



VisionLabs
MACHINES CAN SEE

LUNA Index Module

Administrator manual

v.5.57.0

Contents

Glossary	8
1 Overview	9
2 General concepts	10
2.1 Index	10
2.1.1 Associating index with descriptor version	10
2.1.2 Index structure	11
2.1.3 Index building task creation progress	11
2.1.4 Index creation process	11
2.2 Matching	12
2.2.1 Matching requests	13
2.2.2 Matching process	13
3 Service interaction	14
4 Index services	16
4.1 Index Manager service	16
4.1.1 Background routines	16
4.1.2 Index Manager storage	16
4.1.2.1 Work with multiple instances	16
4.1.3 Requests to service	17
4.2 Indexer service	18
4.3 Indexed Matcher service	19
4.3.1 Synchronization of matching labels in memory	19
4.3.2 Index reloading	20
4.3.3 Refreshing index in memory	21
4.3.4 Index caching	22
5 Matching plugin for Python Matcher Proxy	23
5.1 Matching plugin description	23
5.2 Matching cost	23
5.3 Matching targets	24
6 Monitoring	25
6.1 Data being sent	25
7 Sequence diagrams	28
7.1 Index creation diagram	28
7.2 Diagram of initial index loading into memory	31

7.3	Diagram of matching descriptors	32
7.4	Index reloading diagram	33
7.5	Index refreshing diagram	35
8	Use Redis Sentinel	38
9	API errors	39
9.1	Indexer service errors	39
9.1.1	Code 26100 returned	39
9.1.2	Code 26101 returned	39
9.1.3	Code 26103 returned	39
9.1.4	Code 26104 returned	40
9.1.5	Code 26105 returned	40
9.1.6	Code 26106 returned	40
9.1.7	Code 26107 returned	41
9.1.8	Code 26108 returned	41
9.1.9	Code 26109 returned	41
9.2	Index Manager service errors	41
9.2.1	Code 26201 returned	41
9.2.2	Code 26202 returned	42
9.2.3	Code 26203 returned	42
9.2.4	Code 26204 returned	42
9.3	Index Matcher service errors	43
9.3.1	Code 26301 returned	43
9.3.2	Code 26302 returned	43
9.3.3	Code 26303 returned	43
9.3.4	Code 26304 returned	44
9.3.5	Code 26305 returned	44
9.3.6	Code 26306 returned	44
9.3.7	Code 26307 returned	45
9.3.8	Code 26308 returned	45
10	Configuration parameters of services	46
10.1	Index Manager service configuration	46
10.1.1	LIM_MANAGER_LOGGER section	46
10.1.1.1	log_level	46
10.1.1.2	log_time	46
10.1.1.3	log_to_stdout	46
10.1.1.4	log_to_file	46
10.1.1.5	folder_with_logs	47

10.1.1.6	max_log_file_size	47
10.1.1.7	multiline_stack_trace	47
10.1.1.8	format	47
10.1.2	LIM_MANAGER_INDEXING section	48
10.1.2.1	indexer_origins	48
10.1.2.2	planning_period	48
10.1.2.3	lookup_period	48
10.1.2.4	face_lists > min_indexing_list_size	48
10.1.2.5	face_lists > indexing_lists	49
10.1.2.6	ef_construction	49
10.1.3	LIM_MANAGER_HTTP_SETTINGS section	49
10.1.3.1	request_timeout	49
10.1.3.2	response_timeout	49
10.1.3.3	request_max_size	50
10.1.3.4	keep_alive_timeout	50
10.1.4	LIM_MANAGER_DB section	50
10.1.4.1	db_user	50
10.1.4.2	db_password	50
10.1.4.3	db_host	50
10.1.4.4	db_port	50
10.1.4.5	db_settings > connection_pool_size	51
10.1.4.6	db_number	51
10.1.4.7	sentinel > master_name	51
10.1.4.8	sentinel > sentinels	51
10.1.4.9	sentinel > user	51
10.1.4.10	sentinel > password	51
10.1.5	INFLUX_MONITORING section	52
10.1.5.1	send_data_for_monitoring	52
10.1.5.2	use_ssl	52
10.1.5.3	organization	52
10.1.5.4	token	52
10.1.5.5	bucket	52
10.1.5.6	host	52
10.1.5.7	port	53
10.1.5.8	flushing_period	53
10.1.6	LUNA_FACES_ADDRESS section	53
10.1.6.1	origin	53
10.1.6.2	api_version	53

10.1.7	LUNA_FACES_TIMEOUTS section	53
10.1.7.1	connect	54
10.1.7.2	request	54
10.1.7.3	sock_connect	54
10.1.7.4	sock_read	54
10.1.8	Other	54
10.1.8.1	index_storage_type	54
10.1.8.2	index_storage_local	55
10.1.8.3	storage_time	55
10.1.8.4	lim_manager_active_plugins	55
10.1.8.5	default_face_descriptor_version	55
10.2	Indexed Matcher service configuration	56
10.2.1	LIM_MATCHING section	56
10.2.1.1	ef_search	56
10.2.2	LIM_MATCHER_REFRESH section	56
10.2.2.1	enabled	56
10.2.3	LIM_MATCHER_LOGGER section	56
10.2.3.1	log_level	56
10.2.3.2	log_time	57
10.2.3.3	log_to_stdout	57
10.2.3.4	log_to_file	57
10.2.3.5	folder_with_logs	57
10.2.3.6	max_log_file_size	57
10.2.3.7	multiline_stack_trace	58
10.2.3.8	format	58
10.2.4	LIM_MATCHER_HTTP_SETTINGS section	58
10.2.4.1	request_timeout	58
10.2.4.2	response_timeout	59
10.2.4.3	request_max_size	59
10.2.4.4	keep_alive_timeout	59
10.2.5	LIM_MATCHER_DB section	59
10.2.5.1	db_user	59
10.2.5.2	db_password	59
10.2.5.3	db_host	60
10.2.5.4	db_port	60
10.2.5.5	db_settings > connection_pool_size	60
10.2.5.6	db_number	60
10.2.5.7	sentinel > master_name	60
10.2.5.8	sentinel > sentinels	60

10.2.5.9	sentinel > user	61
10.2.5.10	sentinel > password	61
10.2.6	INFLUX_MONITORING section	61
10.2.6.1	send_data_for_monitoring	61
10.2.6.2	use_ssl	61
10.2.6.3	organization	61
10.2.6.4	token	61
10.2.6.5	bucket	62
10.2.6.6	host	62
10.2.6.7	port	62
10.2.6.8	flushing_period	62
10.2.7	LUNA_FACES_ADDRESS section	62
10.2.7.1	origin	62
10.2.7.2	api_version	63
10.2.8	LUNA_FACES_TIMEOUTS section	63
10.2.8.1	connect	63
10.2.8.2	request	63
10.2.8.3	sock_connect	63
10.2.8.4	sock_read	63
10.2.9	LUNA_LICENSES_ADDRESS section	64
10.2.9.1	origin	64
10.2.9.2	api_version	64
10.2.10	Other	64
10.2.10.1	lim_matcher_cache	64
10.2.10.2	index_storage_type	64
10.2.10.3	index_storage_local	65
10.2.10.4	lim_matcher_active_plugins	65
10.2.10.5	default_face_descriptor_version	65
10.3	Indexer service configuration	66
10.3.1	LIM_INDEXER_LOGGER section	66
10.3.1.1	log_level	66
10.3.1.2	log_time	66
10.3.1.3	log_to_stdout	66
10.3.1.4	log_to_file	66
10.3.1.5	folder_with_logs	67
10.3.1.6	max_log_file_size	67
10.3.1.7	multiline_stack_trace	67
10.3.1.8	format	67

10.3.2	INFLUX_MONITORING section	68
10.3.2.1	send_data_for_monitoring	68
10.3.2.2	use_ssl	68
10.3.2.3	organization	68
10.3.2.4	token	68
10.3.2.5	bucket	69
10.3.2.6	host	69
10.3.2.7	port	69
10.3.2.8	flushing_period	69
10.3.3	LUNA_FACES_ADDRESS section	69
10.3.3.1	origin	69
10.3.3.2	api_version	70
10.3.4	LUNA_FACES_TIMEOUTS section	70
10.3.4.1	connect	70
10.3.4.2	request	70
10.3.4.3	sock_connect	70
10.3.4.4	sock_read	70
10.3.5	LIM_INDEXER_HTTP_SETTINGS section	71
10.3.5.1	request_timeout	71
10.3.5.2	response_timeout	71
10.3.5.3	request_max_size	71
10.3.5.4	keep_alive_timeout	71
10.3.6	Other	71
10.3.6.1	index_storage_type	71
10.3.6.2	index_storage_local	72
10.3.6.3	lim_indexer_active_plugins	72
10.3.6.4	default_face_descriptor_version	72
10.4	Matching plugin configuration	73
10.4.1	LUNA_INDEXED_LIST_PLUGIN section	73
10.4.1.1	redis_url	73
10.4.1.2	request_timeout	73

Glossary

Term	Description
Descriptor	Set of unique features received from the sample. It is used for matching of faces. Descriptor is not considered personal data.
Index	LUNA Index Module (LIM) entity containing set of received from user-provided images and deployed together for approximate matching. The index is generated by the Indexer service after receiving the task from the Index Manager service.
Matching	The process of comparing descriptor with some descriptor batch that results in similarity scores. Its purpose is to search most similar descriptors to a given one over some user-provided set of descriptors.
Matching label	LIM entity that contains the ID of a list of faces.
Relevant index	Last built index for the list, since the index can be rebuilt if the task is created automatically.
Sample	Image containing a face and corresponding to VisionLabs and other standards.

Abbreviation	Full form
DB	Database
LP	LUNA PLATFORM
LIM	VisionLabs LP5 Index, LUNA Index Module

1 Overview

LUNA Index Module is a module consisting of the Index Manager, Indexer and Indexed Matcher services, designed to speed up the matching of a large number of descriptors. 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 basic principle of the module work 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.

The ability to perform matching using the Indexed Matcher service is controlled using a separate parameter in the LUNA PLATFORM 5 license key. Thus, without a license, you can use the Indexer and Indexed Matcher services, but the built indexes cannot be processed.

This document contains the following main sections:

- [General concepts](#). The section contains:
 - Description of index.
 - Description of how LIM works.
 - Description of the process of creating tasks for building indexes.
 - Description of the index building process.
 - Description of the matching process.

It is recommended to start your acquaintance with LIM from this section.

