. See the section “[Handlers-lambda](#)” for more information.

18. The Lambda service receives a response.
19. The API service receives a response.

20. The user receives a response.

Proxy request for Standalone-lambda

21. The user sends a “[proxy pass request to lambda](#)” request to the API service.

22. The API service sends a request to the Lambda service.

23. The Lambda service sends a request for processing to Kubernetes.

24. The Kubernetes cluster processes user code.

During the operation of Handlers-lambda, interaction with some LUNA PLATFORM services will be performed. See the section “[Standalone-lambda](#)” for more information.

25. The Lambda service receives a response.

26. The API service receives a response.

27. The user receives a response.

10 Database description

All the data of the timestamp type is stored in the RFC 3339 format.

The time used for storing data in the database can be set in the configuration file for each service in the `“STORAGE_TIME”` parameter. You can select `“LOCAL”` or `“UTC”` storage time.

When the `“STORAGE_TIME”` is set to `“LOCAL”` but you receive time in `“UTC”`, data is converted to local time. When `“UTC”` is set but the received time is local it is also converted to `“UTC”`.

10.1 Faces database description

This section describes the Faces database fields.

For more information, see [“Faces service”](#).

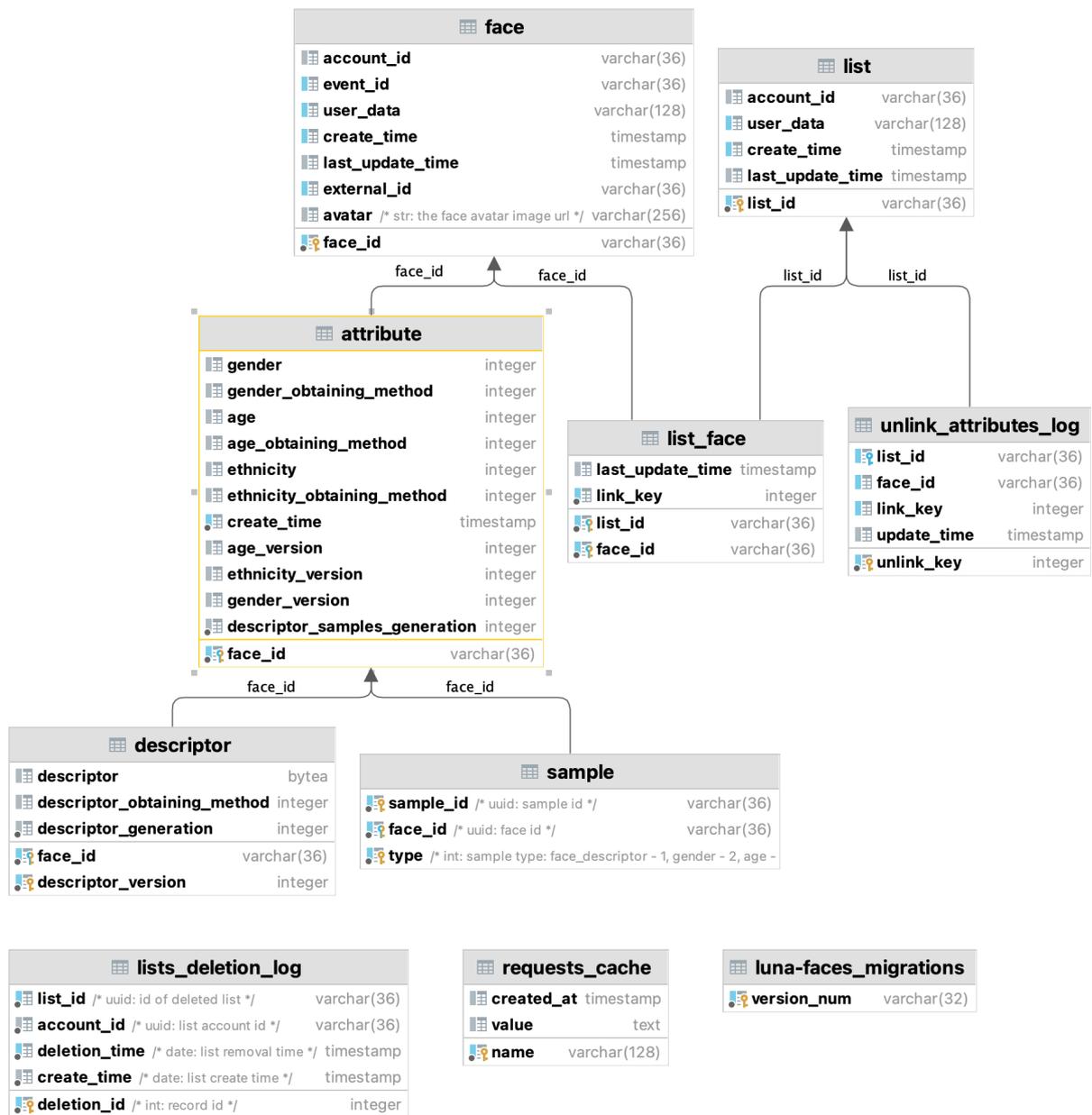


Figure 93: Faces database schema

10.1.1 Attribute table model

The database table model describes face attributes linked to faces.

Name	Type	Description
face_id	varchar(36)	Face ID.

gender	integer	Result of gender estimation based on face image: <ul style="list-style-type: none"> • “0” – Male • “1” – Female
gender_obtaining_method	integer	Algorithm used to estimate gender based on face image.
gender_version	integer	Version of the algorithm for estimating gender based on face image.
age	integer	Result of age estimation based on face image.
age_obtaining_method	integer	Algorithm used to estimate age based on face image.
age_version	integer	Version of the algorithm for estimating age based on face image.
ethnicity	integer	Result of ethnicity estimation.
ethnicity_obtaining_method	integer	Version of the algorithm for estimating ethnicity.
ethnicity_version	integer	Version of the algorithm for estimating ethnicity.
create_time	timestamp	Attribute creation date and time.
account_id	varchar(36)	Account ID to which the attribute belongs.
descriptor_samples_generation	integer	Generation of the samples used. This value changes when the attribute sample is updated. The initial value is 0.

10.1.2 Descriptor table model

The database table model describes descriptors.

Name	Type	Description
attribute_id	varchar(36)	Attribute ID.
descriptor_version	integer	Version of NN that was used to extract the descriptor.
descriptor	bytea	Binary descriptor.
descriptor_obtaining_method	integer	Algorithm that was used to obtain the descriptor.

descriptor_ generation	integer	Generation of the descriptor. This value changes when the attribute descriptor is updated. It shows that the descriptor does not match existing samples. The initial value is 0.
---------------------------	---------	--

10.1.3 Face table model

The database table model describes existing faces.

Name	Type	Description
face_id	varchar(36)	Face ID.
account_id	varchar(36)	Account ID to which the face belongs.
event_id	varchar(36)	Event ID. A reference to the event that created the face.
user_data	varchar(128)	User-defined data for the face.
create_time	timestamp	Time and date of the face creation.
last_ update_ time	timestamp	Time and date of the last face update.
external_id	varchar(36)	External ID. External ID is specified in the create face request or in the event creation request ("face_policy").
avatar	varchar(256)	URL of a photo image that corresponds to the face.

10.1.4 List table model

The database table model describes existing lists.

Name	Type	Description
list_id	varchar(36)	List ID.
account_id	varchar(36)	Account ID to which the list belongs.
user_data	varchar(128)	User data for the list.
create_time	timestamp	Time and date when the list was created.
last_update_time	timestamp	Time and date of the last list update.

10.1.5 List_face table model

The database table model describes the history of attaching faces to lists. If a face was attached to a list a new record appears.

Name	Type	Description
list_id	varchar(36)	List ID.
face_id	varchar(36)	Face ID.
last_update_ time	timestamp	Date and time of the last linking of the face to the list.
link_key	integer	Sequence number of linking the face to the list.

10.1.6 Unlink_attributes_log table model

The database table model describes the history of detaching faces from lists. If a face was detached from a list, a new record appears.

Name	Type	Description
unlink_key	integer	Sequence number of the face and the list unlinking.
list_id	varchar(36)	List ID.
face_id	varchar(36)	Face ID.
link_key	integer	Sequence number of linking of the face and the list.
update_ time	timestamp	Date and time of the last detach of the face from the list.

10.1.7 Sample table model

The database table model describes links between samples and faces.

Name	Type	Description
sample_id	varchar(36)	Sample.
face_id	varchar(36)	Face ID related to the sample.

type	integer	Method of using the sample: <ul style="list-style-type: none"> • “1” — To extract descriptor. • “5” — To create basic attributes.
------	---------	---

10.1.8 Unlink_deletion_log table model

The database table model describes the history of deleting lists. If the list has been deleted, a new entry appears.

Name	Type	Description
list_id	varchar(36)	List ID.
account_id	varchar(36)	Account ID to which the list was attached.
deletion_time	timestamp	Date and time when the list was deleted.
create_time	timestamp	Date and time when the list was created.
deletion_id	integer	Deletion ID.

10.1.9 Requests_cache table model

The database table model describes the cache of the maximum number of faces with associated descriptors or basic attributes for license requests.

Name	Type	Description
created_at	timestamp	Date and time of data caching.
value	text	Encrypted value cache.
name	varchar(36)	Unique cache name.

10.1.10 Luna-faces_migrations table model

Name	Type	Description
version_num	varchar(32)	Parameter required for database migration.

10.2 Events database description

This section describes the fields in the Events database.

For more information, see “Events service”.

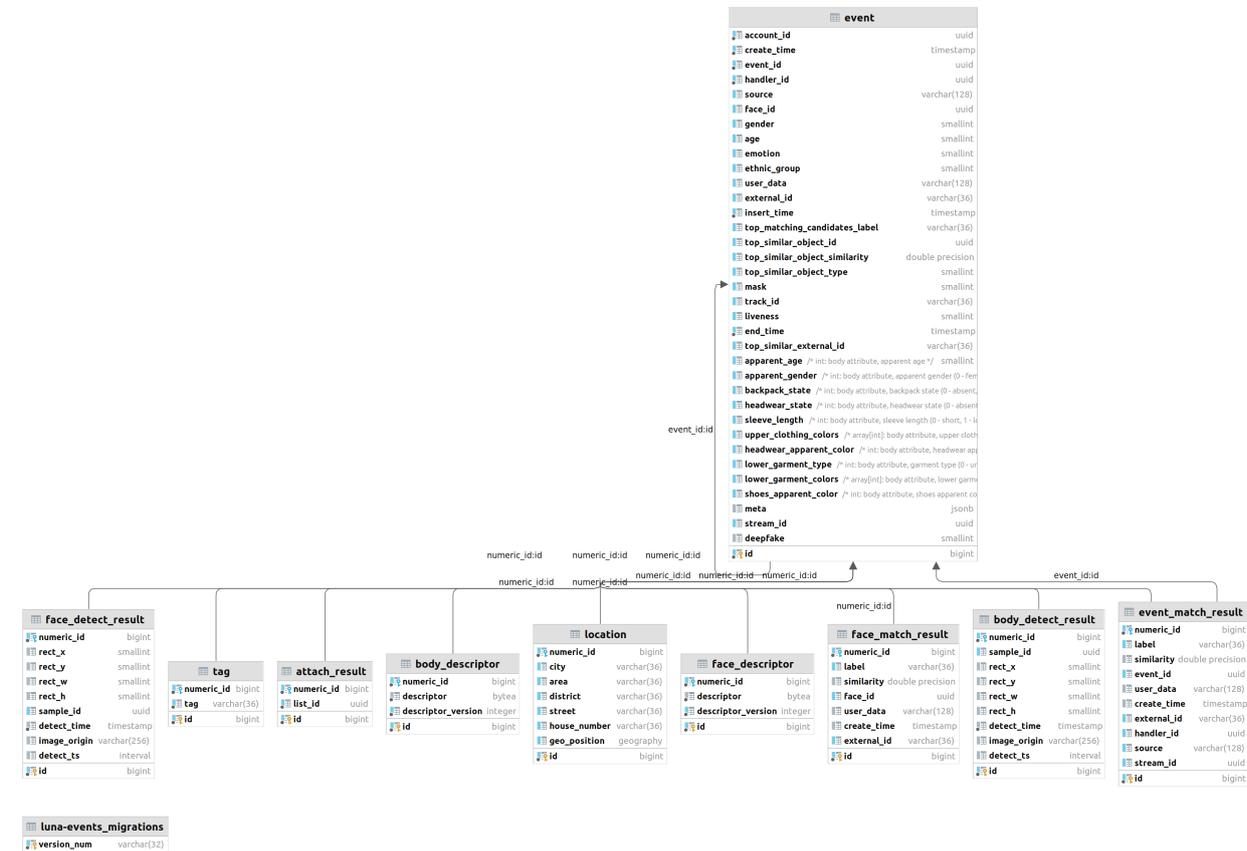


Figure 94: Events database schema

10.2.1 Event table model

The database table model describes existing events. It includes information about created events and faces.

Name	Type	Description
id	bigint	Primary key of the table (digital).
account_id	uuid	Account ID to which the event belongs.
create_time	timestamp	Time code of the event occurrence in the video stream. The parameter is used for the real-time monitoring of events creation.
event_id	uuid	Event ID.

handler_id	uuid	Handler ID that created the event.
source	varchar(128)	Event source. The source is specified in the request for event creation.
face_id	uuid	Face ID of the face corresponding to the event.
gender	smallint	Result of gender estimation based on face image.
age	smallint	Result of age estimation based on face image.
emotion	smallint	Result of emotions estimation.
ethnic_group	smallint	Result of ethnic group estimation.
user_data	varchar(128)	User data for the face corresponding to the event. User data is specified in the request for event creation.
external_id	varchar(36)	External ID of the face corresponding to the event. External ID is specified in the request for event creation.
insert_time	timestamp	Date and time of the event creation.
top_matching_candidates_label	varchar(36)	Label of the group of the candidates used for matching.
top_similar_object_id	uuid	ID of the top similar object from matching results. Matching results are received when the “match_policy” of a handler is enabled.
top_similar_object_similarity	double precision	Similarity score of the top similar object from matching results. Matching results are received when the “match_policy” of a handler is enabled.
top_similar_object_type	smallint	Type of the top similar object: <ul style="list-style-type: none"> • “0” — Face • “1” — Event
mask	smallint	Result of medical mask estimation: <ul style="list-style-type: none"> • “1” — Missing • “2” — Medical mask • “3” — Occluded
track_id	varchar(36)	Track ID. The ID can be specified in the event generation request.

liveness	smallint	Result of Liveness check: <ul style="list-style-type: none"> • “0” — Spoof • “1” — Real • “2” — Unknown
end_time	timestamp	End time code of the event occurrence in the video stream. The parameter is used for the real-time monitoring of events creation. Set equal to “create_time” if not set.
top_similar_external_id	varchar(36)	External ID of the top similar object from matching results. Matching results are received when the “match_policy” of a handler is enabled.
apparent_age	smallint	Result of age estimation based on body image.
apparent_gender	smallint	Result of gender estimation based on body image: <ul style="list-style-type: none"> • “0” — Female • “1” — Male • “2” — Unknown
backpack_state	smallint	Result of backpack state estimation: <ul style="list-style-type: none"> • “0” — Absent • “1” — Present • “2” — Unknown
headwear_state	smallint	Result of headwear state estimation: <ul style="list-style-type: none"> • “0” — Absent • “1” — Present • “2” — Unknown
sleeve_length	smallint	Result of sleeve length estimation: <ul style="list-style-type: none"> • “0” — Short • “1” — Long • “2” — Undefined

upper_ clothing_color	array[int]	Result of upper clothing color estimation: <ul style="list-style-type: none">• “0” – Unknown• “1” – Black• “2” – Blue• “3” – Green• “4” – Gray• “5” – Orange• “6” – Purple• “7” – Red• “8” – White• “9” – Yellow• “10” – Pink• “11” – Brown• “12” – Beige• “13” – Khaki• “14” – Multicolored
--------------------------	------------	--

headwear_ apparent_color	smallint	Result of headwear apparent color estimation: <ul style="list-style-type: none">• “0” – Unknown• “1” – White• “2” – Black• “3” – Other
-----------------------------	----------	---

lower_ garment_type	smallint	Result of lower garment type estimation: <ul style="list-style-type: none">• “0” – Unknown• “1” – Trousers• “2” – Shorts• “3” – Skirt
------------------------	----------	--

lower_ garment_colors	array[int]	Result of lower garment colors estimation: <ul style="list-style-type: none"> • “0” – Unknown • “1” – Black • “2” – Blue • “3” – Green • “4” – Gray • “5” – Orange • “6” – Purple • “7” – Red • “8” – White • “9” – Yellow • “10” – Pink • “11” – Brown • “12” – Beige • “13” – Khaki • “14” – Multicolored
--------------------------	------------	--

shoes_ apparent_color	smallint	Result of shoes apparent color estimation: <ul style="list-style-type: none"> • “0” – Unknown • “1” – White • “2” – Black • “3” – Other
--------------------------	----------	---

meta	jsonb	User meta-information.
------	-------	------------------------

deepfake	smallint	Result of Deepfake estimation: <ul style="list-style-type: none"> • “0” – Spoof • “1” – Real
----------	----------	--

10.2.2 Face_detect_result table model

The database table model describes face detection.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_ id	integer	Foreign key of an event.

Name	Type	Description
rect_x	smallint	Top left corner coordinate of the face bounding rectangle by the “X” axis.
rect_y	smallint	Top left corner coordinate of the face bounding rectangle by the “Y” axis.
rect_w	smallint	Width of the face bounding rectangle.
rect_h	smallint	Height of the face bounding rectangle.
sample_id	varchar(36)	Sample ID.
detect_ time	timestamp	Time of the face detection.
image_ origin	varchar(256)	URL to the source image where the face has occurred.
detect_ts	interval	Time relative to something, such as relative to the beginning of a video file.

10.2.3 Body_detect_result table model

The database table model describes face detection.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_ id	integer	Foreign key of an event.
rect_x	smallint	Top left corner coordinate of the body bounding rectangle by the “X” axis.
rect_y	smallint	Top left corner coordinate of the body bounding rectangle by the “Y” axis.
rect_w	smallint	Width of the body bounding rectangle.
rect_h	smallint	Height of the body bounding rectangle.
sample_id	varchar(36)	Sample ID.
detect_ time	timestamp	Time of the body detection.
image_ origin	varchar(256)	URL to the source image where the body has occurred.

Name	Type	Description
detect_ts	interval	Time relative to something, such as relative to the beginning of a video file.

10.2.4 Face_descriptor table model

The database table model describes face descriptors stored in the database.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_id	bigint	Foreign key of the event.
descriptor	bytea	Binary face descriptor.
descriptor_version	integer	The version of NN that was used to extract the descriptor.

10.2.5 Body_descriptor table model

The database table model describes body descriptors stored in the database.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_id	bigint	Foreign key of the event.
descriptor	bytea	Binary body descriptor.
descriptor_version	integer	The version of NN that was used to extract the descriptor.

10.2.6 Event_match_result table model

The database table model describes matching results received using the matching policy of handler. Each record includes information about an event used for matching and the estimated similarity.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_id	bigint	Foreign key of the event.
id		

Name	Type	Description
label	varchar(36)	The label specified for the matching results.
similarity	double precision	The similarity score received after matching of the event descriptor with the given descriptor.
event_ id	uuid	Event ID.
user_ data	varchar(128)	User data associated with the event.
create_ time	timestamp	Event creation time.
external_ id	varchar(36)	External ID of the face created during event creation.
handler_ id	uuid	ID used for the event creation .
source	varchar(128)	Event source.

10.2.7 Face_match_result table model

The database table model describes matching results received using the matching policy of handler. Each record includes information about the face used for matching and the estimated similarity.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_ id	bigint	Foreign key of the event.
label	varchar(36)	The label specified for the matching results.
similarity	double precision	The similarity score received after matching of the event descriptor with the given face descriptor.
face_id	uuid	Face ID.
user_ data	varchar(128)	User data associated with the face.
create_ time	timestamp	Face creation time.

Name	Type	Description
external_id	varchar(36)	External ID of the face.

10.2.8 Location table model

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_id	bigint	Foreign key of the event.
city	varchar(36)	The city of event occurrence.
area	varchar(36)	The area of event occurrence.
district	varchar(36)	The district of event occurrence.
street	varchar(128)	The street of event occurrence.
house_number	varchar(36)	The house number where the event occurred.
geo_position	geography	Geographical coordinate of the event occurrence (longitude, latitude).

10.2.9 Tag table model

The database table model describes tags for events. Tags are specified in an event creation request.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_id	integer	Foreign key of an event.
tag	varchar(36)	Event tag.

10.2.10 Attach_result table model

The database table model describes the attachment of the face created from an event to a list. The face is created using “face_policy”. The face is attached to a list using “link_to_lists_policy”.

Name	Type	Description
id	bigint	Primary key of the table (digital).
numeric_id	integer	Foreign key of an event.
list_id	uuid	List to which the created face was attached.

10.3 Tasks database

This section describes the fields in the Tasks database.

For more information, see “Tasks service”.

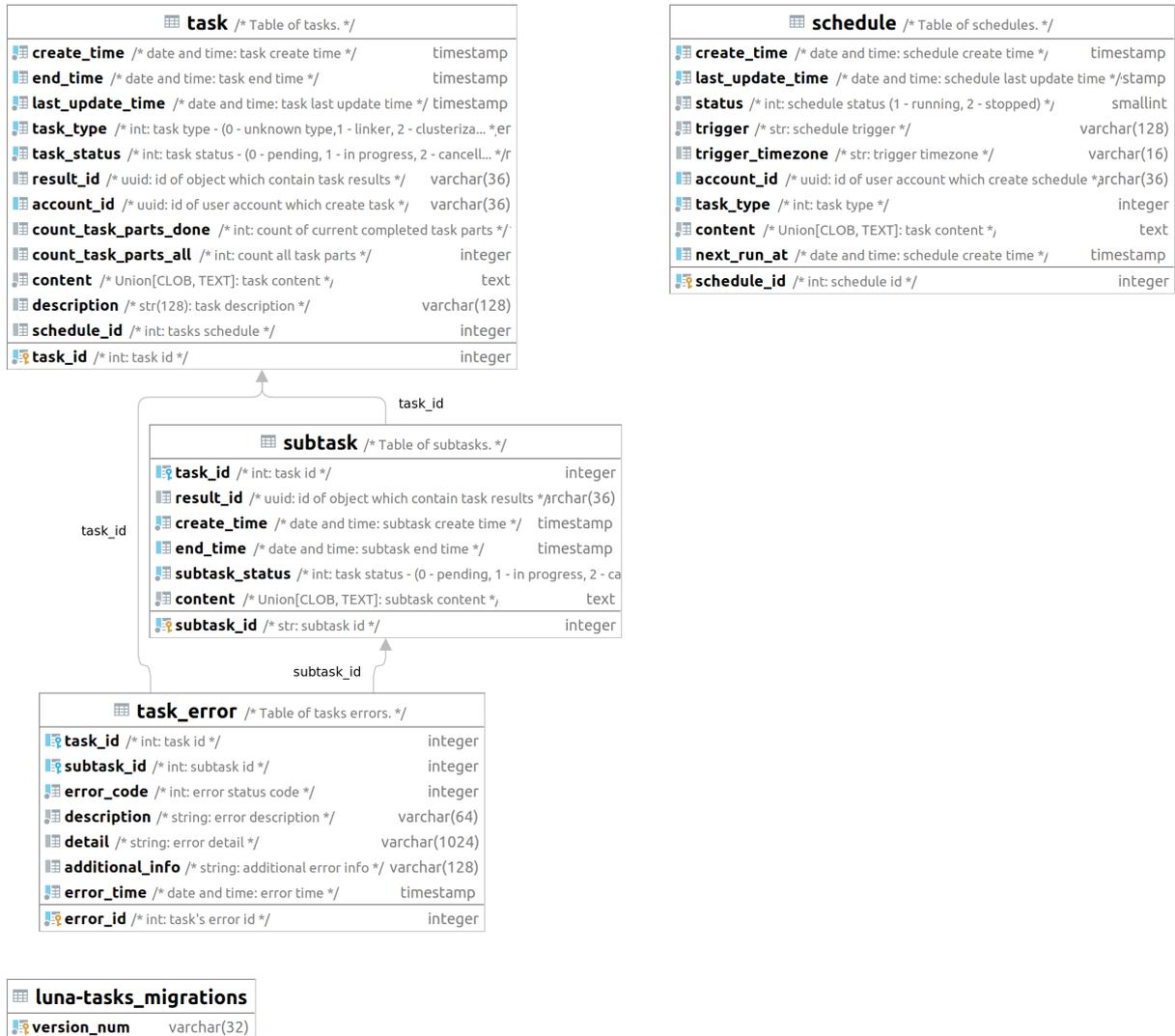


Figure 95: Tasks database schema

10.3.1 Task table model

The database table model describes the created tasks. It includes general information about a task and its content.

Name	Type	Description
------	------	-------------

task_id	integer	Task ID.
create_time	timestamp	Task creation time.
end_time	timestamp	Task end time.
last_update_time	timestamp	Task last update time.
task_type	integer	Task type: <ul style="list-style-type: none"> • “0” – Unknown • “1” – Linker • “2” – Clustering • “3” – Reporter • “4” – Garbage collection • “5” – Additional extraction • “6” – Cross-matching • “7” – ROC-curve calculating • “8” – Exporter • “9” – Estimator
task_status	integer	Task status: <ul style="list-style-type: none"> • “0” – Pending • “1” – In progress • “2” – Cancelled • “3” – Failed • “4” – Collect results • “5” – Done
result_id	varchar(36)	Task result ID.
account_id	varchar(36)	Account ID to which the task belongs.
count_task_parts_done	integer	Number of finished subtasks.
count_task_parts_all	integer	Total number of subtasks.
content	text	Filters and request parameters for the task.
description	varchar(128)	User-defined task description.
schedule_id	integer	Schedule ID.

10.3.2 Subtask table model

The database table model includes information about created subtasks. Depending on a task type there may be one or several subtasks.

Name	Type	Description
subtask_id	integer	Subtask ID.
task_id	integer	ID of the corresponding task.
result_id	varchar(36)	ID of the subtask result.
create_time	timestamp	Subtask creation time.
end_time	timestamp	Subtask end time.
subtask_status	integer	Subtask status: <ul style="list-style-type: none">• “0” – Pending• “1” – In progress• “2” – Cancelled• “3” – Failed• “4” – Collect results• “5” – Done
content	text	Filters and request parameters for a subtask processing.

10.3.3 Task_error table model

The database table model includes information about errors occurred during task processing. Errors are added to the table by Tasks workers.

Name	Type	Description
error_id	integer	Task error ID.
task_id	integer	ID of the corresponding task.
error_code	integer	Error code.
description	varchar(64)	Error short description.
detail	varchar(1024)	Detailed description of the error.
additional_info	varchar(128)	Additional information about the error. It may include lost object IDs or any other useful information.

Name	Type	Description
error_time	timestamp	Time when the error occurred.

10.3.4 Schedule table model

The database table model includes information about the task execution [schedule](#).

Name	Type	Description
schedule_id	integer	Schedule ID.
create_time	timestamp	Date the schedule was created.
last_update_time	timestamp	Date and time of the last schedule changes.
status	smallint	Schedule status: <ul style="list-style-type: none"> • “1” – Started • “2” – Stopped
trigger	varchar(128)	Cron expression.
trigger_timezone	varchar(16)	Time zone (UTC or LOCAL).
account_id	varchar(36)	Account ID that the schedule belongs to.
task_type	integer	Type of task that runs on a schedule.
content	text	Content of the scheduled task (filters, list, etc.)
next_run_at	timestamp	Time of the next task start.

10.3.5 Luna-tasks_migrations table model

Name	Type	Description
version_num	varchar(32)	Parameter required for database migration.

10.4 Handlers database

This section describes the fields in the Handlers database.

For more information about the Handlers service, see [“Handlers service”](#).

handler	
account_id	varchar(36)
create_time	timestamp
last_update_time	timestamp
description	varchar(128)
policies	varchar
handler_type	smallint
lambda_id	varchar(36)
handler_id	varchar(36)

verifier	
account_id	varchar(36)
create_time	timestamp
last_update_time	timestamp
description	varchar(128)
policies	varchar
version	integer
verifier_id	varchar(36)

luna-handlers_migrations	
version_num	varchar(32)

Figure 96: Handlers database schema

10.4.1 Handler table model

Name	Type	Description
handler_id	varchar(36)	UUID4 standard handler ID in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxxx” format.

account_id	varchar(36)	Account ID in UUID4 format that the handler belongs to.
create_time	timestamp	Date and time of handler creation.
last_update_time	timestamp	Date and time of last change of the handler.
description	varchar(128)	Handler description provided by the user.
handler_type	smallint	Handler type: <ul style="list-style-type: none"> • “0” – Static • “1” – Dynamic • “2” – Lambda
lambda_id	smallint	Lambda ID.
policies	varchar(36)	Handler policies.

10.4.2 Verifier table model

Name	Type	Description
account_id	varchar	Account ID in UUID4 format that the verifier belongs to.
create_time	timestamp	Date and time of verifier creation.
description	varchar	Verifier description provided by the user.
verifier_id	varchar(36)	UUID4 standard verifier ID in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxx” format.
last_update_time	timestamp	Date and time of the latest handler modification.
policies	varchar(2048)	Verifier policies.
version	integer	Verifier version.

10.4.3 Luna-handlers_migrations table model

Name	Type	Description
version_num	varchar(32)	Parameter required for database migration.

10.5 Configurator database

This section describes the fields of the Configurator database.

For more information, see “[Configurator service](#)”.

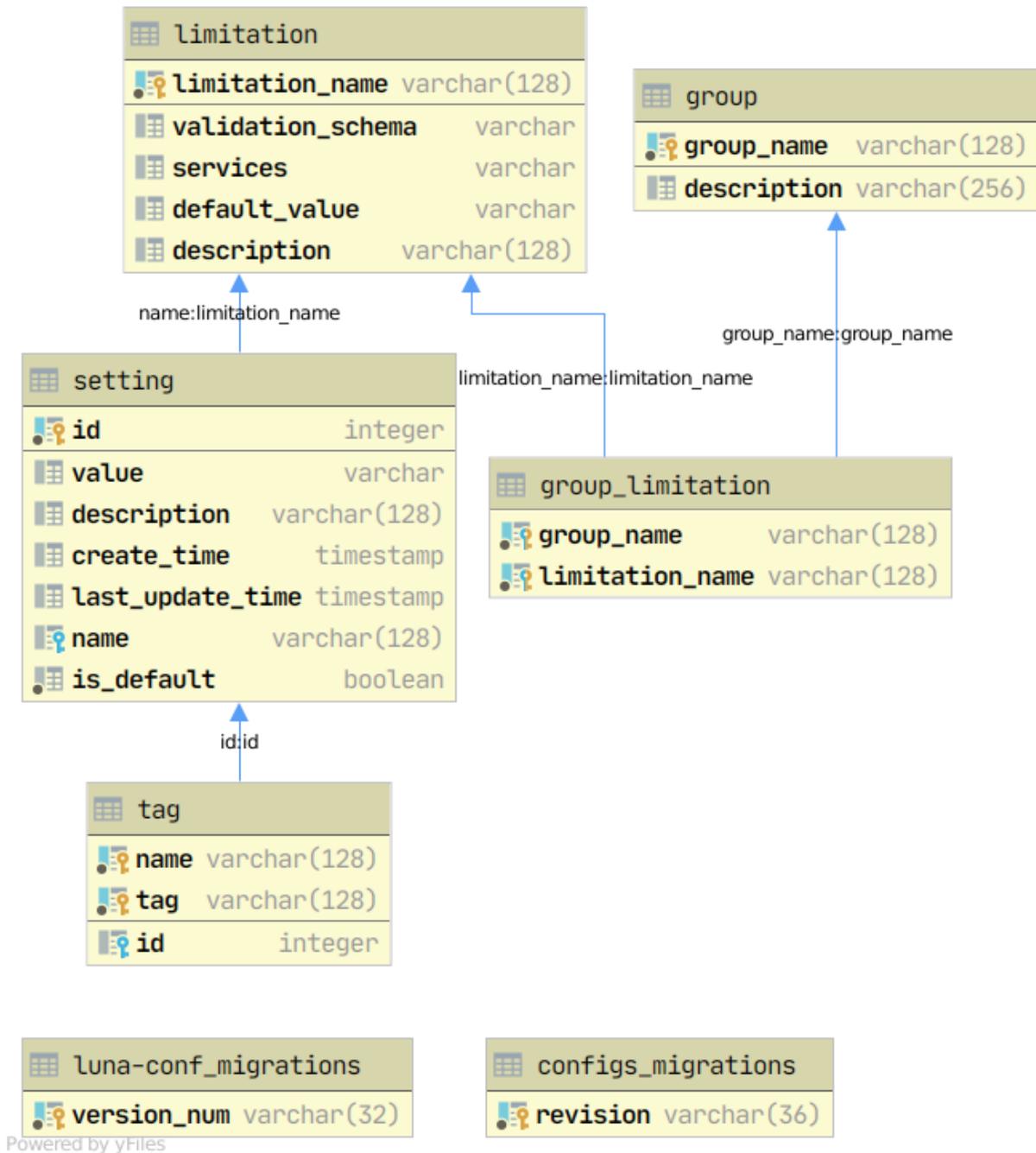


Figure 97: Configurator database schema

10.5.1 Limitation table model

Name	Type	Description
limitation_name	varchar(128)	Name of the limitation.
validation_schema	varchar	Limitation validation schema.
services	varchar	List of services.
default_value	varchar	Default limitation value.
description	varchar(128)	Limitation description.

10.5.2 Settings table model

Name	Type	Description
id	integer	Setting ID.
value	varchar	Setting value.
description	varchar(128)	Settings description.
create_time	timestamp	Setting creation time.
last_update_time	timestamp	Latest time of setting modification.
name	varchar(128)	Setting name.
is_default	boolean	Whether the setting is default.

10.5.3 Tag table model

Name	Type	Description
id	integer	Setting ID.
name	varchar(128)	Setting name.
tag	varchar(128)	Setting tag string.

10.5.4 Group table model

Name	Type	Description
group_name	varchar(128)	Group name.

Name	Type	Description
description	varchar(256)	Group description.

10.5.5 Group_limitation table model

Name	Type	Description
group_name	varchar(128)	Group name.
limitation_name	varchar(128)	Limitation name.

10.5.6 Configs_migration table model

Name	Type	Description
revision	varchar(36)	Revision of settings migration.

10.5.7 Luna-conf_migrations table model

Name	Type	Description
version_num	varchar(32)	Parameter required for database migration.

10.6 Backport3 database

This section describes the fields of the Backport 3 database.

