



VisionLabs LUNA PLATFORM 5

Installation manual

v.5.56.0

Contents

Default ports for services	6
Configuration names for services	7
System requirements	8
Processors	8
CPU	8
GPU	8
Third-party applications	9
Introduction	10
1 Before launch	11
1.1 Distribution unpacking	12
1.2 Symbolic link creation	12
1.3 Changing group and owner for directories	13
1.4 SELinux and Firewall	13
1.5 License activation	13
1.5.1 Actions from License activation manual	14
1.5.2 Specify HASP license settings	14
1.5.3 Specify Guardant license settings	15
1.6 Login to registry	16
1.7 Docker installation	16
1.8 Choose logging method	17
1.8.1 Logging to stdout	17
1.8.2 Logging to file	17
1.9 Calculations using GPU	18
2 Services launch	19
2.1 Monitoring configuration	21
2.1.1 InfluxDB OSS 2	21
2.2 Run third-party services	22
2.2.1 PostgreSQL	22
2.2.2 Redis	23
2.3 Configurator	24
2.3.1 Optional services usage	24
2.3.2 Configurator DB tables creation	24
2.3.3 Run Configurator container	25

2.4	Image Store	26
2.4.1	Image Store container launch	26
2.4.2	Buckets creation	26
2.5	Accounts	29
2.5.1	Accounts DB creation	29
2.5.2	Accounts container launch	29
2.6	Licenses	30
2.6.1	Specify license settings using Configurator	30
2.6.2	Licenses container launch	31
2.7	Faces	32
2.7.1	Faces DB tables creation	32
2.7.2	Faces container launch	32
2.8	Events	33
2.8.1	Events DB tables creation	33
2.8.2	Events container launch	33
2.9	Python Matcher services	34
2.9.1	Use Python Matcher without Python Matcher Proxy	34
2.9.2	Python Matcher container launch	34
2.10	Remote SDK	35
2.10.1	Remote SDK container launch	35
2.11	Handlers	40
2.11.1	Handlers DB tables creation	40
2.11.2	Handlers container launch	40
2.12	Tasks	41
2.12.1	Tasks DB tables creation	41
2.12.2	Tasks and Tasks Worker containers launch	41
2.13	Sender	43
2.13.1	Sender container launch	43
2.14	API	44
2.14.1	API container launch	44
2.14.2	Account creation using API service	44
2.14.3	Create GC task schedule	46
2.15	Admin	47
2.15.1	Admin container launch	47
2.16	Backport 3	48
2.16.1	Backport 3 bucket creation	48
2.16.2	Backport 3 DB tables creation	48
2.16.3	Backport 3 container launch	48
2.16.4	User Interface 3	50

2.17	Backport 4	51
2.17.1	Backport 4 container launch	51
2.17.2	User Interface 4	52
2.18	Lambda	53
2.18.1	Prepare Docker registry	53
2.18.2	Lambda DB tables creation	54
2.18.3	Lambda container launch	54
3	Additional information	55
3.1	Monitoring and logs visualization using Grafana	56
3.1.1	LUNA Dashboards	56
3.1.2	Grafana Loki	56
3.2	Docker commands	58
3.2.1	Show containers	58
3.2.2	Copy files to container	58
3.2.3	Enter container	58
3.2.4	Images names	58
3.2.5	Delete image	58
3.2.6	Stop container	59
3.2.7	Delete container	59
3.3	Launching parameters description	61
3.3.1	Launching services parameters	61
3.3.2	Creating DB parameters	64
3.4	Ways to change service settings	65
3.5	Logging to server	67
3.5.1	Create logs directory	67
3.5.2	Logging activation	67
3.5.3	Mounting directories with logs when starting services	68
3.6	Docker log rotation	70
3.7	Set custom InfluxDB settings	71
3.8	Use Python Matcher with Python Matcher Proxy	73
3.8.1	Python Matcher proxy container launch	73
3.9	System scaling	74
3.9.1	Nginx	75
3.9.2	Launching several containers	76
3.10	Improve performance	78
3.11	External DB	80
3.11.1	PostgreSQL user creation	80
3.11.2	Configurator DB creation	80
3.11.3	Accounts DB creation	81

3.11.4	Handlers DB creation	81
3.11.5	Backport 3 DB creation	82
3.11.6	Faces DB creation	82
3.11.7	Create VLMatch function for Faces DB	83
3.11.8	Events DB creation	85
3.11.9	Create VLMatch function for Events DB	85
3.11.10	PostGIS installation for Events	87
3.11.11	Tasks DB creation	87
3.12	VLMatch library compilation for Oracle	89

Default ports for services

Table 1: Default ports of services

Service name	Port
LUNA PLATFORM API	5000
LUNA PLATFORM Admin	5010
LUNA PLATFORM Image Store	5020
LUNA PLATFORM Faces	5030
LUNA PLATFORM Events	5040
LUNA PLATFORM Tasks	5050
LUNA PLATFORM Tasks Worker	5051
LUNA PLATFORM Configurator	5070
LUNA PLATFORM Sender	5080
LUNA PLATFORM Handlers	5090
LUNA PLATFORM Python Matcher	5100
LUNA PLATFORM Licenses	5120
LUNA PLATFORM Backport 4	5130
LUNA PLATFORM Backport 3	5140
LUNA PLATFORM Accounts	5170
LUNA PLATFORM Lambda	5210
LUNA PLATFORM Remote SDK	5220
LUNA PLATFORM 3 User Interface	4100
LUNA PLATFORM 4 User Interface	4200
Oracle DB	1521
PostgreSQL	5432
Redis DB	6379
InfluxDB	8086
Grafana	3000

Configuration names for services

The table below includes the service names in the Configurator service. Use these parameters to configure your services.

Table 2: Service names in the Configurator service in the “Service name” field

Service	Service name in Configurator
API	luna-api
Licenses	luna-licenses
Faces	luna-faces
Image Store	luna-image-store
Accounts	luna-accounts
Tasks	luna-tasks
Events	luna-events
Sender	luna-sender
Admin	luna-admin
Handlers	luna-handlers
Lambda	luna-lambda
Python Matcher	luna-python-matcher
Backport 3	luna-backport3
Backport 4	luna-backport4

Settings for the Configurator service are set in its configuration file.

System requirements

LUNA PLATFORM is delivered in Docker containers and can be launched on CPU and GPU. Docker images of the LP containers are required for the installation. Internet connection is required on the server for Docker images download, or the images should be downloaded on any other device and moved to the server. It is required to manually specify login and password for Docker images downloading.

LUNA PLATFORM can be launched with a Docker Compose script.

The following Docker and Docker Compose versions are recommended for LP utilization:

- Docker: 20.10.8 (to manually launch containers)
- Docker Compose: 1.29.2 (to automatically launch containers)

Launching LUNA PLATFORM containers is officially supported on CentOS 7/8. Correct work on other systems is not guaranteed. All the procedures in the installation manual are described for CentOS 7.

LUNA PLATFORM service containers use the CentOS Linux 8.3.2011 operating system.

Processors

The configuration below guarantees software package minimum power operating and cannot be used for the production system. System requirements for the production system are calculated based on the intended system load.

CPU

The following minimum system requirements should be met for the LUNA PLATFORM software package installation:

- CPU Intel, 4 physical cores minimum with clock frequency 2.0 GHz or higher. AVX2 instruction set support is required for CPU.
- RAM DDR3 (DDR4 recommended), 8 Gb or higher.
- Free storage size must be 80 Gb or higher.

It is recommended using SSD for databases and Image Store service.

GPU

For GPU acceleration an NVIDIA GPU is required. The following architectures are supported:

- Pascal or newer.

Compute Capability 6.1 or higher is required.

A minimum of 6GB or dedicated video RAM is required. 8 GB or more VRAM recommended.

CUDA of version 11.4 should be installed on the server with the Remote SDK service. The recommended NVIDIA driver is r470.

Third-party applications

The following third-party services are used by default with LUNA PLATFORM 5.

- PostgreSQL is used as a default database for Faces, Configurator, Events, Handlers, Lambda, Tasks, Admin, and Backport3 services.

You can also use the Oracle database instead of PostgreSQL for all services except the Events service. The installation and configuration of Oracle are not described in this manual.

- Redis DB is used for Faces and Sender services.
- InfluxDB is used for monitoring.

Balancers and other software can be used when scaling the system to provide fail-safety. The installation guide provides recommendations on launching Nginx container with a configuration file to balance requests to the API, Faces, Image Store, and Events services.

The following third-party applications versions are recommended for LP launching:

- PostgreSQL: 16
- Oracle: 21c (if used instead PostgreSQL)
- Redis: 7.2
- InfluxDB: 2.0.8-alpine
- Grafana: 8.5.20 (optional)
- Grafana Loki: 2.7.1 (optional)
- Nginx: 1.17.4-alpine (optional)

These versions were tested by VisionLabs specialists. Newer versions can be used if needed, but they are not guaranteed to work.

It is recommended to use the `unzip` package to unpack the distribution. The command to download the package is given in the installation manual.

If you need to use an external database and the `VLMatch` function, you need to download additional dependencies described in the “External DB” section of the installation manual.

PostgreSQL, Redis, InfluxDB, Grafana and Nginx docker containers can be downloaded from the VisionLabs registry.

Introduction

This document describes the general approach for deploying LUNA PLATFORM in Docker containers.

It is considered that installation is performed on the server with CentOS OS, where LP was not installed.

For a successful launch, you need to perform the actions from the sections “[Before launch](#)” and “[Services launch](#)”. The section “[Additional information](#)” provides useful information on the description of service launch parameters, Docker commands, information on launching the Python Matcher Proxy service for using matching plugins and other.

A network license is required to use the LUNA PLATFORM in Docker containers. The license is provided by VisionLabs on request separately from the delivery. The license key is created using the fingerprint of the system. This fingerprint is created based on information about the hardware characteristics of the server. Thus, the received license key will work only on the same server from which the system fingerprint was obtained. LUNA PLATFORM can be activated using one of two utilities - HASP or Guardant. The section “[Activate license](#)” provides instructions for activating the license key for each method.

Firewall and SELinux should be manually configured on the server by the administrator. Their configuration is not described in this document.

No data backup or databases replication is implemented for LP data in this installation.

This document does not include a tutorial for Docker usage. Please refer to the Docker documentation to find more information about Docker:

<https://docs.docker.com>

This document includes an example of LUNA PLATFORM deployment. It implements LUNA PLATFORM minimum power operating for demonstration purposes and cannot be used for the production system.

It is recommended to use orchestration services for the commercial usage of LP. Their utilization is not described in this manual.