- [Service interaction](#). This section provides a interaction diagram of LIM services, which covers the necessary sequence of actions to perform the matching using LIM services.
- [Index services](#). This section provides basic information about LIM services and the nuances of working with them.
- [Matching plugin for Python Matcher Proxy](#). This section describes the operation of the matching plugin built into the Python Matcher Proxy service, which is required to perform the matching.
- [Monitoring](#). This section describes the monitoring process for LIM services.
- [API errors](#). This section provides an extended description of the errors returned by LIM services.
- [Configuration parameters of services](#). This section describes the parameters for all LIM services.

2 General concepts

LUNA Index Module:

- Sends requests for indexing lists with descriptors.
- Performs index building.
- Loads index into memory and performs matching.

LIM contains the following services:

- [Index Manager](#) - Manages index building tasks and coordinates the Indexer service.
- [Indexer](#) - Builds indexes based on the list of descriptors.
- [Indexed Matcher](#) - Performs approximate nearest neighbor (descriptors) matching using indexes built.

See the [Wiki](#) for more information about the nearest neighbor search.

It is required Python Matcher Proxy service with a built-in [matching plugin](#) to work with module. Matching plugin enables you to determine to which service requests from the LUNA API service will be sent - to the Python Matcher service or to the Indexed Matcher service. The Index Manager and Indexed Matcher services require a Redis database.

All LIM services are scalable, which means you can use multiple instances.

If necessary, you can change the logging format of all services to json (see the `FORMAT` setting for each service).

2.1 Index

Index is a collection of user-provided set of descriptors deployed together for approximate matching. It is building as a dependency graph whose vertices are descriptors. Search descriptors in this dependency graph is performed while moving along its vertices (see “[Matching process](#)” section below).

Building the index requires a lot of resources for a long time and is a rather slow process, so you need to correctly set the period for automatic index rebuilding when changes appear in the list (see below).

The size of the list with descriptors controls speed/accuracy trade-off during the index construction and search. Higher values leads to more accurate but slower search. To configure these parameters, use the “[ef_construction](#)” and “[ef_search](#)” settings.

2.1.1 Associating index with descriptor version

LUNA Index Module takes into account the version change of face descriptors. The Indexer service builds the index from the version descriptors specified in the “`DEFAULT_FACE_DESCRIPTOR_VERSION`” setting of the Index Manager service. The Index Manager service automatically rebuilds the index if it does not contain descriptor version information. The Indexed Matcher service only loads indexes that contain

descriptors for the version specified in the “DEFAULT_FACE_DESCRIPTOR_VERSION” setting of the Index Manager service.

2.1.2 Index structure

The index consists of the following files:

- The meta.json file, which contains meta information about the index, including which objects are indexed.
- The index.dat file, which contains binary index data.
- The ids.dat file, which contains an ordered list of object IDs in the index.

Each index has unique name, and it is used as key/folder name.

The default index storage directory is specified for each LIM service in the “index_storage_local” setting of the “OTHER” section of the Configurator service. Note that the directory must be the same for all three services.

2.1.3 Index building task creation progress

The indexing of a set of descriptors is performed out by placing tasks for indexing in a queue. Such tasks are created in the Index Manager service. There are two types of index building tasks - **one-time** and **background**.

One-time type enables you to “[create task](#)” to build the index once using an HTTP request to the Index Manager service. In the request body, you should specify the required “list_id”.

Background type enables you to create index building tasks in the background, where:

- Set of lists is explicitly specified in the “[indexing_lists](#)” setting.
- All existing lists in LP are dynamically indexed, whose number of faces exceeds the number specified in the “[min_indexing_list_size](#)” setting. In this case, the value of the “indexing_lists” setting should take the value “dynamic”. The default value is 50000 faces.

When using the background type, the Index Manager service tracks changes in the number of faces in the lists, interacting with the Faces service. If the number of faces has changed, a new task will be sent to the internal queue.

One task processes only one list.

To disable task building in the background, you need to set the value of the “indexing_lists” setting to [].

2.1.4 Index creation process

Below is the operation process of the index creation:

1. To start indexing, the Index Manager service sends a request to the Indexer service with the necessary parameters - “list_id” and “task_id”. The Indexer service converts these parameters into “label” and “index_id” respectively.
2. When the indexing request is received, the Indexer service starts a separate indexing process. At this point, the Indexer sets its status to “indexing”.
3. When the indexing process is started, the Indexer service fetches the descriptors from the Faces service. Fetching is performed in batches of 1000 items.
4. After all descriptors have been fetched and loaded into memory, Indexer begins building of the index. A directed descriptor dependency graph is created (see “[Index](#)”).
5. Next, when indexing has finished, the index itself is saved using configured backend (filesystem). In the storage, the index is a directory containing some files (see “[Index structure](#)”).
6. After successfully saving the index, the indexing process stops. At this point, the Indexer sets its status to “success”. If the indexing process ended in an error, then the Indexer will set its status to “error”.

Information about stored indexes can be obtained using “[get indexes](#)” or “[get most relevant indexes](#)” to the Index Manager service.

You can view the status of the Indexer service using the “[get tasks](#)” request to the Index Manager service.

Some time after the indexes are stored, all running instances of the Indexed Matcher automatically (re)load those indexes into memory. After the indexes are loaded into memory, you can send requests to match the indexed descriptors sets with the specified matching label.

2.2 Matching

Indexed Matcher loads more relevant indexes from the storage and processes requests for matching. Because the index storage can contain multiple versions of indexes with a specific matching label, the Indexed Matcher service always tries to match against the newer (i.e., more relevant) version.

The index becomes outdated as soon as descriptors are created or deleted in LUNA PLATFORM 5.

In-memory indexes in the Indexed Matcher service are synchronized with the store by a periodic background process called index reloading (see “[Index reloading](#)” section for details).

If any changes were made to the source list, the Indexed Matcher service updates the corresponding indexes in its memory by gradually adding a small number of new descriptors to the index loaded into memory (see “[Refreshing index in memory](#)” section for details).

2.2.1 Matching requests

Matching requests come from the API service to the Matcher Proxy service, which uses the [matching plugin](#) to forward the request to the Indexed Matcher service. The Indexed Matcher service accepts matching requests via [Redis streams](#), performs the matching, and sends the matching result to the [Redis channel](#), from where the result is redirected to the Python Matcher Proxy service and then to the API service.

For requests for each corresponding matching label, there is the stream with the label name. Several running instances of Indexed Matcher with index loaded are the [consumer group](#) for this stream.

2.2.2 Matching process

The Indexed Matcher service moves along the vertices of the dependency graph (index).

After moving to the first vertex of the graph, the service matches the incoming descriptor with all the vertices associated with the current vertex. When the most similar vertex is found, the next matching is made with the vertices associated with it. After several iterations, the most similar vertex is found (i.e., the descriptor with the highest similarity score). The number of operations with such a search is significantly reduced, which increases the search performance a hundred times.

3 Service interaction

Below is a diagram of the interaction of the index module services.

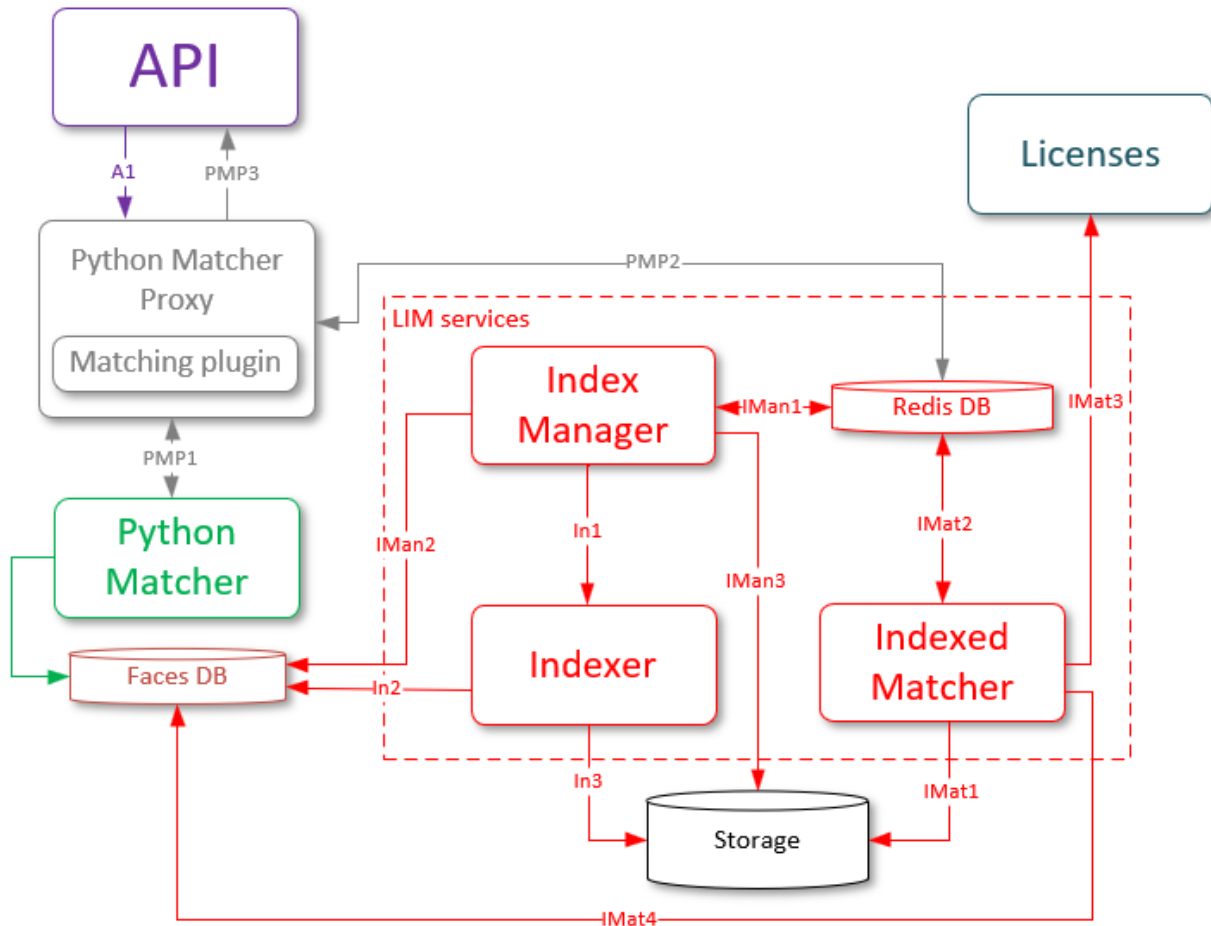


Figure 1: Service Interaction

Before sending the matching request from the API service, the user must create an index, which is created by creating a task to build it. The task is created in the Index Manager service and can be [one-time type](#) or [background type](#).