For more information, see “[Backport 3 service](#)”.

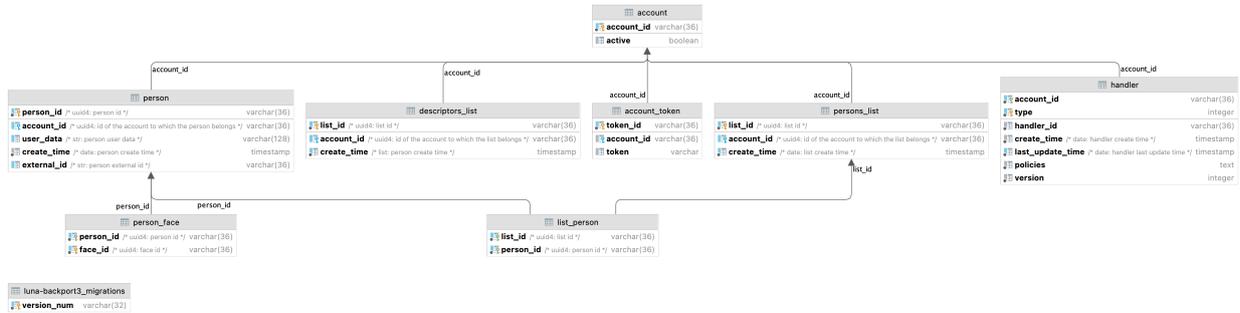


Figure 98: Backport3 database schema

10.6.1 Account table model

Database table model for account.

Name	Type	Description
account_id	varchar(36)	UUID4 standard account ID.
active	boolean	Account status.

10.6.2 Account_token model table

Name	Type	Description
token_id	varchar(36)	Token ID.
account_id	varchar(36)	Account ID, the token is linked to.
token_info	varchar(128)	A string with token data.

10.6.3 Person table model

Name	Type	Description
person_id	varchar(36)	UUID4 standard person ID, in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxx” format.

Name	Type	Description
account_ id	varchar(36)	UUID4 standard ID of the account to which the person belongs, “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxxx” format.
user_ data	varchar(128)	Person user data.
create_ time	timestamp	Date and time of person creation.
external_ id	varchar(36)	Person ID in external system.

10.6.4 Persons_list table model

Name	Type	Description
list_id	varchar(36)	UUID4 standard list ID in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxxx” format.
account_ id	varchar(36)	UUID4 standard account ID, in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxxx” format.
create_ time	timestamp	Date and time of list creation.

10.6.5 Descriptors_list table model

Name	Type	Description
list_id	varchar(36)	UUID4 standard list ID in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxxx” format.
account_ id	varchar(36)	UUID4 standard account ID, the list belongs to, in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxxx” format.
create_ time	timestamp	Date and time of list creation.

10.6.6 List_person table model

Name	Type	Description
list_id	varchar(36)	UUID4 standard list ID in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxxx” format.

Name	Type	Description
person_id	varchar(36)	UUID4 standard person ID, in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxx” format.

10.6.7 Person_face table model

Name	Type	Description
person_id	varchar(36)	UUID4 standard person ID, in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxx” format.
face_id	varchar(36)	UUID4 standard face ID, in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxx” format.

10.6.8 Luna-backport3_migrations table model

Name	Type	Description
version_num	varchar(32)	Parameter required for database migration.

10.6.9 Handler table model

Name	Type	Description
account_id	varchar(36)	UUID4 standard account ID, the handler belongs to, in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxx” format.
type	integer	Handler type.
handler_id	varchar(36)	UUID4 standard handler ID in “xxxxxxxx-xxxx-4xxx-{8-9}xx-xxxxxxxxxxxx” format.
create_time	timestamp	Date and time of handler creation.
last_update_time	timestamp	Date and time of the latest handler modification.
policies	varchar(2048)	handler policies.
version	integer	Handler version.

10.7 Accounts database

This section describes the fields in the Accounts database.

For more information, see [“Accounts service”](#).

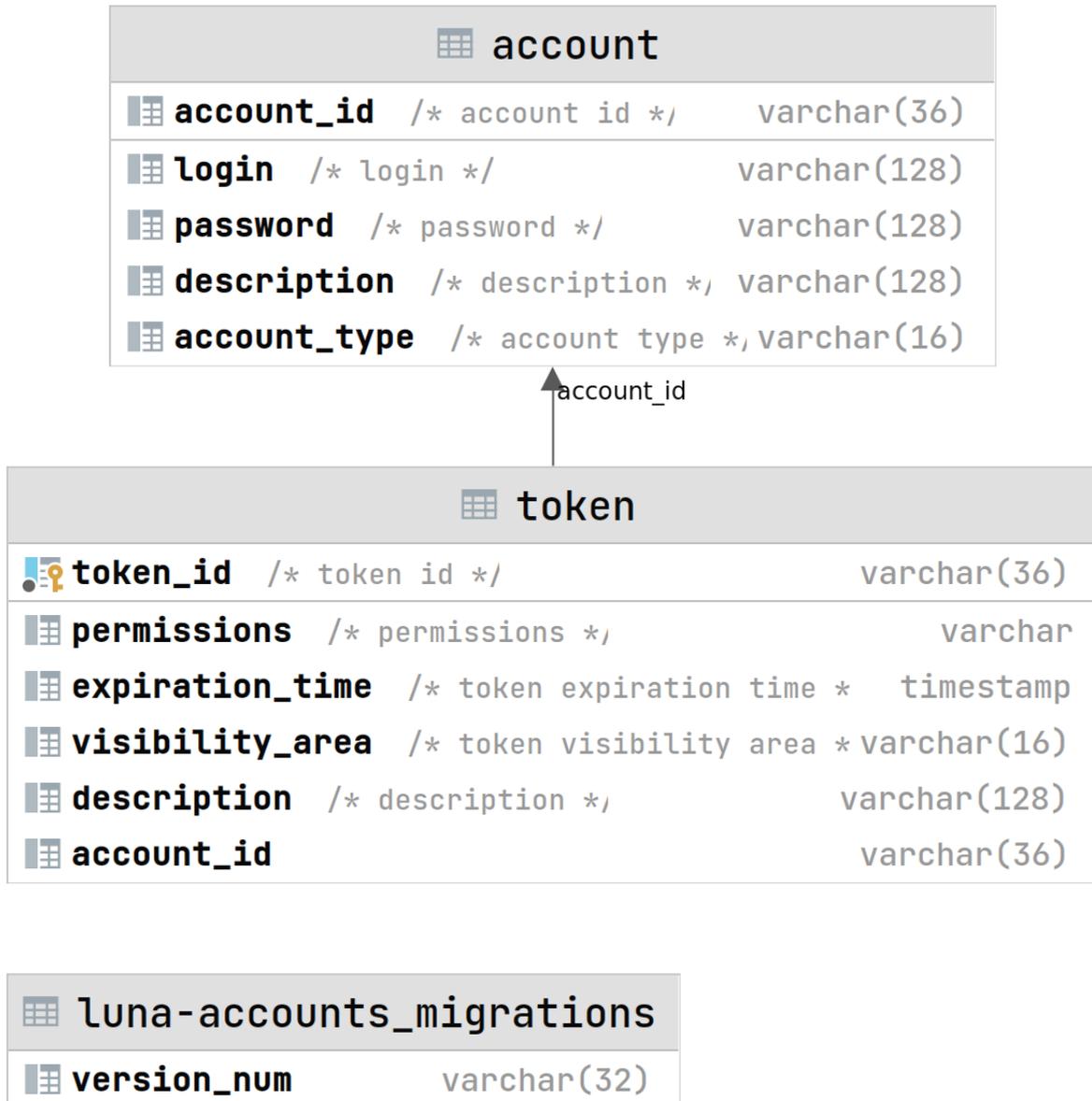


Figure 99: Accounts database schema

10.7.1 Account table model

Name	Type	Description
------	------	-------------

account_id	varchar(36)	Account ID in UUID4 format.
login	varchar(128)	Account login.
password	varchar(128)	Account password.
description	varchar(128)	Account description.
account_type	varchar(16)	Account type: <ul style="list-style-type: none"> • “user” • “advanced_user” • “admin”
create_time	timestamp	Date and time of account creation.
last_update_time	timestamp	Date and time of the latest account modification.

10.7.2 Token table model

Name	Type	Description
token_id	varchar(36)	Token ID in UUID4 format.
permissions	varchar(128)	Token permissions.
expiration_time	varchar(128)	Token expiration time.
description	varchar(128)	Token description.
visibility_area	varchar(128)	Token visibility area: <ul style="list-style-type: none"> • “all” • “account”
account_id	varchar(16)	Account ID in UUID4 format to which the token is linked.

10.7.3 Luna-accounts_migration table model

Name	Type	Description
version_num	varchar(32)	Parameter required for database migration.

10.8 Lambda database

This section describes the fields of the Lambda database.

For more information, see [“Lambda service”](#).

Lambda	
name	varchar(36)
description	varchar(256)
version	integer
status	smallint
account_id	varchar(36)
create_time	timestamp
last_update_time	timestamp
user_commands	varchar
lambda_type	varchar(10)
id	varchar(36)

luna-lambda_migrations	
version_num	varchar(32)

Figure 100: Lambda DB schema

10.8.1 Lambda table model

Name	Type	Description
name	varchar(36)	Lambda name.
description	varchar(256)	Lambda description.

version	integer	Lambda version.
account_id	varchar(16)	Account ID in UUID4 format to which the lambda is linked.
status	smallint	Lambda creation status: <ul style="list-style-type: none"> • “running” • “waiting” • “terminated” • “not_found”
create_time	timestamp	Lambda creation time.
last_update_time	timestamp	Lambda last updated date and time.
user_commands	varchar	List of additional Docker commands for creating a lambda container.
lambda_type	varchar(10)	Lambda type: <ul style="list-style-type: none"> • “handlers” • “standalone”
id	varchar(36)	Lambda ID.

10.8.2 Luna-lambda_migration table model

Name	Type	Description
version_num	varchar(32)	Parameter required for database migration.

11 API errors

The section describes errors, returned by the API service. Each of the errors has a unique code.

The errors can have different reasons.

In case of “Internal server error” or any other unexpected error occurrence, it is recommended to check service logs to find out more information about the error.

11.1 General errors

11.1.1 Code 0 returned

Message:

Success

Source:

General

Description:

This code is returned when there are no errors and the request is successfully processed.

Note that the result may still contain images filtered according to the parameters defined in the request.

11.1.2 Code 1 returned

Error Message:

Internal server error. Unknown error

Error Source:

General errors

Error Description:

An unexpected internal error occurred. The error is not properly processed in the code.

If any additional trace is provided and the error continues to appear, send the trace to VisionLabs technical support.

11.2 HTTP client errors

11.2.1 Code 3 returned

Error Message:

Invalid url {value}

Error Source:

HTTP client errors

Error Description:

Invalid URL was specified in the request.

Check the URL in the request. It may contain spaces or erroneous symbols.

Check API specification for the examples of the URL: [API documentation](#). Select the required version of LUNA PLATFORM using version switcher at the top of the page.

11.2.2 Code 4 returned

Error Message:

Client payload error {value}

Error Source:

HTTP client errors

Error Description:

Client payload error.

Possible causes: the response object was closed before the response received all data. A transfer encoding error occurred.

11.2.3 Code 5 returned

Error Message:

Server fingerprint mismatch {value}

Error Source:

HTTP client errors

Error Description:

Server fingerprint error.

An attempt to connect to an invalid server was performed or an invalid key was used.

11.2.4 Code 6 returned

Error Message:

Socket read timeout. Request timeout on {value} method {value}

Error Source:

HTTP client errors

Error Description:

Socket reading error. Request completion time limit exceeded.

The problem may occur due to the following reasons:

- Problems in the network. Check network.
- Internal server errors. Check service logs.

11.2.5 Code 7 returned

Error Message:

Socket connect timeout Request timeout on {value} method {value}

Error Source:

HTTP client errors

Error Description:

Socket connection error. Request completion time limit exceeded.

The request may take long due to the following reasons:

- Problems in the network. Check network.
- Internal server errors. Check service logs.
- High service load.

11.2.6 Code 8 returned

Error Message:

Connect timeout on {value} method {value}

Error Source:

HTTP client errors

Error Description:

Connection error. The connection time limit was exceeded.

The problem may occur due to the following reasons:

- Problems in the network. Check network.
- Service is unavailable. Check that the service is running and check its logs.

11.2.7 Code 9 returned

Error Message:

Request timeout on {value} method {value}

Error Source:

HTTP client errors

Error Description:

Connection error. Request completion time limit exceeded.

The problem may occur due to the following reasons:

- Problems in the network. Check network.
- Service is unavailable. Check that the service is running and check its logs.

11.2.8 Code 10 returned

Error Message:

Server disconnected {value}

Error Source:

HTTP client errors

Error Description:

The connection with the server is lost.

The problem may occur due to the following reasons:

- Problems in the network. Check network.
- Server is unavailable. Check that the server is running.

11.2.9 Code 11 returned

Error Message:

Server connection error {value}

Error Source:

HTTP client errors

Error Description:

Server connection error.

The connection with the server is lost.

The problem may occur due to the following reasons:

- Problems in the network. Check network.
- Server is unavailable. Check that the server is running.

11.2.10 Code 12 returned

Error Message:

Client proxy connection error {value}

Error Source:

HTTP client errors

Error Description:

Client proxy connection error.

The problem may occur due to the following reasons:

- Problems in the network. Check network.
- Proxy connection is unavailable.

11.2.11 Code 13 returned

Error Message:

Client connector SSL error {value}

Error Source:

HTTP client errors

Error Description:

Client connection via SSL error. Check SSL certificate.

11.2.12 Code 14 returned

Error Message:

Client connector certificate error {value}

Error Source:

HTTP client errors

Error Description:

Client SSL error.

Possible causes are: invalid SSL certificate, outdated client SSL version. Check SSL certificate.

11.2.13 Code 15 returned

Error Message:

Client SSL error {value}

Error Source:

HTTP client errors

Error Description:

Client SSL error.

Possible causes are: invalid SSL certificate, outdated client SSL version. Check SSL certificate.

11.2.14 Code 16 returned

Error Message:

Client connector error {value}

Error Source:

HTTP client errors

Error Description:

Client connection error.

11.2.15 Code 17 returned

Error Message:

Client OS error {value}

Error Source:

HTTP client errors

Error Description:

The error in the client operating system occurred.

Check the client OS.

11.2.16 Code 18 returned

Error Message:

Client connection error {value}

Error Source:

HTTP client errors

Error Description:

Client connection error.

11.2.17 Code 19 returned

Error Message:

Client HTTP proxy error {value}

Error Source:

HTTP client errors

Error Description:

The HTTP proxy server error has occurred on the client-side.

11.2.18 Code 20 returned

Error Message:

WS server handshake error {value}

Error Source:

HTTP client errors

Error Description:

The connection through web sockets failed.

11.2.19 Code 21 returned

Error Message:

Content-Type error {value}

Error Source:

HTTP client errors

Error Description:

Request content-type error. Invalid value in “Content-Type” field in the request header parameters. It does not match with the data provided in the request.

Check the “HEADER PARAMETERS” section in the API documentation for your request. See the list of API specifications here: [API documentation](#). Select the specification for the required service. Check that the specification version corresponds to the currently used LUNA PLATFORM version.

11.2.20 Code 22 returned

Error Message:

Client response error {value}

Error Source:

HTTP client errors

Error Description:

The client response error occurred.

11.2.21 Code 23 returned

Error Message:

HTTP client error {value}

Error Source:

HTTP client errors

Error Description:

Client HTTP error.

11.2.22 Code 24 returned

Error Message:

HTTP client error {value}

Error Source:

HTTP client errors

Error Description:

Client HTTP error. The response was received in incorrect JSON format.

11.3 Legacy errors

11.3.1 Code 5101 returned

Error Message:

Reference uuid is missing. Reference uuid is missing

Error Source:

Legacy errors

Error Description:

Matcher service error.

The provided UUID of the reference for matching was not found. Check that the UUID is correct and that a descriptor exists under that UUID. Check that the reference object is available for the current Account ID.

11.3.2 Code 5102 returned

Error Message:

Reference uuid has no extracted descriptor. Reference uuid has no extracted descriptor

Error Source:

Legacy errors

Error Description:

Matcher service error.

The extracted descriptor for Reference UUID not found. The descriptor was not extracted for the provided reference object or it was extracted using different neural network version.

Check the descriptor existence and its version.

11.4 Database errors

11.4.1 Code 10015 returned

Error Message:

SQL error. SQL request execution failed

Error Source:

Database errors

Error Description:

SQL request failed. A database error has occurred.

The database is not available for an unknown reason, or the required tables are missing. You should request the status of the database, check the availability of the database over the network, check the existence of the required tables in the database.

11.4.2 Code 10016 returned

Error Message:

Database error. Could not connect to database

Error Source:

Database errors

Error Description:

The connection to the database failed.

Check that the database is available. There may be errors during database launch or it is not available in the current network (due to server restrictions or network problems). Check the database settings in services parameters. They may be incorrect.

11.4.3 Code 10017 returned

Error Message:

Database error Database connection timeout error

Error Source:

Database errors

Error Description:

Database connection timeout error.

Check that the database is available. There may be errors during database launch or it is not available in the current network (due to server restrictions or network problems). Check the database settings in services parameters. They may be incorrect.

11.4.4 Code 10018 returned

Error Message:

Database error {value}

Error Source:

Database errors

Error Description:

An error occurred in the database. Check the provided description of the error.

11.5 API service errors

11.5.1 Code 11009 returned

Error Message:

Internal server error

Error Source:

API service errors

Error Description:

The error occurs when an unknown exception comes to the REST resource handler level.

It is recommended to check service logs to find out more information about the error. If any additional trace is provided and the error continues to appear, send the trace to VisionLabs technical support.

11.5.2 Code 11020 returned

Error Message:

Object not found. One or more {value} not found

Error Source:

API service errors

Error Description:

One or more of the specified objects were not found.

Check their existence.

11.5.3 Code 11027 returned

Error Message:

External request failed Failed to download image by url {value}

Error Source:

API service errors

Error Description:

An external request has failed. Failed to get the image by the URL specified in the request body.
Check that the URL is correct and its data is accessible and exists.

11.5.4 Code 11028 returned

Error Message:

Bad/incomplete input data. Bad content type of image {value}

Error Source:

API service errors

Error Description:

Invalid image content in the request. Check the “Content-Type” field in the request.

11.5.5 Code 11029 returned

Error Message:

Bad/incomplete input data. Bad content type of image in multipart body

Error Source:

API service errors

Error Description:

Bad input data in the multipart request. One or several images have an invalid format or size.
Check that the request “Content-Type” is set properly. Check image requirements for the request.
Check the requests in the [API documentation](#).

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.5.6 Code 11030 returned

Error Message:

Bad/incomplete input data. Image size is not equal to 250x250

Error Source:

API service errors

Error Description:

The size of the normalized image in the request is not equal to 250x250. The image is not a sample.

11.5.7 Code 11031 returned

Error Message:

Bad/incomplete input data. Sample with id {value} not found

Error Source:

API service errors

Error Description:

The sample with the specified ID was not found.

Check that the ID is correct and it is created under your current account ID.

11.5.8 Code 11032 returned

Error Message:

Object not found. Face with id {value} does not have attributes yet

Error Source:

API service errors

Error Description:

The Face with the specified ID has no attributes.

The error occurs when matching is performed for a Face without a linked descriptor.

Check that the ID is correct.

If there is no descriptor for the face, you can attach it using request: [PUT face attributes](#) or [create a new face](#). The source image or already existing attribute object is required.

The links lead to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.5.9 Code 11034 returned

Error Message:

Bad/incomplete input data. Descriptor for {value} "{value}" was not extracted

Error Source:

API service errors

Error Description:

The descriptor for the specified object was not extracted. You should verify the correctness of the object ID in the request.

11.5.10 Code 11035 returned

Error Message:

Service name not found. Service name {value} not found

Error Source:

API service errors

Error Description:

The specified service name was not found.

11.5.11 Code 11036 returned

Error Message:

Forbidden. Luna-Account-Id header is required for requests which change the state of system

Error Source:

API service errors

Error Description:

It is required to specify a correct “Luna-Account-Id” header in the request.

The header defines a user who changes the state of the system. Perhaps, the user without access to the database was specified or no account ID was specified.

Check the “HEADER PARAMETERS” section for the request in the [API documentation](#).

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.5.12 Code 11037 returned

Error Message:

Bad/incomplete input data. Luna-Account-Id header is not UUID format: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Error Source:

API service errors

Error Description:

The “Luna-Account-Id” header in the request is not in UUID format.

The required format is xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.

11.5.13 Code 11038 returned

Error Message:

Multiple faces found Multiple faces found in image {value} detect {value}

Error Source:

API service errors

Error Description:

There is more than one face in the image.

The “multiface_policy” value is set to “0” in the request or several faces are not processed in the request by default. Images with several faces are not processed in this case and an error is returned.

Also, this error may appear if more than one face is found in the image and an aggregation attempt is made, with a value of “multiface_policy” other than “2”. When performing aggregation, make sure that the value of the “multiface_policy” parameter is set to “2” in the handler.

11.5.14 Code 11039 returned

Error Message:

Forbidden. Luna Tasks service is disabled

Error Source:

API service errors

Error Description:

A request to the Tasks service cannot be processed.

Using the Tasks service is disabled in the API configuration.

Check the ADDITIONAL_SERVICES_USAGE parameter in the Configurator service or the configuration file.

11.5.15 Code 11040 returned

Error Message:

Forbidden. Luna Events service is disabled

Error Source:

API service errors

Error Description:

The request to the Events service cannot be processed.

Using the Events service is disabled in the API configuration.

Check the “ADDITIONAL_SERVICES_USAGE” parameter in the Configurator service or the configuration file (change this parameter in config for every service, which is using Events service).

11.5.16 Code 11041 returned

Error Message:

Object not found. No one face found with external id {value}

Error Source:

API service errors

Error Description:

The face object with the specified external ID was not found.

Check the external ID and the account ID in the request.

11.5.17 Code 11042 returned

Error Message:

Internal server error. Unhandled exception: {value}

Error Source:

API service errors

Error Description:

An unhandled exception occurred.

It is recommended to check service logs to find out more information about the error.

If any additional trace is provided and the error continues to appear, send the trace to VisionLabs technical support.

11.5.18 Code 11043 returned

Error Message:

Bad/incomplete input data. “{value}”

Error Source:

API service errors

Error Description:

Bad or incomplete input data is provided in the request.

Check the input data format.

11.5.19 Code 11044 returned

Error Message:

Forbidden. {value} turned off on luna-api instance

Error Source:

API service errors

Error Description:

The process is turned off for the API instance. Check the provided information in the API instance configuration.

11.5.20 Code 11045 returned

Error Message:

Forbidden. Only one detection rect for each image available at the moment

Error Source:

API service errors

Error Description:

Only one detection rectangle is available for each image at the moment. The error is returned when several detection rectangles are provided in the request.

11.5.21 Code 11046 returned

Error Message:

Bad/incomplete input data. More than one file named {value} found

Error Source:

API service errors

Error Description:

Several images with the same name were sent in the request. The images must have unique names.

11.5.22 Code 11047 returned

Error Message:

Bad/incomplete input data. Bounding box not available for warp images

Error Source:

API service errors

Error Description:

The bounding box set in the request is not available for samples.

11.5.23 Code 11048 returned

Error Message:

Bad/incomplete input data. Multiplie bounding boxes lists in request

Error Source:

API service errors

Error Description:

There is more than a single bounding box specified in the request. A single bounding box can be specified for an image.

11.5.24 Code 11049 returned

Error Message:

Bad/incomplete input data. {value}

Error Source:

API service errors

Error Description:

Bad or incomplete input data is provided in the multipart request. Check the information in the error message.

11.5.25 Code 11050 returned

Error Message:

Bad/incomplete input data. More than one bounding box for image named {value} found

Error Source:

API service errors

Error Description:

The specified image has several bounding boxes.

11.5.26 Code 11051 returned

Error Message:

Internal server error. Service “static” folder not being loaded

Error Source:

API service errors

Error Description:

The service static directory was not loaded.

11.5.27 Code 11052 returned

Error Message:

Bad/incomplete input data. No one attributes was settled for the extract

Error Source:

API service errors

Error Description:

Both “extract_descriptor” and “extract_basic_attributes” parameters are set to “0”. You should enable at least one parameter in the request.

11.5.28 Code 11053 returned

Error Message:

Multiple human bodies found on image {value} detect {value}

Error Source:

API service errors

Error Description:

There are several human bodies found in the image.

11.5.29 Code 11055 returned

Error Message:

Forbidden. License problem: "{value}"

Error Source:

API service errors

Error Description:

License error. The license has expired or is unavailable.

11.5.30 Code 11056 returned

Error Message:

Bad/incomplete input data. {value}

Error Source:

API service errors

Error Description:

The request is invalid. The request data is incomplete. Check the error message and input data in the request.

11.5.31 Code 11057 returned

Error Message:

Object not found No one event found with external id {value}

Error Source:

API service errors

Error Description:

No event was found with the specified "external_id". Check the specified references for matching in the request body.

11.5.32 Code 11058 returned

Error Message:

Object not found No one event found with track id {value}

Error Source: API service errors

Error Description:

No event was found with the specified “track_id”. Check the specified references for matching in the request body.

11.5.33 Code 11059 returned**Error Message:**

Forbidden Request denied due to token restrictions, required “{value}” permission for “{value}”

Error Source:

API service errors

Error Description:

The request was denied because the specified permissions were missing from the token being used. Check the “permissions” field of the used token.

11.5.34 Code 11060 returned**Error Message:**

Forbidden Access to the resource is denied due to token restrictions. Required “{value}” resource permission

Error Source:

API service errors

Error Description:

Access to the resource is denied due to the missing permission to use the specified resource in the token being used.

Check the “permissions” > “resources” field of the used token.

11.5.35 Code 11061 returned**Error Message:**

Forbidden Authorization with “Luna-Account-Id” header is disabled

Error Source:

API service errors

Error Description:

Authorization with Luna-Account-Id is disabled.

Check the “ALLOW_LUNA_ACCOUNT_AUTH_HEADER” setting in the Configurator service.

11.5.36 Code 11062 returned

Error Message:

Forbidden Specified token corrupted

Error Source:

API service errors

Error Description:

The token is corrupted.

This error can occur when trying to use a token with “visibility_area” = “all” with an account type of “user” that was previously downgraded from “advanced_user” or “admin”.

11.5.37 Code 11063 returned

Error Message:

Forbidden The query parameter account_id usage denied due to account restrictions

Error Source:

API service errors

Error Description:

This error may appear when trying to use the account_id filter with a value other than the user account id with the “user” type.

11.5.38 Code 11064 returned

Error Message:

Forbidden Access to the resource is denied using authorization by token

Error Source:

API service errors

Error Description:

Access to the resource is denied using authorization by token.

11.5.39 Code 11065 returned

Error Message:

Authorization failed, {value}

Error Source:

API service errors

Error Description:

Unsuccessful authorization. The error contains parameters that did not pass authorization.

11.5.40 Code 11066 returned**Error Message:**

Bad/incomplete input data, Luna-Account-Id header not found

Error Source:

API service errors

Error Description:

The required Luna-Account-Id header was not found when trying to create a token in the Accounts service.

11.5.41 Code 11067 returned**Error Message:**

Forbidden, Request denied due to token restrictions. According to emit_events the handler is not whitelisted or handler is blacklisted.

Error Source:

API service errors

Error Description:

Request denied due to token restrictions. The handler_id used is either blacklisted or not whitelisted. Check the “permissions” > “emit_events” > “black_list”/“white_list” field of the used token.

11.5.42 Code 11068 returned**Error Message:**

Forbidden Request denied due to token restrictions. Events generation is not allowed

Error Source:

API service errors

Error Description:

Request denied due to token restrictions. Generation of events for all existing handlers is not allowed. Check the “permissions” > “emit_events” > “allowed” field of the used token.

11.5.43 Code 11069 returned

Error Message:

Forbidden Account id specified in {value} is different from requester account id, that is not acceptable due to account/token visibility area restrictions

Error Source:

API service errors

Error Description:

The account ID set in the specified policy differs from the request account ID, which is not acceptable due to visibility area restrictions.

Make sure that the account IDs match or change the visibility area of the account/token.

11.5.44 Code 11070 returned

Error Message:

Forbidden, Luna Image Store service is disabled

Error Source:

API service errors

Error Description:

The request to the Image Store service cannot be processed.

Using the Image Store service is disabled in the API configuration.

Check the “ADDITIONAL_SERVICES_USAGE” parameter in the Configurator service or the configuration file (change this parameter in config for every service, which is using Image Store service).

11.5.45 Code 11071 returned

Error Message:

Forbidden, Luna Handlers service is disabled

Error Source:

API service errors

Error Description:

The request to the Handlers service cannot be processed.

Using the Handlers service is disabled in the API configuration.

Check the “ADDITIONAL_SERVICES_USAGE” parameter in the Configurator service or the configuration file (change this parameter in config for every service, which is using Handlers service).

11.5.46 Code 11072 returned

Error Message:

Forbidden, Luna Lambda service is disabled

Error Source:

API service errors

Error Description:

The request to the Lambda service cannot be processed.

Using the Lambda service is disabled in the API configuration.

Check the “ADDITIONAL_SERVICES_USAGE” parameter in the Configurator service or the configuration file (change this parameter in config for every service, which is using Lambda service).

11.6 REST API common errors

11.6.1 Code 12002 returned

Error Message:

Bad/incomplete input data. Request does not contain json

Error Source:

REST API common errors

Error Description:

The request does not contain JSON.

Check the JSON syntax in the request body:

- There are extra symbols or spaces.
- The JSON structure is wrong.
- Important syntax elements were missed.

11.6.2 Code 12003 returned

Error Message:

Bad/incomplete input data. Field. {value} not found in json

Error Source:

REST API common errors

Error Description:

The specified field was not found in the JSON body. Check that all required fields were added to the request.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.3 Code 12005 returned

Error Message:

Bad/incomplete input data. Request contain empty json

Error Source:

REST API common errors

Error Description:

The request contains an empty JSON body. A body is required to execute the request.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.4 Code 12010 returned

Error Message:

Bad/incomplete input data. This resource needs "Authorization" authorization headers

Error Source:

REST API common errors

Error Description:

You should specify the "Authorization" header in the request. Generally authorization is required for the Admin service and Backport 3 requests execution.

11.6.5 Code 12012 returned

Error Message:

Bad/incomplete input data. Bad query parameters {value}

Error Source:

REST API common errors

Error Description:

The specified query parameters are set incorrectly.

11.6.6 Code 12013 returned

Error Message:

Resource not found. Page not found

Error Source:

REST API common errors

Error Description:

The HTTP resource was not found.

The URL specified in the request was not found.

For example, the request contains the path “/6/sample” instead of “/6/samples”. Compare the specified path with the path in the documentation.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.7 Code 12014 returned

Error Message:

Bad/incomplete input data. Required parameters {value} not found

Error Source:

REST API common errors

Error Description:

The required query parameters were not found in the request. These parameters are listed in the error.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.8 Code 12016 returned

Error Message:

Bad/incomplete input data. No one parameters {value} not found

Error Source:

REST API common errors

Error Description:

Listed query parameters were not found in the request.

See the [API documentation](#) for the request. Compare your query parameters with the parameters in the “QUERY PARAMETERS” section.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.9 Code 12017 returned

Error Message:

Bad/incomplete input data. Bad content type

Error Source:

REST API common errors

Error Description:

Invalid “Content-Type” header value transmitted in the request.

No header was specified, an invalid header value was specified, or there is a mistake in the header.

See the [API documentation](#) for the request. Compare your header parameters with the parameters in the “HEADER PARAMETERS” section.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.10 Code 12021 returned

Error Message:

Method not allowed. Method not allowed

Error Source:

REST API common errors

Error Description:

The HTTP resource does not support this method. Check the HTTP method specified to send the request.

See the [API documentation](#) for the request. See the method specified for the required request (POST, GET, PATCH, etc.). E.g. POST /6/handlers.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.11 Code 12022 returned**Error Message:**

Bad/incomplete input data. Failed to validate input json. Path: "{value}" message: "{value}"

Error Source:

REST API common errors

Error Description:

There is an error in JSON syntax at the specified path.

The request body contains invalid fields. Check the request for compliance with the template specified the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

The detailed description of the error is given in the field “message”.

11.6.12 Code 12023 returned**Error Message:**

Bad/incomplete input data. Content type is unacceptable allowed types: "{value}"

Error Source:

REST API common errors

Error Description:

The request contains invalid “content-type”. Most likely, there is a typo in the request content-type.

See the [API documentation](#) for the request. Check the available content types.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.13 Code 12024 returned

Error Message:

Bad/incomplete input data. Unsupported media type

Error Source:

REST API common errors

Error Description:

The data type transmitted in the request is not supported.

See the [API documentation](#) for the request. Check the content type available for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.14 Code 12025 returned

Error Message:

Bad/incomplete input data. Specified content type does not match data type

Error Source:

REST API common errors

Error Description:

The content-type specified in the “Content-Type” header does not match the data type.

See the [API documentation](#) for the request. Check the available content types.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.15 Code 12027 returned

Error Message:

Bad/incomplete input data. Failed to validate input json. Message: {value}

Error Source:

REST API common errors

Error Description:

The request body parameters are incorrect. Check the JSON body for compliance with the request requirements. Details are listed in the “Message:” field.

See the [API documentation](#) for the request. Check the “REQUEST BODY SCHEMA” of the request. See the examples provided for the request in the documentation.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.16 Code 12028 returned

Error Message:

Bad/incomplete input data. Failed to parse Flatbuf

Error Source:

REST API common errors

Error Description:

Internal error. Failed to deserialize Flatbuf.

11.6.17 Code 12029 returned

Error Message:

Functionality is not implemented. Required server functionality is not implemented

Error Source:

REST API common errors

Error Description:

The error is returned for the server functionality that is not implemented.

11.6.18 Code 12030 returned

Error Message:

Bad/incomplete input data. Request does not contain data

Error Source:

REST API common errors

Error Description:

The request contains no data. The request body is empty.

See the [API documentation](#) for the request. Check the “REQUEST BODY SCHEMA” of the request and other request parameters. See the examples provided for the request in the documentation.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.19 Code 12031 returned

Error Message:

Bad/incomplete input data. {value}

Error Source:

REST API common errors

Error Description:

The input data is incomplete or wrong.

The detailed description of the error is given in the field “message”.

11.6.20 Code 12032 returned

Error Message:

Internal server error. Document file not found

Error Source:

REST API common errors

Error Description:

The document with the specified name was not found. The error is returned when requesting service documentation.