All the provided commands should be executed using the Bash shell (when you launch commands directly on the server) or in a program for working with network protocols (when you remotely connect to the server), for example, Putty.

A license file is required for LUNA PLATFORM activation. The file is provided by VisionLabs separately upon request.

All actions described in this manual must be performed by the **root** user. This document does not describe the creation of the user with administrator privileges and the following installation by this user.

1 Before launch

Make sure that you are the **root** user before launch!

Before launching the LUNA PLATFORM, you must perform the following actions:

1. [Unpack the LUNA PLATFORM distribution.](#)
2. [Create symbolic link.](#)
3. [Change group and owner for new directories.](#)
4. [Configure SELinux and Firewall.](#)
5. [Activate license.](#)
6. [Login to VisionLabs registry.](#)
7. [Install Docker.](#)
8. [Choose logging method.](#)
9. [Set up GPU computing](#) if you plan to use GPU.

1.1 Distribution unpacking

The distribution package is an archive **luna_v.5.56.0**, where **v.5.56.0** is a numerical identifier, describing the current LUNA PLATFORM version.

The archive includes configuration files, required for installation and exploitation. It does not include Docker images for the services. They should be downloaded from the Internet.

Move the distribution package to the directory on your server before the installation. For example, move the files to `/root/` directory. The directory should not contain any other distribution or license files except the target ones.

Create directory for distribution file unpacking.

```
mkdir -p /var/lib/luna
```

Move the distribution to the created directory.

```
mv /root/luna_v.5.56.0.zip /var/lib/luna
```

Install the unzip archiver if it is necessary.

```
yum install -y unzip
```

Go to the folder with distribution.

```
cd /var/lib/luna
```

Unzip files.

```
unzip luna_v.5.56.0.zip
```

1.2 Symbolic link creation

Create a symbolic link.

The link indicates that the current version of the distribution file is used to run LUNA PLATFORM.

```
ln -s luna_v.5.56.0 current
```

1.3 Changing group and owner for directories

LP services are launched inside the containers by the “luna” user. Therefore, it is required to set permissions for this user to use the mounted volumes.

Go to the LP “example-docker” directory.

```
cd /var/lib/luna/current/example-docker/
```

Create a directory to store settings.

```
mkdir luna_configurator/used_dumps
```

Set permissions for the user with UID 1001 and group 0 to use the mounted directories.

```
chown -R 1001:0 luna_configurator/used_dumps
```

Open the LP root directory.

```
cd /var/lib/luna/
```

Create a directory to store Image Store buckets.

```
mkdir image_store
```

Set permissions for the user with UID 1001 and group 0 to use the mounted directories.

```
chown -R 1001:0 image_store
```

1.4 SELinux and Firewall

You must configure SELinux and Firewall so that they do not block LUNA PLATFORM services.

SELinux and Firewall configurations are not described in this guide.

If SELinux and Firewall are not configured, the installation cannot be performed.

1.5 License activation

To activate the license, follow these steps:

- Follow the steps from [license activation manual](#).

- Set settings for [HASP](#) license or [Guardant](#) license.

1.5.1 Actions from License activation manual

Open the license activation manual and follow the necessary steps.

Note: This action is mandatory. The license will not work without following the steps to activate the license from the corresponding manual.

1.5.2 Specify HASP license settings

For the HASP key, you need to specify the IP address of the licensing server. It can be set in one of two ways:

- In the dump file “platform_settings.json” (see below). The contents of the default settings will be overwritten by the contents of this file when the Configurator service starts.
- In the Licenses service settings in the Configurator user interface (see section “[Specify license settings using Configurator](#)”).

Choose the most convenient method and follow the steps described in the relevant sections.

1.5.2.1 Specify HASP license settings using dump file

Open the “platform_settings.json” file.

```
vi /var/lib/luna/current/extras/conf/platform_settings.json
```

Set the server IP address with your HASP key in the “server_address” field.

```
{
  "value": {
    "vendor": "hasp",
    "server_address": "127.0.0.1"
  },
  "description": "License vendor config",
  "name": "LICENSE_VENDOR",
  "tags": []
},
```

Save the file.

If the license is activated using the HASP key, then two parameters “vendor” and “server_address”

must be specified. If you want to change the HASP protection to Guardant, then you need to add the “license_id” field.

1.5.3 Specify Guardant license settings

For the Guardant key, you need to specify the IP address of the licensing server and the license ID. The settings can be set in one of two ways:

- In the dump file “platform_settings.json” (see below). The contents of the default settings will be overwritten by the contents of this file at the launch stage of the Configurator service.
- In the Licenses service settings in the Configurator user interface (see the section “[Specify license settings using Configurator](#)”).

Choose the most convenient method and follow the steps described in the relevant sections.

1.5.3.1 Specify Guardant license settings using dump file

Open the file “platform_settings.json”.

```
vi /var/lib/luna/current/extras/conf/platform_settings.json
```

Enter the following data:

- IP address of the server with your Guardant key in the “server_address” field.
- License ID in the format 0x<your_license_id>, obtained in the section “Save license ID” of license activation manual, in the “license_id” field.

```
{
  "value": {
    "vendor": "guardant",
    "server_address": "127.0.0.1",
    "license_id": "0x92683BEA"
  },
  "description": "License vendor config",
  "name": "LICENSE_VENDOR",
  "tags": []
},
```

Save the file.

If the license is activated using the Guardant key, then three parameters “vendor”, “server_address” and “license_id” must be specified. If you want to change the Guardant protection to HASP, then you need to delete the “license_id” field.

1.6 Login to registry

When launching containers, you should specify a link to the image required for the container launching. This image will be downloaded from the VisionLabs registry. Before that, you should login to the registry.

Login and password can be requested from the VisionLabs representative.

Enter login <username>.

```
docker login dockerhub.visionlabs.ru --username <username>
```

After running the command, you will be prompted for a password. Enter password.

In the `docker login` command, you can enter the login and password at the same time, but this does not guarantee security because the password can be seen in the command history.

1.7 Docker installation

The Docker installation is described in the [official documentation](#)

You do not need to install Docker if you already have an installed Docker 20.10.8 on your server. Not guaranteed to work with higher versions of Docker.

Quick installation commands are listed below.

Check the official documentation for updates if you have any problems with the installation.

Install dependencies.

```
yum install -y yum-utils device-mapper-persistent-data lvm2
```

Add repository.

```
yum-config-manager --add-repo https://download.docker.com/linux/centos/  
docker-ce.repo
```

Install Docker.

```
yum -y install docker-ce docker-ce-cli containerd.io
```

Launch Docker.

```
systemctl start docker
```

```
systemctl enable docker
```

Check Docker status.

```
systemctl status docker
```

1.8 Choose logging method

There are two methods to output logs in LUNA PLATFORM:

- Standard log output (stdout).
- Log output to a file.

Log output settings are set in the settings of each service in the <SERVICE_NAME>_LOGGER section.

If necessary, you can use both methods of displaying logs.

For more information about the LUNA PLATFORM logging system, see the “Logging” section in the administrator manual.

1.8.1 Logging to stdout

This method is used by default and requires no further action.

It is recommended to configure Docker log rotation to limit log sizes (see “[Docker log rotation](#)”).

1.8.2 Logging to file

Note: When you enable saving logs to a file, you should remember that logs occupy a certain place in the storage, and the process of logging to a file negatively affects system performance.

To use this method, you need to perform the following additional actions:

- **Before launching the services:** Create directories for logs on the server.
- **After launching the services:** Activate log recording and set the location of log storage inside LP service containers.
- **During the launch of services:** Configure synchronization of log directories in the container with logs on the server using the `volume` argument at the start of each container.

Examples of container launch commands in this documentation contain arguments for synchronizing log directories.

Note that the above steps must be performed before, during and after starting the services. Saving logs to file will not work if you perform all actions after starting the containers.

See the instructions for enabling logging to files in the “[Logging the server](#)” section.

1.9 Calculations using GPU

You can use GPU for the general calculations performed by Remote SDK.

Skip this section if you are not going to utilize GPU for your calculations.

You need to install NVIDIA Container Toolkit to use GPU with Docker containers. The example of the installation is given below.

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
```

```
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.repo | tee /etc/yum.repos.d/nvidia-docker.repo
```

```
yum install -y nvidia-container-toolkit
```

```
systemctl restart docker
```

Check the NVIDIA Container toolkit operating by running a base CUDA container (this container is not provided in the LP distribution and should be downloaded from the Internet):

```
docker run --rm --gpus all nvidia/cuda:11.4.3-base-centos7 nvidia-smi
```

See the [NVIDIA documentation](#) for additional information.

Attributes extraction on the GPU is engineered for maximum throughput. The input images are processed in batches. This reduces computation cost per image but does not provide the shortest latency per image.

GPU acceleration is designed for high load applications where request counts per second consistently reach thousands. It won't be beneficial to use GPU acceleration in non-extensively loaded scenarios where latency matters.

2 Services launch

This section gives examples for:

- Databases tables creation.
- Buckets creation.
- Launching of containers.

LUNA PLATFORM services must be launched in the following sequence:

- Databases, Balancers, HASP service and other third-party party software.
- [Configurator](#).
- [Image Store](#).
- [Accounts](#).
- [Licenses](#).
- [Faces](#).
- [Events](#).
- [Python Matcher](#).
- [Python Matcher Proxy](#). The service is disabled by default.
- [Remote SDK](#).
- [Handlers](#).
- [Tasks](#).
- [Sender](#).
- [API](#).
- [Admin](#).

The [Lambda](#) service (disabled by default) can be launched after Licenses and Configurator services.

The following service are used when emulation of LUNA PLATFORM 3 is required only:

- [Backport 3](#).
- [User Interface 3](#).

The following service are used when emulation of LUNA PLATFORM 4 is required only:

- [Backport 4](#).
- [User Interface 4](#).

It is recommended to launch containers one by one and wait for the container status to become “up” (use the `docker ps` command).

Some of these services are optional and you can disable their use. It is recommended to use Events, Tasks, Sender and Admin services by default. See the “[Optional services usage](#)” section for details.

When launching each service, certain parameters are used, for example, `--detach`, `--network`, etc. See the section “[Launching parameters description](#)” for more detailed information about all launch parameters of LUNA PLATFORM services and databases.

See the [“Docker commands”](#) section for details about working with containers.

2.1 Monitoring configuration

Monitoring LUNA PLATFORM services requires running the Influx 2.0.8-alpine database. Below are the commands to launch the InfluxDB container.

For more information, see the “Monitoring” section in the administrator manual.

If necessary, you can configure the visualization of monitoring data using the LUNA Dashboards service, which includes a configured Grafana data visualization system. In addition, you can launch the Grafana Loki tool for advanced work with logs. See the instructions for launching LUNA Dashboards and Grafana Loki in the “[Monitoring and logs visualization using Grafana](#)” section.

2.1.1 InfluxDB OSS 2

Note: If necessary, you can use an external InfluxDB 2.0.8-alpine. In this case, you can skip the command below, but you will have to set [custom settings](#) for each LUNA PLATFORM service.

Use the `docker run` command with these parameters:

```
docker run \  
-e DOCKER_INFLUXDB_INIT_MODE=setup \  
-e DOCKER_INFLUXDB_INIT_BUCKET=luna_monitoring \  
-e DOCKER_INFLUXDB_INIT_USERNAME=luna \  
-e DOCKER_INFLUXDB_INIT_PASSWORD=password \  
-e DOCKER_INFLUXDB_INIT_ORG=luna \  
-e DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=kofqt4Pfqn6o0RBtMDQqVoJLgHoxxDU  
mmhiAZ7JS6VmEnrqZXQhxDhad8AX9tmiJH6CjM7Y1U8p5eSEocGzIA== \  
-v /etc/localtime:/etc/localtime:ro \  
-v /var/lib/luna/influx:/var/lib/influxdb2 \  
--restart=always \  
--detach=true \  
--network=host \  
--name influxdb \  
dockerhub.visionlabs.ru/luna/influxdb:2.0.8-alpine
```

If you need to set the custom settings of the InfluxDB (for example, set the IP address and port when launching InfluxDB on separate server), then you need to change them in the configurations of each LUNA PLATFORM service. See the section “[Set custom InfluxDB settings](#)” for more information.

2.2 Run third-party services

This section describes the launching of databases and message queues in docker containers. They must be launched before LP services.

2.2.1 PostgreSQL

Note: If necessary, you can use an external PostgreSQL database. In this case, you can skip the command below and perform the actions from the section “External DB”.

Use the following command to launch PostgreSQL.

```
docker run \  
--env=POSTGRES_USER=luna \  
--env=POSTGRES_PASSWORD=luna \  
--shm-size=1g \  
-v /var/lib/luna/postgresql/data:/var/lib/postgresql/data/ \  
-v /var/lib/luna/current/example-docker/postgresql/entrypoint-initdb.d:/\  
    docker-entrypoint-initdb.d/ \  
-v /etc/localtime:/etc/localtime:ro \  
--name=postgres \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/postgis-vmatch:16
```

-v /var/lib/luna/current/example-docker/postgresql/entrypoint-initdb.d:/docker-entrypoint-initdb.d/ \ - The “docker-entrypoint-initdb.d” script includes the commands for the creation of services databases. During database creation, a default username and password are automatically used.

-v /var/lib/luna/current/example-docker/postgresql/data:/var/lib/postgresql/data/ - The volume command enables you to mount the “data” folder to the PostgreSQL container. The folder on the server and the folder in the container will be synchronized. The PostgreSQL data from the container will be saved to this directory.

--network=host - If you need to change the port for PostgreSQL, you should change this string to -p 5440:5432. Where the first port 5440 is the local port and 5432 is the port used inside the container.

You should create all the databases for LP services manually if you are going to use an already installed PostgreSQL.

2.2.2 Redis

If you already have Redis installed, skip this step.

Use the following command to launch Redis.

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
--name=redis \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/redis:7.2
```

2.3 Configurator

2.3.1 Optional services usage

The listed below services are not mandatory for LP:

- Events
- Image Store
- Tasks
- Sender
- Handlers
- Python Matcher Proxy (disabled by default)
- Lambda (disabled by default)

You can disable them if their functionality is not required for your tasks.

Use the “ADDITIONAL_SERVICES_USAGE” section in the API service settings in the Configurator service to disable unnecessary services.

You can use the dump file provided in the distribution package to enable/disable services before Configurator launch.

```
vi /var/lib/luna/current/extras/conf/platform_settings.json
```

Disabling any of the services has certain consequences. For more information, see the “Disableable services” section of the administrator manual.

2.3.2 Configurator DB tables creation

Use the `docker run` command with these parameters to create the Configurator database tables.

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /var/lib/luna/current/example-docker/luna_configurator/configs/  
  luna_configurator_postgres.conf:/srv/luna_configurator/configs/config.  
  conf \  
-v /var/lib/luna/current/extras/conf/platform_settings.json:/srv/  
  luna_configurator/used_dumps/platform_settings.json \  
--network=host \  
-v /tmp/logs/configurator:/srv/logs \  
--rm \  
--entrypoint bash \  
dockerhub.visionlabs.ru/luna/luna-configurator:v.2.1.80 \  

```

```
-c "python3 ./base_scripts/db_create.py; cd /srv/luna_configurator/configs/
configs/; python3 -m configs.migrate --config /srv/luna_configurator/
configs/config.conf head; cd /srv; python3 ./base_scripts/db_create.py --
dump-file /srv/luna_configurator/used_dumps/platform_settings.json"
```

Here:

- /var/lib/luna/current/extras/conf/platform_settings.json - Enables you to specify the path to the dump file with LP configurations.
- ./base_scripts/db_create.py; - Creates database structure.
- python3 -m configs.migrate head; - Performs settings migrations in Configurator DB and sets revision for migration. The revision will be required during the upgrade to the new LP5 build.
- --dump-file /srv/luna_configurator/used_dumps/platform_settings.json - Updates settings in the Configurator DB with values from the provided file.

2.3.3 Run Configurator container

Use the `docker run` command with these parameters to launch Configurator:

```
docker run \
--env=PORT=5070 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /etc/localtime:/etc/localtime:ro \
-v /var/lib/luna/current/example-docker/luna_configurator/configs/
luna_configurator_postgres.conf:/srv/luna_configurator/configs/config.
conf \
-v /tmp/logs/configurator:/srv/logs \
--name=luna-configurator \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-configurator:v.2.1.80
```

At this stage, you can activate logging to file if you need to save them on the server (see the [“Logging to server”](#) section).

2.4 Image Store

2.4.1 Image Store container launch

Note: If you are not going to use the Image Store service, do not launch this container and disable the service utilization in Configurator. See section “[Optional services usage](#)”.

Use the following command to launch the Image Store service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5020 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /var/lib/luna/image_store:/srv/local_storage/ \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/image-store:/srv/logs \  
--name=luna-image-store \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-image-store:v.3.10.2
```

Here `-v /var/lib/luna/image_store:/srv/local_storage/` is the data from the specified folder is added to the Docker container when it is launched. All the data from the specified Docker container folder is saved to this directory.

If you already have a directory with LP buckets you should specify it instead of `/var/lib/luna/image_store/`.

2.4.2 Buckets creation

Buckets are required to store data in Image Store. The Image Store service should be launched before the commands execution.

When upgrading from the previous version, it is recommended to launch the bucket creation commands one more time. Hence you make sure that all the required buckets were created.

If the error with code [13006](#) appears during launching of the listed above commands, the bucket is already created.

There are two ways to create buckets in LP.

Run the listed below scripts to create buckets.

Run this script to create general buckets:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/api:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-api:v.6.23.0 \  
python3 ./base_scripts/lis_bucket_create.py -ii --luna-config http://  
localhost:5070/1
```

If you are going to use the Tasks service, use the following command to additionally create the “task-result” in the Image Store service:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/tasks:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.19.2 \  
python3 ./base_scripts/lis_bucket_create.py -ii --luna-config http://  
localhost:5070/1
```

If you are going to use the portraits, use the following command to additionally create the “portraits”.

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/api:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-backport3:v.0.10.2 \  
python3 ./base_scripts/lis_bucket_create.py -ii --luna-config http://  
localhost:5070/1
```

Use direct requests to create required buckets.

The curl utility is required for the following requests.

The “visionlabs-samples” bucket is used for face samples storage. The bucket is required for LP utilization.

```
curl -X POST http://127.0.0.1:5020/1/buckets?bucket=visionlabs-samples
```

The “portraits” bucket is used for portraits storage. The bucket is required for Backport 3 utilization.

```
curl -X POST http://127.0.0.1:5020/1/buckets?bucket=portraits
```

The “visionlabs-bodies-samples” bucket is used for human bodies samples storage. The bucket is required for LP utilization.

```
curl -X POST http://127.0.0.1:5020/1/buckets?bucket=visionlabs-bodies-samples
```

The “visionlabs-image-origin” bucket is used for source images storage. The bucket is required for LP utilization.

```
curl -X POST http://127.0.0.1:5020/1/buckets?bucket=visionlabs-image-origin
```

The “visionlabs-objects” bucket is used for objects storage. The bucket is required for LP utilization.

```
curl -X POST http://127.0.0.1:5020/1/buckets?bucket=visionlabs-objects
```

The “task-result” bucket for the Tasks service. Do not use it if you are not going to use the Tasks service.

```
curl -X POST http://127.0.0.1:5020/1/buckets?bucket=task-result
```

2.5 Accounts

2.5.1 Accounts DB creation

Use the following command to create Accounts DB tables:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/accounts:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-accounts:v.0.2.2 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.5.2 Accounts container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5170 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/accounts:/srv/logs \  
--name=luna-accounts \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-accounts:v.0.2.2
```

2.6 Licenses

Note: To use a trial license, it is required to launch the Licenses service on the same server where trial license is being used.

2.6.1 Specify license settings using Configurator

Follow the steps below to set the settings for [HASP-key](#) or [Guardant-key](#).

Note: Do not perform the steps described below if you have already specified the license settings in the sections [“Specify HASP license settings using dump file”](#) or [“Specify Guardant license settings using dump file”](#).

2.6.1.1 Specify HASP license settings

Note: Perform these actions only if the HASP key is used. See the [“Specify Guardant license settings”](#) section if the Guardant key is used.

To set the license server address, follow these steps:

- Go to the Configurator service interface `http://<configurator_server_ip>:5070/`.
- Specify the “LICENSE_VENDOR” value in the “Setting name” field and click “Apply Filters”.
- Set the IP address of the server with your HASP key in the field “server_address” in the format “127.0.0.1”.
- Click “Save”.

If the license is activated using the HASP key, then two parameters “vendor” and “server_address” must be specified. If you want to change the HASP protection to Guardant, then you need to add the “license_id” field.

2.6.1.2 Specify Guardant license settings

Note: Perform these actions only if the Guardant key is used. See the [“Specify HASP license settings”](#) section if the HASP key is used.

To set the license server address, follow these steps:

- Go to the Configurator service interface `http://<configurator_server_ip>:5070/`.
- Enter the value “LICENSE_VENDOR” in the “Setting name” field and click “Apply Filters”.
- Set the IP address of the server with your Guardant key in the “server_address” field.
- Set the license ID in the format `0x<your_license_id>`, obtained in the section “Save license ID” in the License activation manual, in the “license_id” field.
- Click “Save”.