After creation, the task is sent to the queue in the Redis database (**IMan1**).

In addition to the queue, the Redis database acts as a repository for all index building tasks. It also uses the RedLock mechanism to ensure the operation of multiple instances of the Index Manager service (see the [“Working with multiple instances”](#) section).

The Index Manager service interacts with the Faces database (**IMan2**) to monitor changes in lists if the task of the background type has been created.

Next, the Index Manager sends a request to the Indexer service to build the index (**In1**). After receiving the request, the Indexer service extracts the descriptors from the Faces database (**In2**) and starts building the index. The built index is stored in storage (**In3**).

After successfully creating the index, the status of the task changes to “success”. Before sending the matching request, you need to check the status of the index using the “[get tasks](#)” request to the Index Manager service. When sending user requests to the Index Manager service to get information about indexes, the service will interact with the index storage (**IMan3**).

The Indexed Matcher load the index into memory (**IMat1**), loads the matching label (contains the “list_id” of the loaded into memory index) into the Redis DB (**IMat2**) and listens to the [Redis stream](#) until the matching request appears there.

The Indexed Matcher service constantly monitors changes in face lists by interacting with the Faces (**IMat4**) database. If new changes are made to the list, the Indexed Matcher service gradually adds new descriptors to the corresponding index loaded into memory. In this case, the index located in the repository remains unchanged until it is rebuilt. If necessary, this functionality can be disabled (see section “[Refreshing index in memory](#)”).

After the index is created and loaded into the memory of the Indexed Matcher service, the user performs the matching request in the API service, attaching an image of the reference. This request is redirected to the Python Matcher Proxy service (**A1**), where the matching plugin generates a request of a specific format containing the matching label and the descriptor of the reference, and determines whether the Indexed Matcher service loaded the matching label into the Redis database. Next, the service is selected to perform the matching: - If the matching label is not loaded into the Redis database, then the request is sent to the Python Matcher (**PMP1**). In this case, the matching will be performed using the classical brute-force matching descriptors. - If the matching label is loaded into the Redis database, then the request is sent as a message to the Redis stream. The Indexed Matcher service reads a message from the Redis stream (**IMat2**). After that, the Indexed Matcher service checks the presence of a LUNA PLATFORM 5 license for the possibility of performing the matching, interacting with the Licenses (**IMat3**) service.

See “[Matching plugin for Python Matcher Proxy](#)” for details on choosing a service to perform the matching.

After the matching is completed, the Indexed Matcher service writes the matching results to the [Redis channel](#) (**IMat2**). The Python Matcher Proxy service matching plugin reads the matching results and returns them to the user in the API service (**PMP2**).

See the “[Sequence diagrams](#)” section for a more detailed description of the LUNA Index Module processes.

4 Index services

4.1 Index Manager service

The service manages the process of creating indexes for lists containing face descriptors and performs the following tasks:

- Generates tasks for building the index.
- Sends tasks to the internal queue.
- Retrieves tasks from the internal queue and coordinates the process of sending tasks for indexing to the Indexer service.

It is recommended to run at least two manager instances for redundancy purposes. Since task management is carried out through the Redis, if one manager is down, the second one will be able to continue its work from the instant step.

4.1.1 Background routines

The Index Manager service performs two types of background routines in parallel:

- Planning routine
- Lookup routine

In the planning routine, the Index Manager checks which sets of lists are to be indexed, then creates tasks under a unique “task_id” and places them in the internal queue. The planning procedure is executed with the period (sec) specified in the “[planning_period](#)” setting.

In the lookup routine, the Index Manager checks the status of all running Indexer instances. If any Indexer instance has finished building the index, the Index Manager updates the task information and submits the data for [monitoring](#). If any Indexer instance is ready to accept the task, the Index Manager service retrieves the next task from the internal queue, sends the task to the free Indexer instance, and updates the task information. The lookup routine is performed with the period (sec) specified in the “[lookup_period](#)” setting.

4.1.2 Index Manager storage

All information about the created tasks is stored in the Redis database. Also, using the Redis Redlock mechanism, work with multiple instances is regulated.

4.1.2.1 Work with multiple instances

The multiple instance mode is supported by automatic selection of the master instance based on Redis Redlock.

See <https://redis.io/docs/reference/patterns/distributed-locks/> for details on distributed locks done with Redis.

Only the master instance can perform planning and lookup background routines. The remaining instances can only accept requests for a one-time index creation, as well as issue responses to GET requests.

4.1.3 Requests to service

Interaction with the Index Manager service is performed using HTTP requests. The main requests are listed below:

- “[get queue](#)” - Get list of tasks and their number from the queue.
- “[get tasks](#)” - Get information on tasks:
 - “task_id” - Task ID.
 - “status” - “pending”, “indexing”, “success”, “error”.
 - “create_time” - Index build create time in RFC 3339 format.
 - “start_time” - Index build start time in RFC 3339 format.
 - “end_time” - Index build end time in RFC 3339 format.
 - “indexer” - Address of the server where the Indexer instance that processes the specified task is running.
 - “error” - Error received during index build.
 - “content” - Processed “list_id”.

If necessary, you can filter the received tasks.

- “[create task](#)” - Create task to build the index once.
- “[remove tasks](#)” - Delete tasks. If necessary, you can filter the tasks to be deleted.
- “[get indexes](#)” - Get the number of indexes, as well as the following information for each index:
 - “index_id” - Equal to “task_id”.
 - “index_type” - List only.
 - “label” - Processed “list_id”.
- “[remove indexes](#)” - Delete the index from the repository by ID.
- “[get most relevant indexes](#)” - Get information on the most relevant index, i.e. by the last built index for the list.

See the [OpenAPI specification](#) for more information about requests made to the Index Manager service and other requests.

4.2 Indexer service

The Indexer service is intended to process tasks received by the Index Manager service and perform the [indexes creation process](#).

Requests to the Indexer service are not intended for the user. All requests related to the LUNA Index Module must be made to the Index Manager service (see [“Requests to Index Manager service”](#)).

The deployment of the Indexer service should be done on a separate server, because building an index takes a lot of resources for a long time. One Indexer instance can only build one index at a time, so it is recommended to run multiple indexer instances. The indexer must be also configured with storage, which must be large enough.

4.3 Indexed Matcher service

The Indexed Matcher service loads the most relevant indexes from the index storage (file system) and processes matching requests.

On startup, the Indexed Matcher service loads all indexes of the latest version from the index storage into memory and sets up Redis streams to accept match messages for all matching labels loaded into the index storage.

The Indexed Matcher service always checks for the existence of the list when starting, loading a new index into memory, and refreshing an index in memory. An index without an existing list will be removed from the service's memory.

To speed up access to the index, you can configure index caching in a special folder in the Indexed Matcher service container (caching is disabled by default). Caching is enabled by “[LIM_MATCHER_CACHE](#)” setting.

Requests to the Indexed Matcher service are not intended for the user. All requests related to the LUNA Index Module must be made to the Index Manager service (see “[Requests to Index Manager service](#)”).

Indexed Matcher does not communicate with other LIM services. It only monitors the storage, and when indices appear it loads them into memory. Since matching requests processing is carried out through the Redis streams, any number of matcher instances could be run without any system config updates. The number of Indexed Matcher instances should be determined by performance requirements.

4.3.1 Synchronization of matching labels in memory

The Indexed Matcher service synchronizes matching labels of indexes with Redis keys in its memory. For all labels in memory, the service sets the keys in the following format:

```
matching_label__<label>__<matcher_id>
```

For example, `matching_label__17cdbc41-c7f1-440b-b9ad-aad93c7176ee__127.0.0.1:5200`.

The `<matcher_id>` field in the label key is the host and port of the Indexed Matcher instance. The host is read from the environment variable `VL_LIM_MATCHER_HOST` or, if the variable is not set, it is guessed using the operating system sockets API. Reading these keys from Redis enables the matching plugin to get information about which instances of Indexed Matcher specific index labels were loaded into memory.

Label key being set have a TTL and will expire if not updated again. The presence of such a key in Redis means that some of the running Indexed Matcher instances can process matching requests on the label.

4.3.2 Index reloading

In-memory indexes in the Indexed Matcher service are synchronized with the store by a periodic background process called index reloading.

If the index is removed from storage, the index is also removed from the Indexed Matcher service's memory.

If a new index with a new match label appears in the store, the Indexed Matcher service will attempt to load the new index into memory.

If a new index appears in the store with a newer version of the matching label than the index loaded into memory, the Indexed Matcher service will try to load the new index into memory instead of the old one.

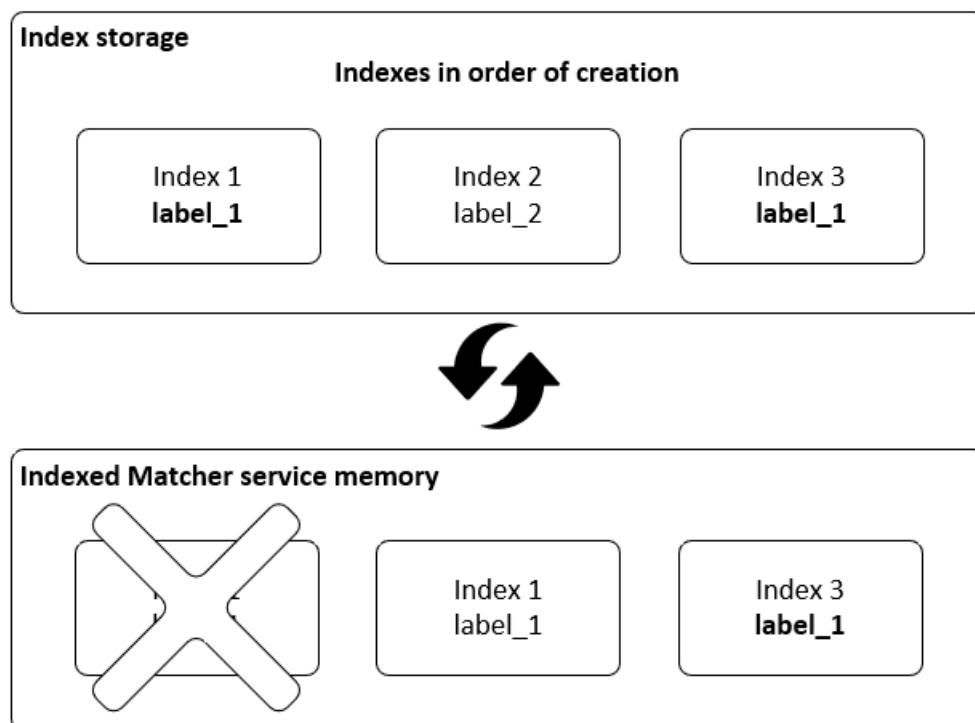


Figure 2: Replacing index with outdated version of list (Index 1) with new one (Index 3)

To ensure that the given index can only be reloaded by one Indexed Matcher service at a time, the Redis Redlock mechanism is used. If a lock is set, the older version of the index is removed from the Indexed Matcher service's memory and the newer one is loaded.