Check that the request is correct and the document name is set properly.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.21 Code 12033 returned

Error Message:

Forbidden. Access to this resource on the server is denied

Error Source:

REST API common errors

Error Description:

The access to the resource on the server is denied.

Check the resource availability. The access to the service may be restricted in the server settings.

11.6.22 Code 12034 returned**Error Message:**

Bad/incomplete input data. Descriptor has incorrect length {value}

Error Source:

REST API common errors

Error Description:

The descriptor has an incorrect length.

Make sure that you are using a descriptor of the appropriate format. Descriptors received from different sources may have different lengths.

Use the “/sdk” resource to receive a raw descriptor of the required format and use it in your requests. See resource [/sdk](#).

See the [API documentation](#) for the request.

The links lead to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.6.23 Code 12035 returned**Error Message:**

Bad/incomplete input data. Failed to parse xpk file

Error Source:

REST API common errors

Error Description:

Failed to parse the descriptor XPK file. Check that the file is correct.

11.6.24 Code 12036 returned**Error Message:**

Bad/incomplete input data. Descriptor version {value} is not registered in the system

Error Source:

REST API common errors

Error Description:

The specified descriptor version is not registered in the system.

A wrong/nonexistent descriptor version is set, or an incompatible service version is used that cannot process this descriptor version.

Check that you use a proper descriptor version. See the available descriptor versions in section [“Neural networks”](#).

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.6.25 Code 12037 returned

Error Message:

Bad/incomplete input data. XPK file does not contain descriptor

Error Source:

REST API common errors

Error Description:

The input data XPK does not contain a descriptor. Check that the file is correct.

11.6.26 Code 12038 returned

Error Message:

Bad/incomplete input data. SDK descriptor is not valid

Error Source:

REST API common errors

Error Description:

SDK descriptor is not valid. Check the provided descriptor data.

Make sure that you are using a descriptor of the appropriate format. Descriptors received from different sources may have different lengths.

Use the [“/sdk”](#) resource to receive a raw descriptor of the required format and use it in your requests.

The link leads to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.6.27 Code 12039 returned

Error Message:

Bad/incomplete input data. Unknown multipart name "{value}"

Error Source:

REST API common errors

Error Description:

There is an unknown multipart name in the request. Check that the name in the "message" field is correct.

11.6.28 Code 12040 returned

Error Message:

Bad/incomplete input data. Duplicate multipart name "{value}"

Error Source:

REST API common errors

Error Description:

There is a duplicated multipart name in the request. Check the name in the "message" field. Duplacate names are not available.

11.6.29 Code 12041 returned

Error Message:

Bad/incomplete input data. Multipart with name "{value}" has bad Content-Type

Error Source:

REST API common errors

Error Description:

The input data with the specified name has bad Content-Type in the multipart request. Check the specified content type.

11.6.30 Code 12042 returned

Error Message:

Gone. Access to this resource on the server is no longer available

Error Source:

REST API common errors

Error Description:

The access to the resource on the server is not available. Check that the corresponding service is working, has correct version and is available. Check that the resource is correct.

11.6.31 Code 12043 returned

Error Message:

Unknown error. Service "{value}" unknown error method: "{value}" url: "{value}"

Error Source:

REST API common errors

Error Description:

Unknown service error has occurred. Check the method and URL provided in the error message.

11.6.32 Code 12044 returned

Error Message:

Bad/incomplete input data. Failed to decompress input body using gzip encoder

Error Source:

REST API common errors

Error Description:

Failed to decompress input body using GZIP encoder.

11.6.33 Code 12045 returned

Error Message:

Bad/incomplete input data. Failed to decompress input body using deflate encoder

Error Source:

REST API common errors

Error Description:

Failed to decompress input body using deflate encoder.

11.6.34 Code 12046 returned**Error Message:**

Bad/incomplete input data. Invalid http request: {value}

Error Source:

REST API common errors

Error Description:

An invalid HTTP request was set.

11.6.35 Code 12047 returned**Error Message:**

Bad/incomplete input data. Failed to validate input msgpack. Path: "{value}", message: "{value}"

Error Source:

REST API common errors

Error Description:

The input data passed in MessagePack format does not match the expected data schema. The specific error message "Path: '{value}', message: '{value}'" provides information about the location and content of the erroneous data.

Check the input data and make sure it matches the expected data schema.

11.6.36 Code 12048 returned**Error Message:**

Bad/incomplete input data. Archive file does not contain plugin.py

Error Source:

REST API common errors

Error Description:

The attached archive does not contain the required `plugin.py` file.

Make sure the file exists in the archive.

11.6.37 Code 12049 returned

Error Message:

Forbidden. Resource is disabled

Error Source:

REST API common errors

Error Description:

Use of the `/metrics` resource is disabled in the “enabled” parameter of the “LUNA_SERVICE_METRICS” section.

Enable the “enabled” option and retry the request after restarting the services.

11.7 Image Store service errors

11.7.1 Code 13003 returned

Error Message:

Object not found. Image with id {value} not found

Error Source:

Image Store service errors

Error Description:

The image with the specified ID was not found in the storage. There is a typo in the ID, the file was deleted or does not exist.

11.7.2 Code 13004 returned

Error Message:

Bad/incomplete input data. Image count exceeded limit 1000

Error Source:

Image Store service errors

Error Description:

The number of images in the deletion request exceeded the limit of 1000.

Reduce the number of deleted images in the request.

11.7.3 Code 13005 returned

Error Message:

Object not found. Bucket with name {value} not found

Error Source:

Image Store service errors

Error Description:

The specified bucket was not found.

Check the existence of the bucket.

The path to the bucket is specified in the Image Store configuration file or Configurator service in the “LOCAL_STORAGE” variable. If there is no bucket, you should manually create it. See section “[Services launch](#)” > “[Buckets creation](#)” in the Installation manual.

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.7.4 Code 13006 returned

Error Message:

Unique constraint error. Bucket with name {value} already exist

Error Source:

Image Store service errors

Error Description:

A bucket with the specified name already exists. The error occurs when trying to create two buckets with the same name. There cannot be two buckets with the same name.

11.7.5 Code 13007 returned

Error Message:

Object not found. Object with id {value} not found

Error Source:

Image Store service errors

Error Description:

The object with the specified ID was not found in the bucket. The object does not exist, it was deleted or there is a typo in the ID.

11.7.6 Code 13008 returned

Error Message:

Unique constraint error. Object with id {value} already exist

Error Source:

Image Store service errors

Error Description:

An object with the specified ID already exists in the bucket.

11.7.7 Code 13009 returned

Error Message:

Bad/incomplete input data. Object count exceeded limit 1000

Error Source:

Image Store service errors

Error Description:

The number of objects transferred to the bucket in the request exceeded the limit of 1000.

11.8 Admin service errors

11.8.1 Code 15012 returned

Error Message:

Object not found. Account with id "{value}" not found

Error Source:

Admin service errors

Error Description:

There is no account with the specified ID in the Admin service.

11.8.2 Code 15013 returned

Error Message:

Unique constraint error. Account with same email or id already exist

Error Source:

Admin service errors

Error Description:

The e-mail with the specified ID already exists.

11.8.3 Code 15014 returned

Error Message:

Bad/incomplete input data. Login or password is incorrect

Error Source:

Admin service errors

Error Description:

Bad input. The login or the password for the account is incorrect.

11.8.4 Code 15015 returned

Error Message:

Bad/incomplete input data, Admin account type required

Error Source:

Admin service errors

Error Description:

The data is incorrect. The request requires the “admin” account type.

11.9 Image Processing Errors

11.9.1 Code 18001 returned

Error Message:

Bad/incomplete input data. Failed convert data from base64 to bytes

Error Source:

Image Processing Errors

Error Description:

Failed to convert BASE64 string to bytes.

The provided BASE64 string is corrupted.

11.9.2 Code 18002 returned

Error Message:

Bad/incomplete input data. Failed convert bytes to {value}

Error Source:

Image Processing Errors

Error Description:

Failed to convert bytes to the specified image format. The request does not include a photo image.

11.9.3 Code 18003 returned

Error Message:

Bad/incomplete input data. Failed read bytes as image

Error Source:

Image Processing Errors

Error Description:

The image bytes cannot be read. The provided image is corrupted or has an invalid format.

11.10 Image Store Service Errors (Local Storage)

11.10.1 Code 19001 returned

Error Message:

Internal server error. Failed to save image in the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not save an image to the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.2 Code 19002 returned

Error Message:

Internal server error. Failed to remove image from the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not delete an image from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.3 Code 19003 returned

Error Message:

Internal server error. Failed to remove image from the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not delete images from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.4 Code 19004 returned

Error Message:

Internal server error. Failed to create bucket

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not create a bucket on the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.5 Code 19005 returned

Error Message:

Internal server error. Failed to get bucket list

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not get a list of buckets on the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.6 Code 19006 returned**Error Message:**

Internal server error. Failed to get image from the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not get an image from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.7 Code 19007 returned**Error Message:**

Internal server error. Failed to save object in the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not get an image from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.8 Code 19008 returned**Error Message:**

Internal server error. Failed to remove object from the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not delete an object from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.9 Code 19009 returned**Error Message:**

Internal server error. Failed to remove objects from the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not delete objects from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.10 Code 19010 returned**Error Message:**

Internal server error. Failed to get object from the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not get an object from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.11 Code 19011 returned**Error Message:**

Internal server error. Failed to get objects from the storage

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not get objects from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.12 Code 19012 returned**Error Message:**

Internal server error. Failed to delete bucket

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not delete a bucket from the local storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.10.13 Code 19013 returned**Error Message:**

Internal server error. Failed to get bucket info

Error Source:

Image Store Service Errors (Local Storage)

Error Description:

Image Store could not get a bucket info.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.11 Image Store Service Errors (S3 storage)**11.11.1 Code 20001 returned****Error Message:**

Internal server error. Failed to save image in the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not save an image to the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.2 Code 20002 returned

Error Message:

Internal server error. Failed to remove image from the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not delete an image from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.3 Code 20003 returned

Error Message:

Internal server error. Failed to remove image from the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not delete images from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.4 Code 20004 returned

Error Message:

Internal server error. Failed to create bucket

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not create a bucket in the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.5 Code 20005 returned

Error Message:

Internal server error. Request to S3 Failed

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Failed to complete the request to the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.6 Code 20006 returned

Error Message:

Internal server error. Request to S3 Forbidden.

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Failed to get access to the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.7 Code 20007 returned

Error Message:

Internal server error. Request time to S3 is longer than the established time

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Request time to the S3 storage exceeded the specified limit. Check the storage availability.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.8 Code 20008 returned

Error Message:

Internal server error. S3 Connection Refused

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Connection to the Image Store (S3) was refused.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.9 Code 20009 returned

Error Message:

Internal server error. Connect time to S3 is longer than the established time

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Connection time to the S3 storage has exceeded the specified limit.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.10 Code 20010 returned

Error Message:

Internal server error. Unknown s3 error

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

An unknown S3 storage error occurred.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.11 Code 20011 returned

Error Message:

Internal server error. Failed to get bucket list

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not get a list of buckets from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.12 Code 20012 returned

Error Message:

Internal server error. Failed to get image from the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not get an image from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.13 Code 20013 returned

Error Message:

Internal server error. Failed to save object in the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not save an object to the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.14 Code 20014 returned

Error Message:

Internal server error. Failed to remove object from the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not delete an object from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.15 Code 20015 returned

Error Message:

Internal server error. Failed to remove object list from the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not delete a list of objects from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.16 Code 20016 returned

Error Message:

Internal server error. Failed to get object from the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not get an object from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.17 Code 20017 returned

Error Message:

Internal server error. Failed to get object list from the storage

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not get objects from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.18 Code 20018 returned

Error Message:

Internal server error. Failed to delete bucket

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not delete a bucket from the S3 storage.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the storage, a problem in Image Store.

11.11.19 Code 20019 returned

Error Message:

Bad/incomplete query Failed to create bucket with specified name

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

The bucket with the given name cannot be created. The name includes unavailable symbols. It cannot be used as a bucket name.

11.11.20 Code 20020 returned

Error Message:

Internal server error. Failed to get bucket info

Error Source:

Image Store Service Errors (S3 storage)

Error Description:

Image Store could not get a bucket info.

Check the Image Store service logs. The error may occur for the following reasons: denial of access, inaccessibility of Image Store over the network, a problem with the disk, a problem in Image Store.

11.12 Faces service errors

11.12.1 Code 22001 returned

Error Message:

Unique constraint error. Face with the same attribute_id already exists

Error Source:

Faces service errors

Error Description:

A Face with the provided “attribute_id” already exists. An attribute can be attached to a single face only.

11.12.2 Code 22002 returned

Error Message:

Object not found. Face with id “{value}” not found

Error Source:

Faces service errors

Error Description:

The Face with the specified “face_id” was not found. A typo was made in the “face_id” field or a Face with such an ID does not exist.

11.12.3 Code 22003 returned

Error Message:

Object not found. List with id “{value}” not found

Error Source:

Faces service errors

Error Description:

The List with the given ID was not found. You should specify the ID of an existing list in the request.

11.12.4 Code 22004 returned

Error Message:

Object not found. One or more faces not found including face with id “{value}”

Error Source:

Faces service errors

Error Description:

One or more Faces were not found, including the Face with the specified ID. A typo was made in the “face_id” field or a face with such an ID does not exist.

11.12.5 Code 22005 returned

Error Message:

Object not found. One or more lists not found including list with id “{value}”

Error Source:

Faces service errors

Error Description:

One or more lists were not found, including the list with the specified ID. The specified lists do not exist. Specify an existing list ID in the request.

11.12.6 Code 22009 returned

Error Message:

Bad/incomplete configuration. Face avatar url is not correct

Error Source:

Faces service errors

Error Description:

The URL in the “avatar” field in the request is incorrect. Check the provided URL. It does not contain data, is corrupted or the data is unavailable without authorization.

11.12.7 Code 22010 returned

Error Message:

Object not found. Attributes with id “{value}” for update not found

Error Source:

Faces service errors

Error Description:

An error occurred while updating the fields of the existing Face. The “attribute_id” field in the request is invalid. The attribute with the specified ID was not found.

The attribute is deleted after the TTL expiration. Create a new attribute to attach to the face.

See information about attributes creation in section [“general concepts”](#) > [“Attribute object”](#) of Administrator’s manual.

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.12.8 Code 22011 returned

Error Message:

Object not found. Attributes with id {value} not found

Error Source:

Faces service errors

Error Description:

Attributes with the specified ID were not found in the database. You should specify an existing “attribute_id” in the request.

The attribute is deleted after the TTL expiration. Create a new attribute to attach to the face.

See information about attributes creation here: [“general concepts” > “Attribute object”](#).

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.12.9 Code 22012 returned

Error Message:

Bad/incomplete input data. Failed to decode descriptor from base64

Error Source:

Faces service errors

Error Description:

Failed to decode the descriptor from the BASE64 format.

If the error repeat, check the BASE64 file. It can be corrupted.

11.12.10 Code 22013 returned

Error Message:

Bad/incomplete input data. Failed to encode descriptor to base64

Error Source:

Faces service errors

Error Description:

Failed to encode the descriptor to the BASE64 format. A software error occurred.

11.12.11 Code 22015 returned

Error Message:

Conflict input data Attribute “{value}” generation should be “{value}” but “{value}” was provided

Error Source:

Faces service errors

Error Description:

The attribute generation does not match the provided one.

The error occurs upon the extraction of the descriptor of a new version. The source descriptor was updated, so the descriptor of a new version cannot be added to the database. It is required to re-extract the descriptor using the source image and the required NN version.

See [“LP Services description”](#) > [“Additional extraction task”](#) in Administrator’s manual for information about additional extraction task.

See [“Additional information”](#) > [“Change neural network used”](#) in Administrator’s manual for information about the neural network switch process.

The links lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.12.12 Code 22016 returned

Error Message:

Bad input data “{value}” is not valid target to get faces. Valid target should be one of {value}.

Error Source:

Faces service errors

Error Description:

Invalid «target» to get Faces.

The valid “target” should be one of the specified in this error.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.12.13 Code 22017 returned

Error Message:

Bad/incomplete input data. Match reference must be specified either as (descriptor) or (attribute_id) or (face_id)

Error Source:

Faces service errors

Error Description:

The provided reference object for matching is incorrect. The reference for matching must be a descriptor, an attribute ID, or a face ID.

11.12.14 Code 22018 returned

Error Message:

Object not found. Descriptor of version {value} is not found for object with id "{value}".

Error Source:

Faces service errors

Error Description:

There is no descriptor with the specified version for the object with the specified ID.

The provided object may include several descriptors of different versions. But there is no descriptor of the version, used in the system.

See the default descriptor version in the "DEFAULT_FACE_DESCRIPTOR_VERSION" parameter in the Configurator service or in the configuration files of LP services.

11.12.15 Code 22020 returned

Error Message:

Internal server error. Corrupted attribute with id "{value}"

Error Source:

Faces service errors

Error Description:

The attribute with this ID is corrupted due to unknown reasons.

11.12.16 Code 22021 returned

Error Message:

Integrity error. Attribute with id "{value}" already exist

Error Source:

Faces service errors

Error Description:

The attribute with the specified ID already exists.

11.12.17 Code 22022 returned

Error Message:

Bad input data. Attribute does not contain “descriptors” and “basic_attributes”

Error Source:

Faces service errors

Error Description:

The input attribute does not contain any data: “descriptors” and “basic_attributes”.

11.12.18 Code 22023 returned

Error Message:

Bad input data. Attribute contains “{value}” samples but corresponding attributes is empty

Error Source:

Faces service errors

Error Description:

The attribute contains samples but does not include data.

11.12.19 Code 22024 returned

Error Message:

Bad input data. Attribute contains descriptors of identical versions: “{value}”

Error Source:

Faces service errors

Error Description:

The attribute contains descriptors of the same version. There cannot be two descriptors of the same version in one attribute object. Only one descriptor per version is available.

11.12.20 Code 22025 returned

Error Message:

Conflict input data. Attribute samples of face (face_id “{value}”) do not match specified ones

Error Source:

Faces service errors

Error Description:

The attribute samples of the face with the specified “face_id” do not match specified samples.

11.12.21 Code 22026 returned**Error Message:**

Unique constraint error. List with id “{value}” already exist

Error Source:

Faces service errors

Error Description:

The list with the specified ID already exists and cannot be created.

11.12.22 Code 22027 returned**Error Message:**

Bad input data. “{value}” is not valid target for face deletion info. Valid target should be one of {value}

Error Source:

Faces service errors

Error Description:

The specified value in the “targets” query parameter does not match the listed available values for this parameter.

Check the “targets” query parameter of the “delete faces with filters” request.

11.13 Events service errors**11.13.1 Code 23001 returned****Error Message:**

Object not found. Event with id {value} not found

Error Source:

Events service errors

Error Description:

The event with the specified ID was not found by Events service.

The event does not exist: it was deleted or there is a typo in the ID.

11.13.2 Code 23002 returned

Error Message:

Object not found. One or more events not found including event with id {value}

Error Source:

Events service errors

Error Description:

One or more events were not found by the Events service, including the specified event. These events do not exist: they were deleted or there are typos in their IDs.

11.13.3 Code 23003 returned

Error Message:

Unique constraint error. One or more event id from {value} already exist

Error Source:

Events service errors

Error Description:

One or more event IDs from the list already exist and cannot be created.
An event should have a unique name.

11.13.4 Code 23004 returned

Error Message:

Unique constraint error. One or more attribute id from "{value}" already exist

Error Source:

Events service errors

Error Description:

One or more listed attribute IDs already exist.

11.13.5 Code 23005 returned

Error Message:

Bad input data "{value}" is not valid target to get events. Valid events should be one of {value}

Error Source:

Events service errors

Error Description:

The specified target is not valid. Specify one of the available targets.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.13.6 Code 23006 returned

Error Message:

Internal server error. Timeout ({value} seconds) for saving events into history database has been expired

Error Source:

Events service errors

Error Description:

The timeout for saving events into the database has been expired. The request took too long due to massive provided data, problems in network or problems with connection to the Database.

See [Advanced PostgreSQL setting](#) for information about PostgreSQL database configuration.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.13.7 Code 23007 returned

Error Message:

Object not found. Human descriptor of version {value} is not found for object with id "{value}"

Error Source:

Events service errors

Error Description:

Human descriptor was not found for the object with the specified ID. The descriptor was not extracted or it was deleted.

11.13.8 Code 23008 returned

Error Message:

Internal server error. Copywriter queue is full

Error Source:

Events service errors

Error Description:

Copywriter queue is full.

11.13.9 Code 23009 returned**Error Message:**

Internal server error. Events are shutting down

Error Source:

Events service errors

Error Description:

Events are shutting down. Check the Events service logs. An error occurred in the service.

11.13.10 Code 23010 returned**Error Message:**

Internal server error. Events saving failed reason: {value}

Error Source:

Events service errors

Error Description:

The saving of an event failed due to an internal server error.

It is recommended to check service logs to find out more information about the error.

If any additional trace is provided and the error continues to appear, send the trace to VisionLabs technical support.

11.13.11 Code 23011 returned**Error Message:**

Bad input data. "{value}" is not valid target for event deletion info. Valid target should be one of {value}

Error Source:

Events service errors

Error Description:

Not valid value is specified in the “target” field in the query parameters. The valid value must be one of the specified ones.

Check the set value for the “target” field.

See the [API documentation](#) for the request.

11.14 Configurator service errors

11.14.1 Code 27001 returned

Error Message:

Object not found. Setting with id “{value}” not found

Error Source:

Configurator service errors

Error Description:

The setting with the specified ID was not found. Check that the specified ID is correct and the setting exists in the Configurator.

11.14.2 Code 27002 returned

Error Message:

Integrity error. Setting with the following fields already exists: name: {value} tag: {value}

Error Source:

Configurator service errors

Error Description:

The setting with the specified fields exists. The “name” and “tag” pair should be unique in the database.

11.14.3 Code 27003 returned

Error Message:

Bad/incomplete input dat. Connection check to service is failed

Error Source:

Configurator service errors

Error Description:

The request to the service failed. Check that the service is running, there are no errors in its logs, and the service is available through network.

11.14.4 Code 27004 returned

Error Message:

Bad/incomplete input data. Not allowed to change tags for default setting

Error Source:

Configurator service errors

Error Description:

You cannot create a tag for the default setting. The default setting can't have a tag. Duplicate the setting to change it and apply a tag.

11.14.5 Code 27005 returned

Error Message:

Bad/incomplete input data. Tag "{value}" for setting "{value}" not found

Error Source:

Configurator service errors

Error Description:

There is no such a tag for the specified setting.

11.14.6 Code 27006 returned

Error Message:

Object not found. Limitation named "{value}" not found

Error Source:

Configurator service errors

Error Description:

The newly created setting should match the configuration template (limitation) that is set for it.

A description of the configuration template is provided in the user interface and dump-file. The file can be received using the corresponding request: ["GET Dump file"](#).

The link leads to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.14.7 Code 27007 returned

Error Message:

Integrity error. Limitation named "{value}" already exists

Error Source:

Configurator service errors

Error Description:

When creating a new template for the settings (limitation) you used the name of an existing template. The name should be unique.

11.14.8 Code 27008 returned

Error Message:

Object not found. Group named "{value}" not found

Error Source:

Configurator service errors

Error Description:

The Group object with the specified name was not found.

You can create a new group using the Configurator UI or the following request: [“New group”](#) in the Configurator API manual.

The link leads to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.14.9 Code 27009 returned

Error Message:

Object not found. One or more groups not found including group named "{value}"

Error Source:

Configurator service errors

Error Description:

One or several Group objects were not found, including the object with the specified name.

You can create a new group using the Configurator UI or the following request: [“New group”](#) in the Configurator manual.

The link leads to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.14.10 Code 27010 returned

Error Message:

Integrity error. Group named "{value}" already exists

Error Source:

Configurator service errors

Error Description:

The Group object with the specified name already exists.

11.14.11 Code 27011 returned

Error Message:

Bad/incomplete input data. Cannot remove default setting "{value}"

Error Source:

Configurator service errors

Error Description:

It is impossible to remove the default setting from Configurator.

11.14.12 Code 27012 returned

Error Message:

Bad/incomplete input data. Not allowed to change default setting name

Error Source:

Configurator service errors

Error Description:

Error when trying to change the default setting name. It is not allowed to change the default setting name.

Duplicate the setting to change it and apply a new name.

11.15 Tasks service errors

11.15.1 Code 28000 returned

Error Message:

Network error Cannot send subtasks to task_workers. Reason: "{value}"

Error Source:

Tasks service errors

Error Description:

Failed to send subtasks to Tasks service workers. The reason is shown in the "Reason" field.

Check that the service worker is running and there are no network problems.

11.15.2 Code 28001 returned

Error Message:

Object not found. Task with id "{value}" not found

Error Source:

Tasks service errors

Error Description:

The task with the specified ID was not found by the Tasks service.

The task was completed, the identifier was set incorrectly or the task with the specified identifier was deleted.

11.15.3 Code 28002 returned

Error Message:

Object not found. One or more tasks not found including task with id {value}

Error Source:

Tasks service errors

Error Description:

The Tasks service did not found the specified tasks, including the task with the specified ID. The task was completed, the ID is set incorrectly or the task was deleted.

11.15.4 Code 28003 returned

Error Message:

Object not found. Task error with id "{value}" not found

Error Source:

Tasks service errors

Error Description:

An error with the ID specified in the request was not found by the Tasks service. The error does not exist.

11.15.5 Code 28004 returned

Error Message:

Object not found. One or more tasks not found including task with id "{value}"

Error Source:

Tasks service errors

Error Description:

One or more errors were not found by Tasks service, including the error with the specified ID.

The ID is incorrect or the error was deleted.

11.15.6 Code 28005 returned

Error Message:

Stop tasks worker On worker shutdown all active tasks on the worker are failed all active subtasks are cancelled

Error Source:

Tasks service errors

Error Description:

The task execution process was interrupted because Tasks worker stopped. All active subtasks were canceled.

Check the status of all Tasks service workers. Some of the workers became unavailable for some reason. There could be problems with the service, network or server.

11.15.7 Code 28006 returned

Error Message:

Internal server error. Failed to update task {value} status. Desired status: {value}

Error Source:

Tasks service errors

Error Description:

Tasks service failed to update the specified task status. The expected status is shown.

11.15.8 Code 28007 returned

Error Message:

Internal server error. Failed to update subtask {value} status. Desired status: {value}

Error Source:

Tasks service errors

Error Description:

Tasks service failed to update the subtask status for unknown reason. The expected status is shown.

11.15.9 Code 28008 returned

Error Message:

Internal server error. Failed to update task {value} progress. Desired progress: {value}

Error Source:

Tasks service errors

Error Description:

Tasks service failed to update the progress of the task. The expected progress is shown.

11.15.10 Code 28009 returned

Error Message:

Attribute is not equal Event ({value}) attribute is not equal {value} corresponding face attribute

Error Source:

Tasks service errors

Error Description:

The event attribute ID and the corresponding Face attribute ID did not match (Linker task).

11.15.11 Code 28010 returned

Error Message:

Objects not found. Objects for clustering not found (empty set)

Error Source:

Tasks service errors

Error Description:

The objects specified for clustering were not found by the Tasks service. Check that the objects exist and their IDs are correct.

11.15.12 Code 28011 returned

Error Message:

Internal server error. Failed to save report to a csv-file

Error Source:

Tasks service errors

Error Description:

Tasks service failed to save a CSV file for the report.

11.15.13 Code 28012 returned

Error Message:

Internal server error. Failed to create archive from a report

Error Source:

Tasks service errors

Error Description:

Tasks service failed to create the archive for the report.

11.15.14 Code 28013 returned

Error Message:

Bad/incomplete input data. Tasks with type "{value}" does not support a build report

Error Source:

Tasks service errors

Error Description:

You cannot create the report (reporter task) for the provided type of task.

11.15.15 Code 28014 returned**Error Message:**

Bad/incomplete input data. Column "{value}" not allowed in report by {value}

Error Source:

Tasks service errors

Error Description:

The specified column cannot be added to the report (reporter task). The column is not available in the reporter task.

See "Create reporter task" in the [API](#) or [Tasks](#) reference manuals.

The links lead to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.15.16 Code 28015 returned**Error Message:**

Bad/incomplete input data. Tasks with status "{value}" does not support a build report

Error Source:

Tasks service errors

Error Description:

A report cannot be created for tasks with the specified status (reporter's task). You should wait for the task to complete to get a report.

You can check task status using request "GET Task": [API service](#) or [Tasks service](#).

The links lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.15.17 Code 28016 returned

Error Message:

Bad/incomplete input data. Clusterization task without result does not support a build report

Error Source:

Tasks service errors

Error Description:

A report cannot be generated for a clusterization task without results. The task was not finished, wrong task ID is provided.

11.15.18 Code 28017 returned

Error Message:

Object not found. Result of the task "{value}" not found

Error Source:

Tasks service errors

Error Description:

The result of the specified task was not found.

Possible reasons: the task result was deleted from Image Store or an error occurred during task execution.

11.15.19 Code 28018 returned

Error Message:

Object not found. Task "{value}" does not have result yet

Error Source:

Tasks service errors

Error Description:

The task has no result yet. You should wait for the task to complete.

You can check task status using the following request: You can check task status using request "GET Task": [API service](#) or [Tasks service](#).

The links lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.15.20 Code 28019 returned

Error Message:

Bad/incomplete input data. Task "{value}" with status {value} cannot be canceled

Error Source:

Tasks service errors

Error Description:

A task with the specified status cannot be canceled.

Depending on the status, the task may have been stopped, completed with an error, or completed.

11.15.21 Code 28020 returned

Error Message:

Object not found. Impossible get a result of task {value} with status {value}

Error Source:

Tasks service errors

Error Description:

It is unable to get the result of the Tasks service task with the specified status.

11.15.22 Code 28021 returned

Error Message:

Forbidden. Service does not support Luna Events as source for tasks

Error Source:

Tasks service errors

Error Description:

The service does not support Luna Events as a data source for tasks.

11.15.23 Code 28022 returned

Error Message:

Objects not found. {value} for cross-matching not found (empty set)

Error Source:

Tasks service errors

Error Description:

The specified object for cross-matching was not found.

The object was deleted or there is a typo in ID.

11.15.24 Code 28023 returned**Error Message:**

Reference uuid is missing: {value}

Error Source:

Tasks service errors

Error Description:

The object with the ID specified for the cross-matching task was not found.

The object was deleted or there is a typo in the ID.

11.15.25 Code 28024 returned**Error Message:**

Reference uuid has no extracted descriptor: {value}

Error Source:

Tasks service errors

Error Description:

There is no extracted descriptor for the object with UUID specified for the cross-matching task.

11.15.26 Code 28025 returned**Error Message:**

Task "{value}" has been cancelled

Error Source:

Tasks service errors

Error Description:

The task was canceled and there is no result for the task.

11.15.27 Code 28026 returned

Error Message:

Failed to read archive {value}

Error Source:

Tasks service errors

Error Description:

The specified archive could not be read.

Check the availability of the archive.

11.15.28 Code 28027 returned

Error Message:

Failed to read file {value}

Error Source:

Tasks service errors

Error Description:

The specified file could not be read.

Check the availability of the file.

11.15.29 Code 28028 returned

Error Message:

Failed to read directory {value}

Error Source:

Tasks service errors

Error Description:

The specified directory could not be read.

Check the availability of the directory.

11.15.30 Code 28029 returned

Error Message:

Network disk is not available {value}

Error Source:

Tasks service errors

Error Description:

The specified network disk is unavailable.
Check the availability of the network disk.

11.15.31 Code 28030 returned**Error Message:**

FTP server authorization error {value}

Error Source:

Tasks service errors

Error Description:

An authorization error occurred while connecting to the specified FTP server.
Check the “user” and “password” parameters.

11.15.32 Code 28031 returned**Error Message:**

FTP server is unreachable {value}

Error Source:

Tasks service errors

Error Description:

The FTP server is unavailable.
Check the availability of the FTP server.

11.15.33 Code 28032 returned**Error Message:**

FTP download error {value}

Error Source:

Tasks service errors

Error Description:

An error occurred while downloading data from FTP server.

Check the entered parameters.

11.15.34 Code 28033 returned

Error Message:

FTP file listing error {value}

Error Source:

Tasks service errors

Error Description:

An error occurred in retrieving the list of files from the FTP server.

Check the entered parameters.

11.15.35 Code 28034 returned

Error Message:

Samba authorization error, {value}

Error Source:

Tasks service errors

Error Description:

Authorization failed for the specified Samba.

Check the entered query parameters.

11.15.36 Code 28035 returned

Error Message:

Samba is unreachable, {value}

Error Source:

Tasks service errors

Error Description:

Samba is unreachable according to the specified data.

Check the entered query parameters.

11.15.37 Code 28036 returned

Error Message:

Samba download error, {value}

Error Source:

Tasks service errors

Error Description:

Download failed for specified Samba.

Check the entered query parameters.

11.15.38 Code 28037 returned

Error Message:

Samba unknown error, {value}

Error Source:

Tasks service errors

Error Description:

Unknown error for specified Samba.

11.15.39 Code 28038 returned

Error Message:

Exporter failed to download data, {value}

Error Source:

Tasks service errors

Error Description:

The error occurred while loading data while executing the specified Exporter task.

This error can occur, for example, if the Faces or Events service returned an error.

11.15.40 Code 28039 returned

Error Message:

Exporter repeatedly failed to download data, {value}

Error Source:

Tasks service errors

Error Description:

Repeated errors occurred while loading data during the specified Exporter task.

This error can occur, for example, if, during the execution of the Exporter task, it was not possible to connect to the Faces or Events service in time and not all data was exported.

11.15.41 Code 28040 returned

Error Message:

Object not found, Schedule with id {value} not found

Error Source:

Tasks service errors

Error Description:

The schedule with the specified ID was not found.

Check the correctness of the entered identifier.

You can get a list of all schedules using the “[get tasks schedules](#)” request.

11.15.42 Code 28041 returned

Error Message:

Encryption error, Authorization data is incorrect

Error Source:

Tasks service errors

Error Description:

Encryption error encrypting existing passwords and tokens passed in the Estimator task or in the “notification_policy”.

Make sure that the environment variables FERNET_PASSPHRASE and SALT were specified during the migration of the Tasks database and during the launch of the Tasks service container.

For more information, see the section “Additional password and token protection” in the administrator manual.

11.16 Sender service errors

11.16.1 Code 29001 returned

Error Message:

Forbidden. Cannot subscribe for events without "Luna-Account-Id" header set

Error Source:

Sender service errors

Error Description:

The request does not include the "Luna-Account-Id" header required to subscribe to the events.

The same "Luna-Account-Id" header as for the request for events creation "[generate events](#)" should be used.