If the license is activated using the Guardant key, then three parameters “vendor”, “server_address” and “license_id” must be specified. If you want to change the Guardant protection to HASP, then you need to delete the “license_id” field.

2.6.2 Licenses container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5120 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/licenses:/srv/logs \  
--name=luna-licenses \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-licenses:v.0.9.5
```

2.7 Faces

2.7.1 Faces DB tables creation

Use the following command to create the Faces DB tables:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/faces:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-faces:v.4.10.2 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.7.2 Faces container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5030 \  
--env=WORKER_COUNT=2 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/faces:/srv/logs \  
--name=luna-faces \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-faces:v.4.10.2
```

2.8 Events

2.8.1 Events DB tables creation

Note: If you are not going to use the Events service, do not launch this container and disable the service utilization in Configurator. See section “[Optional services usage](#)”.

Use the following command to create the Events DB tables:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/events:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-events:v.4.11.3 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.8.2 Events container launch

Note: If you are not going to use the Events service, do not launch this container and disable the service utilization in Configurator. See section “[Optional services usage](#)”.

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5040 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/events:/srv/logs \  
--name=luna-events \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-events:v.4.11.3
```

2.9 Python Matcher services

For matching tasks, you can use either only the Python Matcher service, or additionally use the Python Matcher Proxy service, which redirects matching requests to either the Python Matcher service or matching plugins. This section describes how to use Python Matcher without Python Matcher Proxy.

You need to use the Python Matcher Proxy service only if you are going to use matching plugins. Using Python Matcher Proxy and running the corresponding docker container are described in the [“Use Python Matcher with Python Matcher Proxy”](#) section.

See the description and usage of matching plugins in the administrator manual.

2.9.1 Use Python Matcher without Python Matcher Proxy

The Python Matcher service with matching by the Faces DB is enabled by default after launching.

The Python Matcher service with matching by the Events is also enabled by default. You can disable it by specifying “USE_LUNA_EVENTS = 0” in the “ADDITIONAL_SERVICES_USAGE” settings of Configurator (see [“Optional services usage”](#) section). Thus, the Events service will not be used for LUNA PLATFORM.

The Python Matcher that matches using the matcher library is enabled when “CACHE_ENABLED” is set to “true” in the “DESCRIPTORS_CACHE” setting.

A single image is downloaded for the Python Matcher service and the Python Matcher Proxy service.

2.9.2 Python Matcher container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5100 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/python-matcher:/srv/logs \  
--name=luna-python-matcher \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-python-matcher:v.1.8.2
```

2.10 Remote SDK

2.10.1 Remote SDK container launch

You can run the Remote SDK service utilizing CPU (set by default) or GPU.

By default, the Remote SDK service is launched with all estimators and detectors enabled. If necessary, you can disable the use of some estimators or detectors when launching the Remote SDK container. Disabling unnecessary estimators enables you to save RAM or GPU memory, since when the Remote SDK service launches, the possibility of performing these estimates is checked and neural networks are loaded into memory. If you disable the estimator or detector, you can also remove its neural network from the Remote SDK container. See the “Enable/disable several estimators and detectors” section of the administrator manual for more information.

Run the Remote SDK service using one of the following commands according to the utilized processing unit.

2.10.1.1 Run Remote SDK utilizing CPU

Use the following command to launch the Remote SDK service using CPU:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5220 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/remote-sdk:/srv/logs \  
--network=host \  
--name=luna-remote-sdk \  
--restart=always \  
--detach=true \  
dockerhub.visionlabs.ru/luna/luna-remote-sdk:v.0.4.0
```

2.10.1.2 Run Remote SDK utilizing GPU

The Remote SDK service does not utilize GPU by default. If you are going to use the GPU, then you should enable its use for the Remote SDK service in the Configurator service.

If you need to use the GPU for all estimators and detectors at once, then you need to use the “global_device_class” parameter in the “LUNA_REMOTE_SDK_RUNTIME_SETTINGS” section. All estimators and detectors will use the value of this parameter if the “device_class” parameter of their

settings like "LUNA_REMOTE_SDK_<estimator-or-detector-name>_SETTINGS.runtime_settings" is set to "global" (by default for all estimators and detectors).

If you need to use the GPU for a specific estimator or detector, then you need to use the "device_class" parameter in sections like "LUNA_REMOTE_SDK_<estimator/detector-name>_SETTINGS.runtime_settings".

See section "[Calculations using GPU](#)" for additional requirements for GPU utilization.

Use the following command to launch the Remote SDK service using GPU:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5220 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--gpus device=0 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/remote-sdk:/srv/logs \  
--network=host \  
--name=luna-remote-sdk \  
--restart=always \  
--detach=true \  
dockerhub.visionlabs.ru/luna/luna-remote-sdk:v.0.4.0
```

Here `--gpus device=0` is the parameter specifies the used GPU device and enables GPU utilization. A single GPU can be utilized per Remote SDK instance. Multiple GPU utilization per instance is not available.

2.10.1.3 Run slim version of Remote SDK

You can run a slim version of the Remote SDK service that contains only configuration files without neural networks. It is assumed that the user himself will add the neural networks he needs to the container.

The launch of the slim version of the Remote SDK service is intended for advanced users.

To successfully launch the Remote SDK container with a custom set of neural networks, you need to perform the following actions:

- Request the required neural networks from VisionLabs.
- Place neural networks in a folder with LUNA PLATFORM installed.
- Assign appropriate rights to neural network files.
- Mount neural network files to the `/srv/fsdk/data` folder of the Remote SDK container.
- Using the arguments of the variable "EXTEND_CMD" to explicitly specify which of the neural networks should be used.

Using the “enable-all-estimators-by-default” flag for the “EXTEND_CMD” variable, you can disable the use of all neural networks (estimators) by default, and then use special flags to explicitly specify which neural networks should be used. If you do not specify this flag or set the value “--enable-all-estimators-by-default=1”, the Remote SDK service will try to find all neural networks in the container. If one of the neural networks is not found, the Remote SDK service will not start.

List of available estimators:

Argument	Description
--enable-all-estimators-by-default	Enable all estimators by default.
--enable-human-detector	Simultaneous detector of bodies and bodies.
--enable-face-detector	Face detector.
--enable-body-detector	Body detector.
--enable-face-landmarks5-estimator	Face landmarks5 estimator.
--enable-face-landmarks68-estimator	Face landmarks68 estimator.
--enable-head-pose-estimator	Head pose estimator.
--enable-liveness-estimator	Liveness estimator.
--enable-fisheye-estimator	FishEye effect estimator.
--enable-face-detection-background-estimator	Image background estimator.
--enable-face-warp-estimator	Face sample estimator.
--enable-body-warp-estimator	Body sample estimator.
--enable-quality-estimator	Image quality estimator.
--enable-image-color-type-estimator	Face color type estimator.
--enable-face-natural-light-estimator	Natural light estimator.
--enable-eyes-estimator	Eyes estimator.
--enable-gaze-estimator	Gaze estimator.
--enable-mouth-attributes-estimator	Mouth attributes estimator.
--enable-emotions-estimator	Emotions estimator.
--enable-mask-estimator	Mask estimator.
--enable-glasses-estimator	Glasses estimator.
--enable-eyebrow-expression-estimator	Eyebrow estimator.
--enable-red-eyes-estimator	Red eyes estimator.
--enable-headwear-estimator	Headwear estimator.

Argument	Description
<code>--enable-basic-attributes-estimator</code>	Basic attributes estimator.
<code>--enable-face-descriptor-estimator</code>	Face descriptor extraction estimator.
<code>--enable-body-descriptor-estimator</code>	Body descriptor extraction estimator.
<code>--enable-body-attributes-estimator</code>	Body attributes estimator.
<code>--enable-people-count-estimator</code>	People count estimator.
<code>--enable-deepfake-estimator</code>	Deepfake estimator.

See the detailed information on enabling and disabling certain estimators in the section “Enable/disable several estimators and detectors” of the administrator manual.

Below is an example of a command to assign rights to a neural network file:

```
chown -R 1001:0 /var/lib/luna/current/<neural_network_name>.plan
```

Example of a command to run Remote SDK container with mounting neural networks for face detection and face descriptor extraction:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5220 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
--env=EXTEND_CMD="--enable-all-estimators-by-default=0 --enable-face-
detector=1 --enable-face-descriptor-estimator=1" \
-v /var/lib/luna/current/cnn59b_cpu-avx2.plan:/srv/fsdk/data/cnn59b_cpu-avx2
.plan \
-v /var/lib/luna/current/FaceDet_v3_a1_cpu-avx2.plan:/srv/fsdk/data/
FaceDet_v3_a1_cpu-avx2.plan \
-v /var/lib/luna/current/FaceDet_v3_redetect_v3_cpu-avx2.plan:/srv/fsdk/data
/FaceDet_v3_redetect_v3_cpu-avx2.plan \
-v /var/lib/luna/current/slnet_v3_cpu-avx2.plan:/srv/fsdk/data/slnet_v3_cpu-
avx2.plan \
-v /var/lib/luna/current/LNet_precise_v2_cpu-avx2.plan:/srv/fsdk/data/
LNet_precise_v2_cpu-avx2.plan \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/remote-sdk:/srv/logs \
```

```
--network=host \  
--name=luna-remote-sdk \  
--restart=always \  
--detach=true \  
dockerhub.visionlabs.ru/luna/luna-remote-sdk:v.0.4.0
```

2.11 Handlers

Note: If you are not going to use the Handlers service, do not launch this container and disable the service utilization in Configurator. See section “[Optional services usage](#)”.

2.11.1 Handlers DB tables creation

Use the following command to create the Handlers DB tables:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/handlers:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-handlers:v.3.4.2 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.11.2 Handlers container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5090 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/handlers:/srv/logs \  
--name=luna-handlers \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-handlers:v.3.4.2
```

2.12 Tasks

Note: If you are not going to use the Tasks service, do not launch the Tasks container and the Tasks Worker container. Disable the service utilization in Configurator. See section “[Optional services usage](#)”.

2.12.1 Tasks DB tables creation

Use the following command to create Tasks DB tables:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/tasks:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.19.2 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.12.2 Tasks and Tasks Worker containers launch

Tasks service image includes the Tasks service and the Tasks Worker. They both must be launched.

The “task-result” bucket should be created for the Tasks service before the service launch. The buckets creation is described in the “[Buckets creation](#)”.

If it is necessary to use the Estimator task using a network disk, then you should first mount the directory with images from the network disk into special directories of Tasks and Tasks Worker containers. See the “Estimator task” section in the administrator manual for details.