If there is a problem loading the index, for example, lack of memory, an appropriate message is sent to the logs and monitoring.

When the index is reloaded, the Indexed Matcher service does not accept matching requests for the corresponding label. However, only one Indexed Matcher can reload the index for a particular label at a time. Therefore, it is recommended to run multiple instances of the Indexed Matcher in order to be able to match all labels at any time.

See the sequence diagram for index reloading in the “[Index reloading diagram](#)” section.

4.3.3 Refreshing index in memory

By default, the Indexed Matcher service monitors lists with faces for changes. If new changes are made to the list, the Indexed Matcher service updates the corresponding indexes in its memory by gradually adding a small number of descriptors.

The use of this functionality is controlled by the “[enabled](#)” setting.

This information is described for an index that is already loaded into the memory of the Indexed Matcher service. The index used and the index in the storage may differ.

When the index is updated in memory, the Indexed Matcher service stops matching on that index, but continues to accept new match requests for that index. By adding a small number of descriptors (no more than 10 descriptors at a time) to the index in memory, the matching process is performed with minimal interruption. However, it should be taken into account that if elements are inserted into the list too often (dozens and hundreds of additions), this will affect a significant degradation in the speed of work, up to an almost complete stop of the matching process.

During the index update, the Index Matcher service outputs the following information to the logs:

```
Refresh index for: 2d5832ad-8c8f-415f-a0b4-d12d69fabd60
Sync: 5->6, 0->0
Refresh index for: 2d5832ad-8c8f-415f-a0b4-d12d69fabd60 has finished
successfully
```

where:

- 2d5832ad-8c8f-415f-a0b4-d12d69fabd60 - List ID.
- 5->6- Information about downloading packets with descriptors (1 packet equals 10 descriptors) from the Faces database. Here 6 is the total number of packages that need to be downloaded from the database, and 5 is the current number of downloaded packages. Thus, the message 5->6 means that synchronization will continue and another packet will be downloaded.
- 0->0 - Information about deleting packets with descriptors (1 packet equals 10 descriptors) from the index in memory. The principle of operation is similar to downloading packages from the Faces database.

The speed of updating the index in memory depends on the size of the current index.

If this functionality is used, then it is not necessary and not recommended to perform frequent index rebuilds. Accordingly, it is recommended to increase the [planning routine period](#) (“[planning_period](#)” setting). However, adding new faces to the index in memory is slower than rebuilding the index, so it makes no sense to use this function if a very large number of faces have been added to the list. In this case, it is easier to rebuild the index again.

Unlinking faces from the list does not remove those faces from the index in memory. In this case, the descriptors are marked as unsearchable, so the index retains the storage space allocated to them.

See the sequence diagram for index refreshing in the “[Index refreshing diagram](#)” section.

4.3.4 Index caching

You can enable index caching to speed up the process of loading data into the memory of the Indexed Matcher service. Using caching enables you not to load the index into memory from the Storage, but to load it from the cached directory in case of an unexpected restart of the Indexed Matcher service.

Caching is enabled when specifying an intermediate directory for storing and loading indexes in the “[lim_matcher_cache](#)” setting of the Configurator service. By default, the directory is not specified, i.e. caching is disabled.

Intermediate directory must be located at local file system (using things like GlusterFS or NFS might cause bugs). Every time Indexed Matcher service reloads its indexes it tries to clean up cache directory by removing old generation of list indexes. Cache system has locking mechanism. In case of multiple instances of Indexed Matcher running on the same host and sharing the same directory for cache, locking will prevent downloading of the same indexes multiple times. It means, index storage will be hit exactly one time when data is being sent between Indexed Matcher services host and the storage.

5 Matching plugin for Python Matcher Proxy

Using the matching plugin, the Python Matcher Proxy service can redirect matching requests from the API service to either the Python Matcher service or the Indexed Matcher service. The principle of operation of the plugin and the description of the choice of the service in which the matching will be performed are described below.

The matching plugin is already integrated into the Python Matcher Proxy Docker container, you just need to enable it (see the installation manual).

5.1 Matching plugin description

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.

5.2 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 infinity. If there are several plugins, then the matching cost value will be calculated for each plugin. If the matching label is loaded in the Redis database, then a certain query complexity will be calculated and the matching plugin will be used. If the label is not loaded, then the Python Matcher service will be used.

5.3 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 enriched response with specified targets to the user.
- 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 to the user.

Several matching plugins can be used in LUNA PLATFORM. See the “Matching plugins” section of the LUNA PLATFORM administrator manuals for details.

6 Monitoring

Monitoring is implemented as sending data to the InfluxDB. Monitoring is enabled in the services by default.

It is also possible to use LUNA Dashboards (the “luna_index_module” directory) and Grafana Loki for LIM services. See detailed information about LUNA PLATFORM monitoring, LUNA Dashboards and Grafana Loki in the “Monitoring” section of the LUNA PLATFORM administrator manual.

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. Grafana Loki is a log aggregation system that enables you to flexibly work with LUNA PLATFORM logs in Grafana.

6.1 Data being sent

The types of monitoring events are different for each service. Below is a table showing all types of events for each service:

Service	Types of events
Index Manager	All HTTP requests, all failed HTTP requests, index building .
Indexer	All HTTP requests, all failed HTTP requests.
Indexed Matcher	All HTTP requests, all failed HTTP requests, index reloading , matching request (pass through Redis).

Every event is a point in the time series. The point is represented using the following data:

- Series name (requests or errors)
- Timestamp of the request start
- Tags
- Fields

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.

Saving data for **requests series** is triggered on every request. Each point contains data about the

corresponding request (execution time and etc.).

- Tags

Tag name	Description
service	Always “lim-manager”.
route	Concatenation of a request method and a request resource (GET:/version).
status_code	HTTP status code of response.

- Fields

Field name	Description
request_id	Request ID.
execution_time	Request execution time.

Saving data for **errors series** is triggered when a request fails. Each point contains error_code.

- Tags

Tag name	Description
service	“lim-indexer”, “lim-manager”, or “lim-matcher”.
route	Concatenation of a request method and a request resource (GET:/version).
status_code	HTTP status code of response.
error_code	LIM error code.

- Fields

Field name	Description
request_id	Request ID.

Saving data for **index processing** is started when an error occurs during index building.

- Tags

Tag name	Description
service	“lim-manager”, or “lim-matcher”.
socket_address	Service address in the format <host>:<port>.
stage	Always “build_index”.
label	Index matching label (list_id).
error_code	LIM error code (0 - request was completed successfully).

- Fields

Field name	Description
index_id	Index unique ID.
pending	Time spent in the internal queue (sec).
duration	Index processing (i.e. building / loading / dropping) time, in seconds.
generation	Index generation (unix timestamp).

Saving data for **index matching** is started when a matching is performed.

- Tags

Tag name	Description
service	Always “lim-matcher”.
socket_address	Service address in the format <host>:<port>.
label	Index matching label (list_id).
error_code	LIM error code (0 - request was completed successfully).

- Fields

Field name	Description
request_id	Request ID.
index_id	Index unique ID.
execution_time	Matching request execution time, in seconds.

7 Sequence diagrams

This section provides sequence diagrams for basic LIM operations.

7.1 Index creation diagram

The index is built after creating a task to build it. There are two types of creating index building tasks - [one-time](#) and [background](#).

Below is a sequence diagram for both types of task creation.