See the "HEADER PARAMETERS" section of the resource `"/ws"`:

- [API service](#)
- [Sender service](#)

The links lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.16.2 Code 29002 returned

Error Message:

Internal server error. Failed to subscribe to chanel {value}

Error Source:

Sender service errors

Error Description:

Sender could not subscribe to the channel.

11.16.3 Code 29003 returned

Error Message:

Internal server error. Failed to process input message

Error Source:

Sender service errors

Error Description:

Sender could not process the incoming image.

Check that the incoming image is not corrupted and it has proper format.

11.16.4 Code 29004 returned

Error Message:

Internal server error. Failed to publish message to ws

Error Source:

Sender service errors

Error Description:

Sender could not publish a message by web socket.

11.16.5 Code 29005 returned

Error Message:

Service Unavailable. Redis disconnected

Error Source:

Sender service errors

Error Description:

The Sender service is unavailable. Redis DB was disconnected. Check that Redis is up and running and also accessible through the network.

11.17 Python Matcher service errors

11.17.1 Code 31000 returned

Error Message:

Bad/incomplete input data. Account id from query parameters does not match the one from {value} filters

Error Source:

Python Matcher service errors

Error Description:

The account ID specified in the query parameters does not match the account ID from the specified filters. The data belongs to different Account IDs.

Change the account ID in the request to the required account ID.

11.17.2 Code 31001 returned

Error Message:

Bad/incomplete input data. Descriptor has the version ({value}) which does not match with version ({value}) which is supposed to use for the matching

Error Source:

Python Matcher service errors

Error Description:

The version of the processed descriptor does not match the version currently used for the matching.

See the default descriptor version in the “DEFAULT_FACE_DESCRIPTOR_VERSION” parameter in the Configurator service or in the configuration files of LP services. The processed descriptors should be of the same version.

You should extract descriptors of a new version using the source image using additional extraction task [“Additional information” > “Launch Additional extraction task”](#) or by creating new descriptors [“General concepts” > “Attributes creation and saving”](#).

The links lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.17.3 Code 31002 returned

Error Message:

Bad/incomplete input data. Cross match filters presume too many received objects. Current general limit — {value}

Error Source:

Python Matcher service errors

Error Description:

Too many objects were sent for the cross-matching task. The available limit is shown in the error description.

11.17.4 Code 31003 returned

Error Message:

Internal server error. Unknown cross matching error

Error Source:

Python Matcher service errors

Error Description:

An unknown cross-matching error occurred.

11.17.5 Code 31005 returned

Error Message:

Bad/incomplete input data. Matching between different versions {value} is not allowed

Error Source:

Python Matcher service errors

Error Description:

Matching between different descriptors versions is not allowed.

You should extract descriptors of a new version using the source image using additional extraction task [“Additional information”](#) > [“Launch Additional extraction task”](#) or by creating new descriptors [“General concepts”](#) > [“Attributes creation and saving”](#).

The links lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.17.6 Code 31006 returned

Error Message:

Internal server error. Unexpected behavior of the “{value}” matcher: {value}

Error Source:

Python Matcher service errors

Error Description:

Unexpected behavior of the given matcher.

The error occurs when using plugins. A plugin error is displayed, which occurs because the plugin author did not take into account the requirements for using plugins.

11.17.7 Code 31007 returned

Error Message:

Bad/incomplete input data. Amount of specified/required references exceeds limit: {value}

Error Source:

Python Matcher service errors

Error Description:

The number of specified/required references has been exceeded.

Check the corresponding configuration in the Python Matcher service.

11.17.8 Code 31008 returned

Error Message:

Bad/incomplete input data. User with account type user is not allowed to use different account id

Error Source:

Python Matcher service errors

Error Description:

Accounts with type “user” cannot use the “account_id” other than the current one.

To be able to use other “account_id” account type “advanced_user” or “admin” is required.

11.18 Licenses service errors

11.18.1 Code 33001 returned

Error Message:

License problem Failed to check HASP License feature {value}

Error Source:

Licenses service errors

Error Description:

A license problem occurred. The request for the feature availability failed. Check your license.

Possible reasons: the license was not applied, you use a wrong license, connection to the license server could not be established.

11.18.2 Code 33002 returned

Error Message:

License problem Failed to get value of HASP License feature {value}

Error Source:

Licenses service errors

Error Description:

A license problem occurred. The request for the feature value failed. Check your license. Possible reasons: the license was not applied, you use a wrong license, connection to the license server could not be established.

11.18.3 Code 33003 returned**Error Message:**

License problem No value found for required HASP License feature {value}

Error Source:

Licenses service errors

Error Description:

A license problem occurred. The license does not include required feature.

Possible reasons:

- The required feature was not added to the license.
- A wrong license is used. A wrong license file was applied.
- A new license was not applied.
- Connection to the license server could not be established.

11.18.4 Code 33004 returned**Error Message:**

License problem Failed to consume {value}

Error Source:

Licenses service errors

Error Description:

The specified licensed LP feature cannot be used because the limit on the number of transactions was exceeded. Check your license.

11.18.5 Code 33005 returned**Error Message:**

License problem Failed to consume {value}: license expired

Error Source:

Licenses service errors

Error Description:

The specified licensed LP feature cannot be used because the license has expired.

Check your license.

11.18.6 Code 33006 returned

Error Message:

Bad input data. "{value}" is not valid target to get license features. Valid target should be one of {values}.

Error Source:

Licenses service errors

Error Description:

The value specified in the request body does not belong to any of the available values.

See the [API documentation](#) for the “get license” request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.18.7 Code 33007 returned

Error Message:

License problem. Failed to initialize license client {value}.

Error Source:

Licenses service errors

Error Description:

License problem. Failed to connect, or the license expired.

Check your license.

11.19 Handlers service errors

11.19.1 Code 34000 returned

Error Message:

Object not found. Handler with id {value} not found

Error Source:

Handlers service errors

Error Description:

The handler with the specified ID was not found.

Possible reasons:

- A wrong account ID was specified in the request. There is no handler with such an ID for the specified account ID.
- The handler was deleted.
- There is a typo in the handler ID.

You can create a new handler using request [“Create handler”](#).

The links lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.19.2 Code 34001 returned

Error Message:

Forbidden. Descriptor version {value} is not supported

Error Source:

Handlers service errors

Error Description:

The descriptor with the specified version is not supported.

Check that you use a proper descriptor version. See the available descriptor versions in section [“Neural networks”](#) in administrator manual.

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.19.3 Code 34002 returned

Error Message:

Internal server error. Cannot load handler with id “{value}”. Handler is corrupted

Error Source:

Handlers service errors

Error Description:

The system cannot load the handler with the specified ID. The handler is corrupted.

Create a new handler using request [“Create handler”](#).

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.19.4 Code 34003 returned

Error Message:

Bad/incomplete input data. Aggregation is not supported for raw descriptors

Error Source:

Handlers service errors

Error Description:

Aggregation is not supported for raw descriptors.

The error occurs when you specify aggregation and provide descriptors instead of images. Aggregation is performed using images.

See section “[LP Services description](#)” > “[Aggregation](#)” in Administrator’s manual.

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.19.5 Code 34004 returned

Error Message:

Object not found. Verifier with id {value} not found

Error Source:

Handlers service errors

Error Description:

Verifier with the specified ID was not found.

Possible reasons:

- A wrong account ID was specified in the request. There is no verifier with such an ID for the specified account ID.
- The verifier was deleted.
- There is a typo in the verifier ID.

You can create a new verifier using request “[Create verifier](#)”.

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.19.6 Code 34005 returned

Error Message:

Bad/incomplete input data. Candidates limit exceeded: {value} > {value}

Error Source:

Handlers service errors

Error Description:

Candidates limit exceeded. A limited number of candidates can be specified in the request.

See the “PLATFORM_LIMITS” setting in the Configurator service or in services configuration files.

11.19.7 Code 34006 returned

Error Message:

Bad/incomplete input data. No candidates specified

Error Source:

Handlers service errors

Error Description:

No candidates were specified in the request.

See the [API documentation](#) for the request.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.19.8 Code 34007 returned

Error Message:

Forbidden. Allowed use only dynamic handler

Error Source:

Handlers service errors

Error Description:

It is allowed to use a dynamic handler only. The used handler is not a dynamic handler.

Get the current handler by ID using request: “[Get handler](#)”.

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.19.9 Code 34008 returned

Error Message:

Multiple bodies found in image {value}, detect {value}

Error Source:

Handlers service errors

Error Description:

More than one body is found in the image and an aggregation attempt is made with a value of “multiface_policy” other than “2”. When performing aggregation, make sure that the value of the “multiface_policy” parameter is set to “2” in the handler.

11.20 Backport 4 service errors

11.20.1 Code 35000 returned

Error Message:

Bad/incomplete input data. Attributes gc is not available

Error Source:

Backport 4 service errors

Error Description:

Attributes gc is not available.

11.20.2 Code 35001 returned

Error Message:

Forbidden. Luna Sender service is disabled

Error Source:

Backport 4 service errors

Error Description:

Sender service is disabled.

Check the ADDITIONAL_SERVICES_USAGE settings in the Configurator service or in the configuration file.

See section [“Configuration parameters of services”](#) > [“Additional services usage”](#) in Administrator’s manual.

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.20.3 Code 35002 returned

Error Message:

Handler is not supported Invalid handler with id “{value}”

Error Source:

Backport 4 service errors

Error Description:

The handler is not supported. E.g., a handler with extended policies was used in Backport 4. It cannot be processed by the service.

See the handlers creation requests:

- For API service see [resource “/handlers”](#)
- For Backport 4 service see [resource “/handlers”](#)
- For Backport 3 service see [section “Handlers”](#)

The links lead to the latest documentation version. Check that you are reading the documentation for your current LUNA PLATFORM version.

11.21 Backport 3 service errors

11.21.1 Code 4003 returned

Error Message:

No faces found

Error Source:

Backport 3 service errors

Error Description:

There are no faces found.

11.21.2 Code 11002 returned

Error Message:

Authorization failed Account corresponding login/password not found

Error Source:

Backport 3 service errors

Error Description:

The account was not found for the Backport 3 service. There is no account with the provided login/password in the system.

11.21.3 Code 11004 returned

Error Message:

Account is suspended

Error Source:

Backport 3 service errors

Error Description:

The account is not active in the Backport 3 service.

11.21.4 Code 11011 returned

Error Message:

Unique constraint error. An account with given email already exists

Error Source:

Backport 3 service errors

Error Description:

The account with given email already exists.

11.21.5 Code 11012 returned

Error Message:

Extract policy error {value}

Error Source:

Backport 3 service errors

Error Description:

An error occurred during extraction. There are several faces in the image.

11.21.6 Code 11018 returned

Error Message:

Object not found. Face descriptor of version {value} is not found for object with id "{value}".

Error Source:

Backport 3 service errors

Error Description:

The descriptor with the given ID was not found.

Check that the ID is correct.

11.21.7 Code 11022 returned

Error Message:

Object not found. Token not found

Error Source:

Backport 3 service errors

Error Description:

The token was not found for Backport 3.

11.21.8 Code 12001 returned

Error Message:

Bad/incomplete input data. Object in query is not UUID4 format: xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx

Error Source:

Backport 3 service errors

Error Description:

The provided ID for the object is not in UUID4 format. Check that the provided IDs are correct.

11.21.9 Code 12018 returned

Error Message:

Bad/incomplete input data. Unsupported param "{value}"

Error Source:

Backport 3 service errors

Error Description:

The query parameter is unsupported.

See the [API documentation](#) for the request. Compare your query parameters with the parameters in the “QUERY PARAMETERS” section.

The link lead to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

11.21.10 Code 22007 returned**Error Message:**

Object not found. Person with id “{value}” not found

Error Source:

Backport 3 service errors

Error Description:

The person with the given ID was not found in the database.

11.21.11 Code 22008 returned**Error Message:**

Unique constraint error. This face is already attached to the person

Error Source:

Backport 3 service errors

Error Description:

The face is already attached to a person.

11.21.12 Code 36001 returned**Error Message:**

Bad/incomplete input data. Expected list of {value}. Got list of {value}. List id “{value}”

Error Source:

Backport 3 service errors

Error Description:

Wrong list type is provided. Check that the list with correct data type is used. The error is generally returned when using Backport 3 service.

11.21.13 Code 36002 returned

Error Message:

Bad/incomplete input data. Bad format basic authorization header format: Basic base64(login:password)

Error Source:

Backport 3 service errors

Error Description:

Bad format of basic authorization header. The authorization header has incorrect login or password provided. The error is generally returned when using Backport 3 service.

11.21.14 Code 36003 returned

Error Message:

Face does not satisfy thresholds limits. No faces found.

Error Source:

Backport 3 service errors

Error Description:

No faces that satisfy the specified threshold limits were found. Check the thresholds specified in the request.

11.21.15 Code 36004 returned

Error Message:

Bad/incomplete input data. Face was not linked to the person

Error Source:

Backport 3 service errors

Error Description:

A face was not linked to the person. The error is generally returned when using Backport 3 service.

You should link a face to the person to perform request.

11.21.16 Code 36005 returned

Error Message:

Forbidden. WS subscription is disabled

Error Source:

Backport 3 service errors

Error Description:

WS subscription is disabled.

Check your WS subscription.

11.22 Application server errors

11.22.1 Code 37001 returned

Error Message:

Internal server error. Service did not process the request within {value} seconds

Error Source:

Application server errors

Error Description:

The service did not process the request within the specified period.

The error may occur due to:

- High system load.
- Lack of server resources.
- Network problems.

11.22.2 Code 37002 returned

Error Message:

Request timeout Service did not receive the complete request message within {value} seconds

Error Source:

Application server errors

Error Description:

The service did not receive a complete request message within the specified period.

The error may occur due to:

- High system load.
- Lack of server resources.
- Network problems.

11.22.3 Code 37003 returned

Error Message:

Request payload is too large

Error Source:

Application server errors

Error Description:

Request payload is too high.

11.22.4 Code 37004 returned

Error Message:

Internal server error. Unknown web application error: {value}

Error Source:

Application server errors

Error Description:

Unknown web application error.

11.22.5 Code 37005 returned

Error Message:

Client closed request

Error Source:

Application server errors

Error Description:

The client-side was disconnected.

11.23 Healthcheck error

11.23.1 Code 38001 returned

Error Message:

Health check error {value}

Error Source:

Healthcheck error

Error Description:

Service health check error.

11.24 Cached Matcher error

11.24.1 Code 40001 returned

Error Message:

Object not found. List with id "{value}" not found in the cache

Error Source:

Cached Matcher error

Error Description:

The list with the given ID was not found in the cache.

11.25 Accounts service errors

11.25.1 Code 41001 returned

Error Message:

Unique constraint error, Account with login "{value}" already exists

Error Source:

Accounts service errors

Error Description:

Account with the specified email already exists.

11.25.2 Code 41002 returned

Error Message:

Unique constraint error, Account with account_id "{value}" already exists

Error Source:

Accounts service errors

Error Description:

Account with the specified account_id already exists.

11.25.3 Code 41003 returned

Error Message:

Object not found, Token with id "{value}" not found

Error Source:

Accounts service errors

Error Description:

Token with the specified token_id does not exist.

11.25.4 Code 41004 returned

Error Message:

Forbidden, Account with account_id "{value}" has admin type and its type can't be changed

Error Source:

Accounts service errors

Error Description:

Account with the specified account_id has an account type of "admin". You do not have enough rights to change this type of account.

11.25.5 Code 41005 returned

Error Message:

JWT token not found, Specified JWT token doesn't exist

Error Source:

Accounts service errors

Error Description:

The specified JWT token does not exist.

11.25.6 Code 41006 returned

Error Message:

Account credentials wrong, Account login or password are incorrect

Error Source:

Accounts service errors

Error Description:

Error verifying credentials. The login or password for the account is incorrect.

11.25.7 Code 41007 returned**Error Message:**

Account credentials wrong, Token has been expired

Error Source:

Accounts service errors

Error Description:

Error verifying credentials. The token has expired.

Check the “expiration_time” field of the token.

11.25.8 Code 41008 returned**Error Message:**

Bad/incomplete input data, Specified token corrupted

Error Source:

Accounts service errors

Error Description:

The specified token is corrupted. Check if the entered token is correct.

11.25.9 Code 41009 returned**Error Message:**

Bad/incomplete input data, Specified permissions (“{values}”) are unacceptable valid for “{value}” account type

Error Source:

Accounts service errors

Error Description:

The specified permissions cannot be granted to the account of the specified type.

11.25.10 Code 41010 returned

Error Message:

Bad input data, "{values}" is not valid target to get account(s). Valid target should be one of {value}.

Error Source:

Accounts service errors

Error Description:

The listed values for the “targets” query parameter are not valid for specifying in the “[get account](#)” or “[get accounts](#)” request.

See the error description for a list of possible values.

11.26 Lambda service errors

11.26.1 Code 42001 returned

Error Message:

Object not found, Lambda with id "{value}" not found

Error Source:

Lambda service errors

Error Description:

Lambda with the specified ID was not found.

Check that the Lambda has been created.

11.26.2 Code 42002 returned

Error Message:

Lambda image creation not found, Lambda image creation pod not found. Lambda id "{value}"

Error Source:

Lambda service errors

Error Description:

The Docker image for the specified Lambda was not found.

Check the image creation logs using the “[get lambda image creation logs](#)” request.

11.26.3 Code 42003 returned

Error Message:

Unique constraint error, Lambda with name "{value}" already exists

Error Source:

Lambda service errors

Error Description:

Lambda with the specified name already exists. Each lambda must have a unique name.

Delete the old lambda using the [“delete lambda”](#) request or update the existing lambda using [“put lambda”](#) or [“patch lambda”](#) requests.

11.26.4 Code 42004 returned

Error Message:

Lambda exception, "{value}"

Error Source:

Lambda service errors

Error Description:

An exception occurred in the user module.

11.26.5 Code 42005 returned

Error Message:

Lambda validation exception, "{value}"

Error Source:

Lambda service errors

Error Description:

Lambda validation exception.

Make sure that the archive containing the module meets the requirements.

11.26.6 Code 42006 returned

Error Message:

Lambda update exception, "{value}"

Error Source:

Lambda service errors

Error Description:

Lambda update exception.

Make sure that the archive containing the module meets the requirements.

11.26.7 Code 42007 returned

Error Message:

Lambda wrong type exception, Lambda type must be "{}" but it is "{}"

Error Source:

Lambda service errors

Error Description:

Invalid lambda type.

See the error description and the “parameters” > “lambda_type” parameter of the “create lambda” request for available types.

11.26.8 Code 42008 returned

Error Message:

Kubernetes exception, {}

Error Source:

Lambda service errors

Error Description:

The indicated Kubernetes error.

11.27 Remote SDK service errors

11.27.1 Code 43001 returned

Error Message:

External request failed. Failed to download video by url "{value}"

Error Source:

Remote SDK service errors

Error Description:

An external request to the specified video file was not successful.

This error can occur for various reasons, including:

- Problems connecting to the Internet or the server where the video is located. If the server is unavailable or has performance issues, the request may fail.
- Restrictions on the server side. Some servers may block or restrict access to video files for certain users, IP addresses, or geographic regions.
- Problems with access rights. If the video is protected by access rights and requires authorization to view or download, the request may fail.

11.27.2 Code 43002 returned**Error Message:**

Bad/incomplete configuration. Failed to initialize video decoder "{value}"

Error Source:

Remote SDK service errors

Error Description:

An error initializing the video decoder on the GPU.

11.27.3 Code 43003 returned**Error Message:**

Bad/incomplete input data. Unsupported video format

Error Source:

Remote SDK service errors

Error Description:

Unsupported video format.

See the full list of formats on the CPU in [official FFmpeg documentation](#).

11.27.4 Code 43004 returned**Error Message:**

Bad/incomplete input data. Error occurred while decoding video by url "{value}"

Error Source:

Remote SDK service errors

Error Description:

Error decoding the video at the specified URL.

This error can occur for various reasons, including incorrect, incomplete, or corrupted video data.

11.27.5 Code 43005 returned

Error Message:

Bad/incomplete input data. Video file by url "{value}" has too large size {value}

Error Source:

Remote SDK service errors

Error Description:

The video file located at the specified URL is too large.

The maximum file size is set in the “max_size” setting of the “LUNA_REMOTE_SDK_VIDEO_SETTINGS” section of the Remote SDK service.

11.27.6 Code 43006 returned

Error Message:

Forbidden. Unsupported analytics {value}

Error Source:

Remote SDK service errors

Error Description:

The specified video analytics is not supported.

See the description of the “videosdk” request in the OpenAPI specification for a list of possible video analytics.

11.27.7 Code 43007 returned

Error Message:

Internal server error. Video analytics processing failed: {value}

Error Source:

Remote SDK service errors

Error Description:

An internal error occurred when performing video analysis. See the contents of the error.

11.27.8 Code 43008 returned

Error Message:

Bad/incomplete input data“:{value}”

Error Source:

Remote SDK service errors

Error Description:

An error validating the parameters specified in the request body, or an error loading the analytics module due to the fact that the module was not found or does not comply with the protocol.

Check the request body in the OpenAPI specification.

11.28 SDK errors

11.28.1 Code 99999 returned

Error Message:

Unknown fsdk core error, {value}

Error Source:

SDK errors

Error Description:

Unknown FSDK core error.

11.28.2 Code 100001 returned

Error Message:

Buffer is empty {value}

Error Source:

SDK errors

Error Description:

The buffer is empty.

11.28.3 Code 100002 returned

Error Message:

Buffer is null {value}

Error Source:

SDK errors

Error Description:

The buffer is null.

11.28.4 Code 100003 returned

Error Message:

Buffer is full {value}

Error Source:

SDK errors

Error Description:

The buffer is full.

11.28.5 Code 100004 returned

Error Message:

Descriptors are incompatible {value}

Error Source:

SDK errors

Error Description:

The specified descriptors are incompatible. The descriptors have different versions and cannot be matched.

11.28.6 Code 100005 returned

Error Message:

Internal error {value}

Error Source:

SDK errors

Error Description:

An internal error occurred.

11.28.7 Code 100006 returned**Error Message:**

Invalid buffer size {value}

Error Source:

SDK errors

Error Description:

Invalid buffer size.

11.28.8 Code 100007 returned**Error Message:**

Invalid descriptor {value}

Error Source:

SDK errors

Error Description:

Invalid descriptor. Check the descriptor file.

11.28.9 Code 100008 returned**Error Message:**

Invalid descriptor batch {value}

Error Source:

SDK errors

Error Description:

Invalid descriptors batch. Check the descriptors batch.

11.28.10 Code 100009 returned**Error Message:**

Invalid detection {value}

Error Source:

SDK errors

Error Description:

Invalid detection.

11.28.11 Code 100010 returned**Error Message:**

Invalid image {value}

Error Source:

SDK errors

Error Description:

Invalid image. Check that the image file is not corrupted and can be processed by the system.

11.28.12 Code 100011 returned**Error Message:**

Invalid image format {value}

Error Source:

SDK errors

Error Description:

Invalid image format.

11.28.13 Code 100012 returned**Error Message:**

Invalid image size {value}

Error Source:

SDK errors

Error Description:

Invalid image size.

11.28.14 Code 100013 returned

Error Message:

Invalid input {value}

Error Source:

SDK errors

Error Description:

Invalid input.

11.28.15 Code 100014 returned

Error Message:

Invalid landmarks 5 {value}

Error Source:

SDK errors

Error Description:

Invalid landmarks5.

11.28.16 Code 100015 returned

Error Message:

Invalid landmarks 68 {value}

Error Source:

SDK errors

Error Description:

Invalid landmarks68.

11.28.17 Code 100016 returned

Error Message:

Invalid rectangle {value}

Error Source:

SDK errors

Error Description:

Invalid rectangle.

11.28.18 Code 100017 returned

Error Message:

Invalid settings provider {value}

Error Source:

SDK errors

Error Description:

Invalid settings provider.

11.28.19 Code 100018 returned

Error Message:

Licensing issue {value}

Error Source:

SDK errors

Error Description:

Licensing issue.

11.28.20 Code 100019 returned

Error Message:

Module is not initialized {value}

Error Source:

SDK errors

Error Description:

Module is not initialized.

11.28.21 Code 100020 returned

Error Message:

Module is not ready {value}

Error Source:

SDK errors

Error Description:

Module is not ready.

11.28.22 Code 100021 returned

Error Message:

Error during initialization fdk image {value}

Error Source:

SDK errors

Error Description:

FSDK image initialization error.

11.28.23 Code 100022 returned

Error Message:

Error during image loadin {value}

Error Source:

SDK errors

Error Description:

Image loading error.

11.28.24 Code 100023 returned

Error Message:

Error during image saving {value}

Error Source:

SDK errors

Error Description:

Image saving error.

11.28.25 Code 100024 returned

Error Message:

Archive image error {value}

Error Source:

SDK errors

Error Description:

Archive image error.

11.28.26 Code 100025 returned

Error Message:

Invalid detection {value}

Error Source:

SDK errors

Error Description:

Invalid detection.

11.28.27 Code 100026 returned

Error Message:

Image conversion not implemented {value}

Error Source:

SDK errors

Error Description:

Image conversion not implemented.

11.28.28 Code 100027 returned

Error Message:

Bad input image data pointer {value}

Error Source:

SDK errors

Error Description:

Bad input image data pointer.

11.28.29 Code 100028 returned

Error Message:

Bad input image data size {value}

Error Source:

SDK errors

Error Description:

Bad input image data size.

11.28.30 Code 100029 returned

Error Message:

Unsupported image format {value}

Error Source:

SDK errors

Error Description:

Unsupported image format.

11.28.31 Code 100030 returned

Error Message:

Invalid image height {value}

Error Source:

SDK errors

Error Description:

Invalid image height.

11.28.32 Code 100031 returned

Error Message:

Bad path for image saving / loading {value}

Error Source:

SDK errors

Error Description:

Bad path for image saving/loading.

11.28.33 Code 100032 returned

Error Message:

Error at image memory opening {value}

Error Source:

SDK errors

Error Description:

Error at image memory opening.

11.28.34 Code 100033 returned

Error Message:

Unsupported image type {value}

Error Source:

SDK errors

Error Description:

Unsupported image type.

11.28.35 Code 100034 returned

Error Message:

Invalid image width {value}

Error Source:

SDK errors

Error Description:

Invalid image width.

11.28.36 Code 100035 returned

Error Message:

Batching error {value}

Error Source:

SDK errors

Error Description:

Batching error.

11.28.37 Code 110001 returned

Error Message:

Creation descriptor error {value}

Error Source:

SDK errors

Error Description:

Descriptor creation error.

11.28.38 Code 110002 returned

Error Message:

Creation descriptor error {value}

Error Source:

SDK errors

Error Description:

Descriptors batch creation error.

11.28.39 Code 110003 returned

Error Message:

Creation core image error {value}

Error Source:

SDK errors

Error Description:

Core image creation error.

11.28.40 Code 110004 returned

Error Message:

Estimation descriptor error {value}

Error Source:

SDK errors

Error Description:

Descriptor estimation error.

11.28.41 Code 110005 returned

Error Message:

Estimation descriptor error {value}

Error Source:

SDK errors

Error Description:

Descriptors batch estimation error.

11.28.42 Code 110006 returned

Error Message:

Estimation basic attributes error {value}

Error Source:

SDK errors

Error Description:

Basic attributes estimation error.

11.28.43 Code 110007 returned

Error Message:

Batch estimation basic attributes error {value}

Error Source:

SDK errors

Error Description:

Basic attributes batch estimation error.

11.28.44 Code 110008 returned

Error Message:

Estimation AGS error {value}

Error Source:

SDK errors

Error Description:

AGS estimation error.

11.28.45 Code 110009 returned

Error Message:

Estimation head pose error {value}

Error Source:

SDK errors

Error Description:

Head pose estimation error.

11.28.46 Code 110011 returned

Error Message:

Estimation eyes gaze error {value}

Error Source:

SDK errors

Error Description:

Gaze estimation error.

11.28.47 Code 110012 returned

Error Message:

Estimation emotions error {value}

Error Source:

SDK errors

Error Description:

Emotions estimation error.

11.28.48 Code 110013 returned

Error Message:

Estimation warp quality error {value}

Error Source:

SDK errors

Error Description:

Sample quality estimation error.

11.28.49 Code 110014 returned

Error Message:

Estimation mouth state error {value}

Error Source:

SDK errors

Error Description:

Mouth state estimation error.

11.28.50 Code 110015 returned

Error Message:

Estimation eyes error {value}

Error Source:

SDK errors

Error Description:

Eyes estimation error.

11.28.51 Code 110016 returned

Error Message:

Creation warped image error {value}

Error Source:

SDK errors

Error Description:

Sample creation error.

11.28.52 Code 110017 returned

Error Message:

Landmarks transformation error {value}

Error Source:

SDK errors

Error Description:

Landmarks transformation error.

11.28.53 Code 110018 returned

Error Message:

Detect one face error {value}

Error Source:

SDK errors

Error Description:

Single face detection error.

11.28.54 Code 110019 returned

Error Message:

Detect faces error {value}

Error Source:

SDK errors

Error Description:

Several faces detection error.

11.28.55 Code 110020 returned

Error Message:

High memory usage {value}

Error Source:

SDK errors

Error Description:

High memory usage error.

11.28.56 Code 110021 returned

Error Message:

Detect one human body error {value}

Error Source:

SDK errors

Error Description:

Single human body detection error.

11.28.57 Code 110022 returned

Error Message:

Detect humans bodies error {value}

Error Source:

SDK errors

Error Description:

Humans bodies detection error.

11.28.58 Code 110023 returned

Error Message:

Estimation mask error {value}

Error Source:

SDK errors

Error Description:

Mask estimation error has occurred.

11.28.59 Code 110024 returned

Error Message:

Filtered aggregation error {value}

Error Source:

SDK errors

Error Description:

Aggregation error related to the specified threshold.

11.28.60 Code 1100010 returned

Error Message:

Estimation ethnicities error {value}

Error Source:

SDK errors

Error Description:

Ethnicity estimation error.

11.29 RPC errors

11.29.1 Code 120004 returned

Error Message:

The command does not exist {value}

Error Source:

RPC errors

Error Description:

The command does not exist.

11.29.2 Code 120005 returned

Error Message:

Internal RPC error {value}

Error Source:

RPC errors

Error Description:

Internal RPC error.

11.29.3 Code 120006 returned

Error Message:

The command is not valid {value}

Error Source:

RPC errors

Error Description:

The command is not valid.

11.30 Estimation errors

11.30.1 Code 200003 returned

Error Message:

Bad/incomplete input data. Not supported descriptor version {value}

Error Source:

Estimation errors

Error Description:

Descriptor of the specified version is not supported.

Make sure that the supported descriptor version is specified in the “DEFAULT_FACE_DESCRIPTOR_VERSION” or “DEFAULT_HUMAN_DESCRIPTOR_VERSION” settings.

11.30.2 Code 200004 returned

Error Message:

Internal error. Estimator {value} initialization fail

Error Source:

Estimation errors

Error Description:

Error initializing the specified estimator.

Check for the presence of a neural network for the estimator in the Handlers container.

11.30.3 Code 200005 returned

Error Message:

Internal error. {value} estimator was not initialized

Error Source:

Estimation errors

Error Description:

The specified estimator was not initialized when the Handlers container was started.

Restart the container with the enable argument for this estimator. See [“Enable/disable several estimators and detectors”](#).

The link leads to the latest documentation version. Make sure that you are reading the documentation for your current LUNA PLATFORM version.

12 Configuration parameters of services

12.1 Configurator configuration

The section describes the Configurator service parameters.

The Configurator service itself is configured using the configuration file “./example-docker/luna_configurator/configs/”.

12.1.1 LUNA_CONFIGURATOR_DB section

In this section, the settings for connecting to the database of the service Configurator are set.

12.1.1.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.1.1.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.1.1.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.1.1.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.1.1.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.1.1.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_configurator.

12.1.1.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.1.1.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_configurator?
    some_option=some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_configurator” — Database name.
- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres01,postgres02/luna_configurator”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.1.2 LUNA_CONFIGURATOR_LOGGER section

This section sets the logging settings for the logging.

12.1.2.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: string.

Default value: INFO.

12.1.2.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.1.2.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.1.2.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: boolean.

Default value: false.

12.1.2.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.1.2.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.1.2.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.1.2.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.1.3 LUNA_CONFIGURATOR_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.1.3.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: `integer (seconds)`.

Default value: `60`.

12.1.3.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.1.3.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.1.3.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.1.4 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

12.1.4.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

12.1.4.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

12.1.4.3 organization

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

12.1.4.4 token

The parameter sets InfluxDB authentication token.

Setting format: string.

12.1.4.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

12.1.4.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

12.1.4.7 port

The parameter sets InfluxDB port.

Setting format: string.

Default value: 8086.

12.1.4.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.1.5 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.1.5.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: boolean.

Default value: false.

12.1.5.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.1.5.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.1.6 Other

12.1.6.1 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.2 API service configuration

The section describes the API service parameters.

You can configure the service using the Configurator service.

12.2.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the API service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_api/configs/” of the corresponding container.

12.2.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_api/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.2.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.2 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

[12.2.2.1 send_data_for_monitoring](#)

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

[12.2.2.2 use_ssl](#)

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

[12.2.2.3 organization](#)

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

[12.2.2.4 token](#)

The parameter sets InfluxDB authentication token.

Setting format: string.

[12.2.2.5 bucket](#)

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

[12.2.2.6 host](#)

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

[12.2.2.7 port](#)

The parameter sets InfluxDB port.

Setting format: string.

Default value: 8086.

12.2.2.8 `flushing_period`

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.2.3 `LUNA_API_LOGGER` section

This section sets the logging settings for the logging.

12.2.3.1 `log_level`

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: string.

Default value: INFO.

12.2.3.2 `log_time`

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.2.3.3 `log_to_stdout`

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.2.3.4 `log_to_file`

The parameter enables you to save logs to a file. The directory with log files is specified in the “`folder_with_logs`” parameter.

Setting format: boolean.

Default value: false.

12.2.3.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.2.3.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.2.3.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: boolean.

Default value: true.

12.2.3.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.2.4 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

12.2.4.1 origin

The parameter sets the protocol, IP address and port of the Faces service.

The IP address “127.0.0.1” means that the Faces service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Faces service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5030`.

12.2.4.2 api_version

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: `integer`.

Default value: 3.

12.2.5 LUNA_FACES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

12.2.5.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Faces service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.2.5.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.5.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.2.5.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.6 LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section

In this section, the bucket for storing [face samples](#) settings are set.

12.2.6.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.2.6.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.6.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: visionlabs-samples.

12.2.7 LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [face samples](#).

12.2.7.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [face samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.2.7.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.8 LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section

In this section, the bucket for storing [body samples](#) settings are set.