2.12.2.1 Tasks Worker launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5051 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--env=SERVICE_TYPE="tasks_worker" \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/tasks-worker:/srv/logs \  
--name=luna-tasks-worker \  

```

```
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.19.2
```

2.12.2.2 Tasks launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5050 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/tasks:/srv/logs \  
--name=luna-tasks \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.19.2
```

2.13 Sender

2.13.1 Sender container launch

Note: If you are not going to use the Sender service, do not launch this container and disable the service utilization in Configurator. See section [“Optional services usage”](#).

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5080 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/sender:/srv/logs \  
--name=luna-sender \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-sender:v.2.10.2
```

2.14 API

2.14.1 API container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5000 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--name=luna-api \  
--restart=always \  
--detach=true \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/api:/srv/logs \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-api:v.6.23.0
```

2.14.2 Account creation using API service

The account is created using an HTTP request to the “create account” resource of the API service.

You can also create an account using the Admin service. This method requires an existing login and password (or the default login and password) and enables you to create an “admin” account. See the “Admin service” section of the administrator manual for details.

To create the account using a request to the API service, you need to provide the following mandatory data:

- “login” — Email address.
- “password” — Password.
- “account_type” — Account type (“user” or “advanced_user”).

Create the account using your authentication details.

Example of CURL-request to the “create account” resource:

```
curl --location --request POST 'http://127.0.0.1:5000/6/accounts' \  
--header 'Content-Type: application/json' \  
--data '{  
  "login": "user@mail.com",  
  "password": "password",
```

```
"account_type": "user",  
"description": "description"  
'
```

It is necessary to replace the authentication data from the example with your own.

To work with tokens, you must have an account.

2.14.3 Create GC task schedule

Before you start working with the LUNA PLATFORM, you can create a schedule for the Garbage collection task.

To do this, make a “create tasks schedule” request to the API service, specifying the necessary rules for the schedule.

An example of a schedule creation command for an account created at the stage of [creating an account using the API service](#) is given below.

The example sets a schedule for the Garbage collection task for events older than 30 days with the removal of the samples and the source images. The task will be repeated **once a day at 05:30 am**.

```
curl --location --request POST 'http://127.0.0.1:5000/6/tasks/schedules' \
--header 'Authorization: Basic dXNlckBtYWlsLmNvbTpwYXNzd29yZA==' \
--header 'Content-Type: application/json' \
--data '{
  "task": {
    "task_type": 4,
    "content": {
      "target": "events",
      "filters": {
        "create_time__lt": "now-30d"
      },
      "remove_samples": true,
      "remove_image_origins": true
    }
  },
  "trigger": {"cron": "30 5 * * *", "cron_timezone": "utc"},
  "behaviour": {"start_immediately": false, "create_stopped": false}
}'
```

If necessary, you can create a schedule without automatically activating it. To do this, specify the parameter “create_stopped”: “true”. In this case, after creating the schedule, it must be activated manually using the “action” = “start” parameter of the “patch tasks schedule” request.

For more information, see the “Running scheduled tasks” section of the administrator manual.

2.15 Admin

2.15.1 Admin container launch

Note: If you are not going to use the Admin service, do not launch this container.

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5010 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/admin:/srv/logs \  
--name=luna-admin \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-admin:v.5.5.2
```

Monitoring data about the number of executed requests is saved in the luna-admin bucket of the InfluxDB. To enable data saving use the following command:

```
docker exec -it luna-admin python3 ./base_scripts/influx2_cli.py  
create_usage_task --luna-config http://127.0.0.1:5070/1
```

2.16 Backport 3

The section describes launching of Backport 3 service.

The service is not mandatory for utilizing LP5 and is required for emulation of LP 3 API only.

2.16.1 Backport 3 bucket creation

Use the following command to create the “portraits” bucket for Backport 3:

```
curl -X POST http://127.0.0.1:5020/1/buckets?bucket=portraits
```

2.16.2 Backport 3 DB tables creation

Use the following command to create DB tables for Backport 3:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/backport3:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-backport3:v.0.10.2 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.16.3 Backport 3 container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5140 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--name=luna-backport3 \  
--restart=always \  
--detach=true \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/backport3:/srv/logs \  
--network=host
```

`dockerhub.visionlabs.ru/luna/luna-backport3:v.0.10.2`

2.16.4 User Interface 3

The User Interface 3 is used with the Backport 3 service only.

2.16.4.1 User Interface 3 container launch

Use the following command to launch the service:

```
docker run \  
--env=PORT=4100 \  
--env=LUNA_API_URL=http://127.0.0.1:5140 \  
--name=luna-ui-3 \  
--restart=always \  
--detach=true \  
--network=host \  
-v /etc/localtime:/etc/localtime:ro \  
dockerhub.visionlabs.ru/luna/luna3-ui:v.0.5.10
```

Here:

- --env=LUNA_API_URL - Specifies the URL of the Backport 3 service.
- --env=PORT - Specifies the port of the User Interface 3 service.

2.17 Backport 4

The section describes launching of Backport 4 service.

The service is not mandatory for utilizing LP5 and is required for emulation of LP 4 API only.

2.17.1 Backport 4 container launch

Use the following command to launch the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5130 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--name=luna-backport4 \  
--restart=always \  
--detach=true \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/backport4:/srv/logs \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-backport4:v.1.4.2
```

2.17.2 User Interface 4

The User Interface 4 is used with the Backport 4 service only.

2.17.2.1 User Interface 4 container launch

Note: You should have the account with account type **user** before launching the User Interface 4 container. Its login and password in Base64 format will be used to work with the user interface.

Use the following command to launch the service:

```
docker run \  
--env=PORT=4200 \  
--env=LUNA_API_URL=http://<server_external_ip>:5130 \  
--env=BASIC_AUTH=dXNlcjBtYWlsLmNvbTpwYXNzd29yZA== \  
--name=luna-ui-4 \  
--restart=always \  
--detach=true \  
--network=host \  
-v /etc/localtime:/etc/localtime:ro \  
dockerhub.visionlabs.ru/luna/luna4-ui:v.0.1.5
```

Here:

- `--env=PORT` - Sets the port for running User Interface 4.
- `--env=BASIC_AUTH` - Sets the Basic authorization for the account which data is displayed in the user interface. It is necessary to convert `login:password` created at the stage [“Account creation using API service”](#) to Base64 format. The account type should be set to **user**.
- `--env=LUNA_API_URL` - Sets the URL of the Backport 4 service.

You should use the external IP of the service, not localhost.

You should specify the Backport 4 service port (5130 is set by default).

2.18 Lambda

Working with the Lambda service is possible only when deploying LUNA PLATFORM services in Kubernetes. To use it, you need to deploy LUNA PLATFORM services in Kubernetes yourself or consult VisionLabs specialists. Use the commands below as reference information.

Note: If you are not going to use the Lambda service, do not run this container.

Enable the use of the Lambda service (see the section [“Using optional services”](#)).

2.18.1 Prepare Docker registry

It is necessary to prepare a registry for storing Lambda docker images. Transfer the base images and Kaniko executor image to your registry using the following commands.

Upload the images from the remote repository to the local image storage.

```
docker pull dockerhub.visionlabs.ru/luna/lpa-lambda-base-fsdk:v.0.0.45
```

```
docker pull dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.0.45
```

```
docker pull dockerhub.visionlabs.ru/luna/kaniko-executor:latest
```

Add new names to the images by replacing `new-registry` on their own. The names of the base images in the user registry must be the same as in the `dockerhub.visionlabs.ru/luna` registry.

```
docker tag dockerhub.visionlabs.ru/luna/lpa-lambda-base-fsdk:v.0.0.45 new-registry/lpa-lambda-base-fsdk:v.0.0.45
```

```
docker tag dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.0.45 new-registry/lpa-lambda-base:v.0.0.45
```

```
docker tag dockerhub.visionlabs.ru/luna/kaniko-executor:latest new-registry/kaniko-executor:latest
```

Push local images to your remote repository by replacing `new-registry` on their own.

```
docker push new-registry/lpa-lambda-base-fsdk:v.0.0.45
```

```
docker push new-registry/lpa-lambda-base:v.0.0.45
```

```
docker push new-registry/kaniko-executor:latest
```

2.18.2 Lambda DB tables creation

Use the following command to create the Lambda DB tables:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/lambda:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-lambda:v.0.2.0 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.18.3 Lambda container launch

Use the following command to start the service:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5210 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/lambda:/srv/logs \  
--name=luna-lambda \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-lambda:v.0.2.0
```

3 Additional information

This section provides the following additional information:

- [Monitoring and logs visualization using Grafana.](#)
- [Useful commands for working with Docker.](#)
- [Description of the parameters for launching LUNA PLATFORM services and creating databases.](#)
- [Ways to change service settings.](#)
- [Actions to enable saving LP service logs to files.](#)
- [Configuring Docker log rotation.](#)
- [Setting custom InfluxDB settings.](#)
- [Using Python Matcher service with Python Matcher Proxy service.](#)
- [System scaling.](#)
- [Improve system performance.](#)
- [Manual creation of databases when using an external PostgreSQL database \(including creation of the VLMatch function for the Faces and Events database\).](#)
- [Compiling the VLMatch library for Oracle.](#)

3.1 Monitoring and logs visualization using Grafana

Monitoring visualization is performed by the LUNA Dashboards service, which contains the Grafana monitoring data visualization platform with configured LUNA PLATFORM dashboards.

If necessary, you can install customized dashboards for Grafana separately. See the “LUNA Dashboards” section in the administrator manual for more information.

Together with Grafana, you can use the Grafana Loki log aggregation system, which enables you to flexibly work with LUNA PLATFORM logs. The Promtail agent is used to deliver LUNA PLATFORM logs to Grafana Loki (for more information, see the “Grafana Loki” section in the administrator manual).

3.1.1 LUNA Dashboards

Note: To work with Grafana you need to use InfluxDB version 2.

3.1.1.1 Run LUNA Dashboards container

Use the `docker run` command with these parameters to run Grafana:

```
docker run \  
--restart=always \  
--detach=true \  
--network=host \  
--name=grafana \  
-v /etc/localtime:/etc/localtime:ro \  
dockerhub.visionlabs.ru/luna/luna-dashboards:v.0.0.9
```

Use “`http://IP_ADDRESS:3000`” to go to the Grafana web interface when the LUNA Dashboards and InfluxDB containers are running.

3.1.2 Grafana Loki

Note: Grafana Loki requires LUNA Dashboards to be running.