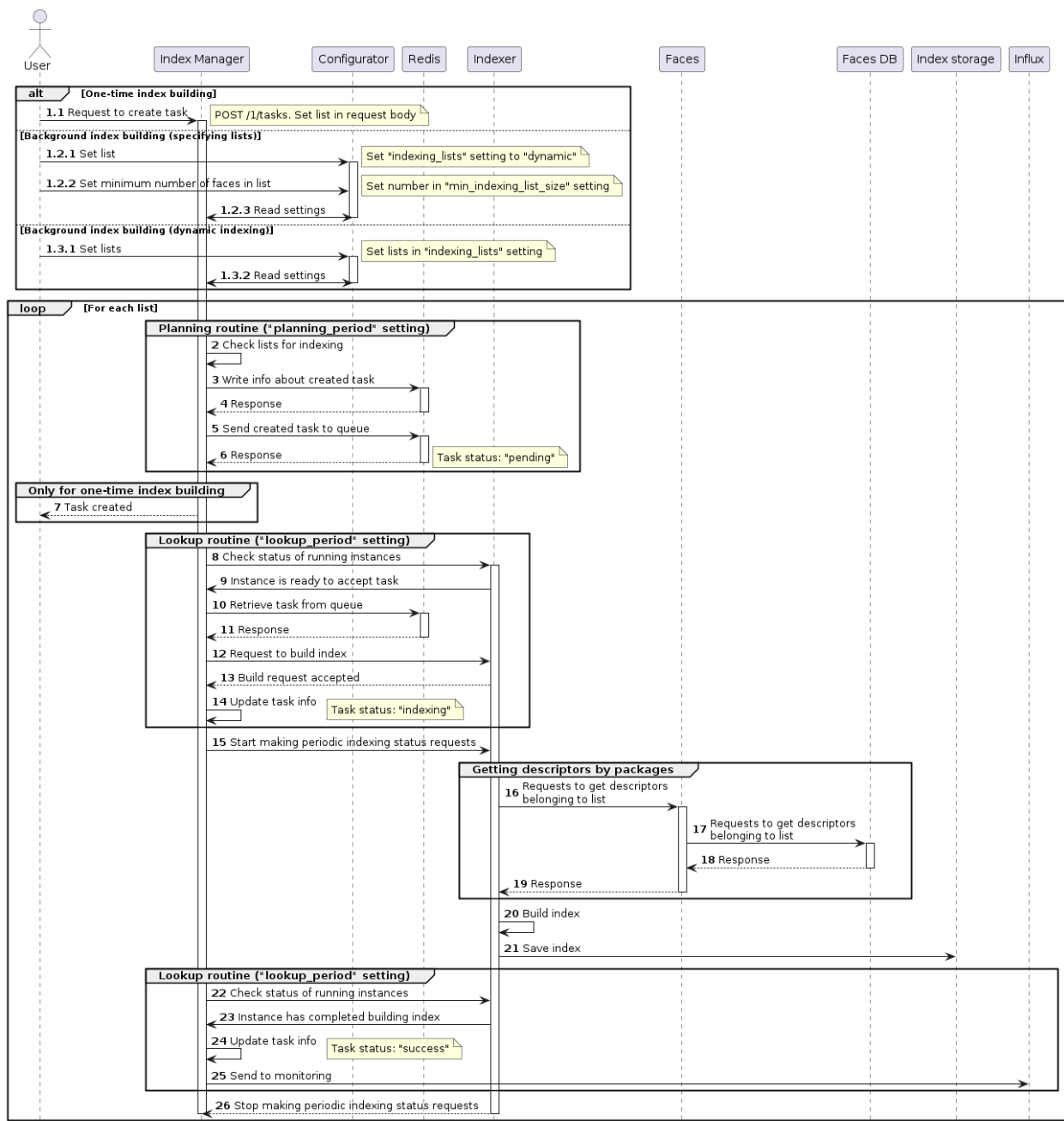


Figure 3: Index creation diagram

- **(1.1)** Sending “create task” request to **one-time** index building.
- **(1.2.1)** Enabling **dynamic indexing** of lists by setting the “dynamic” value for the “indexing_lists” setting.
- **(1.2.2)** Setting the minimum number of faces in the list by which the index will be built (default 50,000) in the “min_indexing_list_size” setting.

- (1.2.3) Reading the settings values from the Configurator service.
- **(1.3.1)** Setting a set of lists in the “indexing_lists” setting.
- (1.3.2) Reading the setting value from the Configurator service.
- (2) Beginning of the [planning routine](#) (regulated by setting “[planning_period](#)”). Checking which sets of lists should be indexed.
- (3) Saving information about the task being created in the Redis storage.
- (4) Response that the task was placed in storage.
- (5) Sending the task to an internal queue.
- (6) Response that the task was queued. At this stage, the status of the task is set to “pending”.
- (7) Response that the task was created. Only for [one-time](#) index building (see (1.1)).
- (8) Beginning of [lookup routine](#) (regulated by setting “[lookup_period](#)”). Checking the status of running Indexer instances.
- (9) Response that the Indexer instance is ready to start index creation.
- (10) Retrieving the task from the internal queue.
- (11) Response that the task was retrieved.
- (12) Request to build a index on the created task.
- (13) Response that the request to build the index on the created task was accepted.
- (14) Updating task information. At this stage, the status of the task is set to “indexing”.
- (15) Start making periodic indexing status requests.
- (16) Making requests to the Faces service to get descriptors belonging to the list.
- (17) Redirecting requests to the Faces database.
- (18) Uploading data from the Faces database.
- (19) Redirecting the uploaded data to the Faces service.
- (20) Index building process.
- (21) Saving the index to the Index storage.
- (22) Continuation of [lookup routine](#). Checking the status of running Indexer instances.
- (23) Response that the instance has finished building the index.
- (24) Updating task information. At this stage, the status of the task is set to “success”.
- (25) Sending to [Influx database](#).
- (26) End making of periodic indexing status requests.

You can find out the current status of the task using the request “[get tasks](#)” after it has been created.

7.2 Diagram of initial index loading into memory

Before the matching starts, the index is loaded into the Indexed Matcher memory service.

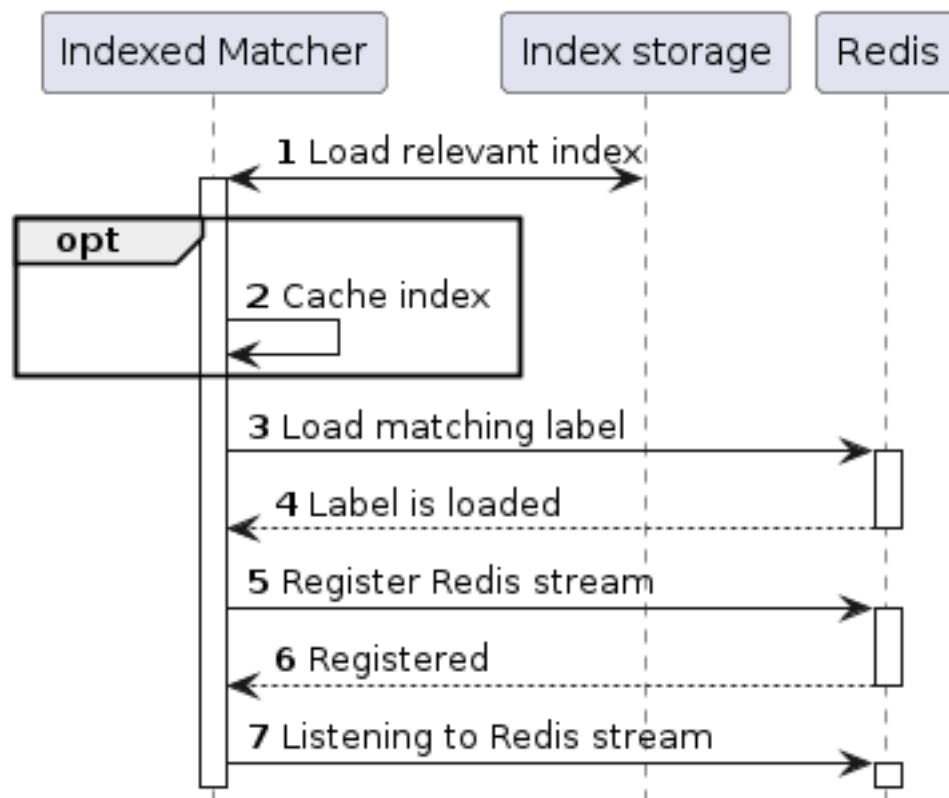


Figure 4: Diagram of initial index loading into memory

- (1) Loading the most relevant index from the Storage into the memory of the Indexed Matcher service.
- (2) Caching the index into an intermediate directory. The directory is specified in the “lim_matcher_cache” setting of the Indexed Matcher service. By default, caching is disabled. See the “[Index caching](#)” section for details.
- (3) Loading matching label in Redis. The label name is the same as the list ID.
- (4) Response that the label is loaded.
- (5) Register a [Redis stream](#) as a message handler for the appropriate matching label. At this point, the stream is given the name of the matching label (i.e. the name of the list id).
- (6) Response that the stream is registered.

- (7) Waiting for matching request that contains a matching label with exactly the same name as the one loaded in Redis (3) and for which the stream was registered (5).

7.3 Diagram of matching descriptors

This diagram assumes that the index is already loaded into the memory of the Indexed Matcher service (see “Diagram of initial index loading into memory”).

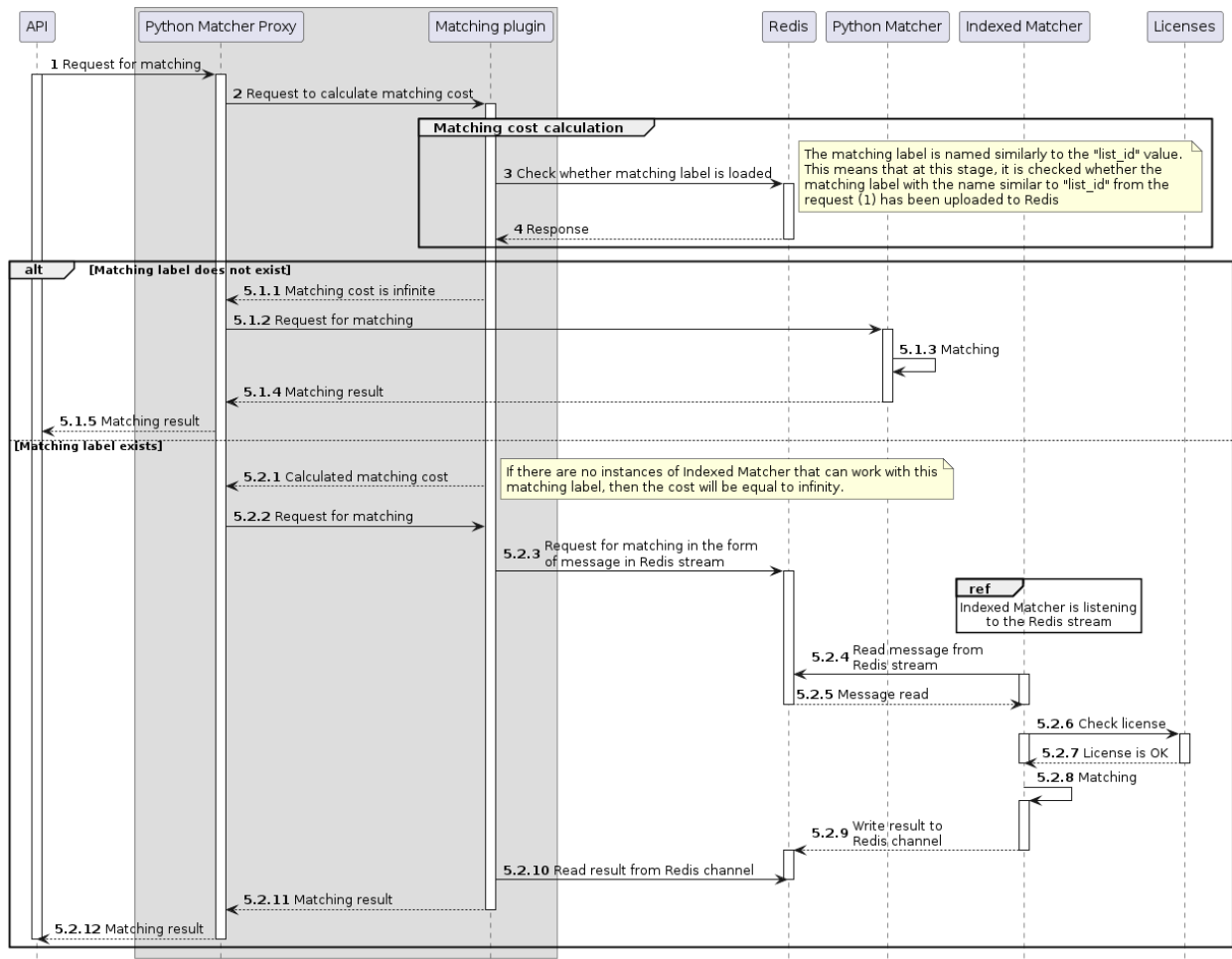


Figure 5: Diagram of matching descriptors

- (1) Sending the matching request.
- (2) Request to calculate the matching cost of request.
- (3) Checking whether the matching label is loaded to Redis.
- (4) Response.
- (5.1.1) If the matching label exists, then the matching cost of the request is considered infinite and the matching plugin returns the corresponding information to the Python Matcher Proxy