12.2.8.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.2.8.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.2.8.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-bodies-samples`.

12.2.9 `LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with `body samples`.

12.2.9.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with `body samples`. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 30.

12.2.9.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.2.10 `LUNA_IMAGE_STORE_IMAGES_ADDRESS` section

In this section, the bucket for storing `source images` settings are set.

12.2.10.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: `http://127.0.0.1:5020`.

12.2.10.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.10.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: `visionlabs-image-origin`.

12.2.11 LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [source images](#).

12.2.11.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [source images](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.2.11.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.12 LUNA_IMAGE_STORE_OBJECTS_ADDRESS section

In this section, the bucket for storing [objects](#) settings are set.

12.2.12.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.2.12.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.12.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: visionlabs-objects.

12.2.13 LUNA_IMAGE_STORE_OBJECTS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [objects](#) .

12.2.13.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [objects](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.2.13.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.14 LUNA_SENDER_ADDRESS section

This section sets the connection settings for the Sender service.

12.2.14.1 origin

The parameter sets the protocol, IP address and port of the Sender service.

The IP address “127.0.0.1” means that the Sender service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Sender service running.

Setting format: string.

Default value: http://127.0.0.1:5080.

12.2.14.2 api_version

The parameter sets the version of the Sender service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.15 ADDITIONAL_SERVICES_USAGE section

12.2.15.1 luna_events

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.15.2 luna_tasks

The parameter sets the possibility of using the Tasks service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.15.3 luna_handlers

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.15.4 luna_sender

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.15.5 luna_matcher_proxy

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.15.6 luna_image_store

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.15.7 luna_lambda

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.16 LUNA_EVENTS_ADDRESS section

This section sets the connection settings for the Events service.

12.2.16.1 origin

The parameter sets the protocol, IP address and port of the Events service.

The IP address “127.0.0.1” means that the Events service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Events service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5040`.

12.2.16.2 `api_version`

The parameter sets the version of the Events service. The available API version is “2”.

Setting format: `integer`.

Default value: 2.

12.2.17 `LUNA_EVENTS_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Events service.

12.2.17.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Events service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.2.17.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.2.17.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: `integer (seconds)`.

Default value: 10.

12.2.17.4 `sock_read`

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: `integer` (seconds).

Default value: 60.

12.2.18 `LUNA_HANDLERS_ADDRESS` section

This section sets the connection settings for the Handlers service.

12.2.18.1 `origin`

The parameter sets the protocol, IP address and port of the Handlers service.

The IP address “127.0.0.1” means that the Handlers service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Handlers service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5090`.

12.2.18.2 `api_version`

The parameter sets the version of the Handlers service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.2.19 `LUNA_HANDLERS_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Handlers service.

12.2.19.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Handlers service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer` (seconds).

Default value: 20.

12.2.19.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.19.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.2.19.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.20 LUNA_PYTHON_MATCHER_ADDRESS section

This section sets the connection settings for the Python Matcher service.

12.2.20.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher service.

The IP address “127.0.0.1” means that the Python Matcher service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher service running.

Setting format: string.

Default value: http://127.0.0.1:5100.

12.2.20.2 api_version

The parameter sets the version of the Python Matcher service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.21 LUNA_PYTHON_MATCHER_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Python Matcher service.

12.2.21.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Python Matcher service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.2.21.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.21.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.2.21.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.22 LUNA_MATCHER_PROXY_ADDRESS section

This section sets the connection settings for the Python Matcher Proxy service.

12.2.22.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher Proxy service.

The IP address “127.0.0.1” means that the Python Matcher Proxy service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher Proxy service running.

Setting format: string.

Default value: `http://127.0.0.1:5110`.

12.2.22.2 api_version

The parameter sets the version of the Python Matcher Proxy service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.23 LUNA_PYTHON_MATCHER_PROXY_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Python Matcher Proxy service.

12.2.23.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Python Matcher Proxy service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.2.23.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.23.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.2.23.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.24 LUNA_TASKS_ADDRESS section

This section sets the connection settings for the Tasks service.

12.2.24.1 origin

The parameter sets the protocol, IP address and port of the Tasks service.

The IP address “127.0.0.1” means that the Tasks service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Tasks service running.

Setting format: string.

Default value: http://127.0.0.1:5050.

12.2.24.2 api_version

The parameter sets the version of the Tasks service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.25 LUNA_TASKS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Tasks service.

12.2.25.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Tasks service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.2.25.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.25.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.2.25.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.26 LUNA_LICENSES_ADDRESS section

This section sets the connection settings for the Licenses service.

12.2.26.1 origin

The parameter sets the protocol, IP address and port of the Licenses service.

The IP address “127.0.0.1” means that the Licenses service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Licenses service running.

Setting format: string.

Default value: http://127.0.0.1:5120.

12.2.26.2 api_version

The parameter sets the version of the Licenses service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.2.27 LUNA_ACCOUNTS_ADDRESS section

This section sets the connection settings for the Accounts service.

12.2.27.1 origin

The parameter sets the protocol, IP address and port of the Accounts service.

The IP address “127.0.0.1” means that the Accounts service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Accounts service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5170`.

12.2.27.2 api_version

The parameter sets the version of the Accounts service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.2.28 LUNA_ACCOUNTS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Accounts service.

12.2.28.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Accounts service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.2.28.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.2.28.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.2.28.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.29 LUNA_REMOTE_SDK_ADDRESS section

This section sets the connection settings for the Remote SDK service.

12.2.29.1 origin

The parameter sets the protocol, IP address and port of the Remote SDK service.

The IP address "127.0.0.1" means that the Remote SDK service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Remote SDK service running.

Setting format: string.

Default value: http://127.0.0.1:5220.

12.2.29.2 api_version

The parameter sets the version of the Remote SDK service. The available API version is "1".

Setting format: integer.

Default value: 1.

12.2.30 LUNA_REMOTE_SDK_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Remote SDK service.

12.2.30.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Remote SDK service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.2.30.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.2.30.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.2.30.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.31 LUNA_LAMBDA_ADDRESS section

This section sets the connection settings for the Lambda service.

12.2.31.1 origin

The parameter sets the protocol, IP address and port of the Lambda service.

The IP address “127.0.0.1” means that the Lambda service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Lambda service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5210`.

12.2.31.2 `api_version`

The parameter sets the version of the Lambda service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.2.32 `LUNA_LAMBDA_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Lambda service.

12.2.32.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Lambda service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.2.32.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.2.32.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: `integer (seconds)`.

Default value: 10.

12.2.32.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.2.33 EXTERNAL_LUNA_API_ADDRESS section

This section sets the address of the API service, which will be displayed in the “external_url” parameter in the response body to the creation of various objects. As a result, it will be possible to get an absolute reference to the created object.

Example of the response body to [list creation](#):

```
{
  "list_id": "3b94d026-2434-4a52-b1e7-da4a97fc0398",
  "url": "/6/lists/3b94d026-2434-4a52-b1e7-da4a97fc0398",
  "external_url": "http://127.0.0.1:5000/6/lists/3b94d026-2434-4a52-b1e7-
    da4a97fc0398"
}
```

Here:

- “url” is a relative link to the list
- “external_url” is an absolute link to the list, consisting of the parameters “origin”, “api_version” and “url”

This enables you to use the links from the API service responses for your own purposes without knowing the exact address of the service. It also enables you to transmit links in a convenient format, through which you can immediately get their contents.

12.2.33.1 origin

The parameter sets the protocol, IP address and port of the API service displayed in the “external_url” parameter in the body of the response to the creation of various objects.

Setting format: string.

Default value: http://127.0.0.1:5000.

12.2.33.2 api_version

The parameter sets the API version of the API service displayed in the “external_url” parameter in the body of the response to the creation of various objects.

Setting format: integer.

Default value: 6.

12.2.34 LUNA_API_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.2.34.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.2.34.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.2.34.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.2.34.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.2.35 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.2.35.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.2.35.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.2.35.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.2.36 Other

12.2.36.1 `luna_api_active_plugins`

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[
  "plugin_1",
  "plugin_2",
  "plugin_3"
]
```

The list should contain file names without the extension (`.py`).

Setting format: `integer`.

Default value: `1`.

12.2.36.2 luna_api_plugins_settings

The parameter sets the settings for plugins.

At the moment, the settings are available only for the “luna-streams.py” plugin. For more information, see the section “Proxying requests to LUNA Streams via LUNA API” in the FaceStream administrator manual.

Important: Plugins must be enabled in the “luna_api_active_plugins” setting.

Setting format: object.

Default value: not set.

Settings for luna-streams plugin

The following settings are available for the “luna-streams.py” plugin:

- “luna-streams-address” > “origin” — Protocol, IP address and port of the LUNA Streams service.
- “luna-streams-address” > “api_version” — API version of the LUNA Streams service.
- “luna-streams-timeouts” > “connect” — Timeout for establishing a connection when sending an HTTP request to the LUNA Streams service.
- “luna-streams-timeouts” > “request” — General timeout for completing the entire HTTP request.
- “luna-streams-timeouts” > “sock_connect” — Timeout for establishing a connection at the socket level.
- “luna-streams-timeouts” > “sock_read” — Timeout for reading data from the socket after a successful connection.

Example of setting the “luna_api_plugins_settings” setting for the “luna-streams.py” plugin:

```
{
  "luna-streams": {
    "luna-streams-address": {
      "origin": "http://127.0.0.1:5160/1",
      "api_version": 1
    },
    "luna-streams-timeouts": {
      "request": 60,
      "connect": 20,
      "sock_connect": 10,
      "sock_read": 60
    }
  }
}
```

If the “luna_api_plugins_settings” setting is not set or any of the above settings for the plugin are not set, the default values will be applied. The default values are shown in the example above.

12.2.36.3 `allow_luna_account_auth_header`

The parameter enables authorization by the “Luna-Account-Id” header (**LunaAccountIdAuth**), which specifies the “account_id” generated after account creation.

This authorization was taken as a basis before version 5.30.0 and has the lowest priority compared to other methods.

In [OpenAPI specification](#) the “Luna-Account-Id” header is marked with the word **Deprecated**.

See the detailed information about authorization in the section [“Accounts, tokens and authorization types”](#).

Setting format: integer (“0” or “1”).

Default value: 1.

12.2.36.4 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.2.36.5 `default_face_descriptor_version`

The parameter sets the version of the face descriptor to use.

Setting format: string.

Default value: 59.

Note: For more information about descriptor versions, see [“Neural networks”](#).

12.3 Admin service configuration

The section describes the Admin service parameters.

You can configure the service using the Configurator service.

12.3.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Admin service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_admin/configs/” of the corresponding container.

12.3.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_admin/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.3.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.3.2 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

12.3.2.1 [send_data_for_monitoring](#)

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: `1`.

12.3.2.2 [use_ssl](#)

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: `0`.

12.3.2.3 [organization](#)

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.3.2.4 [token](#)

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.3.2.5 [bucket](#)

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.3.2.6 [host](#)

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.3.2.7 [port](#)

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.3.2.8 `flushing_period`

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.3.3 `LUNA_ADMIN_LOGGER` section

This section sets the logging settings for the logging.

12.3.3.1 `log_level`

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: string.

Default value: INFO.

12.3.3.2 `log_time`

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.3.3.3 `log_to_stdout`

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.3.3.4 `log_to_file`

The parameter enables you to save logs to a file. The directory with log files is specified in the “`folder_with_logs`” parameter.

Setting format: boolean.

Default value: false.

12.3.3.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.3.3.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.3.3.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: boolean.

Default value: true.

12.3.3.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.3.4 LUNA_ACCOUNTS_ADDRESS section

This section sets the connection settings for the Accounts service.

12.3.4.1 origin

The parameter sets the protocol, IP address and port of the Accounts service.

The IP address “127.0.0.1” means that the Accounts service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Accounts service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5170`.

12.3.4.2 api_version

The parameter sets the version of the Accounts service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.3.5 LUNA_API_ADDRESS section

This section sets the connection settings for the API service.

12.3.5.1 origin

The parameter sets the protocol, IP address and port of the API service.

The IP address “127.0.0.1” means that the API service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the API service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5000`.

12.3.5.2 `api_version`

The parameter sets the version of the API service. The available API version is “6”.

Setting format: `integer`.

Default value: 6.

12.3.6 `LUNA_FACES_ADDRESS` section

This section sets the connection settings for the Faces service.

12.3.6.1 `origin`

The parameter sets the protocol, IP address and port of the Faces service.

The IP address “127.0.0.1” means that the Faces service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Faces service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5030`.

12.3.6.2 `api_version`

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: `integer`.

Default value: 3.

12.3.7 `LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS` section

In this section, the bucket for storing [face samples](#) settings are set.

12.3.7.1 `origin`

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.3.7.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.3.7.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-samples`.

12.3.8 `LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS` section

In this section, the bucket for storing [body samples](#) settings are set.

12.3.8.1 `origin`

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.3.8.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.3.8.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: visionlabs-bodies-samples.

12.3.9 LUNA_IMAGE_STORE_IMAGES_ADDRESS section

In this section, the bucket for storing [source images](#) settings are set.

12.3.9.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.3.9.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.3.9.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: visionlabs-image-origin.

12.3.10 LUNA_IMAGE_STORE_TASK_RESULT_ADDRESS section

In this section, the bucket for storing [tasks results](#) settings are set.

12.3.10.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.3.10.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.10.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `task-result`.

12.3.11 `LUNA_SENDER_ADDRESS` section

This section sets the connection settings for the Sender service.

12.3.11.1 `origin`

The parameter sets the protocol, IP address and port of the Sender service.

The IP address “127.0.0.1” means that the Sender service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Sender service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5080`.

12.3.11.2 `api_version`

The parameter sets the version of the Sender service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.12 `LUNA_EVENTS_ADDRESS` section

This section sets the connection settings for the Events service.

12.3.12.1 origin

The parameter sets the protocol, IP address and port of the Events service.

The IP address “127.0.0.1” means that the Events service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Events service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5040`.

12.3.12.2 api_version

The parameter sets the version of the Events service. The available API version is “2”.

Setting format: `integer`.

Default value: 2.

12.3.13 LUNA_TASKS_ADDRESS section

This section sets the connection settings for the Tasks service.

12.3.13.1 origin

The parameter sets the protocol, IP address and port of the Tasks service.

The IP address “127.0.0.1” means that the Tasks service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Tasks service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5050`.

12.3.13.2 api_version

The parameter sets the version of the Tasks service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.14 LUNA_HANDLERS_ADDRESS section

This section sets the connection settings for the Handlers service.

12.3.14.1 origin

The parameter sets the protocol, IP address and port of the Handlers service.

The IP address “127.0.0.1” means that the Handlers service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Handlers service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5090`.

12.3.14.2 api_version

The parameter sets the version of the Handlers service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.15 LUNA_REMOTE_SDK_ADDRESS section

This section sets the connection settings for the Remote SDK service.

12.3.15.1 origin

The parameter sets the protocol, IP address and port of the Remote SDK service.

The IP address “127.0.0.1” means that the Remote SDK service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Remote SDK service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5220`.

12.3.15.2 api_version

The parameter sets the version of the Remote SDK service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.16 LUNA_PYTHON_MATCHER_ADDRESS section

This section sets the connection settings for the Python Matcher service.

12.3.16.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher service.

The IP address “127.0.0.1” means that the Python Matcher service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5100`.

12.3.16.2 api_version

The parameter sets the version of the Python Matcher service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.17 LUNA_MATCHER_PROXY_ADDRESS section

This section sets the connection settings for the Python Matcher Proxy service.

12.3.17.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher Proxy service.

The IP address “127.0.0.1” means that the Python Matcher Proxy service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher Proxy service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5110`.

12.3.17.2 api_version

The parameter sets the version of the Python Matcher Proxy service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.18 LUNA_LICENSES_ADDRESS section

This section sets the connection settings for the Licenses service.

12.3.18.1 origin

The parameter sets the protocol, IP address and port of the Licenses service.

The IP address “127.0.0.1” means that the Licenses service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Licenses service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5120`.

12.3.18.2 api_version

The parameter sets the version of the Licenses service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.19 LUNA_LAMBDA_ADDRESS section

This section sets the connection settings for the Lambda service.

12.3.19.1 origin

The parameter sets the protocol, IP address and port of the Lambda service.

The IP address “127.0.0.1” means that the Lambda service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Lambda service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5210`.

12.3.19.2 api_version

The parameter sets the version of the Lambda service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.3.20 LUNA_ADMIN_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Admin service.

12.3.20.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Admin service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.3.20.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.3.20.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.3.20.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.3.21 ADDITIONAL_SERVICES_USAGE section

12.3.21.1 luna_events

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.21.2 luna_tasks

The parameter sets the possibility of using the Tasks service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.21.3 luna_handlers

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.21.4 luna_sender

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.21.5 luna_matcher_proxy

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.21.6 luna_image_store

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.21.7 luna_lambda

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.3.22 LUNA_ADMIN_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.3.22.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.3.22.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.3.22.3 `request_max_size`

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.3.22.4 `keep_alive_timeout`

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.3.23 `LUNA_SERVICE_METRICS` section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.3.23.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: boolean.

Default value: false.

12.3.23.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: string.

Default value: prometheus.

12.3.23.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.3.24 Other

12.3.24.1 `luna_admin_active_plugins`

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.4 Faces service configuration

The section describes the Faces service parameters.

You can configure the service using the Configurator service.

12.4.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Faces service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_faces/configs/” of the corresponding container.

12.4.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_faces/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.4.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.4.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.4.2 LUNA_FACES_DB section

In this section, the settings for connecting to the database of the service Faces are set.

12.4.2.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.4.2.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.4.2.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.4.2.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.4.2.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.4.2.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: `luna_faces`.

12.4.2.7 `connection_pool_size`

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “`max_connections`” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: `string`.

Default value: `10`.

12.4.2.8 `dsn`

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “`dsn`” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “`dsn`” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_faces?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “`luna:luna`” — Username and password for connecting to PostgreSQL.
- “`@postgres01:5432,postgres02:5432`” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“`postgres01`”) on port 5432. If this fails, it will try to connect to the second host (“`postgres02`”) also on port 5432.
- “`/luna_faces`” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_faces”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.4.3 LUNA_ATTRIBUTES_DB section

12.4.3.1 user

The parameter sets the user name of the Redis database.

Setting format: string.

Default value: not specified.

12.4.3.2 password

The parameter sets the password of the Redis database.

Setting format: string.

Default value: not specified.

12.4.3.3 host

The parameter sets Redis database IP address.

Setting format: string.

Default value: 127.0.0.1.

12.4.3.4 port

The parameter sets the port number on which Redis waits for incoming network connections and listens for them to execute commands from clients.

Setting format: integer.

Default value: 6379.

12.4.3.5 number

The parameter sets the number of the Redis database. Each number corresponds to a separate database, which enables you to separate the data.

Setting format: integer.

Default value: 0.

12.4.3.6 sentinel > master_name

The parameter sets the name of the Redis database master, which is monitored and managed by the Sentinel system.

Setting format: string.

Default value: luna_attributes.

12.4.3.7 sentinel > sentinels

The parameter sets the list of addresses and ports of Sentinel servers that will be used by clients to detect and monitor the Redis database.

Setting format: list > string.

Default value: [].

12.4.3.8 sentinel > user

The parameter sets the user name of the Sentinel server.

Setting format: string.

Default value: Not specified.

12.4.3.9 sentinel > password

The parameter sets the password of the Sentinel server user.

Setting format: string.

Default value: Not specified.

12.4.4 ATTRIBUTES_STORAGE_POLICY section

This section sets the settings for the Faces service related to storing [temporary attributes](#).

12.4.4.1 default_ttl

The parameter sets the default lifetime of temporary attributes.

Setting format: integer (seconds).

Default value: 300.

12.4.4.2 max_ttl

The parameter sets the maximum time to live of temporary attributes.

Setting format: integer (seconds).

Default value: 86400.

12.4.5 LUNA_LICENSES_ADDRESS section

This section sets the connection settings for the Licenses service.

12.4.5.1 origin

The parameter sets the protocol, IP address and port of the Licenses service.

The IP address “127.0.0.1” means that the Licenses service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Licenses service running.

Setting format: string.

Default value: http://127.0.0.1:5120.

12.4.5.2 api_version

The parameter sets the version of the Licenses service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.4.6 LUNA_FACES_LOGGER section

This section sets the logging settings for the logging.

12.4.6.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: string.

Default value: INFO.

12.4.6.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.4.6.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.4.6.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: boolean.

Default value: false.

12.4.6.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.4.6.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: `integer`.

Default value: `1024`

12.4.6.7 `multiline_stack_trace`

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.4.6.8 `format`

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.4.7 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.4.7.1 `send_data_for_monitoring`

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: `1`.

12.4.7.2 `use_ssl`

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: `0`.

12.4.7.3 `organization`

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.4.7.4 `token`

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.4.7.5 `bucket`

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.4.7.6 `host`

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.4.7.7 `port`

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.4.7.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.4.8 LUNA_FACES_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.4.8.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.4.8.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.4.8.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.4.8.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.4.9 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.4.9.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.4.9.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.4.9.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.4.10 Other

12.4.10.1 `database_number`

The parameter specifies the Faces database number in the context of the replication setup.

Database replication is the process of creating and maintaining copies of data from one database to another.

The value “0” means that the current Faces database does not participate in replication. In other words, it is not a data source for any other database, and the data in it is not replicated to other databases.

For more information about data replication, see the “[Replication](#)” section of the Faces development manual.

Setting format: `integer` (“0”, “1” or “2”).

Default value: `0`.

12.4.10.2 `default_face_descriptor_version`

The parameter sets the version of the face descriptor to use.

Setting format: `string`.

Default value: 59.

Note: For more information about descriptor versions, see [“Neural networks”](#).

12.4.10.3 `use_material_views`

The parameter enables you to use materialized views, which makes it possible to speed up the request to the resource `/lists/linkkeys` of the Faces service.

Materialized views are objects in the database that contain the results of executing a query to one or more tables. Unlike conventional (virtual) views, materialized views actually store data in a table, which allows you to speed up query execution, especially for complex aggregating or computational operations.

See the principle of using this parameter in the [“Materialized”](#) section the Faces service development manual.

Setting format: `integer` (“0” or “1”).

Default value: 0.

12.4.10.4 `luna_faces_db_ping_max_count`

This parameter sets the maximum number of database connection checks for each request.

If the database is unavailable, an error is returned. If the value ≤ 0 is set, the checks are not performed.

Setting format: `integer`.

Default value: 0.

12.4.10.5 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: `string`.

Default value: LOCAL.

12.4.10.6 luna_faces_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.5 Image Store service configuration

The section describes the Image Store service parameters.

You can configure the service using the Configurator service.

12.5.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Image Store service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_image_store/configs/” of the corresponding container.

12.5.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_image_store/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.5.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.5.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.5.2 LUNA_IMAGE_STORE_LOGGER section

This section sets the logging settings for the logging.

12.5.2.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.5.2.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.5.2.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.5.2.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: `boolean`.

Default value: `false`.

12.5.2.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.5.2.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.5.2.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: boolean.

Default value: true.

12.5.2.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: string.

Default value: default.

12.5.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

[12.5.3.1 send_data_for_monitoring](#)

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

[12.5.3.2 use_ssl](#)

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

[12.5.3.3 organization](#)

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

[12.5.3.4 token](#)

The parameter sets InfluxDB authentication token.

Setting format: string.

[12.5.3.5 bucket](#)

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

[12.5.3.6 host](#)

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

12.5.3.7 port

The parameter sets InfluxDB port.

Setting format: string.

Default value: 8086.

12.5.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.5.4 LUNA_IMAGE_STORE_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.5.4.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.5.4.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.5.4.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.5.4.4 `keep_alive_timeout`

The parameter sets the timeout for maintaining HTTP activity.

Setting format: `integer` (seconds).

Default value: 15.

12.5.5 S3

This section contains parameters for interacting with S3 storage.

To use these parameters, you need to enable the use of S3 storage in the `“storage_type”` parameter.

To use S3, you must specify the following parameters:

- `“host”`
- `“aws_public_access_key”`
- `“aws_secret_access_key”`
- `“authorization_signature”`

12.5.5.1 `host`

The parameter sets the URL address to establish a connection with the S3 storage.

Setting format: `string`

Default value: `http://localhost:7480`

12.5.5.2 `region`

The parameter specifies the region to be used when interacting with S3 storage.

The region can affect the availability and performance of various resources in AWS S3.

Setting format: `string`

Default value: Not specified

12.5.5.3 `aws_public_access_key`

The parameter specifies the public access key used for authentication when accessing the S3 storage.

This key is provided by AWS and is used to identify the client.

Setting format: `string`

Default value: Not specified

12.5.5.4 `aws_secret_access_key`

The parameter sets the secret access key, which, in combination with the public key, provides authentication when accessing the S3 storage.

Setting format: `string`

Default value: Not specified

12.5.5.5 `authorization_signature`

The parameter determines the method used to create an authentication signature when performing operations with S3.

Two values can be specified:

- “s3v4” — using the AWS S3 Version 4 signature algorithm.
- “s3v2” — using the AWS S3 Version 2 signature algorithm.

Setting format: `string`

Default value: `s3v4`

12.5.5.6 `request_timeout`

The parameter sets the maximum time within which a request to the S3 storage must be completed. If the request does not complete within this time, it will be canceled.

Setting format: `integer (seconds)`

Default value: `60`

12.5.5.7 `connect_timeout`

The parameter sets the maximum waiting time for establishing a connection with the S3 storage. If the connection is not established within this time, it will be considered unsuccessful.

Setting format: `integer (seconds)`

Default value: `30`

12.5.5.8 `verify_ssl`

The parameter determines whether to perform SSL certificate verification when establishing a secure (HTTPS) connection with the S3 storage. If the value is `true`, the SSL certificate will be verified. If the value is `false`, verification will be disabled, which may lead to security issues.

Setting format: `boolean`

Default value: `true`

12.5.6 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See [“Monitoring”](#) for details.

12.5.6.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.5.6.2 metrics_format

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.5.6.3 extra_labels

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.5.7 Other

12.5.7.1 storage_type

The parameter sets the type of storage Image Store. There are two types available:

- “S3” — using S3 cloud storage
- “LOCAL” — using local storage (SSD or HDD)

When using S3, you must additionally set the settings from the [“S3”](#) section.

When using local storage, you must additionally set the path to the directory with images in the [“local_storage”](#) setting.

Setting format: `integer` (“0” or “1”).

Default value: `1`.

12.5.7.2 local_storage

The parameter sets the path and name of the directory for storing images. The relative path starts from the directory with the application.

To use this parameter, you need to enable the use of local storage in the “storage_type” parameter.

Setting format: string.

Default value: ./local_storage.

12.5.7.3 default_image_extension

The parameter sets the image format of the sample obtained after normalization of the source image.

Two values are available:

- “jpg” — save the sample in JPG format
- “png” — save the sample in PNG format

Note that the format of the transmitted source image does not depend on the format of the sample obtained after processing. If the value of this setting is “jpg”, then a sample in JPG format will still be saved for the source PNG image.

Setting format: string.

Default value: jpg.

12.5.7.4 luna_image_store_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.6 Tasks service configuration

The section describes the Tasks service parameters.

You can configure the service using the Configurator service.

12.6.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Tasks service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_tasks/configs/” of the corresponding container.

12.6.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_tasks/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.6.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.6.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.6.2 LUNA_TASKS_DB section

In this section, the settings for connecting to the database of the service Tasks are set.

12.6.2.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.6.2.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.6.2.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.6.2.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.6.2.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.6.2.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: `luna_tasks`.

12.6.2.7 `connection_pool_size`

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “`max_connections`” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: `string`.

Default value: `10`.

12.6.2.8 `dsn`

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “`dsn`” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “`dsn`” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_tasks?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “`luna:luna`” — Username and password for connecting to PostgreSQL.
- “`@postgres01:5432,postgres02:5432`” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“`postgres01`”) on port 5432. If this fails, it will try to connect to the second host (“`postgres02`”) also on port 5432.
- “`/luna_tasks`” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_tasks”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.6.3 LUNA_TASKS_LOGGER section

This section sets the logging settings for the logging.

12.6.3.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.6.3.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.6.3.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.6.3.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: `boolean`.

Default value: `false`.

12.6.3.5 `folder_with_logs`

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “`log_to_file`” parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.6.3.6 `max_log_file_size`

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “`log_to_file`” parameter.

If necessary, you can configure Docker log rotation. See the section “`Docker log rotation`” in the LUNA PLATFORM installation manual.

Setting format: `integer`.

Default value: `1024`

12.6.3.7 `multiline_stack_trace`

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.6.3.8 `format`

The parameter defines the format of the output logs. The following values are available:

- “`default`” — standard output format of the LUNA PLATFORM logs.

- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.6.4 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.6.4.1 `send_data_for_monitoring`

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: `1`.

12.6.4.2 `use_ssl`

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: `0`.

12.6.4.3 `organization`

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.6.4.4 `token`

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.6.4.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

12.6.4.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

12.6.4.7 port

The parameter sets InfluxDB port.

Setting format: string.

Default value: 8086.

12.6.4.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.6.5 LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section

In this section, the bucket for storing [face samples](#) settings are set.

12.6.5.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.6.5.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.6.5.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-samples`.

12.6.6 `LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS` section

In this section, the bucket for storing [body samples](#) settings are set.

12.6.6.1 `origin`

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.6.6.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.6.6.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-bodies-samples`.

12.6.7 LUNA_IMAGE_STORE_IMAGES_ADDRESS section

In this section, the bucket for storing [source images](#) settings are set.

12.6.7.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.6.7.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.6.7.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-image-origin`.

12.6.8 LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [face samples](#).

12.6.8.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [face samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 30.

12.6.8.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.6.9 LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [body samples](#).

12.6.9.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [body samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.6.9.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.6.10 LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [source images](#).

12.6.10.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [source images](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.6.10.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.6.11 LUNA_IMAGE_STORE_TASK_RESULT_ADDRESS section

In this section, the bucket for storing [tasks results](#) settings are set.

12.6.11.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.6.11.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.6.11.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: task-result.

12.6.12 LUNA_IMAGE_STORE_TASK_RESULT_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [tasks results](#).

12.6.12.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [tasks results](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.6.12.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.6.13 LUNA_TASKS_LOAD_EXTERNAL_ARCHIVE_TIMEOUTS section

This section sets timeouts for executing HTTP requests to external resources for downloading archives. Each parameter defines the maximum amount of time that is expected to perform a certain operation within an HTTP request.

12.6.13.1 connect

The parameter sets the timeout for establishing a connection to an external resource. If the connection is not established at the set time, the operation will be interrupted.

Setting format: integer (seconds).

Default value: 20.

12.6.13.2 request

The parameter sets the timeout for waiting for a response to an HTTP request that is sent to get an archive from an external resource.

Setting format: integer (seconds).

Default value: 100.

12.6.13.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.6.13.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 100.

12.6.14 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

12.6.14.1 origin

The parameter sets the protocol, IP address and port of the Faces service.

The IP address “127.0.0.1” means that the Faces service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Faces service running.

Setting format: string.

Default value: http://127.0.0.1:5030.

12.6.14.2 api_version

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: integer.

Default value: 3.

12.6.15 LUNA_FACES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

12.6.15.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Faces service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.6.15.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.6.15.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.6.15.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.6.16 LUNA_PYTHON_MATCHER_ADDRESS section

This section sets the connection settings for the Python Matcher service.

12.6.16.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher service.

The IP address “127.0.0.1” means that the Python Matcher service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher service running.

Setting format: string.

Default value: http://127.0.0.1:5100.

12.6.16.2 api_version

The parameter sets the version of the Python Matcher service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.6.17 LUNA_MATCHER_PROXY_ADDRESS section

This section sets the connection settings for the Python Matcher Proxy service.

12.6.17.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher Proxy service.

The IP address “127.0.0.1” means that the Python Matcher Proxy service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher Proxy service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5110`.

12.6.17.2 api_version

The parameter sets the version of the Python Matcher Proxy service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.6.18 LUNA_TASKS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Tasks service.

12.6.18.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Tasks service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.6.18.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.6.18.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.6.18.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.6.19 LUNA_EVENTS_ADDRESS section

This section sets the connection settings for the Events service.

12.6.19.1 origin

The parameter sets the protocol, IP address and port of the Events service.

The IP address “127.0.0.1” means that the Events service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Events service running.

Setting format: string.

Default value: http://127.0.0.1:5040.

12.6.19.2 api_version

The parameter sets the version of the Events service. The available API version is “2”.

Setting format: integer.

Default value: 2.

12.6.20 LUNA_EVENTS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Events service.

12.6.20.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Events service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.6.20.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.6.20.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.6.20.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.6.21 LUNA_HANDLERS_ADDRESS section

This section sets the connection settings for the Handlers service.

12.6.21.1 origin

The parameter sets the protocol, IP address and port of the Handlers service.

The IP address “127.0.0.1” means that the Handlers service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Handlers service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5090`.

12.6.21.2 `api_version`

The parameter sets the version of the Handlers service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.6.22 LUNA_HANDLERS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Handlers service.

12.6.22.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Handlers service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.6.22.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.6.22.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: `integer (seconds)`.

Default value: 10.

12.6.22.4 `sock_read`

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: `integer` (seconds).

Default value: 60.

12.6.23 `PLATFORM_LIMITS` section

This section defines restrictions associated with matching operations between faces, events, or attributes. These parameters control the number of objects that can be included in a matching operation and also limit the number of results returned in the response.

These settings are useful for balancing performance and resource usage.

Parameters from the “`match`” subsection define restrictions associated with [matching](#) operations.

Parameters from the “`cross_match`” subsection define restrictions associated with cross-matching operations (see “[Cross-matching](#)” task description).

For the Tasks service, the “`match`” subsection is not used. The matching limit settings for regular matching operations are set in the “`PLATFORM_LIMITS`” section of the Python Matcher service.

12.6.23.1 `cross_match > short_array_filter_limit`

The parameter sets the maximum number of objects (maximum array size) specified in all filters, except for the filters “`face_ids`”, “`event_ids`” and “`attribute_ids`” in the Cross-matching task.

For example, the maximum array size for the filters “`sources`”, “`tags`”, “`cities`”, etc. In the OpenAPI specification, such filters are marked with the example `[1 .. 1000] items`.

Setting format: `integer`.

Default value: 1000.

12.6.23.2 `cross_match > array_filter_limit`

The parameter sets the maximum number of objects (maximum array size) specified in the filters “`face_ids`”, “`event_ids`” and “`attribute_ids`” in the Cross-matching task.

In the OpenAPI specification, such filters are marked with the example `[1 .. 20000] items`.

Setting format: `integer`.

Default value: 20000.

12.6.23.3 `cross_match` > `result_candidate_limit`

The parameter sets the maximum number of matching results (maximum array size) for the list of candidates returned in the response of the request to get the result of the Cross-matching task.

In the matching request body, you can also limit the number of candidates returned in the response using the “limit” parameter, but the “result_candidate_limit” parameter takes precedence. This means that if the value “result_candidate_limit” = 4 and “limit” = 6, an error with code “31007” will be returned.

Setting format: integer.

Default value: 20000.

12.6.23.4 `cross_match` > `general_limit`

The parameter sets a general limit on the number of cross-matchings. It is applied to the product of the number of candidates and references that participate in the matching operation, taking into account filters. If the product exceeds the value “general_limit”, the request will fail.

For example, if the value of “general_limit” is “20”, the number of candidate faces in the array “face_ids” is “7”, the number of reference faces in the array “face_ids” is “5” and all other filters do not reduce the actual number of candidate faces, the cross-matching request will not be executed, because the total number of cross-matchings will be equal to “35” (5 x 7). If some filter reduces the total number of candidate faces by “3”, then the request will be executed successfully, because the total number of cross-matchings will be equal to “20” (5 x (7 - 3)).

Setting format: integer.

Default value: 100000.

12.6.24 `EXTERNAL_LUNA_API_ADDRESS` section

This section is intended for correct processing of references to objects created using the “/images” and “/objects” resources in the API service. This section specifies the address and API version of the API service.

If the “content” > “source” > “reference” parameter of the resource “/tasks/estimator” specifies the URL and version of the API service of the “object” (ZIP archive) type object that matches the address and version of the API from the section “EXTERNAL_LUNA_API_ADDRESS” of the Tasks service settings, then this object will be loaded using the Image Store service directly, rather than sending a request to the API service with subsequent redirection to the Image Store service.

Format example: “http://10.15.3.144:5000/6/images/141d2706-8baf-433b-82eb-8c7fada847da”, where “http://10.15.3.144:5000” must match the value from the “origin” setting “, and the value “6” must match the value of the “api_version” setting in the “EXTERNAL_LUNA_API_ADDRESS” section.

To avoid errors, you must configure this section in the Tasks settings before using URLs to objects of type “objects” or “images” as an input data source.

12.6.24.1 origin

The parameter sets the protocol, IP address and port of the API service.

See the description of the operating logic in the “[EXTERNAL_LUNA_API_ADDRESS section](#)” section.

Setting format: string.

Default value: `http://127.0.0.1:5000`.

12.6.24.2 api_version

The parameter sets the API version of the API service.

See the description of the operating logic in the “[EXTERNAL_LUNA_API_ADDRESS section](#)” section.

Setting format: integer.

Default value: 6.

12.6.25 LUNA_TASKS_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.6.25.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.6.25.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.6.25.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.6.25.4 `keep_alive_timeout`

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.6.26 `TASKS_REDIS_DB_ADDRESS` section

12.6.26.1 `user`

The parameter sets the user name of the Redis database.

Setting format: string.

Default value: not specified.

12.6.26.2 `password`

The parameter sets the password of the Redis database.

Setting format: string.

Default value: not specified.

12.6.26.3 `host`

The parameter sets Redis database IP address.

Setting format: string.

Default value: 127.0.0.1.

12.6.26.4 `port`

The parameter sets the port number on which Redis waits for incoming network connections and listens for them to execute commands from clients.

Setting format: integer.

Default value: 6379.

12.6.26.5 `number`

The parameter sets the number of the Redis database. Each number corresponds to a separate database, which enables you to separate the data.

Setting format: integer.

Default value: 0.

12.6.26.6 sentinel > master_name

The parameter sets the name of the Redis database master, which is monitored and managed by the Sentinel system.

Setting format: string.

Default value: luna_tasks.

12.6.26.7 sentinel > sentinels

The parameter sets the list of addresses and ports of Sentinel servers that will be used by clients to detect and monitor the Redis database.

Setting format: list > string.

Default value: [].

12.6.26.8 sentinel > user

The parameter sets the user name of the Sentinel server.

Setting format: string.

Default value: Not specified.

12.6.26.9 sentinel > password

The parameter sets the password of the Sentinel server user.

Setting format: string.

Default value: Not specified.

12.6.27 ADDITIONAL_SERVICES_USAGE section

12.6.27.1 luna_events

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.6.27.2 luna_handlers

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.6.27.3 luna_sender

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.6.27.4 luna_matcher_proxy

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.6.27.5 luna_image_store

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.6.27.6 `luna_lambda`

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.6.28 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See [“Monitoring”](#) for details.

12.6.28.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: boolean.

Default value: false.

12.6.28.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: string.

Default value: prometheus.

12.6.28.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.6.29 Other

12.6.29.1 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: `string`.

Default value: LOCAL.

12.6.29.2 `max_error_count_per_task`

The parameter sets the maximum number of errors for each saved task.

Setting format: `integer`.

Default value: 100000.

12.6.29.3 `tasks_to_faces_requests_concurrency`

The parameter sets the maximum number of simultaneous requests (parallel operations) from the Tasks worker(s) to the Faces service at one time.

This is a limit on the number of requests that can be processed at the same time.

Setting format: `integer`.

Default value: 5.

12.6.29.4 `tasks_to_image_store_requests_concurrency`

The parameter sets the maximum number of simultaneous requests (parallel operations) from the Tasks worker(s) to the Image-Store service at one time.

This is a limit on the number of requests that can be processed at the same time.

Setting format: `integer`.

Default value: 100.

12.6.29.5 `tasks_to_handlers_requests_concurrency`

The parameter sets the maximum number of simultaneous requests (parallel operations) from the Tasks worker(s) to the Handlers service at one time.

This is a limit on the number of requests that can be processed at the same time.

Setting format: `integer`.

Default value: 8.

12.6.29.6 luna_tasks_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.7 Events service configuration

The section describes the Events service parameters.

You can configure the service using the Configurator service.

12.7.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Events service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_events/configs/” of the corresponding container.

12.7.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_events/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.7.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.7.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.7.2 LUNA_EVENTS_LOGGER section

This section sets the logging settings for the logging.

12.7.2.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.7.2.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.7.2.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.7.2.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: `boolean`.

Default value: `false`.

12.7.2.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.7.2.6 `max_log_file_size`

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “`log_to_file`” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: `integer`.

Default value: `1024`

12.7.2.7 `multiline_stack_trace`

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.7.2.8 `format`

The parameter defines the format of the output logs. The following values are available:

- “`default`” — standard output format of the LUNA PLATFORM logs.
- “`json`” — output of logs in json format.
- “`ecs`” — output of logs in ECS format (Elastic Common Schema).

When using the “`ecs`” value, the following fields will be used:

- “`http.response.status_code`” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “`http.response.execution_time`” — contains information about the time taken to execute the request and receive the response.
- “`http.request.method`” — contains the HTTP request method (GET, POST, PUT, etc.).
- “`url.path`” — contains the path in the request’s URL.
- “`error.code`” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.7.3 `LUNA_EVENTS_DB` section

In this section, the settings for connecting to the database of the service Events are set.

12.7.3.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.7.3.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.7.3.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.7.3.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.7.3.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.7.3.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: `luna_events`.

12.7.3.7 `connection_pool_size`

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “`max_connections`” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: `string`.

Default value: `10`.

12.7.3.8 `dsn`

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “`dsn`” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “`dsn`” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_events?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “`luna:luna`” — Username and password for connecting to PostgreSQL.
- “`@postgres01:5432,postgres02:5432`” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“`postgres01`”) on port 5432. If this fails, it will try to connect to the second host (“`postgres02`”) also on port 5432.
- “`/luna_events`” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_events”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.7.4 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.7.4.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

12.7.4.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

12.7.4.3 organization

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.7.4.4 token

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.7.4.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.7.4.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.7.4.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.7.4.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.7.5 LUNA_EVENTS_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.7.5.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: `integer (seconds)`.

Default value: `60`.

12.7.5.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: `integer (seconds)`.

Default value: 600.

12.7.5.3 `request_max_size`

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.7.5.4 `keep_alive_timeout`

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.7.6 `LUNA_SERVICE_METRICS` section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.7.6.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: boolean.

Default value: false.

12.7.6.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: string.

Default value: prometheus.

12.7.6.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.7.7 Other

12.7.7.1 storage_time

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.7.7.2 default_face_descriptor_version

The parameter sets the version of the face descriptor to use.

Setting format: string.

Default value: 59.

Note: For more information about descriptor versions, see “[Neural networks](#)”.

12.7.7.3 default_body_descriptor_version

The parameter sets the version of the body descriptor to use.

For more information about descriptor version, see “[Neural networks](#)”.

Setting format: string.

Default value: 110.

Note: For more information about descriptor versions, see “[Neural networks](#)”.

12.7.7.4 save_events_timeout

This parameter sets the timeout for events to be saved in the Events database.

Values of “0” or negative mean that the service will save events without any waiting time limits.

Using this parameter prevents excessive use of computing resources.

Setting format: float (seconds).

Default value: 2.

12.7.7.5 luna_events_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
    "plugin_1",  
    "plugin_2",  
    "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.8 Sender service configuration

The section describes the Sender service parameters.

You can configure the service using the Configurator service.

12.8.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Sender service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_sender/configs/” of the corresponding container.

12.8.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_sender/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.8.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.8.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.8.2 LUNA_SENDER_LOGGER section

This section sets the logging settings for the logging.

12.8.2.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.8.2.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.8.2.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.8.2.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: `boolean`.

Default value: `false`.

12.8.2.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.8.2.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.8.2.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: boolean.

Default value: true.

12.8.2.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: string.

Default value: default.

12.8.3 REDIS_DB_ADDRESS section

12.8.3.1 user

The parameter sets the user name of the Redis database.

Setting format: `string`.

Default value: not specified.

12.8.3.2 `password`

The parameter sets the password of the Redis database.

Setting format: `string`.

Default value: not specified.

12.8.3.3 `host`

The parameter sets Redis database IP address.

Setting format: `string`.

Default value: `127.0.0.1`.

12.8.3.4 `port`

The parameter sets the port number on which Redis waits for incoming network connections and listens for them to execute commands from clients.

Setting format: `integer`.

Default value: `6379`.

12.8.3.5 `number`

The parameter sets the number of the Redis database. Each number corresponds to a separate database, which enables you to separate the data.

Setting format: `integer`.

Default value: `0`.

12.8.3.6 `channel`

The parameter sets the Redis channel to which the service subscribes.

Setting format: `integer`.

Default value: `luna-sender`.

12.8.3.7 [sentinel > master_name](#)

The parameter sets the name of the Redis database master, which is monitored and managed by the Sentinel system.

Setting format: `string`.

Default value: `luna_sender_master`.

12.8.4 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.8.4.1 [enabled](#)

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.8.4.2 [metrics_format](#)

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.8.4.3 [extra_labels](#)

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.8.4.4 [sentinel > sentinels](#)

The parameter sets the list of addresses and ports of Sentinel servers that will be used by clients to detect and monitor the Redis database.

Setting format: `list > string`.

Default value: `[]`.

12.8.4.5 `sentinel > user`

The parameter sets the user name of the Sentinel server.

Setting format: `string`.

Default value: Not specified.

12.8.4.6 `sentinel > password`

The parameter sets the password of the Sentinel server user.

Setting format: `string`.

Default value: Not specified.

12.8.5 **INFLUX_MONITORING** section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.8.5.1 `send_data_for_monitoring`

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

12.8.5.2 `use_ssl`

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

12.8.5.3 `organization`

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.8.5.4 `token`

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.8.5.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.8.5.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.8.5.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.8.5.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.8.6 LUNA_SENDER_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.8.6.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: `integer (seconds)`.

Default value: `60`.

12.8.6.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: `integer (seconds)`.

Default value: 600.

12.8.6.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.8.6.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.8.7 Other

12.8.7.1 luna_sender_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.9 Licenses service configuration

The section describes the Licenses service parameters.

You can configure the service using the Configurator service.

12.9.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Licenses service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_licenses/configs/” of the corresponding container.

12.9.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_licenses/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.9.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.9.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.9.2 LUNA_LICENSES_LOGGER section

This section sets the logging settings for the logging.

12.9.2.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.9.2.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.9.2.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.9.2.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: `boolean`.

Default value: `false`.

12.9.2.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.9.2.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.9.2.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: boolean.

Default value: true.

12.9.2.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: string.

Default value: default.

12.9.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

[12.9.3.1 send_data_for_monitoring](#)

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

[12.9.3.2 use_ssl](#)

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

[12.9.3.3 organization](#)

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

[12.9.3.4 token](#)

The parameter sets InfluxDB authentication token.

Setting format: string.

[12.9.3.5 bucket](#)

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

[12.9.3.6 host](#)

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

12.9.3.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.9.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.9.4 LICENSE_VENDOR section

This section sets licensing settings.

See the detailed license activation information in the installation manual and in the activation manual.

For more information about the license, see [“License information”](#).

12.9.4.1 vendor

This parameter determines which license vendor will be used. Two options are available:

- `“hasp”`
- `“guardant”`

Setting format: `string`.

Default value: `hasp`.

12.9.4.2 server_address

The parameter sets the IP address at which the Licenses service will search for a server for license management.

The address must be specified without protocol and port.

The value `“127.0.0.1”` means that the server is located on the local computer. If you are using an external server for licensing, you should change this value to your server address.

Setting format: `string`.

Default value: `127.0.0.1`.

12.9.4.3 license_id

Note: The parameter is used only for the vendor “guardant” additionally with the parameters “server_address” and “vendor”. If the vendor “hasp” is used, then this parameter must be disabled.

The parameter sets the license ID in the format 0x<your_license_id>. The license ID can be found in the Guardant user interface at http://<your_host_address>:3189/ on the “Dongles” tab.

Setting format: string.

Default value: not specified.

12.9.5 LUNA_LICENSES_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.9.5.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.9.5.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.9.5.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.9.5.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.9.6 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.9.6.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.9.6.2 metrics_format

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.9.6.3 extra_labels

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.9.7 Other

12.9.7.1 luna_licenses_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[
  "plugin_1",
  "plugin_2",
  "plugin_3"
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.10 Python Matcher service configuration

The section describes the Python Matcher service parameters.

You can configure the service using the Configurator service.

12.10.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Python Matcher service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_python_matcher/configs/” of the corresponding container.

12.10.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_python_matcher/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.10.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.10.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.10.2 LUNA_PYTHON_MATCHER_LOGGER section

This section sets the logging settings for the logging.

12.10.2.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.10.2.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.10.2.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.10.2.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: `boolean`.

Default value: `false`.

12.10.2.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.10.2.6 `max_log_file_size`

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “`log_to_file`” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: `integer`.

Default value: `1024`

12.10.2.7 `multiline_stack_trace`

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.10.2.8 `format`

The parameter defines the format of the output logs. The following values are available:

- “`default`” — standard output format of the LUNA PLATFORM logs.
- “`json`” — output of logs in json format.
- “`ecs`” — output of logs in ECS format (Elastic Common Schema).

When using the “`ecs`” value, the following fields will be used:

- “`http.response.status_code`” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “`http.response.execution_time`” — contains information about the time taken to execute the request and receive the response.
- “`http.request.method`” — contains the HTTP request method (GET, POST, PUT, etc.).
- “`url.path`” — contains the path in the request’s URL.
- “`error.code`” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.10.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

12.10.3.1 [send_data_for_monitoring](#)

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

12.10.3.2 [use_ssl](#)

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

12.10.3.3 [organization](#)

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

12.10.3.4 [token](#)

The parameter sets InfluxDB authentication token.

Setting format: string.

12.10.3.5 [bucket](#)

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

12.10.3.6 [host](#)

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

12.10.3.7 port

The parameter sets InfluxDB port.

Setting format: string.

Default value: 8086.

12.10.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.10.4 LUNA_FACES_DB section

In this section, the settings for connecting to the database of the service Faces are set.

12.10.4.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.10.4.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.10.4.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.10.4.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.10.4.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.10.4.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_faces.

12.10.4.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.10.4.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_faces?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_faces” — Database name.
- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres01,postgres02/luna_faces”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.10.5 LUNA_ATTRIBUTES_DB section

12.10.5.1 user

The parameter sets the user name of the Redis database.

Setting format: string.

Default value: not specified.

12.10.5.2 password

The parameter sets the password of the Redis database.

Setting format: string.

Default value: not specified.

12.10.5.3 host

The parameter sets Redis database IP address.

Setting format: `string`.

Default value: `127.0.0.1`.

12.10.5.4 port

The parameter sets the port number on which Redis waits for incoming network connections and listens for them to execute commands from clients.

Setting format: `integer`.

Default value: `6379`.

12.10.5.5 number

The parameter sets the number of the Redis database. Each number corresponds to a separate database, which enables you to separate the data.

Setting format: `integer`.

Default value: `0`.

12.10.5.6 sentinel > master_name

The parameter sets the name of the Redis database master, which is monitored and managed by the Sentinel system.

Setting format: `string`.

Default value: `luna_attributes`.

12.10.5.7 sentinel > sentinels

The parameter sets the list of addresses and ports of Sentinel servers that will be used by clients to detect and monitor the Redis database.

Setting format: `list > string`.

Default value: `[]`.

12.10.5.8 `sentinel > user`

The parameter sets the user name of the Sentinel server.

Setting format: `string`.

Default value: Not specified.

12.10.5.9 `sentinel > password`

The parameter sets the password of the Sentinel server user.

Setting format: `string`.

Default value: Not specified.

12.10.6 **ADDITIONAL_SERVICES_USAGE** section

12.10.6.1 `luna_events`

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: `integer` (“0” or “1”).

Default value: 1.

12.10.6.2 `luna_handlers`

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: `integer` (“0” or “1”).

Default value: 1.

12.10.6.3 `luna_sender`

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.10.6.4 luna_matcher_proxy

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.10.6.5 luna_image_store

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.10.6.6 luna_lambda

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.10.7 LUNA_EVENTS_DB section

In this section, the settings for connecting to the database of the service Events are set.

12.10.7.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: `string`.

Default value: `postgres`.

12.10.7.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: `string`.

Default value: `127.0.0.1`.

12.10.7.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: `string`.

Default value: `5432`.

12.10.7.4 db_user

The parameter sets the database username.

Setting format: `string`.

Default value: `luna`.

12.10.7.5 db_password

The parameter sets the database password.

Setting format: `string`.

Default value: `luna`.

12.10.7.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_events.

12.10.7.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.10.7.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_events?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_events” — Database name.
- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_events”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.10.8 LUNA_PYTHON_MATCHER_ADDRESS section

This section sets the connection settings for the Python Matcher service.

12.10.8.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher service.

The IP address “127.0.0.1” means that the Python Matcher service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5100`.

12.10.8.2 api_version

The parameter sets the version of the Python Matcher service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.10.9 LUNA_PYTHON_MATCHER_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.10.9.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.10.9.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.10.9.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.10.9.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.10.10 PLATFORM_LIMITS section

This section defines restrictions associated with matching operations between faces, events, or attributes. These parameters control the number of objects that can be included in a matching operation and also limit the number of results returned in the response.

These settings are useful for balancing performance and resource usage.

Parameters from the “match” subsection define restrictions associated with [matching](#) operations.

Parameters from the “cross_match” subsection define restrictions associated with cross-matching operations (see “[Cross-matching](#)” task description).

12.10.10.1 match > array_filter_limit

The parameter sets the maximum number of objects (maximum array size) specified in the “face_ids”, “event_ids” and “attribute_ids” filters.

Setting format: integer.

Default value: 1000.

12.10.10.2 match > reference_limit

The parameter sets the maximum number of references (maximum array size) that can be specified in a matching request.

You must select the value of this parameter in accordance with your business cases, because it consumes system resources significantly. For example, you should not use the default value of “30” if you only need to work with three references. See “[Resource optimization](#)” for more tips on optimizing resources.

Setting format: integer.

Default value: 30.

12.10.10.3 match > candidate_limit

The parameter sets the maximum number of candidates (maximum array size) that can be specified in a matching request.

You must select the value of this parameter in accordance with your business cases, because it consumes system resources significantly. For example, you should not use the default value of “30” if you only need to work with three candidates. See “[Resource optimization](#)” for more tips on optimizing resources.

Setting format: integer.

Default value: 30.

12.10.10.4 match > result_candidate_limit

The parameter sets the maximum number of matching results (maximum array size) for the list of candidates returned in the matching request response.

In the matching request body, you can also limit the number of candidates returned in the response using the “limit” parameter, but the “result_candidate_limit” parameter takes precedence. This means that if the value “result_candidate_limit” = 4 and “limit” = 6, an error with code “31007” will be returned.

Setting format: integer.

Default value: 100.

12.10.10.5 `cross_match > short_array_filter_limit`

The parameter sets the maximum number of objects (maximum array size) specified in all filters, except for the filters “face_ids”, “event_ids” and “attribute_ids” in the Cross-matching task.

For example, the maximum array size for the filters “sources”, “tags”, “cities”, etc. In the OpenAPI specification, such filters are marked with the example [1 .. 1000] items.

Setting format: integer.

Default value: 1000.

12.10.10.6 `cross_match > array_filter_limit`

The parameter sets the maximum number of objects (maximum array size) specified in the filters “face_ids”, “event_ids” and “attribute_ids” in the Cross-matching task.

In the OpenAPI specification, such filters are marked with the example [1 .. 20000] items.

Setting format: integer.

Default value: 20000.

12.10.10.7 `cross_match > result_candidate_limit`

The parameter sets the maximum number of matching results (maximum array size) for the list of candidates returned in the response of the request to get the result of the Cross-matching task.

In the matching request body, you can also limit the number of candidates returned in the response using the “limit” parameter, but the “result_candidate_limit” parameter takes precedence. This means that if the value “result_candidate_limit” = 4 and “limit” = 6, an error with code “31007” will be returned.

Setting format: integer.

Default value: 20000.

12.10.10.8 `cross_match > general_limit`

The parameter sets a general limit on the number of cross-matchings. It is applied to the product of the number of candidates and references that participate in the matching operation, taking into account filters. If the product exceeds the value “general_limit”, the request will fail.

For example, if the value of “general_limit” is “20”, the number of candidate faces in the array “face_ids” is “7”, the number of reference faces in the array “face_ids” is “5” and all other filters do not reduce the actual number of candidate faces, the cross-matching request will not be executed, because the total number of cross-matchings will be equal to “35” (5 x 7). If some filter reduces the total number of candidate faces by “3”, then the request will be executed successfully, because the total number of cross-matchings will be equal to “20” (5 x (7 - 3)).

Setting format: integer.

Default value: 100000.

12.10.11 DESCRIPTORS_CACHE section

This section sets the list caching settings when performing a list matching.

Using caching increases matching performance.

For more information, see [“List caching”](#).

12.10.11.1 cache_enabled

The parameter enables list caching.

Setting format: boolean.

Default value: true.

12.10.11.2 updating_cache_interval

The parameter sets the cache update interval. All new descriptors will be added in one iteration after the specified time.

Setting format: integer (seconds).

Default value: 2.

12.10.11.3 matching_settings > thread_count

The parameter sets the number of threads that will be used to match cache data.

If the value is set to “0”, the automatic method of selecting the number of threads will be selected.

Setting format: integer (non-negative value).

Default value: 0.

12.10.11.4 matching_settings > tasks_count

The parameter sets the number of workers processing the queue for sending matching requests to Cached Matcher.

Setting format: integer.

Default value: 10.

12.10.11.5 [matching_settings > batch_size](#)

The parameter sets the maximum number of matching requests in Cached Matcher within a single batch.

Setting format: integer.

Default value: 20.

12.10.11.6 [rpc_settings > timeouts > connect_timeout](#)

The parameter sets the maximum waiting time for establishing a connection with Cached Matcher. If the connection is not established within this time, it will be considered unsuccessful.

Setting format: integer (seconds).

Default value: 20.

12.10.11.7 [rpc_settings > timeouts > request_timeout](#)

The parameter sets the maximum time during which the request to Cached Matcher should be executed. If the request does not complete within this time, it will be canceled.

Setting format: integer (seconds).

Default value: 60.

12.10.11.8 [rpc_settings > timeouts > response_timeout](#)

The parameter sets the maximum waiting time for a response from Cached Matcher. If the response is not received within this time, the operation will be considered unsuccessful.

Setting format: integer (seconds).

Default value: 60.

12.10.11.9 [rpc_settings > pool_size](#)

The parameter sets the number of connections between the Python Matcher and Cached Matcher services.

Setting format: integer (seconds).

Default value: 100.

12.10.11.10 [cached_data > faces_lists > exclude](#)

The parameter sets the lists that will not be cached.

To disable the setting, you need to set the value “[]”.