3.1.2.1 Run Grafana Loki container

Use the `docker run` command with these parameters to run Grafana Loki:

```
docker run \  
--name=loki \  
--restart=always \  
--detach=true \  
--network=host \  

```

```
-v /etc/localtime:/etc/localtime:ro \  
dockerhub.visionlabs.ru/luna/loki:2.7.1
```

3.1.2.2 Run Promtail container

Use the `docker run` command with these parameters to run Promtail:

```
docker run \  
-v /var/lib/luna/current/example-docker/logging/promtail.yml:/etc/promtail/  
  luna.yml \  
-v /var/lib/docker/containers:/var/lib/docker/containers \  
-v /etc/localtime:/etc/localtime:ro \  
--name=promtail \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/promtail:2.7.1 \  
-config.file=/etc/promtail/luna.yml -client.url=http://127.0.0.1:3100/loki/  
  api/v1/push -client.external-labels=job=containerlogs,pipeline_id=,job_id  
  =,version=
```

Here:

- `-v /var/lib/luna/current/example-docker/logging/promtail.yml:/etc/promtail/luna.yml` - Mounting the configuration file to the Promtail container.
- `-config.file=/etc/promtail/luna.yml` - Flag with the address of the configuration file.
- `-client.url=http://127.0.0.1:3100/loki/api/v1/push` - Flag with the address of deployed Grafana Loki.
- `-client.external-labels=job=containerlogs,pipeline_id=,job_id=,version=` - Static labels to add to all logs sent to Grafana Loki.

3.2 Docker commands

3.2.1 Show containers

To show the list of launched Docker containers use the command:

```
docker ps
```

To show all the existing Docker containers use the command:

```
docker ps -a
```

3.2.2 Copy files to container

You can transfer files into the container. Use the `docker cp` command to copy a file into the container.

```
docker cp <file_location> <container_name>:<folder_inside_container>
```

3.2.3 Enter container

You can enter individual containers using the following command:

```
docker exec -it <container_name> bash
```

To exit the container, use the command:

```
exit
```

3.2.4 Images names

You can see all the names of the images using the command:

```
docker images
```

3.2.5 Delete image

If you need to delete an image:

- Run the `docker images` command.

- Find the required image, for example `dockerhub.visionlabs.ru/luna/luna-image-store`.
- Copy the corresponding image ID from the IMAGE ID, for example, “61860d036d8c”.
- Specify it in the deletion command:

```
docker rmi -f 61860d036d8c
```

Delete all the existing images.

```
docker rmi -f $(docker images -q)
```

3.2.6 Stop container

You can stop the container using the command:

```
docker stop <container_name>
```

Stop all the containers:

```
docker stop $(docker ps -a -q)
```

3.2.7 Delete container

If you need to delete a container:

- Run the “`docker ps`” command.
- Stop the container (see [Stop container](#)).
- Find the required image, for example `dockerhub.visionlabs.ru/luna/luna-image-store`.
- Copy the corresponding container ID from the CONTAINER ID column, for example, “23f555be8f3a”.
- Specify it in the deletion command:

```
docker container rm -f 23f555be8f3a
```

Delete all the containers.

```
docker container rm -f $(docker container ls -aq)
```

3.2.7.1 Check service logs

You can use the following command to show logs for the service:

```
docker logs <container_name>
```

3.3 Launching parameters description

When launching a Docker container for a LUNA PLATFORM service you should specify additional parameters required for the service launching.

The parameters specific for a particular container are described in the section about this container launching.

All the parameters given in the service launching example are required for proper service launching and utilization.

3.3.1 Launching services parameters

Example command of launching LP services containers:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=<Port_of_the_launched_service> \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/<service>:/srv/logs/ \  
--name=<service_container_name> \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/<service-name>:<version>
```

The following parameters are used when launching LP services containers:

- `docker run` - Command for running the selected image as a new container.
- `dockerhub.visionlabs.ru/luna/<service-name>:<version>` - Sets the image required for the container launching.

Links to download the container images you need are available in the description of the corresponding container launching.

- `--network=host` - Sets that a network is not simulated and the server network is used. If you need to change the port for third-party party containers, you should change this string to `-p 5440:5432`. Where the first port 5440 is the local port and 5432 is the port used inside the container. The example is given for PostgreSQL.

- `--env=` - Sets the environment variables required to run the container (see the “[Service arguments](#)” section).
- `--name=<service_container_name>` - Sets the name of the launched container. The name must be unique. If there is a container with the same name, an error will occur.
- `--restart=always` - Sets a restart policy. The daemon will always restart the container regardless of the exit status.
- `--detach=true` - Run the container in the background mode.
- `-v` - Enables you to mount the content of a server folder into a volume in the container. Thus their contents will synchronize. The following general data is mounted:
- `/etc/localtime:/etc/localtime:ro` - Sets the current time zone used by the system in the container.
- `/tmp/logs/<service>:/srv/logs/` - Enables copying of the folder with service logs to your server `/tmp/logs/<service>` directory. You can change the directory where the logs will be saved according to your needs.

3.3.1.1 Service arguments

Each service in LUNA PLATFORM has its own launch arguments. These arguments can be passed through:

- Setting a flag for the launch script (`run.py`) of the corresponding service.
- Setting environment variables (`--env`) on the Docker command line.

Some arguments can only be passed by setting a flag. For the Handlers and Remote SDK services, it is possible to use the environment variable “`EXTEND_CMD`” to explicitly pass flags. See the example of using the “`EXTEND_CMD`” variable in the “[Run slim version of Remote SDK](#)” section.

For example, using the `--help` flag you can get a list of all available arguments. An example of passing an argument to an API service:

```
docker run --rm dockerhub.visionlabs.ru/luna/luna-api:v.6.23.0 python3 /srv/luna_api/run.py --help
```

List of main arguments:

Launch flag	Environment variable	Description
<code>--port</code>	<code>PORT</code>	Port on which the service will listen for connections.
<code>--workers</code>	<code>WORKER_COUNT</code>	Number of workers for the service.

<code>--log_suffix</code>	LOG_SUFFIX LOG_SUFFIX	Suffix added to log file names (with the option to write logs to a file enabled).
<code>--config-reload</code>	RELOAD_CONFIG	Enable automatic configuration reload. See “Automatic configurations reload” in the LUNA PLATFORM 5 administrator manual.
<code>--pulling-time</code>	RELOAD_CONFIG_INTERVAL	Configuration checking period (default 10 seconds). See “Automatic configurations reload” in the LUNA PLATFORM 5 administrator manual.
<code>--luna-config</code> <code>--luna-config</code>	CONFIGURATOR_HOST, CONFIGURATOR_PORT	Address of the Configurator service for downloading settings. For <code>--luna-config</code> it is sent in the format <code>http://localhost:5070/1</code> . For environment variables, the host and port are set explicitly. If the argument is not given, the default configuration file will be used.
<code>--config</code>	None	Path to the file with service configurations.
<code>--<config_name></code>	None	Tag of the specified configuration in the Configurator. When setting this configuration, the value of the tagged configuration will be used. Example: <code>--INFLUX_MONITORING TAG_1</code> Note: You must pre-tag the appropriate configuration in. Configurator. Note: Only works with the <code>--luna-config</code> flag.

The list of arguments may vary depending on the service.

It is also possible to override the settings of services at their start using environment variables.

The VL_SETTINGS prefix is used to redefine the settings. Examples:

- `--env=VL_SETTINGS.INFLUX_MONITORING.SEND_DATA_FOR_MONITORING=0`. Using the environment variable from this example will set the “SEND_DATA_FOR_MONITORING” setting for the INFLUX_MONITORING section to “0”.
- `--env=VL_SETTINGS.OTHER.STORAGE_TIME=LOCAL`. For non-compound settings (settings

that are located in the “OTHER” section in the configuration file), you must specify the “OTHER” prefix. Using the environment variable from this example will set the value of the “STORAGE_TIME” setting (if the service uses this setting) to “LOCAL”.

3.3.2 Creating DB parameters

Example command of launching containers for database migration or database creation:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/<service>:/srv/logs/ \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/<service-name>:<version> \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

The following parameters are used when launching containers for database migration or database creation:

Here:

- `--rm` - Sets if the container is deleted after all the specified scripts finish processing.
- `python3 ./base_scripts/db_create.py` - Sets Python version and a script `db_create.py` launched in the container. The script is used for the database structure creation.
- `--luna-config http://localhost:5070/1` - Sets where the launched script should receive configurations. By default, the service requests configurations from the Configurator service.

3.4 Ways to change service settings

There are three general ways to change settings stored in the Configurator DB:

- Using Configurator GUI.
- Using Configurator API requests.
- Using dump file.

Settings can be changed after the Configurator service was launched.

The settings of Configurator itself are stored in the “luna_configurator_postgres.conf” file that is downloaded to the Configurator container during its launch.

Configurator Interface

You can enter the configurator GUI and change setting. By default, the following address on localhost are used: <Configurator_server_address>:5070.

Configurator API

You can use Configurator API to update settings. See the Configurator OpenAPI specification.

Dump file

You can receive a dump file with all the LP services settings. Use one of the following commands:

```
wget -O /var/lib/luna/settings_dump.json 127.0.0.1:5070/1/dump
```

or

```
curl 127.0.0.1:5070/1/dump > /var/lib/luna/settings_dump.json
```

You should specify correct Configurator service address and port.

You should delete the “limitations” section from the file. You will not be able to apply your updated dump file if the section remains.

```
"limitations": [  
  ...  
],
```

Edit parameters in the “settings” section.

```
"settings": [  
  ...  
],
```

Copy the dump file to your Configurator container.

```
docker cp /var/lib/luna/settings_dump.json luna-configurator:/srv/
```

Apply the file:

```
docker exec luna-configurator python3 ./base_scripts/db_create.py --dump-  
file /srv/settings_dump.json
```

3.5 Logging to server

To enable saving logs to the server, you should:

- Create directories for logs on the server.
- Activate log recording and set the location of log storage inside LP service containers.
- Configure synchronization of log directories in the container with logs on the server using the `volume` argument at the start of each container.

3.5.1 Create logs directory

Below are examples of commands for creating directories for saving logs and assigning rights to them for all LUNA PLATFORM services.