- service.
- (5.1.2) Request for matching to the Python Matcher service.
 - (5.1.3) Matching process in the Python Matcher service.
 - (5.1.4) Response about matching result to the Python Matcher Proxy.
 - (5.1.5) Response about matching result to the API.
 - **(5.2.1)** If the matching label not exists, then the certain matching cost of the request is calculated and the response is returned to the Python Matcher Proxy service.
 - (5.2.2) Sending the matching request to the matching plugin.
 - (5.2.3) Converting the request and sending it as a message to the [Redis stream](#).
 - (5.2.4) The Indexed Matcher service is listening to the Redis stream (see [“Diagram of initial index loading into memory”](#)). As soon as the matching request appears, which contains the matching label with exactly the same name as the one that was previously uploaded to Redis and for which the stream was registered, then Indexed Matcher reads the message for matching.
 - (5.2.5) Response that the message was read.
 - (5.2.6) Performing license verification.
 - (5.2.7) Response about the license status (this diagram does not reflect the case of invalid license).
 - (5.2.8) Performing the matching process.
 - (5.2.9) Recording the matching result in the [Redis channel](#).
 - (5.2.10) The matching plugin reads the matching result from the Redis channel.
 - (5.2.11) Sending the matching result to the Python Matcher Proxy.
 - (5.2.12) Sending the matching result to the API.

7.4 Index reloading diagram

This diagram reflects only the sequence of work in the case when an index with a newer version of the matching label appears in the Index storage. If the index disappears from Index storage, then after checking the status of the index, the Indexed Matcher service deletes it from memory. This functionality is enabled using the “enabled = 1” setting (enabled by default) of the “LIM_MATCHER_REFRESH” section of the Indexed Matcher service settings.

It is recommended to read the section [“Index reloading”](#).

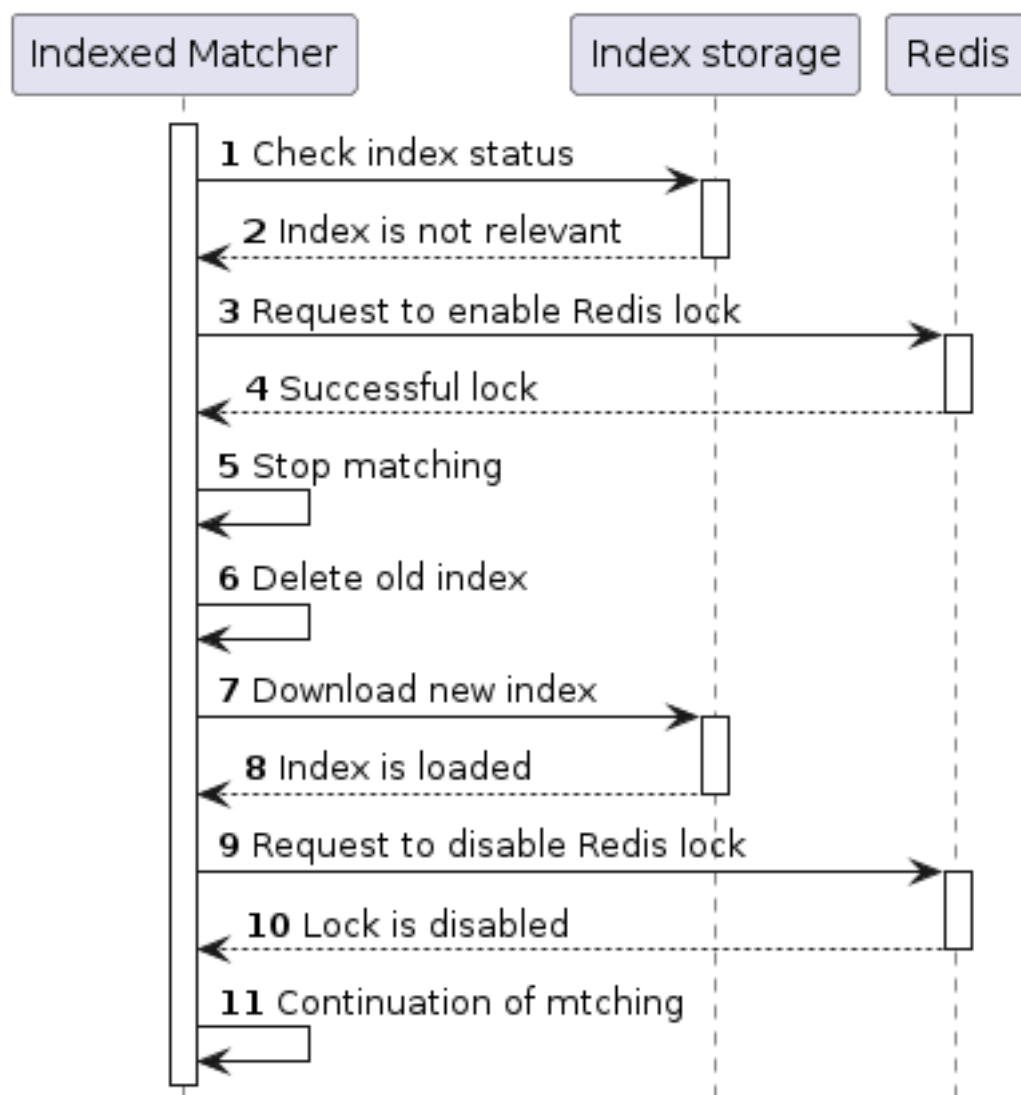


Figure 6: Index reloading diagram

- (1) Checking whether the current index in the Index storage was changed. If the Index storage has a more recent index at the time of saving for the same matching label and with the desired version of the descriptor, then this will be taken into account as the fact that the current index is not relevant.
- (2) Response that the index is not relevant.
- (3) Request to enable Redis lock.
- (4) Response about successful Redis lock. If it is currently impossible to perform the lock, then Indexed Matcher will perform lock attempts until the next attempt succeeds.
- (5) If the lock is successful, then stop matching using the current index.

- (6) Deleting the old index from the Indexed Matcher memory.
- (7) Loading a new index from the Index storage into the Indexed Matcher service memory.
- (8) Response that the index is loaded into memory.
- (9) Request to disable Redis lock.
- (10) Response that the lock is disabled.
- (11) Continuation of the matching.

7.5 Index refreshing diagram

Below is a sequence diagram for the process of refreshing an index in the memory of the Indexed Matcher service.

This information is described for an index that is already loaded into the memory of the Indexed Matcher service. The index in the service's memory and the index in the storage may differ. It is recommended to read the section [“Refreshing index in memory”](#).

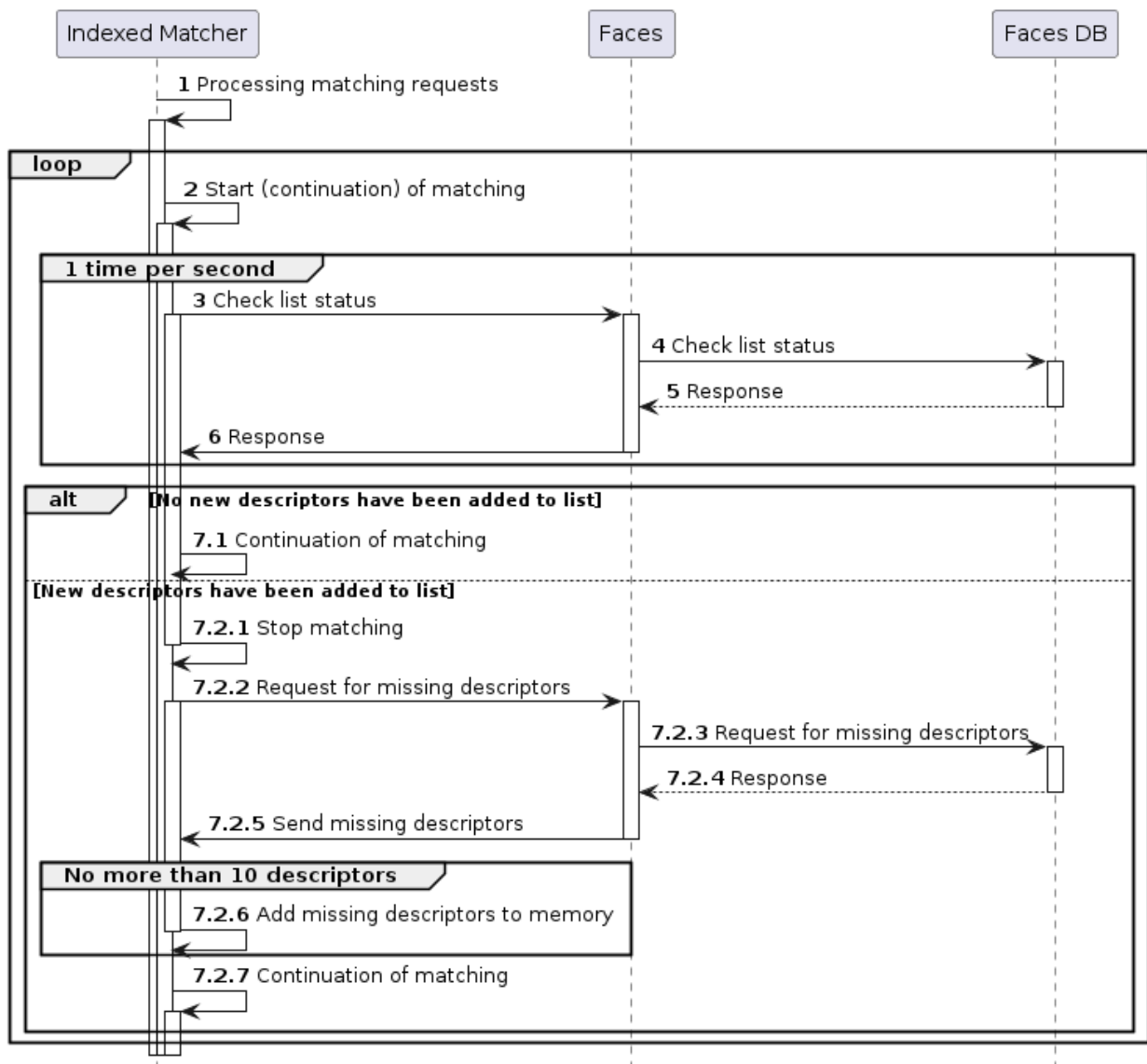


Figure 7: Index refreshing diagram

- (1) Receiving processing of matching requests.
- (2) Starting of matching.
- (3) Request to check the state of the list to Faces (request is performed every second).
- (4) Redirecting the request to check the state of the list to the Faces database.
- (5) Response.
- (6) Response.
 - **(7.1)** If no new descriptors was added to the list, then the matching will continue until the next check of the list status (3).