Setting format: string.

Default value: [].

12.10.11.11 `cached_data > faces_lists > include`

The parameter sets the lists to be cached.

Setting format: `string`.

The default value is not set, which means that all existing lists will be processed.

12.10.12 `LUNA_SERVICE_METRICS` section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.10.12.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.10.12.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.10.12.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.10.13 Other

12.10.13.1 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.

- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.10.13.2 luna_python_matcher_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.10.13.3 default_face_descriptor_version

The parameter sets the version of the face descriptor to use.

Setting format: string.

Default value: 59.

Note: For more information about descriptor versions, see “[Neural networks](#)”.

12.10.13.4 default_body_descriptor_version

The parameter sets the version of the body descriptor to use.

For more information about descriptor version, see “[Neural networks](#)”.

Setting format: string.

Default value: 110.

Note: For more information about descriptor versions, see “[Neural networks](#)”.

12.11 Python Matcher Proxy service configuration

The section describes the Python Matcher Proxy service parameters.

You can configure the service using the Configurator service.

12.11.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Python Matcher Proxy service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_python_matcher/configs/” of the corresponding container.

12.11.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_python_matcher/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.11.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.11.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.11.2 LUNA_PYTHON_MATCHER_PROXY_LOGGER section

This section sets the logging settings for the logging.

12.11.2.1 log_level

The parameter sets the level of debug printing, by priority: "ERROR", "WARNING", "INFO", "DEBUG".

Setting format: `string`.

Default value: `INFO`.

12.11.2.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- "LOCAL" — displays the local time of the system on which the logs are being recorded.
- "UTC" — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.11.2.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.11.2.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the "folder_with_logs" parameter.

Setting format: `boolean`.

Default value: `false`.

12.11.2.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the "log_to_file" parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.11.2.6 `max_log_file_size`

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “`log_to_file`” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: `integer`.

Default value: `1024`

12.11.2.7 `multiline_stack_trace`

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.11.2.8 `format`

The parameter defines the format of the output logs. The following values are available:

- “`default`” — standard output format of the LUNA PLATFORM logs.
- “`json`” — output of logs in json format.
- “`ecs`” — output of logs in ECS format (Elastic Common Schema).

When using the “`ecs`” value, the following fields will be used:

- “`http.response.status_code`” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “`http.response.execution_time`” — contains information about the time taken to execute the request and receive the response.
- “`http.request.method`” — contains the HTTP request method (GET, POST, PUT, etc.).
- “`url.path`” — contains the path in the request’s URL.
- “`error.code`” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.11.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

12.11.3.1 [send_data_for_monitoring](#)

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

12.11.3.2 [use_ssl](#)

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

12.11.3.3 [organization](#)

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

12.11.3.4 [token](#)

The parameter sets InfluxDB authentication token.

Setting format: string.

12.11.3.5 [bucket](#)

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

12.11.3.6 [host](#)

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

12.11.3.7 port

The parameter sets InfluxDB port.

Setting format: string.

Default value: 8086.

12.11.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: integer (seconds).

Default value: 1.

12.11.4 PLATFORM_LIMITS section

This section defines restrictions associated with matching operations between faces, events, or attributes. These parameters control the number of objects that can be included in a matching operation and also limit the number of results returned in the response.

These settings are useful for balancing performance and resource usage.

Parameters from the “match” subsection define restrictions associated with [matching](#) operations.

Parameters from the “cross_match” subsection define restrictions associated with cross-matching operations (see “[Cross-matching](#)” task description).

12.11.4.1 match > array_filter_limit

The parameter sets the maximum number of objects (maximum array size) specified in the “face_ids”, “event_ids” and “attribute_ids” filters.

Setting format: integer.

Default value: 1000.

12.11.4.2 match > reference_limit

The parameter sets the maximum number of references (maximum array size) that can be specified in a matching request.

You must select the value of this parameter in accordance with your business cases, because it consumes system resources significantly. For example, you should not use the default value of “30” if you only need to work with three references. See “[Resource optimization](#)” for more tips on optimizing resources.

Setting format: integer.

Default value: 30.

12.11.4.3 `match > candidate_limit`

The parameter sets the maximum number of candidates (maximum array size) that can be specified in a matching request.

You must select the value of this parameter in accordance with your business cases, because it consumes system resources significantly. For example, you should not use the default value of “30” if you only need to work with three candidates. See “[Resource optimization](#)” for more tips on optimizing resources.

Setting format: integer.

Default value: 30.

12.11.4.4 `match > result_candidate_limit`

The parameter sets the maximum number of matching results (maximum array size) for the list of candidates returned in the matching request response.

In the matching request body, you can also limit the number of candidates returned in the response using the “limit” parameter, but the “result_candidate_limit” parameter takes precedence. This means that if the value “result_candidate_limit” = 4 and “limit” = 6, an error with code “31007” will be returned.

Setting format: integer.

Default value: 100.

12.11.4.5 `cross_match > short_array_filter_limit`

The parameter sets the maximum number of objects (maximum array size) specified in all filters, except for the filters “face_ids”, “event_ids” and “attribute_ids” in the Cross-matching task.

For example, the maximum array size for the filters “sources”, “tags”, “cities”, etc. In the OpenAPI specification, such filters are marked with the example [1 .. 1000] items.

Setting format: integer.

Default value: 1000.

12.11.4.6 `cross_match > array_filter_limit`

The parameter sets the maximum number of objects (maximum array size) specified in the filters “face_ids”, “event_ids” and “attribute_ids” in the Cross-matching task.

In the OpenAPI specification, such filters are marked with the example [1 .. 20000] items.

Setting format: integer.

Default value: 20000.

12.11.4.7 `cross_match > result_candidate_limit`

The parameter sets the maximum number of matching results (maximum array size) for the list of candidates returned in the response of the request to get the result of the Cross-matching task.

In the matching request body, you can also limit the number of candidates returned in the response using the “limit” parameter, but the “result_candidate_limit” parameter takes precedence. This means that if the value “result_candidate_limit” = 4 and “limit” = 6, an error with code “31007” will be returned.

Setting format: `integer`.

Default value: 20000.

12.11.4.8 `cross_match > general_limit`

The parameter sets a general limit on the number of cross-matchings. It is applied to the product of the number of candidates and references that participate in the matching operation, taking into account filters. If the product exceeds the value “general_limit”, the request will fail.

For example, if the value of “general_limit” is “20”, the number of candidate faces in the array “face_ids” is “7”, the number of reference faces in the array “face_ids” is “5” and all other filters do not reduce the actual number of candidate faces, the cross-matching request will not be executed, because the total number of cross-matchings will be equal to “35” (5 x 7). If some filter reduces the total number of candidate faces by “3”, then the request will be executed successfully, because the total number of cross-matchings will be equal to “20” (5 x (7 - 3)).

Setting format: `integer`.

Default value: 100000.

12.11.5 LUNA_PYTHON_MATCHER_DB section

In this section, the settings for connecting to the database of the service Python-Matcher are set.

12.11.5.1 `db_type`

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: `string`.

Default value: `postgres`.

12.11.5.2 `db_host`

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.11.5.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.11.5.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.11.5.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.11.5.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_python_matcher.

12.11.5.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.11.5.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_python_matcher?
    some_option=some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_python_matcher” — Database name.
- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_python_matcher”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.11.6 LUNA_PROXY_TO_PYTHON_MATCHER_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Python Matcher Proxy service.

12.11.6.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Python Matcher Proxy service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.11.6.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 300.

12.11.6.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.11.6.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 300.

12.11.7 LUNA_PYTHON_MATCHER_PROXY_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.11.7.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.11.7.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.11.7.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.11.7.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.11.7.5 luna_python_matcher_proxy_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[
  "plugin_1",
  "plugin_2",
  "plugin_3"
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.11.8 LUNA_FACES_DB section

In this section, the settings for connecting to the database of the service Faces are set.

12.11.8.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.11.8.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.11.8.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.11.8.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.11.8.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.11.8.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_faces.

12.11.8.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.11.8.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_faces?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_faces” — Database name.
- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_faces”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.11.9 LUNA_EVENTS_DB section

In this section, the settings for connecting to the database of the service Events are set.

12.11.9.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: `string`.

Default value: `postgres`.

12.11.9.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: `string`.

Default value: `127.0.0.1`.

12.11.9.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: `string`.

Default value: 5432.

12.11.9.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.11.9.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.11.9.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_events.

12.11.9.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.11.9.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_events?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_events” — Database name.
- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres01,postgres02/luna_events”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.11.10 LUNA_ATTRIBUTES_DB section

12.11.10.1 user

The parameter sets the user name of the Redis database.

Setting format: string.

Default value: not specified.

12.11.10.2 password

The parameter sets the password of the Redis database.

Setting format: `string`.

Default value: not specified.

12.11.10.3 `host`

The parameter sets Redis database IP address.

Setting format: `string`.

Default value: `127.0.0.1`.

12.11.10.4 `port`

The parameter sets the port number on which Redis waits for incoming network connections and listens for them to execute commands from clients.

Setting format: `integer`.

Default value: `6379`.

12.11.10.5 `number`

The parameter sets the number of the Redis database. Each number corresponds to a separate database, which enables you to separate the data.

Setting format: `integer`.

Default value: `0`.

12.11.10.6 `sentinel > master_name`

The parameter sets the name of the Redis database master, which is monitored and managed by the Sentinel system.

Setting format: `string`.

Default value: `luna_attributes`.

12.11.10.7 `sentinel > sentinels`

The parameter sets the list of addresses and ports of Sentinel servers that will be used by clients to detect and monitor the Redis database.

Setting format: `list > string`.

Default value: `[]`.

12.11.10.8 `sentinel > user`

The parameter sets the user name of the Sentinel server.

Setting format: `string`.

Default value: Not specified.

12.11.10.9 `sentinel > password`

The parameter sets the password of the Sentinel server user.

Setting format: `string`.

Default value: Not specified.

12.11.11 **ADDITIONAL_SERVICES_USAGE** section

12.11.11.1 `luna_events`

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: `integer` (“0” or “1”).

Default value: 1.

12.11.11.2 `luna_handlers`

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: `integer` (“0” or “1”).

Default value: 1.

12.11.11.3 `luna_sender`

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

[12.11.11.4 luna_matcher_proxy](#)

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

[12.11.11.5 luna_image_store](#)

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

[12.11.11.6 luna_lambda](#)

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.11.12 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.11.12.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.11.12.2 metrics_format

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.11.12.3 extra_labels

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.11.13 Other

12.11.13.1 storage_time

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.11.13.2 `default_face_descriptor_version`

The parameter sets the version of the face descriptor to use.

Setting format: `string`.

Default value: 59.

Note: For more information about descriptor versions, see [“Neural networks”](#).

12.11.13.3 `default_body_descriptor_version`

The parameter sets the version of the body descriptor to use.

For more information about descriptor version, see [“Neural networks”](#).

Setting format: `string`.

Default value: 110.

Note: For more information about descriptor versions, see [“Neural networks”](#).

12.12 Handlers service configuration

The section describes the Handlers service parameters.

You can configure the service using the Configurator service.

12.12.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Handlers service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_handlers/configs/” of the corresponding container.

12.12.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_handlers/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.12.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.12.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.12.2 LUNA_HANDLERS_DB section

In this section, the settings for connecting to the database of the service Handlers are set.

12.12.2.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.12.2.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.12.2.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.12.2.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.12.2.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.12.2.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: `luna_handlers`.

12.12.2.7 `connection_pool_size`

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “`max_connections`” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: `string`.

Default value: `10`.

12.12.2.8 `dsn`

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “`dsn`” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “`dsn`” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_handlers?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “`luna:luna`” — Username and password for connecting to PostgreSQL.
- “`@postgres01:5432,postgres02:5432`” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“`postgres01`”) on port 5432. If this fails, it will try to connect to the second host (“`postgres02`”) also on port 5432.
- “`/luna_handlers`” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_handlers”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.12.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.12.3.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

12.12.3.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

12.12.3.3 organization

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.12.3.4 token

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.12.3.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.12.3.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.12.3.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.12.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.12.4 LUNA_HANDLERS_LOGGER section

This section sets the logging settings for the logging.

12.12.4.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.12.4.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.12.4.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.12.4.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: boolean.

Default value: false.

12.12.4.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.12.4.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.12.4.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.12.4.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.12.5 LUNA_REMOTE_SDK_ADDRESS section

This section sets the connection settings for the Remote SDK service.

12.12.5.1 origin

The parameter sets the protocol, IP address and port of the Remote SDK service.

The IP address “127.0.0.1” means that the Remote SDK service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Remote SDK service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5220`.

12.12.5.2 `api_version`

The parameter sets the version of the Remote SDK service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.12.6 `LUNA_REMOTE_SDK_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Remote SDK service.

12.12.6.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Remote SDK service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.12.6.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.12.6.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: `integer (seconds)`.

Default value: 10.

12.12.6.4 `sock_read`

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: `integer (seconds)`.

Default value: 60.

12.12.7 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

12.12.7.1 origin

The parameter sets the protocol, IP address and port of the Faces service.

The IP address “127.0.0.1” means that the Faces service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Faces service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5030`.

12.12.7.2 api_version

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: `integer`.

Default value: 3.

12.12.8 LUNA_FACES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

12.12.8.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Faces service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.12.8.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.12.8.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.12.8.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.12.9 LUNA_LAMBDA_UNIT_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Lambda Unit service.

12.12.9.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Lambda Unit service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.12.9.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.12.9.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.12.9.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.12.10 LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section

In this section, the bucket for storing [face samples](#) settings are set.

12.12.10.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.12.10.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.12.10.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: visionlabs-samples.

12.12.11 LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [face samples](#).

12.12.11.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [face samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.12.11.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.12.12 LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section

In this section, the bucket for storing [body samples](#) settings are set.

12.12.12.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.12.12.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.12.12.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-bodies-samples`.

12.12.13 LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with `body samples`.

12.12.13.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with `body samples`. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: `30`.

12.12.13.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: `60`.

12.12.14 LUNA_IMAGE_STORE_IMAGES_ADDRESS section

In this section, the bucket for storing `source images` settings are set.

12.12.14.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.12.14.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.12.14.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-image-origin`.

12.12.15 `LUNA_IMAGE_STORE_IMAGES_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [source images](#) .

12.12.15.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [source images](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 30.

12.12.15.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.12.16 `ADDITIONAL_SERVICES_USAGE` section

12.12.16.1 `luna_events`

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.12.16.2 luna_handlers

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.12.16.3 luna_sender

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.12.16.4 luna_matcher_proxy

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.12.16.5 luna_image_store

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.12.16.6 luna_lambda

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.12.17 LUNA_EVENTS_ADDRESS section

This section sets the connection settings for the Events service.

12.12.17.1 origin

The parameter sets the protocol, IP address and port of the Events service.

The IP address “127.0.0.1” means that the Events service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Events service running.

Setting format: string.

Default value: http://127.0.0.1:5040.

12.12.17.2 api_version

The parameter sets the version of the Events service. The available API version is “2”.

Setting format: integer.

Default value: 2.

12.12.18 LUNA_EVENTS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Events service.

12.12.18.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Events service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.12.18.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.12.18.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.12.18.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.12.19 LUNA_PYTHON_MATCHER_ADDRESS section

This section sets the connection settings for the Python Matcher service.

12.12.19.1 `origin`

The parameter sets the protocol, IP address and port of the Python Matcher service.

The IP address “127.0.0.1” means that the Python Matcher service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5100`.

12.12.19.2 `api_version`

The parameter sets the version of the Python Matcher service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.12.20 `LUNA_PYTHON_MATCHER_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Python Matcher service.

12.12.20.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Python Matcher service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.12.20.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.12.20.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.12.20.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.12.21 LUNA_MATCHER_PROXY_ADDRESS section

This section sets the connection settings for the Python Matcher Proxy service.

12.12.21.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher Proxy service.

The IP address “127.0.0.1” means that the Python Matcher Proxy service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher Proxy service running.

Setting format: string.

Default value: http://127.0.0.1:5110.

12.12.21.2 api_version

The parameter sets the version of the Python Matcher Proxy service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.12.22 LUNA_PYTHON_MATCHER_PROXY_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Python Matcher Proxy service.

12.12.22.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Python Matcher Proxy service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

[12.12.22.2 request](#)

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

[12.12.22.3 sock_connect](#)

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

[12.12.22.4 sock_read](#)

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.12.23 REDIS_DB_ADDRESS section

[12.12.23.1 user](#)

The parameter sets the user name of the Redis database.

Setting format: string.

Default value: not specified.

[12.12.23.2 password](#)

The parameter sets the password of the Redis database.

Setting format: string.

Default value: not specified.

12.12.23.3 host

The parameter sets Redis database IP address.

Setting format: `string`.

Default value: `127.0.0.1`.

12.12.23.4 port

The parameter sets the port number on which Redis waits for incoming network connections and listens for them to execute commands from clients.

Setting format: `integer`.

Default value: `6379`.

12.12.23.5 number

The parameter sets the number of the Redis database. Each number corresponds to a separate database, which enables you to separate the data.

Setting format: `integer`.

Default value: `0`.

12.12.23.6 channel

The parameter sets the Redis channel to which the service subscribes.

Setting format: `integer`.

Default value: `luna-sender`.

12.12.23.7 sentinel > master_name

The parameter sets the name of the Redis database master, which is monitored and managed by the Sentinel system.

Setting format: `string`.

Default value: `luna_sender_master`.

12.12.23.8 sentinel > sentinels

The parameter sets the list of addresses and ports of Sentinel servers that will be used by clients to detect and monitor the Redis database.

Setting format: `list > string`.

Default value: `[]`.

12.12.23.9 `sentinel > user`

The parameter sets the user name of the Sentinel server.

Setting format: `string`.

Default value: Not specified.

12.12.23.10 `sentinel > password`

The parameter sets the password of the Sentinel server user.

Setting format: `string`.

Default value: Not specified.

12.12.24 **FETCH_EXTERNAL_IMAGE_TIMEOUTS** section

This section represents a set of parameters for setting timeouts during the execution of HTTP requests to external resources for uploading images. Each of the parameters in this setting determines the maximum waiting time for a specific operation during the execution of the request.

12.12.24.1 `connect`

The parameter sets the timeout for establishing a connection to an external resource. If the connection is not established at the set time, the operation will be interrupted.

Setting format: `integer (seconds)`.

Default value: 10.

12.12.24.2 `request`

The parameter sets the timeout for waiting for a response to an HTTP request that is sent to get an image from an external resource.

Setting format: `integer (seconds)`.

Default value: 10.

12.12.24.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: `integer (seconds)`.

Default value: 10.

12.12.24.4 `sock_request`

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: `integer` (seconds).

Default value: 10.

12.12.25 `ATTRIBUTES_STORAGE_POLICY` section

This section sets the settings for the Handlers service related to storing [temporary attributes](#).

12.12.25.1 `default_ttl`

The parameter sets the default lifetime of temporary attributes.

Setting format: `integer` (seconds).

Default value: 300.

12.12.25.2 `max_ttl`

The parameter sets the maximum time to live of temporary attributes.

Setting format: `integer` (seconds).

Default value: 86400.

12.12.26 `LUNA_HANDLERS_LIMITS` section

This section sets restrictions for some operations performed using the Handlers service.

Increasing these limits may lead to a decrease in performance.

12.12.26.1 `received_images_limit`

The parameter sets the maximum number of images that can be specified in the requests [“generate events”](#), [“detect faces”](#) and [“perform verification”](#) using the [“multipart/form-data”](#) scheme.

For batch image processing, you need to use [Estimator task](#).

Setting format: `integer`.

Default value: 8.

12.12.26.2 [raw_event_detections_limit](#)

The parameter sets the maximum number of detections that can be specified in the “[save event](#)” request.

Setting format: integer.

Default value: 100.

12.12.26.3 [raw_event_arrays_limit](#)

The parameter sets the maximum length of arrays in the “[save event](#)” request body.

Setting format: integer.

Default value: 30.

12.12.26.4 [result_candidate_limit](#)

The parameter sets the maximum number of descriptor matching results in the “[generate events](#)” request.

Setting format: integer.

Default value: 100.

12.12.27 [EXTERNAL_LUNA_API_ADDRESS](#) section

This section is intended for correct processing of references to objects created using the “[/images](#)” and “[/objects](#)” resources in the API service. This section specifies the address and API version of the API service.

If as input for resources “[/detector](#)”, “[handlers/{handler_id}/events](#)” and “[verifiers/{verifier_id}/verification](#)” specifies the URL and version of the API service of the “images” type object that matches the address and version of the API from the “EXTERNAL_LUNA_API_ADDRESS” section of the Handlers service settings, then these objects will be loaded using the Image Store service directly, and not send a request to the API service from subsequent redirection to the Image Store service.

Format example: “[http://10.15.3.144:5000/6/images/141d2706-8baf-433b-82eb-8c7fada847da](#)”, where “[http://10.15.3.144:5000](#)” must match the value from the “origin” setting “, and the value “6” must match the value of the “api_version” setting in the “EXTERNAL_LUNA_API_ADDRESS” section.

To avoid errors, you must configure this section in the Tasks settings before using URLs to objects of type “objects” or “images” as an input data source.

12.12.27.1 [origin](#)

The parameter sets the protocol, IP address and port of the API service.

See the description of the operating logic in the “[EXTERNAL_LUNA_API_ADDRESS](#) section” section.

Setting format: string.

Default value: http://127.0.0.1:5000.

12.12.27.2 [api_version](#)

The parameter sets the API version of the API service.

See the description of the operating logic in the “[EXTERNAL_LUNA_API_ADDRESS section](#)” section.

Setting format: integer.

Default value: 6.

12.12.28 [LUNA_HANDLERS_HTTP_SETTINGS section](#)

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.12.28.1 [request_timeout](#)

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.12.28.2 [response_timeout](#)

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.12.28.3 [request_max_size](#)

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.12.28.4 [keep_alive_timeout](#)

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.12.29 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.12.29.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.12.29.2 metrics_format

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.12.29.3 extra_labels

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.12.30 Other

12.12.30.1 luna_handlers_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[
  "plugin_1",
  "plugin_2",
  "plugin_3"
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.12.30.2 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.12.30.3 `default_face_descriptor_version`

The parameter sets the version of the face descriptor to use.

Setting format: string.

Default value: 59.

Note: For more information about descriptor versions, see “[Neural networks](#)”.

12.12.30.4 `default_body_descriptor_version`

The parameter sets the version of the body descriptor to use.

For more information about descriptor version, see “[Neural networks](#)”.

Setting format: string.

Default value: 110.

Note: For more information about descriptor versions, see “[Neural networks](#)”.

12.13 Backport 3 service configuration

The section describes the Backport 3 service parameters.

You can configure the service using the Configurator service.

12.13.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Backport 3 service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_backport3/configs/” of the corresponding container.

12.13.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_backport3/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.13.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.13.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.13.2 LUNA_BACKPORT3_DB section

In this section, the settings for connecting to the database of the service Backport3 are set.

12.13.2.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.13.2.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.13.2.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.13.2.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.13.2.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.13.2.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_backport3.

12.13.2.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.13.2.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_backport3?some_option
        =some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_backport3” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_backport3”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.13.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.13.3.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

12.13.3.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

12.13.3.3 organization

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.13.3.4 token

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.13.3.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.13.3.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.13.3.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.13.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.13.4 LUNA_BACKPORT3_LOGGER section

This section sets the logging settings for the logging.

12.13.4.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.13.4.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.13.4.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.13.4.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: boolean.

Default value: false.

12.13.4.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.13.4.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.13.4.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.13.4.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.13.5 LUNA_API_ADDRESS section

This section sets the connection settings for the API service.

12.13.5.1 origin

The parameter sets the protocol, IP address and port of the API service.

The IP address “127.0.0.1” means that the API service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the API service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5000`.

12.13.5.2 `api_version`

The parameter sets the version of the API service. The available API version is “6”.

Setting format: `integer`.

Default value: 6.

12.13.6 `LUNA_API_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the API service.

12.13.6.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the API service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.13.6.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.13.6.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: `integer (seconds)`.

Default value: 10.

12.13.6.4 `sock_read`

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: `integer (seconds)`.

Default value: 60.

12.13.7 LUNA_IMAGE_STORE_PORTRAITS_ADDRESS section

In this section, the bucket for storing [portraits](#) settings are set.

12.13.7.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.13.7.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.13.7.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `portraits`.

12.13.8 LUNA_IMAGE_STORE_PORTRAITS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [portraits](#).

12.13.8.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [portraits](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: `30`.

12.13.8.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.13.9 BACKPORT3_EVENTS_DB_ADDRESS section

12.13.9.1 user

The parameter sets the user name of the Redis database.

Setting format: string.

Default value: not specified.

12.13.9.2 password

The parameter sets the password of the Redis database.

Setting format: string.

Default value: not specified.

12.13.9.3 host

The parameter sets Redis database IP address.

Setting format: string.

Default value: 127.0.0.1.

12.13.9.4 port

The parameter sets the port number on which Redis waits for incoming network connections and listens for them to execute commands from clients.

Setting format: integer.

Default value: 6379.

12.13.9.5 number

The parameter sets the number of the Redis database. Each number corresponds to a separate database, which enables you to separate the data.

Setting format: integer.

Default value: 0.

12.13.9.6 channel

The parameter sets the Redis channel to which the service subscribes.

Setting format: integer.

Default value: luna-backport3.

12.13.9.7 sentinel > master_name

The parameter sets the name of the Redis database master, which is monitored and managed by the Sentinel system.

Setting format: string.

Default value: luna_backport3_master.

12.13.9.8 sentinel > sentinels

The parameter sets the list of addresses and ports of Sentinel servers that will be used by clients to detect and monitor the Redis database.

Setting format: list > string.

Default value: [].

12.13.9.9 sentinel > user

The parameter sets the user name of the Sentinel server.

Setting format: string.

Default value: Not specified.

12.13.9.10 sentinel > password

The parameter sets the password of the Sentinel server user.

Setting format: string.

Default value: Not specified.

12.13.10 LUNA_BACKPORT3_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.13.10.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.13.10.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.13.10.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.13.10.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.13.11 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.13.11.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: boolean.

Default value: false.

12.13.11.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: string.

Default value: prometheus.

12.13.11.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.13.12 Other

12.13.12.1 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.13.12.2 `luna_backport3_active_plugins`

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.13.12.3 [use_samples_as_portraits](#)

The parameter enables you to use samples instead of portraits in order not to store both types of entities.

Setting format: boolean.

Default value: true.

12.13.12.4 [backport3_enable_portraits](#)

The parameter enables you to disable the possibility of using portraits, but leave the possibility of using the rest of the functionality of the Image Store service. If the use of the Image Store service is disabled in the “ADDITIONAL_SERVICES_USAGE” setting, then this setting should also be disabled.

Setting format: boolean.

Default value: true.

12.13.12.5 [backport3_enable_ws_events](#)

The parameter enables web sockets support for Backport3.

Setting format: boolean.

Default value: true.

12.13.12.6 [max_candidate_in_response](#)

The parameter sets the maximum number of candidates in responses to matching requests.

Setting format: integer.

Default value: 5.

12.14 Backport 4 service configuration

The section describes the Backport 4 service parameters.

You can configure the service using the Configurator service.

12.14.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Backport 4 service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_backport4/configs/” of the corresponding container.

12.14.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_backport4/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.14.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.14.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.14.2 LUNA_BACKPORT4_DB section

In this section, the settings for connecting to the database of the service Backport4 are set.

12.14.2.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.14.2.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.14.2.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.14.2.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.14.2.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.14.2.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: luna_backport3.

12.14.2.7 connection_pool_size

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “max_connections” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: string.

Default value: 10.

12.14.2.8 dsn

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “dsn” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “dsn” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_backport3?some_option
    =some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “luna:luna” — Username and password for connecting to PostgreSQL.
- “@postgres01:5432,postgres02:5432” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“postgres01”) on port 5432. If this fails, it will try to connect to the second host (“postgres02”) also on port 5432.
- “/luna_backport3” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_backport3”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.14.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.14.3.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

12.14.3.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

12.14.3.3 organization

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.14.3.4 token

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.14.3.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.14.3.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.14.3.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.14.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.14.4 LUNA_BACKPORT4_LOGGER section

This section sets the logging settings for the logging.

12.14.4.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.14.4.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.14.4.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.14.4.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: boolean.

Default value: false.

12.14.4.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.14.4.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.14.4.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.14.4.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.14.5 LUNA_API_ADDRESS section

This section sets the connection settings for the API service.

12.14.5.1 origin

The parameter sets the protocol, IP address and port of the API service.

The IP address “127.0.0.1” means that the API service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the API service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5000`.

12.14.5.2 `api_version`

The parameter sets the version of the API service. The available API version is “6”.

Setting format: `integer`.

Default value: 6.

12.14.6 `LUNA_API_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the API service.

12.14.6.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the API service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.14.6.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.14.6.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: `integer (seconds)`.

Default value: 10.

12.14.6.4 `sock_read`

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: `integer (seconds)`.

Default value: 60.

12.14.7 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

12.14.7.1 origin

The parameter sets the protocol, IP address and port of the Faces service.

The IP address “127.0.0.1” means that the Faces service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Faces service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5030`.

12.14.7.2 api_version

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: `integer`.

Default value: 3.

12.14.8 LUNA_FACES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

12.14.8.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Faces service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.14.8.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.14.8.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.14.8.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.14.9 ATTRIBUTES_STORAGE_POLICY section

This section sets the settings for the Backport4 service related to storing [temporary attributes](#).

12.14.9.1 default_ttl

The parameter sets the default lifetime of temporary attributes.

Setting format: integer (seconds).

Default value: 300.

12.14.9.2 max_ttl

The parameter sets the maximum time to live of temporary attributes.

Setting format: integer (seconds).

Default value: 86400.

12.14.10 LUNA_BACKPORT4_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.14.10.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.14.10.2 `response_timeout`

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: `integer` (seconds).

Default value: `600`.

12.14.10.3 `request_max_size`

The parameter sets the maximum size of the request.

Setting format: `integer` (bytes).

Default value: `1073741824`.

12.14.10.4 `keep_alive_timeout`

The parameter sets the timeout for maintaining HTTP activity.

Setting format: `integer` (seconds).

Default value: `15`.

12.14.11 `LUNA_SERVICE_METRICS` section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.14.11.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.14.11.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.14.11.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.14.12 Other

12.14.12.1 `luna_backport4_active_plugins`

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.15 Accounts service configuration

The section describes the Accounts service parameters.

You can configure the service using the Configurator service.

12.15.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Accounts service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_accounts/configs/” of the corresponding container.

12.15.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_accounts/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.15.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.15.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.15.2 LUNA_ACCOUNTS_DB section

In this section, the settings for connecting to the database of the service Accounts are set.

12.15.2.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.15.2.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.15.2.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.15.2.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.15.2.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.15.2.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: `luna_accounts`.

12.15.2.7 `connection_pool_size`

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “`max_connections`” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: `string`.

Default value: `10`.

12.15.2.8 `dsn`

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “`dsn`” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “`dsn`” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_accounts?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “`luna:luna`” — Username and password for connecting to PostgreSQL.
- “`@postgres01:5432,postgres02:5432`” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“`postgres01`”) on port 5432. If this fails, it will try to connect to the second host (“`postgres02`”) also on port 5432.
- “`/luna_accounts`” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_accounts”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.15.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.15.3.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

12.15.3.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

12.15.3.3 organization

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.15.3.4 token

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.15.3.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.15.3.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.15.3.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.15.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.15.4 LUNA_ACCOUNTS_LOGGER section

This section sets the logging settings for the logging.

12.15.4.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.15.4.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.15.4.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.15.4.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: boolean.

Default value: false.

12.15.4.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.15.4.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.15.4.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.15.4.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.15.5 LUNA_ACCOUNTS_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.15.5.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: `integer (seconds)`.

Default value: `60`.

12.15.5.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.15.5.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.15.5.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

12.15.6 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.15.6.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: boolean.

Default value: false.

12.15.6.2 metrics_format

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: string.

Default value: prometheus.

12.15.6.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.15.7 Other

12.15.7.1 `luna_accounts_active_plugins`

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.15.7.2 `storage_time`

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.16 Remote SDK service configuration

The section describes the Remote SDK service parameters.

You can configure the service using the Configurator service.

12.16.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Remote SDK service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_remote_sdk/configs/” of the corresponding container.

12.16.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_remote_sdk/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.16.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.16.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.16.2 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see [“Monitoring”](#) section.

12.16.2.1 `send_data_for_monitoring`

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: `1`.

12.16.2.2 `use_ssl`

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: `0`.

12.16.2.3 `organization`

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.16.2.4 `token`

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.16.2.5 `bucket`

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.16.2.6 `host`

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.16.2.7 `port`

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.16.2.8 `flushing_period`

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer` (seconds).

Default value: `1`.

12.16.3 `LUNA_REMOTE_SDK_LOGGER` section

This section sets the logging settings for the logging.

12.16.3.1 `log_level`

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.16.3.2 `log_time`

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.16.3.3 `log_to_stdout`

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.16.3.4 `log_to_file`

The parameter enables you to save logs to a file. The directory with log files is specified in the “`folder_with_logs`” parameter.

Setting format: `boolean`.

Default value: `false`.

12.16.3.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.16.3.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.16.3.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: boolean.

Default value: true.

12.16.3.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.16.4 LUNA_LICENSES_ADDRESS section

This section sets the connection settings for the Licenses service.

12.16.4.1 origin

The parameter sets the protocol, IP address and port of the Licenses service.

The IP address “127.0.0.1” means that the Licenses service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Licenses service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5120`.

12.16.4.2 api_version

The parameter sets the version of the Licenses service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.16.5 LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS section

In this section, the bucket for storing [face samples](#) settings are set.

12.16.5.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.16.5.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.16.5.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-samples`.

12.16.6 `LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [face samples](#).

12.16.6.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [face samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 30.

12.16.6.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.16.7 LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section

In this section, the bucket for storing [body samples](#) settings are set.

12.16.7.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.16.7.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.16.7.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-bodies-samples`.

12.16.8 LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [body samples](#).

12.16.8.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [body samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 30.

12.16.8.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.16.9 LUNA_IMAGE_STORE_IMAGES_ADDRESS section

In this section, the bucket for storing [source images](#) settings are set.

12.16.9.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.16.9.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.16.9.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: visionlabs-image-origin.

12.16.10 LUNA_IMAGE_STORE_IMAGES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [source images](#).

12.16.10.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [source images](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.16.10.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.16.11 ADDITIONAL_SERVICES_USAGE section

12.16.11.1 luna_events

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.16.11.2 luna_handlers

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.16.11.3 luna_sender

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.16.11.4 luna_matcher_proxy

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.16.11.5 luna_image_store

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.16.11.6 luna_lambda

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.16.12 LUNA_REMOTE_SDK_RUNTIME_SETTINGS section

This section sets global settings for all estimators/detectors.

12.16.12.1 global_device_class

The parameter sets the device type (“cpu” or “gpu”) for all estimators/detectors with the parameter value “device_class” = “global”.

Setting format: string.

Default value: cpu.

12.16.12.2 num_threads

The parameter sets the number of CPU processor threads to be used by the service when performing estimation/detection. It is desirable that the value of this parameter corresponds to the number of physical CPU cores to maximise performance. However, too many threads may cause performance degradation due to additional overhead.

Setting format: integer.

Default value: 4.

12.16.12.3 num_compute_streams

The parameter sets the number of computation threads on the GPU processor that will be used by the service when performing estimation/detection. It is recommended to choose the value corresponding to the characteristics of a particular GPU and the task. It should be taken into account that NVIDIA may change the behaviour of this parameter in different software versions, so it is recommended to perform testing to determine the optimal value.

Setting format: integer.

Default value: 6.

12.16.13 LUNA_REMOTE_SDK_FACE_DETECTOR_SETTINGS section

This section sets individual operation settings for [face detector](#).

[12.16.13.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.13.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.13.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.13.4 estimator_settings > min_face_size](#)

The parameter sets the minimum face size in pixels.

The maximum face size is equal to the minimum face size multiplied by 32.

Setting format: integer.

Default value: 50.

[12.16.13.5 estimator_settings > redetect_face_target_size](#)

The parameter sets the target face size for redetection.

Setting format: integer.

Default value: 80.

[12.16.13.6 estimator_settings > redetect_tensor_size](#)

The parameter sets the target face size for redetection after preprocessing.

Detection is considered unsuccessful if the score is below the specified value.

Setting format: integer.

Default value: 64.

[12.16.13.7 estimator_settings > redetect_score_threshold](#)

The parameter sets the redetection estimating threshold.

Redetection is considered unsuccessful if the score is below the specified value.

Setting format: number.

Default value: 0.3.

[12.16.13.8 estimator_settings > score_threshold](#)

The parameter sets the estimation threshold.

Detection is considered unsuccessful if the score is below the specified value.

Setting format: number.

Default value: 0.5.

[12.16.14 LUNA_REMOTE_SDK_GAZE_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [gaze estimator](#).

[12.16.14.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.14.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.14.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.15 LUNA_REMOTE_SDK_QUALITY_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [image quality estimator](#).

[12.16.15.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.15.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.15.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.16 LUNA_REMOTE_SDK_MOUTH_ATTRIBUTES_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [mouth attributes estimator](#).

[12.16.16.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.16.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.16.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.17 LUNA_REMOTE_SDK_EMOTIONS_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [emotions estimator](#).

12.16.17.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

12.16.17.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: `integer`.

Default value: `10`.

12.16.17.3 runtime_settings > num_compute_streams

The parameter sets the number of workers to perform the estimation.

Setting format: `integer`.

Default value: `1`.

12.16.18 LUNA_REMOTE_SDK_BASIC_ATTRIBUTES_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [basic attributes estimator](#).

12.16.18.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

12.16.18.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: `integer`.

Default value: `10`.

[12.16.18.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.19 LUNA_REMOTE_SDK_EYES_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [eyes attributes estimator](#).

[12.16.19.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.19.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.19.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.20 LUNA_REMOTE_SDK_HEAD_POSE_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [head pose estimator](#).

[12.16.20.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.20.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.20.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.21 LUNA_REMOTE_SDK_FACE_DESCRIPTOR_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [face descriptor estimator](#).

[12.16.21.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.21.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.21.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.22 LUNA_REMOTE_SDK_MASK_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [mask estimator](#).

[12.16.22.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.22.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.22.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.23 LUNA_REMOTE_SDK_LIVENESS_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [Liveness estimator](#).

[12.16.23.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.23.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.23.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.23.4 estimator_settings > real_threshold

The parameter sets the “[Liveness threshold](#)”, below which the system will consider the result an attack on biometric presentation and the result will be “[spooft](#)”.

Setting format: float.

Default value: 0.5.

12.16.23.5 estimator_settings > quality_threshold

The parameter sets the “[Quality threshold](#)”, below which no check will be performed and the result will be “[unknown](#)”.

Setting format: float.

Default value: 0.5.

12.16.24 LUNA_REMOTE_SDK_GLASSES_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [glasses estimator](#).

12.16.24.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

12.16.24.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

12.16.24.3 runtime_settings > num_compute_streams

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.25 LUNA_REMOTE_SDK_FACE_WARP_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [face sample estimator](#).

[12.16.25.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.25.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.25.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.26 LUNA_REMOTE_SDK_FACE_LANDMARKS68_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [68 landmarks estimator](#).

[12.16.26.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.26.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.26.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.27 LUNA_REMOTE_SDK_FACE_LANDMARKS5_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [5 landmarks estimator](#).

12.16.27.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

12.16.27.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

12.16.27.3 runtime_settings > num_compute_streams

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.28 LUNA_REMOTE_SDK_IMAGE_COLOR_TYPE_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [face color type estimator](#).

12.16.28.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

12.16.28.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

12.16.28.3 [runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: `integer`.

Default value: 1.

12.16.29 [LUNA_REMOTE_SDK_HEADWEAR_ESTIMATOR_SETTINGS](#) section

This section sets individual operation settings for [headwear estimator](#).

12.16.29.1 [runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

12.16.29.2 [runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: `integer`.

Default value: 10.

12.16.29.3 [runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: `integer`.

Default value: 1.

12.16.30 [LUNA_REMOTE_SDK_FACE_NATURAL_LIGHT_ESTIMATOR_SETTINGS](#) section

This section sets individual operation settings for [natural light estimator](#).

12.16.30.1 [runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

[12.16.30.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.30.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.31 LUNA_REMOTE_SDK_FISHEYE_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [FishEye effect estimator](#).

[12.16.31.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.31.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.31.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.32 LUNA_REMOTE_SDK_EYEBROW_EXPRESSION_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [eyebrow estimator](#).

[12.16.32.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.32.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.32.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.33 LUNA_REMOTE_SDK_RED_EYES_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [red eyes effect estimator](#).

[12.16.33.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.33.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.33.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.34 LUNA_REMOTE_SDK_FACE_DETECTION_BACKGROUND_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [image background estimator](#).

12.16.34.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

12.16.34.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

12.16.34.3 runtime_settings > num_compute_streams

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.35 LUNA_REMOTE_SDK_IMAGE_ORIENTATION_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [image orientation estimator](#).

12.16.35.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

12.16.35.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

12.16.35.3 [runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: `integer`.

Default value: 1.

12.16.36 [LUNA_REMOTE_SDK PORTRAIT_STYLE_ESTIMATOR_SETTINGS](#) section

This section sets individual operation settings for [shoulders position estimator](#).

12.16.36.1 [runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

12.16.36.2 [runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: `integer`.

Default value: 10.

12.16.36.3 [runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: `integer`.

Default value: 1.

12.16.37 [LUNA_REMOTE_SDK BODY_DETECTOR_SETTINGS](#) section

This section sets individual operation settings for [human body detector](#).

12.16.37.1 [runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

[12.16.37.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.37.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.37.4 estimator_settings > image_size](#)

The parameter sets the maximum frame size in pixels after scaling along the largest side of the frame.

Setting format: integer.

Default value: 640.

[12.16.37.5 estimator_settings > redetect_score_threshold](#)

The parameter sets the redetection estimating threshold.

Redetection is considered unsuccessful if the score is below the specified value.

Setting format: number.

Default value: 0.12.

[12.16.37.6 estimator_settings > score_threshold](#)

The parameter sets the estimating threshold.

Detection is considered unsuccessful if the score is below the specified value.

Setting format: number.

Default value: 0.5.

[12.16.38 LUNA_REMOTE_SDK_BODY_DESCRIPTOR_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [human body descriptor estimator](#).

[12.16.38.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.38.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.38.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

[12.16.39 LUNA_REMOTE_SDK_BODY_WARP_ESTIMATOR_SETTINGS section](#)

This section sets individual operation settings for [human body sample estimator](#).

[12.16.39.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.39.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.39.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.40 LUNA_REMOTE_SDK_BODY_LANDMARKS_SETTINGS section

This section sets individual operation settings for body landmarks estimator.

12.16.40.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

12.16.40.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

12.16.40.3 runtime_settings > num_compute_streams

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.41 LUNA_REMOTE_SDK_BODY_ATTRIBUTES_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [body parameters estimator](#).

12.16.41.1 runtime_settings > device_class

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

12.16.41.2 runtime_settings > num_threads

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.41.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: `integer`.

Default value: `1`.

12.16.42 LUNA_REMOTE_SDK_HUMAN_DETECTOR_SETTINGS section

This section sets individual operation settings for face and body detector.

[12.16.42.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

[12.16.42.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: `integer`.

Default value: `10`.

[12.16.42.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: `integer`.

Default value: `1`.

12.16.43 LUNA_REMOTE_SDK_PEOPLE_COUNT_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [people count estimator](#).

[12.16.43.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: `string`.

Default value: `cpu`.

[12.16.43.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.43.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.44 LUNA_REMOTE_SDK_DEEPFAKE_ESTIMATOR_SETTINGS section

This section sets individual operation settings for [Deepfake estimator](#).

[12.16.44.1 runtime_settings > device_class](#)

The parameter sets the type of device to perform the estimation (“cpu”, “gpu” or “global”)

Setting format: string.

Default value: cpu.

[12.16.44.2 runtime_settings > num_threads](#)

The parameter sets the size of the batch to perform the estimation.

Setting format: integer.

Default value: 10.

[12.16.44.3 runtime_settings > num_compute_streams](#)

The parameter sets the number of workers to perform the estimation.

Setting format: integer.

Default value: 1.

12.16.45 FETCH_EXTERNAL_IMAGE_TIMEOUTS section

This section represents a set of parameters for setting timeouts during the execution of HTTP requests to external resources for uploading images. Each of the parameters in this setting determines the maximum waiting time for a specific operation during the execution of the request.

12.16.45.1 connect

The parameter sets the timeout for establishing a connection to an external resource. If the connection is not established at the set time, the operation will be interrupted.

Setting format: integer (seconds).

Default value: 10.

12.16.45.2 request

The parameter sets the timeout for waiting for a response to an HTTP request that is sent to get an image from an external resource.

Setting format: integer (seconds).

Default value: 10.

12.16.45.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.16.45.4 sock_request

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 10.

12.16.46 EXTERNAL_LUNA_API_ADDRESS section

This section is intended for correct processing of references to objects created using the “/images” and “/objects” resources in the API service. This section specifies the address and API version of the API service.

If as input for resources “/iso” and “/sdk” specifies the URL and version of the API service of the “images” type object that matches the address and version of the API from the “EXTERNAL_LUNA_API_ADDRESS” section of the Remote SDK service settings, then these objects will be loaded using the Image Store service directly, and not send a request to the API service from subsequent redirection to the Image Store service.

Format example: “http://10.15.3.144:5000/6/images/141d2706-8baf-433b-82eb-8c7fada847da”, where “http://10.15.3.144:5000” must match the value from the “origin” setting “, and the value “6” must match the value of the “api_version” setting in the “EXTERNAL_LUNA_API_ADDRESS” section.

To avoid errors, you must configure this section in the Tasks settings before using URLs to objects of type “objects” or “images” as an input data source.

12.16.46.1 origin

The parameter sets the protocol, IP address and port of the API service.

See the description of the operating logic in the “EXTERNAL_LUNA_API_ADDRESS section” section.

Setting format: string.

Default value: http://127.0.0.1:5000.

12.16.46.2 api_version

The parameter sets the API version of the API service.

See the description of the operating logic in the “EXTERNAL_LUNA_API_ADDRESS section” section.

Setting format: integer.

Default value: 6.

12.16.47 LUNA_REMOTE_SDK_LIMITS section

This section sets restrictions for some operations performed using the Remote SDK service.

Increasing these limits may lead to a decrease in performance.

12.16.47.1 received_images_limit

The parameter sets the maximum number of images that can be specified in the “sdk” and “iso” requests using the “multipart/form-data” scheme.

For batch image processing, you need to use [Estimator task](#).

Setting format: integer.

Default value: 8.

12.16.48 LUNA_REMOTE_SDK_VIDEO_SETTINGS section

This section sets the video processing settings for performing video analytics.

12.16.48.1 `decoder_device_class`

The parameter sets the type of video decoder device. The following options are available:

- “cpu”
- “gpu” (in priority)
- “auto”

Setting format: `string`.

Default value: `auto`.

12.16.48.2 `decoder_worker_count`

The parameter sets the number of workers of the video decoder.

Setting format: `integer (seconds)`.

Default value: `10`.

12.16.48.3 `storage`

The parameter sets a temporary directory in the Remote SDK container where the video will be saved to perform video analysis.

After performing the video analysis, the video will be deleted.

Setting format: `string`.

Default value: `./videos`.

12.16.48.4 `max_size`

The parameter sets the maximum video size for performing video analytics.

Setting format: `integer`.

Default value: `1024 (megabytes)`.

LUNA_SERVICE_METRICS section {#luna_service_metrics_remote_sdk}

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.16.48.5 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.16.48.6 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: string.

Default value: prometheus.

12.16.48.7 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.16.49 `FACE_QUALITY_SETTINGS` section

This section specifies the thresholds at which LUNA PLATFORM will consider that the check has passed.

For Natural Light, Fish Eye, Color Type and Red Eyes estimators, you can also set a threshold that interprets the estimator's response.

See "[Image check](#)" for details.

Important: These thresholds are default. Setting thresholds in the "face_quality" parameter group overrides the value of these thresholds.

12.16.49.1 `image_format`

The parameter sets standard thresholds for checking [image format](#).

Setting format: array.

Default value:

```
"threshold": [  
  "JPEG"  
  "JPEG2000"  
  "PNG"  
]
```

12.16.49.2 illumination

The parameter sets standard thresholds for checking [degree of lighting uniformity](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.3,  
  "max": 1  
}
```

12.16.49.3 specularity

The parameter sets standard thresholds for checking [degree of specularity absence](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.3,  
  "max": 1  
}
```

12.16.49.4 blurriness

The parameter sets standard thresholds for checking [degree of blurring](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.61,  
  "max": 1  
}
```

12.16.49.5 dark

The parameter sets standard thresholds for checking [degree to which the photo is not darkened](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 0.5,
  "max": 1
}
```

12.16.49.6 light

The parameter sets standard thresholds for checking [degree to which the photo is not overexposed](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 0.57,
  "max": 1
}
```

12.16.49.7 head_yaw

The parameter sets standard thresholds for checking [head pose yaw angle](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": -5,
  "max": 5
}
```

12.16.49.8 head_pitch

The parameter sets standard thresholds for checking [head pose pitch angle](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": -5,
  "max": 5
}
```

12.16.49.9 head_roll

The parameter sets standard thresholds for checking [head pose roll angle](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": -8,  
  "max": 8  
}
```

12.16.49.10 gaze_yaw

The parameter sets standard thresholds for checking [gaze yaw angle](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": -5,  
  "max": 5  
}
```

12.16.49.11 gaze_pitch

The parameter sets standard thresholds for checking [gaze pitch angle](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": -5,  
  "max": 5  
}
```

12.16.49.12 mouth_smiling

The parameter sets standard thresholds for checking [degree of the presence of a smile](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 0,
  "max": 0.5
}
```

12.16.49.13 [mouth_occluded](#)

The parameter sets standard thresholds for checking [degree of presence of mouth occlusion](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 0,
  "max": 0.5
}
```

12.16.49.14 [mouth_open](#)

The parameter sets standard thresholds for checking [degree of presence of an open mouth](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 0,
  "max": 0.5
}
```

12.16.49.15 [glasses](#)

The parameter sets standard thresholds for checking the [most likely state of glasses](#).

Setting format: array.

Default value:

```
"threshold": [
  "no_glasses",
  "eyeglasses"
]
```

12.16.49.16 eyes

The parameter sets standard thresholds for checking [the position of each eye](#).

Setting format: array.

Default value:

```
"threshold": [  
  "open"  
]
```

12.16.49.17 head_horizontal_center

The parameter sets standard thresholds for checking [horizontal face position](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.45,  
  "max": 0.55  
}
```

12.16.49.18 head_vertical_center

The parameter sets standard thresholds for checking [vertical face position](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.3,  
  "max": 0.5  
}
```

12.16.49.19 head_width

The parameter sets standard thresholds for checking [head width](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 0.5,
  "max": 0.75
}
```

12.16.49.20 [head_height](#)

The parameter sets standard thresholds for checking [head height](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 0.6,
  "max": 0.9
}
```

12.16.49.21 [eye_distance](#)

The parameter sets standard thresholds for checking [distance between eyes](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 90
}
```

12.16.49.22 [image_width](#)

The parameter sets standard thresholds for checking [image width](#).

Setting format: object.

Default value:

```
"threshold": {
  "min": 180,
  "max": 1920
}
```

12.16.49.23 [image_height](#)

The parameter sets standard thresholds for checking [image height](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 180,  
  "max": 1080  
}
```

12.16.49.24 [aspect_ratio](#)

The parameter sets standard thresholds for checking [aspect ratio](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.74,  
  "max": 0.8  
}
```

12.16.49.25 [face_width](#)

The parameter sets standard thresholds for checking [face width](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 180  
}
```

12.16.49.26 [face_height](#)

The parameter sets standard thresholds for checking [face height](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 180  
}
```

12.16.49.27 [indent_left](#)

The parameter sets standard thresholds for checking [indents from the left edge of the image](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 20  
}
```

12.16.49.28 [indent_right](#)

The parameter sets standard thresholds for checking [indents from the right edge of the image](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 20  
}
```

12.16.49.29 [indent_upper](#)

The parameter sets standard thresholds for checking [indents from the top edge of the image](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 20  
}
```

12.16.49.30 [indent_lower](#)

The parameter sets standard thresholds for checking [indents from the bottom edge of the image](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 20  
}
```

12.16.49.31 [image_size](#)

The parameter sets standard thresholds for checking [image size](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 5120,  
  "max": 2097152  
}
```

12.16.49.32 [illumination_uniformity](#)

The parameter sets standard thresholds for checking [illumination uniformity according to the ICAO standard](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.5  
}
```

12.16.49.33 [dynamic_range](#)

The parameter specifies standard thresholds for checking [dynamic range](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.5  
}
```

12.16.49.34 [eyebrows](#)

The parameter sets standard thresholds for checking [eyebrow condition](#).

Setting format: array.

Default value:

```
"threshold": [  
  "neutral"  
]
```

12.16.49.35 [shoulders](#)

The parameter sets standard thresholds for checking [shoulders-position](#).

Setting format: array.

Default value:

```
"threshold": [  
  "parallel"  
]
```

12.16.49.36 [smile](#)

The parameter sets standard thresholds for checking [smile state](#).

Setting format: array.

Default value:

```
"threshold": [  
  "none"  
]
```

12.16.49.37 [headwear](#)

The parameter sets standard thresholds for checking [headwear condition](#).

Setting format: array.

Default value:

```
"threshold": [  
  "none"  
]
```

12.16.49.38 [natural_light](#)

The parameter sets standard thresholds for checking [naturalness of lighting](#).

Default threshold value at which LUNA PLATFORM will consider the check completed:

```
"threshold": 1
```

Default threshold value that interprets the estimator response:

```
"min": 0.5
```

12.16.49.39 [fish_eye](#)

The parameter sets standard thresholds for checking [the presence of fisheye effect](#).

Default threshold value at which LUNA PLATFORM will consider the check completed:

```
"threshold": 0
```

Default threshold value that interprets the estimator response:

```
"min": 0.5
```

12.16.49.40 [red_eye](#)

The parameter sets standard thresholds for checking [the presence of red-eye effect](#).

Default threshold value at which LUNA PLATFORM will consider the check completed:

```
"threshold": 0
```

Default threshold value that interprets the estimator response:

```
"min": 0.5
```

12.16.49.41 [color_type](#)

The parameter specifies standard thresholds for checking [face-based image color type](#).

Default threshold value at which LUNA PLATFORM will consider the check completed:

```
"threshold": [  
  "color"  
]
```

Default threshold value that interprets the estimator response:

```
"infrared": {  
  "threshold": {  
    "min": 0.5  
  }  
},  
"color": {  
  "threshold": {  
    "min": 0.5  
  }  
}
```

12.16.49.42 [background_lightness](#)

The parameter sets standard thresholds for checking [background brightness](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.2  
}
```

12.16.49.43 [background_uniformity](#)

The parameter sets standard thresholds for checking [background uniformity](#).

Setting format: object.

Default value:

```
"threshold": {  
  "min": 0.5  
}
```

```
}
```

12.16.50 Other

12.16.50.1 luna_remote_sdk_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.16.50.2 storage_time

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.16.50.3 default_face_descriptor_version

The parameter sets the version of the face descriptor to use.

Setting format: string.

Default value: 59.

Note: For more information about descriptor versions, see “[Neural networks](#)”.

12.16.50.4 default_body_descriptor_version

The parameter sets the version of the body descriptor to use.

For more information about descriptor version, see “[Neural networks](#)”.

Setting format: `string`.

Default value: `110`.

Note: For more information about descriptor versions, see [“Neural networks”](#).

12.16.50.5 `luna_remote_sdk_detector_type`

The parameter sets the type of detector used.

Currently, only one type of detector is available — `“FACE_DET_V3”`.

Setting format: `string`.

Default value: `FACE_DET_V3`.

12.16.50.6 `luna_remote_sdk_use_auto_rotation`

This parameter allows you to enable auto-orientation of the rotated image.

See the description of auto-orientation in the [“Auto-orientation of rotated image”](#) section.

Setting format: `boolean`.

Default value: `true`.

12.17 Lambda service configuration

The section describes the Lambda service parameters.

You can configure the service using the Configurator service.

12.17.1 LUNA_CONFIGURATOR section

This section sets the settings for connecting the Lambda service to the Configurator service.

This section will not be visible in the [Configurator service user interface](#). The parameters can be changed only in the configuration file “config.conf” located in the directory “/srv/luna_lambda/configs/” of the corresponding container.

12.17.1.1 use_configurator

The parameter allows you to enable the use of the Configurator service.

Using the Configurator service, the configuration of LP services is simplified. The service stores all the necessary settings for all LP services in one place.

If this parameter is disabled, the settings from the “config.conf” file located in the “/srv/luna_lambda/configs/” directory of the corresponding container will be used.

Setting format: integer (“0” or “1”).

Default value: 1.

12.17.1.2 luna_configurator_origin

The parameter sets the protocol, IP address and port of the Configurator service.

Setting format: string.

Default value: http://127.0.0.1:5070.

12.17.1.3 luna_configurator_api

The parameter sets the version of the Configurator API service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.17.2 LUNA_LAMBDA_DB section

In this section, the settings for connecting to the database of the service Lambda are set.

12.17.2.1 db_type

The parameter sets the type of database used. The following types are available:

- “postgres” — PostgreSQL database type
- “oracle” — Oracle database type

Setting format: string.

Default value: postgres.

12.17.2.2 db_host

The parameter sets the IP address of the server with the database.

Setting format: string.

Default value: 127.0.0.1.

12.17.2.3 db_port

The parameter sets database listener port.

The default port for “postgres” type is 5432

The default port for “oracle” type is 1521

Setting format: string.

Default value: 5432.

12.17.2.4 db_user

The parameter sets the database username.

Setting format: string.

Default value: luna.

12.17.2.5 db_password

The parameter sets the database password.

Setting format: string.

Default value: luna.

12.17.2.6 db_name

The parameter sets the database name for “postgres” and the name of the SID for “oracle” type to connect to.

Setting format: string.

Default value: `luna_lambda`.

12.17.2.7 `connection_pool_size`

The parameter sets the database connection pool size. The actual number of connections may be greater than the value of this setting by 1.

If necessary, in the “`max_connections`” configuration of the PostgreSQL configuration file, you can set the maximum number of simultaneous connections to the database server. See the section “[Advanced PostgreSQL setting](#)” for more details.

Setting format: `string`.

Default value: `10`.

12.17.2.8 `dsn`

The parameter sets the DSN connection string for connecting to the database.

DSN is a connection string that identifies and points to the data source (database) to which you want to establish a connection.

Settings such as multiple hosts, authentication data, port, and other parameters can be set in the DSN string.

The settings depend on the type of database. Multiple hosts are supported only with PostgreSQL.

By default, the “`dsn`” parameter is not displayed in the “Settings” tab in the Configurator. You can check the list of all available parameters for a section on the “Limitations” tab.

Below is an example of specifying the “`dsn`” parameter:

```
{
  "dsn": "luna:luna@postgres01:5432,postgres02:5432/luna_lambda?some_option=
    some_value"
  "db_settings": {
    "connection_pool_size": 5
  }
}
```

Here:

- “`luna:luna`” — Username and password for connecting to PostgreSQL.
- “`@postgres01:5432,postgres02:5432`” — Comma-separated list of hosts and ports. This means that the service will try to connect to the first host (“`postgres01`”) on port 5432. If this fails, it will try to connect to the second host (“`postgres02`”) also on port 5432.
- “`/luna_lambda`” — Database name.

- “?some_option=some_value” — Optional parameters for connection.

If necessary, you can combine the DSN string and the classic settings, but the DSN string is a higher priority. You can partially fill in the DSN string (for example, “postgres 01,postgres02/luna_lambda”), and then the missing parameters will be filled in from the values of the parameters “db_host”, “db_port”, “db_name”, “db_user” and “db_password”.

At startup, the service will create a pool of connections to one of the available DSN hosts. In case of problems with establishing a connection after several unsuccessful attempts, the service will again try to set up a connection pool to any of the available DSN hosts.

12.17.3 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.17.3.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

12.17.3.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

12.17.3.3 organization

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

12.17.3.4 token

The parameter sets InfluxDB authentication token.

Setting format: `string`.

12.17.3.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

12.17.3.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

12.17.3.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

12.17.3.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer (seconds)`.

Default value: `1`.

12.17.4 LUNA_LAMBDA_LOGGER section

This section sets the logging settings for the logging.

12.17.4.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.17.4.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.17.4.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: boolean.

Default value: true

12.17.4.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: boolean.

Default value: false.

12.17.4.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.17.4.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.17.4.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.17.4.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.17.5 ADDITIONAL_SERVICES_USAGE section

12.17.5.1 luna_events

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: `integer` (“0” or “1”).

Default value: `1`.

12.17.5.2 luna_handlers

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.17.5.3 luna_sender

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.17.5.4 luna_matcher_proxy

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.17.5.5 luna_image_store

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.17.5.6 luna_lambda

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see “Disableable services”.

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.17.6 LUNA_LAMBDA_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.17.6.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: integer (seconds).

Default value: 60.

12.17.6.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: integer (seconds).

Default value: 600.

12.17.6.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

12.17.6.4 `keep_alive_timeout`

The parameter sets the timeout for maintaining HTTP activity.

Setting format: `integer` (seconds).

Default value: 15.

12.17.7 LAMBDA_S3 section

This section contains parameters for interaction with S3-storage, in which modified archives with lambda should be stored.

To use S3, you must specify the following parameters:

- `"host"`
- `"aws_public_access_key"`
- `"aws_secret_access_key"`
- `"authorization_signature"`
- `"bucket"`

12.17.7.1 `host`

The parameter sets the URL address to establish a connection with the S3 storage.

Setting format: `string`

Default value: `http://localhost:7480`

12.17.7.2 `region`

The parameter specifies the region to be used when interacting with S3 storage.

The region can affect the availability and performance of various resources in AWS S3.

Setting format: `string`

Default value: Not specified

12.17.7.3 `aws_public_access_key`

The parameter specifies the public access key used for authentication when accessing the S3 storage.

This key is provided by AWS and is used to identify the client.

Setting format: `string`

Default value: Not specified

12.17.7.4 `aws_secret_access_key`

The parameter sets the secret access key, which, in combination with the public key, provides authentication when accessing the S3 storage.

Setting format: `string`

Default value: Not specified

12.17.7.5 `authorization_signature`

The parameter determines the method used to create an authentication signature when performing operations with S3.

Two values can be specified:

- “s3v4” — using the AWS S3 Version 4 signature algorithm.
- “s3v2” — using the AWS S3 Version 2 signature algorithm.

Setting format: `string`

Default value: `s3v4`

12.17.7.6 `request_timeout`

The parameter sets the maximum time within which a request to the S3 storage must be completed. If the request does not complete within this time, it will be canceled.

Setting format: `integer (seconds)`

Default value: `60`

12.17.7.7 `connect_timeout`

The parameter sets the maximum waiting time for establishing a connection with the S3 storage. If the connection is not established within this time, it will be considered unsuccessful.

Setting format: `integer (seconds)`

Default value: `30`

12.17.7.8 `verify_ssl`

The parameter determines whether to perform SSL certificate verification when establishing a secure (HTTPS) connection with the S3 storage. If the value is `true`, the SSL certificate will be verified. If the value is `false`, verification will be disabled, which may lead to security issues.

Setting format: `boolean`

Default value: `true`

12.17.7.9 bucket

The parameter sets a bucket for storing modified archives with lambda.

Setting format: string.

Default value: lambda_bucket.

12.17.8 CLUSTER_CREDENTIALS section

This section sets the access settings for the Kubernetes cluster.

12.17.8.1 host

The parameter sets the URL of the Kubernetes cluster.

Setting format: string.

Default value: https://127.0.0.1:6443.

12.17.8.2 token

The parameter sets the access token that is used for authentication when connecting to the Kubernetes cluster.

Setting format: string.

Default value: token.

12.17.8.3 certificate_path

The parameter sets the path to the SSL certificate file that is used for secure (HTTPS) connection to the cluster. The default value ./cert.crt indicates the relative path to the certificate file named “cert.crt” in the current directory.

Setting format: string.

Default value: ./cert.crt.

12.17.9 LUNA_LICENSES_ADDRESS section

This section sets the connection settings for the Licenses service.

12.17.9.1 origin

The parameter sets the protocol, IP address and port of the Licenses service.

The IP address “127.0.0.1” means that the Licenses service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Licenses service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5120`.

12.17.9.2 `api_version`

The parameter sets the version of the Licenses service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.17.10 `LUNA_SERVICE_METRICS` section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.17.10.1 `enabled`

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.17.10.2 `metrics_format`

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.17.10.3 `extra_labels`

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.17.11 Other

12.17.11.1 luna_lambda_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[  
  "plugin_1",  
  "plugin_2",  
  "plugin_3"  
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.17.11.2 storage_time

The parameter sets the time format used for records in the database. The following values are available:

- “LOCAL” — displays the local time of the system on which logs are being recorded.
- “UTC” — displays coordinated universal time, which is a standard time and does not depend on the local time zone or seasonal time changes.

Setting format: string.

Default value: LOCAL.

12.17.12 cluster_location

The parameter sets the location of the Kubernetes cluster:

- “internal” — Lambda service works in a Kubernetes cluster and does not require other additional settings.
- “remote” — Lambda service works with a remote Kubernetes cluster and correctly defined settings “[CLUSTER_CREDENTIALS](#)” (host, token and certificate).
- “local” — Lambda service works in the same place where the Kubernetes cluster is running.

In the classic version of working with the Lambda service, it is assumed to use the “internal” parameter.

Setting format: string.

Default value: internal.

12.17.12.1 `lambda_registry`

This parameter sets the Docker registry for storing lambda images. The registry must contain the base images `lpa-lambda-base` and `lpa-lambda-base-fsdk`, as well as the image of the container assembly tool `kaniko-executor`. These images must be transferred from the VisionLabs registry when starting work with lambda. See the detailed information in the installation manual.

Read and write access to this registry is required.

If lambda is running in a Kubernetes cluster, you must grant access to this registry from the cluster.

Setting format: `string`.

Default value: not specified.

12.17.12.2 `lambda_insecure_registries`

This parameter sets a list of insecure Docker registries that require additional security measures or attention when interacting with them.

Setting format: `array > string`.

Default value: not specified.

12.18 Lambda configuration

This section describes the parameters for each lambda unit.

You can configure the service using the Configurator service.

12.18.1 LUNA_LAMBDA_UNIT_LOGGER section

This section sets the logging settings for the logging.

12.18.1.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.18.1.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.18.1.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.18.1.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “folder_with_logs” parameter.

Setting format: `boolean`.

Default value: `false`.

12.18.1.5 `folder_with_logs`

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “`log_to_file`” parameter.

Setting format: `string`.

Default value: `./`

Example:

```
"folder_with_logs": "/srv/logs"
```

12.18.1.6 `max_log_file_size`

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “`log_to_file`” parameter.

If necessary, you can configure Docker log rotation. See the section “`Docker log rotation`” in the LUNA PLATFORM installation manual.

Setting format: `integer`.

Default value: `1024`

12.18.1.7 `multiline_stack_trace`

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: `boolean`.

Default value: `true`.

12.18.1.8 `format`

The parameter defines the format of the output logs. The following values are available:

- “`default`” — standard output format of the LUNA PLATFORM logs.
- “`json`” — output of logs in json format.
- “`ecs`” — output of logs in ECS format (Elastic Common Schema).

When using the “`ecs`” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.18.2 LUNA_LAMBDA_UNIT_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.18.2.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: `integer` (seconds).

Default value: `60`.

12.18.2.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: `integer` (seconds).

Default value: `600`.

12.18.2.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: `integer` (bytes).

Default value: `1073741824`.

12.18.2.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: `integer` (seconds).

Default value: `15`.

12.18.3 LUNA_REMOTE_SDK_ADDRESS section

This section sets the connection settings for the Remote SDK service.

12.18.3.1 origin

The parameter sets the protocol, IP address and port of the Remote SDK service.

The IP address “127.0.0.1” means that the Remote SDK service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Remote SDK service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5220`.

12.18.3.2 api_version

The parameter sets the version of the Remote SDK service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.18.4 LUNA_REMOTE_SDK_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Remote SDK service.

12.18.4.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Remote SDK service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 20.

12.18.4.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.18.4.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.18.4.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.18.5 LUNA_SENDER_ADDRESS section

This section sets the connection settings for the Sender service.

12.18.5.1 origin

The parameter sets the protocol, IP address and port of the Sender service.

The IP address “127.0.0.1” means that the Sender service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Sender service running.

Setting format: string.

Default value: http://127.0.0.1:5080.

12.18.5.2 api_version

The parameter sets the version of the Sender service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.18.6 LUNA_PYTHON_MATCHER_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Python Matcher service.

12.18.6.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Python Matcher service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.18.6.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.18.6.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.18.6.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.18.7 LUNA_PYTHON_MATCHER_PROXY_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Python Matcher Proxy service.

12.18.7.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Python Matcher Proxy service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.18.7.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.18.7.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.18.7.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.18.8 LUNA_PYTHON_MATCHER_ADDRESS section

This section sets the connection settings for the Python Matcher service.

12.18.8.1 origin

The parameter sets the protocol, IP address and port of the Python Matcher service.

The IP address “127.0.0.1” means that the Python Matcher service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher service running.

Setting format: string.

Default value: http://127.0.0.1:5100.

12.18.8.2 `api_version`

The parameter sets the version of the Python Matcher service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.18.9 `LUNA_MATCHER_PROXY_ADDRESS` section

This section sets the connection settings for the Python Matcher Proxy service.

12.18.9.1 `origin`

The parameter sets the protocol, IP address and port of the Python Matcher Proxy service.

The IP address “127.0.0.1” means that the Python Matcher Proxy service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Python Matcher Proxy service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5110`.

12.18.9.2 `api_version`

The parameter sets the version of the Python Matcher Proxy service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.18.10 `LUNA_IMAGE_STORE_IMAGES_ADDRESS` section

In this section, the bucket for storing [source images](#) settings are set.

12.18.10.1 `origin`

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.18.10.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

12.18.10.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-image-origin`.

12.18.11 `LUNA_IMAGE_STORE_IMAGES_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [source images](#) .

12.18.11.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [source images](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: 30.

12.18.11.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: `integer (seconds)`.

Default value: 60.

12.18.12 `LUNA_IMAGE_STORE_FACES_SAMPLES_ADDRESS` section

In this section, the bucket for storing [face samples](#) settings are set.

12.18.12.1 `origin`

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: `string`.

Default value: `http://127.0.0.1:5020`.

12.18.12.2 `api_version`

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: `integer`.

Default value: `1`.

12.18.12.3 `bucket`

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: `string`.

Default value: `visionlabs-samples`.

12.18.13 `LUNA_IMAGE_STORE_FACES_SAMPLES_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [face samples](#).

12.18.13.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [face samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: `integer (seconds)`.

Default value: `30`.

12.18.13.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.18.14 LUNA_IMAGE_STORE_BODIES_SAMPLES_ADDRESS section

In this section, the bucket for storing [body samples](#) settings are set.

12.18.14.1 origin

The parameter sets the protocol, IP address and port of the Image Store service.

The IP address “127.0.0.1” means that the Image Store service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Image Store service running.

Setting format: string.

Default value: http://127.0.0.1:5020.

12.18.14.2 api_version

The parameter sets the version of the Image Store service. The available API version is “1”.

Setting format: integer.

Default value: 1.

12.18.14.3 bucket

The parameter sets the bucket name.

See the detailed description of packages in the “[Bucket description](#)” section.

Setting format: string.

Default value: visionlabs-bodies-samples.

12.18.15 LUNA_IMAGE_STORE_BODIES_SAMPLES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the bucket with [body samples](#) .

12.18.15.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the bucket with [body samples](#). This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 30.

12.18.15.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.18.16 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

12.18.16.1 origin

The parameter sets the protocol, IP address and port of the Faces service.

The IP address “127.0.0.1” means that the Faces service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Faces service running.

Setting format: string.

Default value: http://127.0.0.1:5030.

12.18.16.2 api_version

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: integer.

Default value: 3.

12.18.17 LUNA_FACES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

12.18.17.1 connect

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Faces service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.18.17.2 request

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.18.17.3 sock_connect

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.18.17.4 sock_read

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.18.18 LUNA_EVENTS_ADDRESS section

This section sets the connection settings for the Events service.

12.18.18.1 origin

The parameter sets the protocol, IP address and port of the Events service.

The IP address “127.0.0.1” means that the Events service located on the server with Configurator will be used. If the service is located on another server, then in this parameter you need to specify the correct IP address of the server with the Events service running.

Setting format: string.

Default value: http://127.0.0.1:5040.

12.18.18.2 `api_version`

The parameter sets the version of the Events service. The available API version is “2”.

Setting format: integer.

Default value: 2.

12.18.19 LUNA_EVENTS_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Events service.

12.18.19.1 `connect`

The parameter sets the timeout for establishing a connection when sending an HTTP request to the Events service. This is the timeout during which the client tries to establish a connection with the service.

Setting format: integer (seconds).

Default value: 20.

12.18.19.2 `request`

The parameter sets a general timeout for the entire HTTP request. It includes the time to establish a connection, send a request, receive a response, and close the connection. If the entire process takes longer than specified in this parameter, the request will be aborted.

Setting format: integer (seconds).

Default value: 60.

12.18.19.3 `sock_connect`

The parameter sets the timeout for establishing a connection at the socket level. If the socket level connection is not established at the set time, the operation will be aborted.

Setting format: integer (seconds).

Default value: 10.

12.18.19.4 `sock_read`

The parameter sets the timeout for reading data from the socket after a successful connection. If the data does not arrive at the set time, the read operation will be interrupted.

Setting format: integer (seconds).

Default value: 60.

12.18.20 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

12.18.20.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

12.18.20.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

12.18.20.3 organization

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

12.18.20.4 token

The parameter sets InfluxDB authentication token.

Setting format: string.

12.18.20.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

12.18.20.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

12.18.20.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: 8086.

12.18.20.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer` (seconds).

Default value: 1.

12.18.21 ADDITIONAL_SERVICES_USAGE section

12.18.21.1 luna_events

The parameter sets the possibility of using the Events service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: `integer` (“0” or “1”).

Default value: 1.

12.18.21.2 luna_handlers

The parameter sets the possibility of using the Handlers service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: `integer` (“0” or “1”).

Default value: 1.

12.18.21.3 luna_sender

The parameter sets the possibility of using the Sender service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.18.21.4 [luna_matcher_proxy](#)

The parameter sets the possibility of using the Python Matcher Proxy service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.18.21.5 [luna_image_store](#)

The parameter sets the possibility of using the Image Store service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.18.21.6 [luna_lambda](#)

The parameter sets the possibility of using the Lambda service.

Enabling/disabling this service may affect the operation of other services. For more information, see [“Disableable services”](#).

The installation manual contains the section “Optional services usage”, which describes how to configure the use of optional services before launching the LUNA PLATFORM.

Setting format: integer (“0” or “1”).

Default value: 1.

12.18.22 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.18.22.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.18.22.2 metrics_format

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.18.22.3 extra_labels

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.18.23 Other

12.18.23.1 luna_lambda_unit_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[
  "plugin_1",
  "plugin_2",
  "plugin_3"
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.

12.19 Tasks-lambda configuration

This section describes the parameters for each lambda tasks unit. Most of the settings are similar to the settings described in the “[Tasks service configuration](#)” section. The individual settings are described below.

You can configure the service using the Configurator service.

12.19.1 LUNA_LAMBDA_TASKS_UNIT_LOGGER section

This section sets the logging settings for the logging.

12.19.1.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

12.19.1.2 log_time

The parameter sets the time format used in log entries. The following values are available:

- “LOCAL” — displays the local time of the system on which the logs are being recorded.
- “UTC” — displays Coordinated Universal Time, which is a time standard and is not affected by local time zones or seasonal time changes.

Setting format: `string`.

Default value: `LOCAL`.

12.19.1.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

12.19.1.4 log_to_file

The parameter enables you to save logs to a file. The directory with log files is specified in the “`folder_with_logs`” parameter.

Setting format: `boolean`.

Default value: `false`.

12.19.1.5 folder_with_logs

The parameter sets the directory in which logs are stored. The relative path starts from the application directory.

To use this parameter, you must enable the “log_to_file” parameter.

Setting format: string.

Default value: ./

Example:

```
"folder_with_logs": "/srv/logs"
```

12.19.1.6 max_log_file_size

The parameter sets the maximum size of the log file in MB before performing its rotation (0 — do not use rotation).

To use this parameter, you must enable the “log_to_file” parameter.

If necessary, you can configure Docker log rotation. See the section “Docker log rotation” in the LUNA PLATFORM installation manual.

Setting format: integer.

Default value: 1024

12.19.1.7 multiline_stack_trace

The parameter enables multi-line stack tracing in logs. When the parameter is enabled, information about the call stack is recorded in the logs so that each stack frame is placed on a separate line, which improves readability. If the parameter is disabled, information about the call stack is recorded on one line, which may make logs less convenient for analysis.

Setting format: boolean.

Default value: true.

12.19.1.8 format

The parameter defines the format of the output logs. The following values are available:

- “default” — standard output format of the LUNA PLATFORM logs.
- “json” — output of logs in json format.
- “ecs” — output of logs in ECS format (Elastic Common Schema).

When using the “ecs” value, the following fields will be used:

- “http.response.status_code” — contains the HTTP response status code (e.g., 200, 404, 500, etc.).
- “http.response.execution_time” — contains information about the time taken to execute the request and receive the response.
- “http.request.method” — contains the HTTP request method (GET, POST, PUT, etc.).
- “url.path” — contains the path in the request’s URL.
- “error.code” — contains the error code if the request results in an error.

Setting format: `string`.

Default value: `default`.

12.19.2 LUNA_LAMBDA_TASKS_UNIT_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

12.19.2.1 request_timeout

The parameter sets the duration of time between the instant when a new open TCP connection is passed to the server, and the instant when the whole HTTP request is received.

Setting format: `integer` (seconds).

Default value: `60`.

12.19.2.2 response_timeout

The parameter sets the duration of time between the instant the server passes the HTTP request to the app, and the instant a HTTP response is sent to the client.

Setting format: `integer` (seconds).

Default value: `600`.

12.19.2.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: `integer` (bytes).

Default value: `1073741824`.

12.19.2.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: `integer` (seconds).

Default value: `15`.

12.19.3 LUNA_SERVICE_METRICS section

This section enables and configures the collection of metrics in the Prometheus format.

See “[Monitoring](#)” for details.

12.19.3.1 enabled

The parameter enables metrics collection.

If metrics collection is disabled, a request to the `/metrics` resource will return an appropriate message.

Setting format: `boolean`.

Default value: `false`.

12.19.3.2 metrics_format

The parameter sets the metrics format.

Currently only the Prometheus format is supported.

See the [official Prometheus documentation](#) for more details.

Setting format: `string`.

Default value: `prometheus`.

12.19.3.3 extra_labels

The parameter specifies custom label types.

Setting format: `label_name=label_value`.

Default value is not set.

12.19.4 Other

12.19.4.1 luna_lambda_tasks_unit_active_plugins

The parameter sets a list of plugins that the service should use.

The names are given in the following format:

```
[
  "plugin_1",
  "plugin_2",
  "plugin_3"
]
```

The list should contain file names without the extension (.py).

Setting format: integer.

Default value: 1.