```
mkdir -p /tmp/logs/configurator /tmp/logs/image-store /tmp/logs/accounts /  
tmp/logs/faces /tmp/logs/licenses /tmp/logs/events /tmp/logs/python-  
matcher /tmp/logs/handlers /tmp/logs/remote-sdk /tmp/logs/tasks /tmp/logs  
/tasks-worker /tmp/logs/sender /tmp/logs/api /tmp/logs/admin /tmp/logs/  
backport3 /tmp/logs/backport4
```

```
chown -R 1001:0 /tmp/logs/configurator /tmp/logs/image-store /tmp/logs/  
accounts /tmp/logs/faces /tmp/logs/licenses /tmp/logs/events /tmp/logs/  
python-matcher /tmp/logs/handlers /tmp/logs/remote-sdk /tmp/logs/tasks /  
tmp/logs/tasks-worker /tmp/logs/sender /tmp/logs/api /tmp/logs/admin /tmp  
/logs/backport3 /tmp/logs/backport4
```

If you need to use the Python Matcher Proxy service, then you need to additionally create the `/tmp/logs/python-matcher-proxy` directory and set its permissions.

3.5.2 Logging activation

3.5.2.1 LP services logging activation

To enable logging to file, you need to set the `log_to_file` and `folder_with_logs` settings in the `<SERVICE_NAME>_LOGGER` section of the settings for each service.

Automatic method (before/after starting Configurator)

To update logging settings, you can use the `logging.json` settings file provided with the distribution package.

Run the following command after starting the Configurator service:

```
docker cp /var/lib/luna/current/extras/conf/logging.json luna-configurator:/
srv/luna_configurator/used_dumps/logging.json
```

Update your logging settings with the copied file.

```
docker exec -it luna-configurator python3 ./base_scripts/db_create.py --dump
-file /srv/luna_configurator/used_dumps/logging.json
```

Manual method (after starting Configurator)

Go to the Configurator service interface (127.0.0.1:5070) and set the logs path in the container in the `folder_with_logs` parameter for all services whose logs need to be saved. For example, you can use the path `/srv/logs`.

Set the `log_to_file` option to `true` to enable logging to a file.

3.5.2.2 Configurator service logging activation (before/after Configurator start)

The Configurator service settings are not located in the Configurator user interface, they are located in the following file:

```
/var/lib/luna/current/example-docker/luna_configurator/configs/
luna_configurator_postgres.conf
```

You should change the logging parameters in this file before starting the Configurator service or restart it after making changes.

Set the path to the logs location in the container in the `FOLDER_WITH_LOGS = ./` parameter of the file. For example, `FOLDER_WITH_LOGS = /srv/logs`.

Set the `log_to_file` option to `true` to enable logging to a file.

3.5.3 Mounting directories with logs when starting services

The log directory is mounted with the following argument when starting the container:

```
-v <server_logs_folder>:<container_logs_folder> \
```

where `<server_logs_folder>` is the directory created in the [create logs directory](#) step, and `<container_logs_folder>` is the directory created in the [activate logging](#) step.

Example of command to launch the API service with mounting a directory with logs:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5000 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--name=luna-api \  
--restart=always \  
--detach=true \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/api:/srv/logs \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-api:v.6.23.0
```

The example container launch commands in this documentation contain these arguments.

3.6 Docker log rotation

To limit the size of logs generated by Docker, you can set up automatic log rotation. To do this, add the following data to the `/etc/docker/daemon.json` file:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m",
    "max-file": "5"
  }
}
```

This will allow Docker to store up to 5 log files per container, with each file being limited to 100MB.

After changing the file, you need to restart Docker:

```
systemctl reload docker
```

The above changes are the default for any newly created container, they do not apply to already created containers.

3.7 Set custom InfluxDB settings

If you are going to use InfluxDB OSS 2, then you need to update the monitoring settings in Configurator service.

There are the following settings for InfluxDB OSS 2:

```
"send_data_for_monitoring": 1,
"use_ssl": 0,
"flushing_period": 1,
"host": "127.0.0.1",
"port": 8086,
"organization": "<ORGANIZATION_NAME>",
"token": "<TOKEN>",
"bucket": "<BUCKET_NAME>",
"version": <DB_VERSION>
```

You can update InfluxDB settings in the Configurator service by following these steps:

- Open the following file:

```
vi /var/lib/luna/current/extras/conf/influx2.json
```

- Set required data.
- Save changes.
- Copy the file to the influxDB container:

```
docker cp /var/lib/luna/current/extras/conf/influx2.json luna-configurator:/srv/
```

- Update settings in the Configurator.

```
docker exec -it luna-configurator python3 ./base_scripts/db_create.py --dump
-file /srv/influx2.json
```

You can also manually update settings in the Configurator service user interface.

The Configurator service configurations are set separately.

- Open the file with the Configurator configurations:

```
vi /var/lib/luna/current/example-docker/luna_configurator/configs/
luna_configurator_postgres.conf
```

- Set required data.
- Save changes.
- Restart Configurator:

```
docker restart luna-configurator
```

3.8 Use Python Matcher with Python Matcher Proxy

As mentioned earlier, along with the Python Matcher service, you can additionally use the Python Matcher Proxy service, which will redirect matching requests either to the Python Matcher service or to the matching plugins. Plugins may significantly improve matching processing performance. For example, it is possible to organize the storage of the data required for matching operations and additional objects fields in separate storage using plugins, which will speed up access to the data compared to the use of the standard LUNA PLATFORM database.

To use the Python Matcher service with Python Matcher Proxy, you should additionally launch the appropriate container, and then set a certain setting in the Configurator service. Follow the steps below only if you are going to use matching plugins.

See the description and usage of matching plugins in the administrator manual.

3.8.1 Python Matcher proxy container launch

Use the following command to launch the service:

After starting the container, you need to set the "luna_matcher_proxy":true parameter in the "ADDITIONAL_SERVICES_USAGE" section in the Configurator service.

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5110 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--env=SERVICE_TYPE="proxy" \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/python-matcher-proxy:/srv/logs \  
--name=luna-python-matcher-proxy \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-python-matcher:v.1.8.2
```

After launching the container, you need to set the following value in the Configurator service.

```
ADDITIONAL_SERVICES_USAGE = "luna_matcher_proxy":true
```

3.9 System scaling

All LP services are linearly scalable and can be located on several services.

You can run additional containers with LP services to improve performance and fail-safety. The number of services and the characteristics of servers depend on your tasks.

To increase performance, you may either improve the performance of a single server or increase the number of servers used by distributing most resource-intensive components of the system.

Balancers are used for the distribution of requests among the launched service instances. This approach provides the necessary processing speed and the required fail-safety level for specific customer's tasks. In the case of a node failure, the system will not stop: requests will be redirected to another node.

The image below shows two instances of the Faces service balanced by Nginx. Nginx receives requests on port 5030 and routes them to Faces instances. The faces services are launched on ports 5031 and 5032.

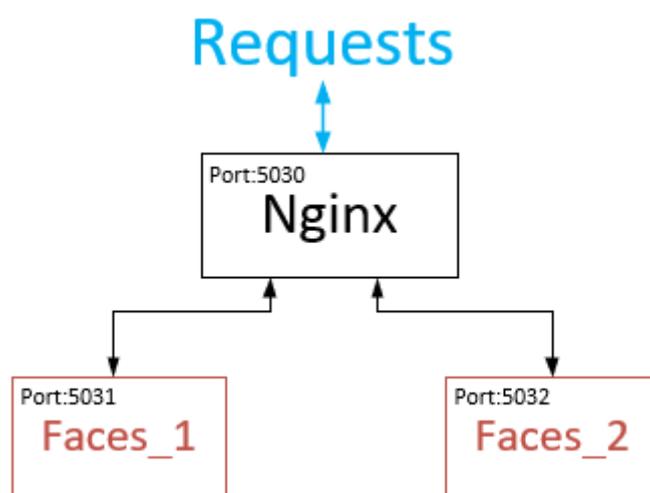


Figure 1: Faces service balancing

It is strongly recommended to regularly back up databases to a separate server regardless of the fail-safety level of the system. It allows you not to lose data in case of unforeseen circumstances.

MQs, databases, and balancers used by LUNA PLATFORM are products of third-party developers. You should configure them according to the recommendations of the corresponding vendors.

The Remote SDK service and the Python Matcher service perform the most resource-intensive operations.

The Remote SDK service performs mathematical image transformations and descriptors extraction. The operations require significant computational resources. Both CPU and GPU can be used for computations.

GPU usage is preferable since it improves the processing of requests. However, not all types of video cards are supported.

The Python Matcher service performs matching with lists. Matching requires CPU resources, however, you also should allocate as much RAM as possible for each Python Matcher instance. The RAM is used to store descriptors received from a database. Thus matcher does not require to request each descriptor from the database.

When distributing instances on several servers, you should consider the performance of each server. For example, if a large task is executed by several Python Matcher instances, and one of the instances is on the server with low performance, this can slow down the execution of the entire task.

For each instance of the service, you can set the number of workers. The greater the number of workers, the more resources and memory are consumed by the service instance. See the detailed information in the “Worker processes” section of the LUNA PLATFORM administrator manual.

3.9.1 Nginx

Nginx is required when you use several instances of LUNA PLATFORM services.

3.9.1.1 Nginx configuration

The Nginx configuration file includes parameters for balancing of requests to API, Faces, Image Store and Events.

Check the “/var/lib/luna/current/extras/conf/nginx.conf” configuration file before running NGINX.

Nginx listens requests on the specified ports and sends the requests to the available instances of services. The following servers and ports are set in Nginx configuration file.

Service name	Port set in Nginx	Ports for the launched services
API	5000	5001-5004
Faces	5030	5031-5034
Image Store	5020	5021-5024
Events	5040	5041-5044

3.9.1.2 Run Nginx container

Note: You should configure the file for using with your launched services.

Use the `docker run` command with these parameters to launch the Nginx container:

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /var/lib/luna/current/extras/conf/nginx.conf:/etc/nginx/nginx.conf \
--name=nginx \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/nginx:1.17.4-alpine
```

Here:

- `-v /var/lib/luna/current/extras/conf/nginx.conf:/etc/nginx/nginx.conf` - Nginx configuration file used for Nginx launching.

See the example of Nginx configuration file for more services instances launch here: `“/var/lib/luna/current/extras/co`

3.9.2 Launching several containers

There are two steps required for launching several instances of the same LP service

1. Run several containers of the service.

You must launch the required number of service by using the corresponding command for the service.

For example, for the API service you must run the following command with updated parameters.

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=<port> \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/<folder_name>:/srv/logs \
--name=<name> \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-api:v.6.23.0
```

When running several similar containers the following parameters of the containers must differ:

- `--env=PORT=<port>` - Specified port for similar containers must differ. You must specify an available port for the instance. For example, “5001”, “5002”. The “5000” port will be specified for the Nginx balancer.
- `/tmp/logs/<folder_name>:/srv/logs` - Specified folder name for logs must differ to distinguish logs for different service instances.

- `--name=<container_name>` - Name of the launched container must differ as it is prohibited to launch two containers with the same name. For example, “api_1”, “api_2”.
- `--gpus device=0` - CORE services usually utilize different GPU devices. Thus you should specify different device numbers.