- **(7.2.1)** If new descriptors were added to the list, the matchings stops.
- (7.2.2) Request missing descriptors to the Faces service.
- (7.2.3) Redirecting the request to the Faces database.
- (7.2.4) Response containing missing descriptors.
- (7.2.5) Response containing missing descriptors.
- (7.2.6) Adding missing descriptors to the index located in the memory of the Indexed Matcher service (no more than 10 descriptors at a time).
- (7.2.7) Continuation of the matching until the next check of the state of the list (3).

8 Use Redis Sentinel

Redis Sentinel is a component in the High Availability Management System for the Redis database. It is used to detect failures in Redis master nodes and automatically reconfigure the system to ensure continuous operation.

To use Redis Sentinel, you need to fill in the “sentinels” section from the “[LIM_MANAGER_DB](#)” and “[LIM_MATCHER_DB](#)” sections.

Also, the use of Redis Sentinel can be specified in the “[LUNA_INDEXED_LIST_PLUGIN](#)” section, which is responsible for connecting the [matching plugin](#) with Redis when [calculating matching cost](#) of the request.

Example of filling in the “[LUNA_INDEXED_LIST_PLUGIN](#)” section:

```
LUNA_INDEXED_LIST_PLUGIN = {"REDIS_URL": "redis+sentinel://localhost:26379,localhost:26378/indexed_matcher?username=master_username&password=master_password", "REQUEST_TIMEOUT": 60}
```

Here:

- “redis+sentinel://” - Prefix indicating the use of the Redis protocol and Redis Sentinel for connection.
- “localhost:26379,localhost:26378” - List of Sentinel addresses and ports that the service will use to detect the status of Redis master nodes and coordinate actions in case of failures.
- “/indexed_matcher” - Path to a specific Redis database.
- “?username=master_username&password=master_password” - Auth data.

9 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.

9.1 Indexer service errors

9.1.1 Code 26100 returned

Error Message:

Internal server error

Error Source:

Indexer service errors

Error Description:

An unknown error has occurred.

9.1.2 Code 26101 returned

Error Message:

Indexer busy

Error Source:

Indexer service errors

Error Description:

The Indexer service is currently unable to process the request.

Wait for the index to finish processing, or start another instance of the Indexer service.

9.1.3 Code 26103 returned

Error Message:

Build failed

Error Source:

Indexer service errors

Error Description:

An error occurred while creating the index. Try again.

9.1.4 Code 26104 returned

Error Message:

Build process failed. Build process died

Error Source:

Indexer service errors

Error Description:

Index build failed.

This error could have occurred due to process termination (OOM killer).

9.1.5 Code 26105 returned

Error Message:

Build process failed. List is empty

Error Source:

Indexer service errors

Error Description:

Building the index failed due to the lack of faces associated with the list.

Make sure the faces are attached to the list.

9.1.6 Code 26106 returned

Error Message:

Build process cancelled

Error Source:

Indexer service errors

Error Description:

Index building was canceled with the “/stop” request to the Indexer service.

9.1.7 Code 26107 returned

Error Message:

Build process failed. List not found

Error Source:

Indexer service errors

Error Description:

Index build failed due to missing list.

Make sure the list exists.

9.1.8 Code 26108 returned

Error Message:

Build process failed. Index with specified ID already exists

Error Source:

Indexer service errors

Error Description:

Index build failed because the index with the specified ID already exists.

9.1.9 Code 26109 returned

Error Message:

Build process failed. It's probably due to running out of memory and the OS was triggering the OOM killer.

Error Source:

Indexer service errors

Error Description:

Index build failed. It's probably due to running out of memory and the OS was triggering the OOM killer.

9.2 Index Manager service errors

9.2.1 Code 26201 returned

Error Message:

Task duplicate. Indexing task already exists

Error Source:

Index Manager service errors

Error Description:

An error occurred while trying to create the task. The task with this ID already exists (the task ID is equal to the list ID).

9.2.2 Code 26202 returned

Error Message:

Index duplicate. Index for the most recent content version already exists

Error Source:

Index Manager service errors

Error Description:

An error occurred while trying to create the index. The most relevant index already exists.

9.2.3 Code 26203 returned

Error Message:

Internal server error. Indexer was restarted for internal reasons

Error Source:

Index Manager service errors

Error Description:

The Indexer service restarted for unknown reasons.

9.2.4 Code 26204 returned

Error Message:

Object not found. Index with id "{}" not found in the storage

Error Source:

Index Manager service errors

Error Description:

The index with the specified ID was not found in the storage. Verify that the specified index exists.

You can check the list of existing indexes using the query [“get indexes”](#).

The link leads to the latest version of the documentation. Make sure you read the documentation for your current version of LIM.

9.3 Index Matcher service errors

9.3.1 Code 26301 returned

Error Message:

Bad/incomplete input data. Failed to validate matching request: {value}

Error Source:

Index Matcher service errors

Error Description:

Matching request processing failed due to invalid data specified.

9.3.2 Code 26302 returned

Error Message:

Bad/incomplete input data. Failed to load descriptor bytes {value}

Error Source:

Index Matcher service errors

Error Description:

Error loading descriptor.

The error may occur due to the wrong descriptor format.

9.3.3 Code 26303 returned

Error Message:

Internal server error. Failed to search descriptor {value}

Error Source:

Index Matcher service errors

Error Description:

Internal error while searching for descriptor.

9.3.4 Code 26304 returned

Error Message:

Index not found. Index for label {value} not found

Error Source:

Index Matcher service errors

Error Description:

Index was not found for the specified matching label.

Verify that the index with the ID equal to the matching label has been created.

You can check the list of existing indexes using the “[get indexes](#)” request to the Index Manager service.

The link leads to the latest version of the documentation. Make sure you read the documentation for your current version of LIM.

9.3.5 Code 26305 returned

Error Message:

Descriptor version mismatch. Descriptor of version {value} cannot be searched in index of version {value}

Error Source:

Index Matcher service errors

Error Description:

Version mismatch of the matched descriptors was detected.

Make sure the descriptors versions match. You can upgrade descriptors versions using the Additional extraction task (see “Running the Additional extraction task” section of the LUNA PLATFORM 5 administrator manual).

The default descriptors version is contained in the “DEFAULT_FACE_DESCRIPTOR_VERSION” setting in the Configurator service.

9.3.6 Code 26306 returned

Error Message:

Index processing internal error. Skip load index for label {value} due to index storage damage

Error Source:

Index Matcher service errors

Error Description:

Internal index processing error. The index load for the specified matching label was skipped due to index storage corruption.

9.3.7 Code 26307 returned**Error Message:**

Index processing internal error. Skip load index for label {value} due to insufficient memory

Error Source:

Index Matcher service errors

Error Description:

An internal index processing error. The index load for the specified matching label was skipped due to insufficient free space.

Make sure there is enough free space.

The system can be purged with the `docker system prune` command. By default, stopped containers, layers not related to the images in use, volumes and networks not related to running containers will be deleted.

9.3.8 Code 26308 returned**Error Message:**

Index processing internal error. Skip load index for label {value} due to index corruption

Error Source:

Index Matcher service errors

Error Description:

Internal index processing error. The index load for the specified matching label was skipped due to a corrupted index.

10 Configuration parameters of services

10.1 Index Manager service configuration

The section describes the Index Manager service parameters.

You can configure the service using the Configurator service.

10.1.1 LIM_MANAGER_LOGGER section

This section sets the logging settings for the logging.

10.1.1.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

10.1.1.2 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`.

10.1.1.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

10.1.1.4 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`.