2. Configure your balancer (e.g., Nginx) for routing requests to the services.

For each scaled LP service, you must set a port where Nginx will listen to service requests and real ports of each service instance where Nginx will redirect the requests.

An example of Nginx configuration file can be found here:

“`/var/lib/luna/current/extras/conf/nginx.conf`”.

You can use another balancer, but its utilization is not described in this documentation.

3.10 Improve performance

Follow these steps if you need to improve the performance of LUNA PLATFORM in Docker containers.

Docker security policies impose restrictions on the network interaction of LP services. If you perform the listed below actions and disable the policies, a new network connection of docker containers will be established approximately 20% faster.

Changing the settings may have a negative impact on security. Do not change the settings if are not an advanced Docker user.

Docker must be installed before performing these actions. See “[Docker installation](#)”.

1. Add `--network=host --security-opt seccomp=unconfined` parameters to the launch of each LP service container. The launch commands are described in section “[Services launch](#)”.

The example below is given for the Faces service.

```
docker run --env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5031 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
--name=luna-faces1 \  
--restart=always \  
--detach=true \  
--network=host \  
--security-opt seccomp=unconfined \  
dockerhub.visionlabs.ru/luna/luna-faces:v.4.10.2
```

2. Add "userland-proxy": false to the Docker configuration.

Create configuration file for Docker.

```
echo 'DOCKER_OPTS="--config-file=/etc/docker/daemon.json"' > /etc/default/  
docker
```

Create the file.

```
vi /etc/docker/daemon.json
```

Add the following option to the opened file.

```
{  
  "userland-proxy": false  
}
```

Restart Docker.

```
systemctl restart docker
```

3.11 External DB

This section describes commands required for configuring external PostgreSQL for working with LP. External means that you already have working DB and MQ and want to use them with LP.

You need to specify your external DB and MQ in the configurations of LP services.

Skip this section if you are using PostgreSQL and RabbitMQ from docker containers!

After you perform all the commands from this sections go back to the “[Luna Configurator](#)” section.

Your database user and other parameters may differ. Consider it during the commands execution.

The sections below describes the databases creation. You should use the “db_create.py” script to create the tables structure, after the table is created. The script launch from the container is described for each service in sections above. E. g. see the “[Faces DB tables creation](#)”.

3.11.1 PostgreSQL user creation

Go to the directory.

```
cd /var/
```

Create a database user.

```
runuser -u postgres -- psql -c 'create role luna;'
```

Assign a password to the user.

```
runuser -u postgres -- psql -c "ALTER USER luna WITH PASSWORD 'luna';"
```

3.11.2 Configurator DB creation

Create the database for the Configurator service. It is assumed that the DB user is already created.

Go to the directory.

```
cd /var/
```

- Create the database.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_configurator;'
```

- Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_configurator TO luna;'
```

- Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

3.11.3 Accounts DB creation

Create the database for the Accounts service. It is assumed that the DB user is already created.

Go to the directory.

```
cd /var/
```

- Create the database.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_accounts;'
```

- Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_accounts TO luna;'
```

- Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

3.11.4 Handlers DB creation

Create the database for the Handlers service. It is assumed that the DB user is already created.

Go to the directory.

```
cd /var/
```

Create a database.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_handlers;'
```

Grant privileges to the database and the user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_handlers TO luna;'
```

Enable the user to login into DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

3.11.5 Backport 3 DB creation

Create the database for the Backport 3 service. It is assumed that the DB user is already created.

Go to the directory.

```
cd /var/
```

Create a database.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_backport3;'
```

Grant privileges to the database and the user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE  
luna_backport3 TO luna;'
```

Enable the user to login into DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

3.11.6 Faces DB creation

Create the database for the Faces service. It is assumed that the DB user is already created.

Go to the directory.

```
cd /var/
```

Create a database.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_faces;'
```

Grant privileges to the database and the user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE luna_faces  
TO luna;'
```

Enable the user to login into DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

3.11.7 Create VLMatch function for Faces DB

The Faces service requires the VLMatch additional function to be added to the utilized database. LUNA PLATFORM cannot perform matching calculations without this function. The VLMatch function can be added to the PostgreSQL or Oracle database.

The VLMatch library is compiled for your particular database version.

Do not use the library built for another version of DB. For example, the library build for the PostgreSQL of version 12 cannot be used for the PostgreSQL of version 9.6.

This section describes the function creation for PostgreSQL.

The instruction for the Oracle DB is given in the “[VLMatch for Oracle](#)” section.

3.11.7.1 Build VLMatch

Note: The following instruction describes installation for PostgreSQL 16.

You can find all the required files for the VLMatch user-defined extension (UDx) compilation in the following directory:

```
/var/lib/luna/current/extras/VLMatch/postgres/
```

The following instruction describes installation for PostgreSQL 16.

For VLMatch UDx function compilation one needs to:

1. Install RPM repository:

```
dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

2. Install PostgreSQL:

```
dnf install postgresql16-server
```

3. Install the development environment:

```
dnf install postgresql16-devel
```

4. Install the gcc package:

```
dnf install gcc-c++
```

5. Install CMAKE. The version 3.5 or higher is required.

6. Open the make.sh script using a text editor. It includes paths to the currently used PostgreSQL version. Change the following values (if necessary):

SDK_HOME specifies the path to PostgreSQL home directory. Default value: `/usr/include/postgresql/16/server;`

LIB_ROOT specifies the path to PostgreSQL library root directory. Default value: `/usr/lib/postgresql/16/lib.`

7. Go to the make.sh script directory and run it:

```
cd /var/lib/luna/current/extras/VLMatch/postgres/
```

```
chmod +x make.sh
```

```
./make.sh
```

3.11.7.2 Add VLMatch function to Faces database

The VLMatch function should be applied to the PostgreSQL DB.

- Define the function inside the Faces database.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_faces -c "CREATE FUNCTION
  VLMatch(bytea, bytea, int) RETURNS float8 AS 'VLMatchSource.so', 'VLMatch
  ' LANGUAGE C PARALLEL SAFE;"
```

- Test function by sending re following request to the service database.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_faces -c "SELECT VLMatch('\
x1234567890123456789012345678901234567890123456789012345678901234'::bytea
, '\x0123456789012345678901234567890123456789012345678901234567890123'::
bytea, 32);" 
```

The result returned by the database must be “0.4765625”.

3.11.8 Events DB creation

Create the database for the Events service. It is assumed that the DB user is already created.

Go to the directory.

```
cd /var/
```

Create database.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_events;'
```

Grant privileges to database and the user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE luna_events
TO luna;'
```

Enable the user to login into DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

3.11.9 Create VLMatch function for Events DB

Note: Do not use the library built for another version of DB. For example, the library build for the PostgreSQL of version 12 cannot be used for the PostgreSQL of version 9.6.

The Events service requires the VLMatch additional function to be added to the utilized database. LUNA PLATFORM cannot perform matching calculations without this function. The VLMatch function can be added to the PostgreSQL.

The VLMatch library is compiled for your particular database version.

This section describes the function creation for PostgreSQL. If you use the PostgreSQL database, you have already created and moved the created library during the Faces service launch. See section [“Build VLMatch for PostgreSQL”](#).

3.11.9.1 Add VLMatch function to Events database

The VLMatch function should be applied to the PostgreSQL DB.

Define the function inside the Events database.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_events -c "CREATE FUNCTION
  VLMatch(bytea, bytea, int) RETURNS float8 AS 'VLMatchSource.so', 'VLMatch
  ' LANGUAGE C PARALLEL SAFE;"
```

Test function within call.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_events -c "SELECT VLMatch('\
  x1234567890123456789012345678901234567890123456789012345678901234':::bytea
  , '\x0123456789012345678901234567890123456789012345678901234567890123':::
  bytea, 32);" 
```

The result returned by the database must be “0.4765625”.

3.11.10 PostGIS installation for Events

The Events service requires PostGIS for working with coordinates.

This instruction describes the PostGIS installation for PostgreSQL 12 database. The PostGIS version depends on the PostgreSQL version.

- Install epel-release for access to extended package repository (necessary for RabbitMQ and other dependencies).

```
yum -y install epel-release
```

- Install PostGIS.

```
yum -y install postgis25_12
```

- Activate PostGIS in your database.

```
sudo -u postgres -h 127.0.0.1 -- psql -d luna_events -c "CREATE EXTENSION postgis;"
```

See additional information about PostGIS on its website: <https://postgis.net/>

3.11.11 Tasks DB creation

Create the database for the Tasks service. It is assumed that the DB user is already created.

Go to the directory.

```
cd /var/
```

Create the database.

```
runuser -u postgres -- psql -c 'CREATE DATABASE luna_tasks;'
```

Grant privileges to the database user.

```
runuser -u postgres -- psql -c 'GRANT ALL PRIVILEGES ON DATABASE luna_tasks TO luna;'
```

Allow user to authorize in the DB.

```
runuser -u postgres -- psql -c 'ALTER ROLE luna WITH LOGIN;'
```

3.12 VLMatch library compilation for Oracle

Note: The following instruction describes installation for Oracle 21c.

You can find all the required files for the VLMatch user-defined extension (UDx) compilation in the following directory:

```
/var/lib/luna/current/extras/VLMatch/oracle
```

For VLMatch UDx function compilation one needs to:

1. Install required environment, see [requirements](#):

```
sudo yum install gcc g++
```

2. Change SDK_HOME variable - oracle sdk root (default is \$ORACLE_HOME/bin, check \$ORACLE_HOME environment variable is set) in the makefile:

```
vi /var/lib/luna/current/extras/VLMatch/oracle/make.sh
```

3. Open the directory and run the file "make.sh".

```
cd /var/lib/luna/current/extras/VLMatch/oracle
```

```
chmod +x make.sh
```

```
./make.sh
```

4. Define the library and the function inside the database (from database console):

```
CREATE OR REPLACE LIBRARY VLMatchSource AS '$ORACLE_HOME/bin/VLMatchSource.so';
CREATE OR REPLACE FUNCTION VLMatch(descriptorFst IN RAW, descriptorSnd IN
  RAW, length IN BINARY_INTEGER)
  RETURN BINARY_FLOAT
AS
  LANGUAGE C
  LIBRARY VLMatchSource
  NAME "VLMatch"
  PARAMETERS (descriptorFst BY REFERENCE, descriptorSnd BY REFERENCE,
    length UNSIGNED SHORT, RETURN FLOAT);
```

5. Test function within call (from database console):

```
SELECT VLMatch(HEXTORAW('
1234567890123456789012345678901234567890123456789012345678901234'),
HEXTORAW('
0123456789012345678901234567890123456789012345678901234567890123'), 32)
FROM DUAL;
```

The result returned by the database must be “0.4765625”.