10.1.1.5 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"
```

10.1.1.6 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

10.1.1.7 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.

10.1.1.8 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`.

10.1.2 LIM_MANAGER_INDEXING section

This section sets the indexing settings set by the Index Manager service.

10.1.2.1 indexer_origins

The parameter sets the list of addresses of running instances of the Indexer service.

Setting format: `array > string`.

Default value: `http://127.0.0.1:5180`.

10.1.2.2 planning_period

The parameter sets the period of the [planning routine](#), which checks the sets of lists that need to be indexed.

Setting format: `integer (seconds)`.

Default value: `600`.

10.1.2.3 lookup_period

The parameter sets the period of the [search routine](#), which checks the statuses of all running Indexer instances.

Setting format: `integer (seconds)`.

Default value: `5`.

10.1.2.4 face_lists > min_indexing_list_size

The parameter sets the minimum number of faces in the lists for the list to be indexed.

The parameter is used only when using the parameter “[indexing_lists](#)” with the value “dynamic”.

Setting format: `integer`.

Default value: `50000`.

10.1.2.5 `face_lists > indexing_lists`

The parameter sets a set of lists for indexing.

You can either specify lists explicitly or specify the value “dynamic”. In the latter case, all lists will be processed whose number of face descriptors exceeds the number specified in the “`min_indexing_list_size`” parameter.

Setting format: `array > string`.

Default value: `dynamic`.

10.1.2.6 `ef_construction`

The parameter sets a limit on the number of nearest neighbors considered when constructing the index.

Higher values result in a more accurate graph, but it takes longer to build.

It is recommended to change the parameter together with the “`ef_search`” parameter of the Indexed Matcher service.

Setting format: `integer`.

Default value: `1600`.

10.1.3 `LIM_MANAGER_HTTP_SETTINGS` section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

10.1.3.1 `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`.

10.1.3.2 `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`.

10.1.3.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

10.1.3.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

10.1.4 LIM_MANAGER_DB section

In this section, the settings for connecting to the database of the Index Manager service are set.

10.1.4.1 db_user

The parameter sets the name of the Redis database user.

Setting format: string.

Default value is not set.

10.1.4.2 db_password

This parameter sets the password of the Redis database user.

Setting format: string.

Default value is not set.

10.1.4.3 db_host

The parameter sets the host of the Redis database.

Setting format: string.

Default value: 127.0.0.1.

10.1.4.4 db_port

The parameter sets the port of the Redis database.

Setting format: string.

Default value: 6379.

10.1.4.5 `db_settings > connection_pool_size`

This parameter sets the size of the pool of connections to the Redis database.

Setting format: `string`.

Default value: 100.

10.1.4.6 `db_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.

10.1.4.7 `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: `index_manager`.

10.1.4.8 `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: `[]`.

10.1.4.9 `sentinel > user`

The parameter sets the user name of the Sentinel server.

Setting format: `string`.

Default value: Not specified.

10.1.4.10 `sentinel > password`

The parameter sets the password of the Sentinel server user.

Setting format: `string`.

Default value: Not specified.

10.1.5 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

10.1.5.1 send_data_for_monitoring

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: integer.

Default value: 1.

10.1.5.2 use_ssl

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: integer.

Default value: 0.

10.1.5.3 organization

The parameter sets InfluxDB workspace.

Setting format: string.

Default value: luna.

10.1.5.4 token

The parameter sets InfluxDB authentication token.

Setting format: string.

10.1.5.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: string.

Default value: luna_monitoring.

10.1.5.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: string.

Default value: 127.0.0.1.

10.1.5.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: 8086.

10.1.5.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer` (seconds).

Default value: 1.

10.1.6 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

10.1.6.1 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`.

10.1.6.2 api_version

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: `integer`.

Default value: 3.

10.1.7 LUNA_FACES_TIMEOUTS section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

10.1.7.1 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.

10.1.7.2 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.

10.1.7.3 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.

10.1.7.4 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.

10.1.8 Other

10.1.8.1 index_storage_type

The parameter sets the type of index storage. Currently, only the “LOCAL” option is available.

The value “LOCAL” means that the indexes will be stored in the directory specified in the “[index_storage_local](#)” parameter.

Setting format: string.

Default value: LOCAL.

10.1.8.2 `index_storage_local`

The parameter sets the directory for storing indexes with “LOCAL” [storage type](#).

Setting format: `string`.

Default value: `./local_storage`.

10.1.8.3 `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`.

10.1.8.4 `lim_manager_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`.

10.1.8.5 `default_face_descriptor_version`

The parameter sets the version of the face descriptor to use.

Setting format: `string`.

Default value: `59`.

10.2 Indexed Matcher service configuration

The section describes the Indexed Matcher service parameters.

You can configure the service using the Configurator service.

10.2.1 LIM_MATCHING section

This section sets the settings for indexed matching.

10.2.1.1 ef_search

The parameter sets a limit on the considered number of nearest neighbors when searching for an index.

Higher values result in a more accurate but slower search.

It is recommended to change the parameter together with the “[ef_construction](#)” parameter of the Indexed Manager service.

Setting format: integer.

Default value: 1600.

10.2.2 LIM_MATCHER_REFRESH section

In this section, the settings for [index refreshing](#) in the Indexed Matcher service memory are set.

10.2.2.1 enabled

The parameter allows you to enable [index refreshing](#).

Setting format: integer.

Default value: 1.

10.2.3 LIM_MATCHER_LOGGER section

This section sets the logging settings for the logging.

10.2.3.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: string.

Default value: INFO.

10.2.3.2 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`.

10.2.3.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

10.2.3.4 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`.

10.2.3.5 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"
```

10.2.3.6 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`

10.2.3.7 `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`.

10.2.3.8 `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`.

10.2.4 `LIM_MATCHER_HTTP_SETTINGS` section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

10.2.4.1 `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.

10.2.4.2 `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.

10.2.4.3 `request_max_size`

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

10.2.4.4 `keep_alive_timeout`

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

10.2.5 `LIM_MATCHER_DB` section

In this section, the settings for connecting to the database of the Index Matcher service are set.

10.2.5.1 `db_user`

The parameter sets the name of the Redis database user.

Setting format: string.

Default value is not set.

10.2.5.2 `db_password`

This parameter sets the password of the Redis database user.

Setting format: string.

Default value is not set.

10.2.5.3 db_host

The parameter sets the host of the Redis database.

Setting format: `string`.

Default value: `127.0.0.1`.

10.2.5.4 db_port

The parameter sets the port of the Redis database.

Setting format: `string`.

Default value: `6379`.

10.2.5.5 db_settings > connection_pool_size

This parameter sets the size of the pool of connections to the Redis database.

Setting format: `string`.

Default value: `100`.

10.2.5.6 db_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`.

10.2.5.7 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: `indexed_matcher`.

10.2.5.8 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: `[]`.

10.2.5.9 `sentinel > user`

The parameter sets the user name of the Sentinel server.

Setting format: `string`.

Default value: Not specified.

10.2.5.10 `sentinel > password`

The parameter sets the password of the Sentinel server user.

Setting format: `string`.

Default value: Not specified.

10.2.6 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

10.2.6.1 `send_data_for_monitoring`

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: 1.

10.2.6.2 `use_ssl`

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: 0.

10.2.6.3 `organization`

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

10.2.6.4 `token`

The parameter sets InfluxDB authentication token.

Setting format: `string`.

10.2.6.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

10.2.6.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

10.2.6.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

10.2.6.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer` (seconds).

Default value: `1`.

10.2.7 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

10.2.7.1 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`.

10.2.7.2 `api_version`

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: integer.

Default value: 3.

10.2.8 `LUNA_FACES_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

10.2.8.1 `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.

10.2.8.2 `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.

10.2.8.3 `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.

10.2.8.4 `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.

10.2.9 LUNA_LICENSES_ADDRESS section

This section sets the connection settings for the Licenses service.

10.2.9.1 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`.

10.2.9.2 api_version

The parameter sets the version of the Licenses service. The available API version is “1”.

Setting format: `integer`.

Default value: 1.

10.2.10 Other

10.2.10.1 lim_matcher_cache

The parameter sets the path to the directory with the cache.

For more information about caching, see “[Index caching](#)”.

To disable caching, leave the field empty.

Setting format: `string`.

Default value: not set.

10.2.10.2 index_storage_type

The parameter sets the type of index storage. Currently, only the “LOCAL” option is available.

The value “LOCAL” means that the indexes will be stored in the directory specified in the “[index_storage_local](#)” parameter.

Setting format: `string`.

Default value: LOCAL.

10.2.10.3 `index_storage_local`

The parameter sets the directory for storing indexes with “LOCAL” [storage type](#).

Setting format: string.

Default value: `./local_storage`.

10.2.10.4 `lim_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.

10.2.10.5 `default_face_descriptor_version`

The parameter sets the version of the face descriptor to use.

Setting format: string.

Default value: 59.

10.3 Indexer service configuration

The section describes the Indexer service parameters.

You can configure the service using the Configurator service.

10.3.1 LIM_INDEXER_LOGGER section

This section sets the logging settings for the logging.

10.3.1.1 log_level

The parameter sets the level of debug printing, by priority: “ERROR”, “WARNING”, “INFO”, “DEBUG”.

Setting format: `string`.

Default value: `INFO`.

10.3.1.2 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`.

10.3.1.3 log_to_stdout

The parameter enables you to send logs to standard output (stdout).

Setting format: `boolean`.

Default value: `true`

10.3.1.4 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`.

10.3.1.5 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"
```

10.3.1.6 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

10.3.1.7 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.

10.3.1.8 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`.

10.3.2 INFLUX_MONITORING section

In this section, settings for monitoring are set.

For more information about monitoring, see “[Monitoring](#)” section.

10.3.2.1 `send_data_for_monitoring`

The parameter enables you to enable or disable sending monitoring data to InfluxDB.

Setting format: `integer`.

Default value: `1`.

10.3.2.2 `use_ssl`

The parameter enables you to use HTTPS to connect to InfluxDB.

Setting format: `integer`.

Default value: `0`.

10.3.2.3 `organization`

The parameter sets InfluxDB workspace.

Setting format: `string`.

Default value: `luna`.

10.3.2.4 `token`

The parameter sets InfluxDB authentication token.

Setting format: `string`.

10.3.2.5 bucket

The parameter sets InfluxDB bucket name.

Setting format: `string`.

Default value: `luna_monitoring`.

10.3.2.6 host

The parameter sets IP address of server with InfluxDB.

Setting format: `string`.

Default value: `127.0.0.1`.

10.3.2.7 port

The parameter sets InfluxDB port.

Setting format: `string`.

Default value: `8086`.

10.3.2.8 flushing_period

The parameter sets frequency of sending monitoring data to InfluxDB.

Setting format: `integer` (seconds).

Default value: `1`.

10.3.3 LUNA_FACES_ADDRESS section

This section sets the connection settings for the Faces service.

10.3.3.1 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`.

10.3.3.2 `api_version`

The parameter sets the version of the Faces service. The available API version is “3”.

Setting format: integer.

Default value: 3.

10.3.4 `LUNA_FACES_TIMEOUTS` section

This section sets time intervals for managing the timeouts of HTTP requests that are sent to the Faces service.

10.3.4.1 `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.

10.3.4.2 `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.

10.3.4.3 `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.

10.3.4.4 `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.

10.3.5 LIM_INDEXER_HTTP_SETTINGS section

This section contains parameters responsible for process HTTP connections. More detail see [here](#).

10.3.5.1 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.

10.3.5.2 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.

10.3.5.3 request_max_size

The parameter sets the maximum size of the request.

Setting format: integer (bytes).

Default value: 1073741824.

10.3.5.4 keep_alive_timeout

The parameter sets the timeout for maintaining HTTP activity.

Setting format: integer (seconds).

Default value: 15.

10.3.6 Other

10.3.6.1 index_storage_type

The parameter sets the type of index storage. Currently, only the “LOCAL” option is available.

The value “LOCAL” means that the indexes will be stored in the directory specified in the “[index_storage_local](#)” parameter.

Setting format: string.

Default value: LOCAL.

10.3.6.2 `index_storage_local`

The parameter sets the directory for storing indexes with “LOCAL” [storage type](#).

Setting format: string.

Default value: `./local_storage`.

10.3.6.3 `lim_indexer_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.

10.3.6.4 `default_face_descriptor_version`

The parameter sets the version of the face descriptor to use.

Setting format: string.

Default value: 59.

10.4 Matching plugin configuration

10.4.1 LUNA_INDEXED_LIST_PLUGIN section

This section is responsible for connecting the [matching plugin](#) with Redis when [calculating matching cost](#).

It is possible to specify the Redis Sentinel address. See [“Use Redis sentinel”](#).

10.4.1.1 `redis_url`

The parameter sets the Redis address.

Setting format: `string`.

Default value: `redis://localhost:6379`.

10.4.1.2 `request_timeout`

The parameter sets the connection timeout to Redis.

Setting format: `integer`.

Default value: `60`.