



**VisionLabs**  
MACHINES CAN SEE

# **VisionLabs LUNA PLATFORM 5**

**Migration from 3.3.8 to v.5.56.0**

**v.5.56.0**

## Contents

<b>Default ports for services</b>	<b>5</b>
<b>Configuration names for services</b>	<b>6</b>
<b>System requirements</b>	<b>7</b>
Processors . . . . .	7
CPU . . . . .	7
GPU . . . . .	7
Third-party applications . . . . .	8
<b>Introduction</b>	<b>9</b>
<b>1 Before upgrade</b>	<b>11</b>
1.1 Backups creation . . . . .	12
1.2 Delete old symbolic link . . . . .	12
1.3 Distribution unpacking . . . . .	12
1.4 Symbolic link creation . . . . .	13
1.5 Changing group and owner for directories . . . . .	13
1.6 Move Image Store buckets . . . . .	13
1.7 SELinux and Firewall . . . . .	14
1.8 Create log directory for new services . . . . .	14
1.9 License activation . . . . .	15
1.9.1 Actions from License activation manual . . . . .	15
1.10 Docker installation . . . . .	15
1.11 Calculations using GPU . . . . .	16
1.12 Login to registry . . . . .	17
<b>2 Services launch</b>	<b>18</b>
2.1 Monitoring configuration . . . . .	19
2.1.1 Migration from version 1 . . . . .	19
2.1.2 InfluxDB OSS 2 . . . . .	19
2.2 Run third-party services . . . . .	21
2.2.1 PostgreSQL . . . . .	21
2.2.2 Redis . . . . .	22
2.3 Configurator . . . . .	23
2.3.1 Optional services usage . . . . .	23
2.3.2 Configurator DB tables creation . . . . .	23
2.3.3 Run Configurator container . . . . .	24

2.4	Migration from LUNA PLATFORM 3 to Backport 3 . . . . .	25
2.4.1	Edit configuration file . . . . .	25
2.4.2	Before migration . . . . .	25
2.4.3	Faces DB creation for LUNA PLATFORM 5 . . . . .	26
2.4.4	Change the utilized DB . . . . .	26
2.4.5	Faces DB tables creation . . . . .	26
2.4.6	Backport 3 DB tables creation . . . . .	27
2.4.7	Accounts DB tables creation . . . . .	27
2.4.8	Migration launch . . . . .	27
2.4.9	Migration script description . . . . .	28
2.4.10	Stop LUNA PLATFORM 3 services . . . . .	30
2.5	Image Store . . . . .	31
2.5.1	Samples migrations . . . . .	31
2.5.2	Portraits migration . . . . .	31
2.5.3	Image Store container launch . . . . .	32
2.5.4	Buckets creation . . . . .	32
2.6	Accounts . . . . .	35
2.6.1	Accounts container launch . . . . .	35
2.7	Licenses . . . . .	35
2.7.1	Specify license settings using Configurator . . . . .	35
2.7.2	Licenses container launch . . . . .	36
2.8	Faces . . . . .	37
2.8.1	Faces container launch . . . . .	37
2.9	Events . . . . .	38
2.9.1	Events DB tables creation . . . . .	38
2.9.2	Events container launch . . . . .	38
2.10	Python Matcher services . . . . .	39
2.10.1	Use Python Matcher without Python Matcher Proxy . . . . .	39
2.10.2	Python Matcher container launch . . . . .	39
2.11	Remote SDK . . . . .	40
2.11.1	Remote SDK container launch . . . . .	40
2.12	Handlers . . . . .	45
2.12.1	Handlers DB tables creation . . . . .	45
2.12.2	Handlers container launch . . . . .	45
2.13	Tasks . . . . .	46
2.13.1	Tasks DB tables creation . . . . .	46
2.13.2	Tasks and Tasks Worker containers launch . . . . .	46
2.14	Sender . . . . .	48
2.14.1	Sender container launch . . . . .	48

2.15	API . . . . .	49
2.15.1	API container launch . . . . .	49
2.16	Admin . . . . .	50
2.16.1	Admin container launch . . . . .	50
2.17	Backport 3 . . . . .	51
2.17.1	Backport 3 container launch . . . . .	51
2.17.2	User Interface 3 . . . . .	52
2.18	Lambda . . . . .	53
2.18.1	Prepare Docker registry . . . . .	53
2.18.2	Create Lambda database . . . . .	54
2.18.3	Lambda DB tables creation . . . . .	54
2.18.4	Lambda container launch . . . . .	54
<b>3</b>	<b>Additional information</b>	<b>56</b>
3.1	Monitoring and logs visualization using Grafana . . . . .	57
3.1.1	LUNA Dashboards . . . . .	57
3.1.2	Grafana Loki . . . . .	57
3.2	Docker commands . . . . .	59
3.2.1	Show containers . . . . .	59
3.2.2	Copy files to container . . . . .	59
3.2.3	Enter container . . . . .	59
3.2.4	Images names . . . . .	59
3.2.5	Delete image . . . . .	59
3.2.6	Stop container . . . . .	60
3.2.7	Delete container . . . . .	60
3.3	Launching parameters description . . . . .	62
3.3.1	Launching services parameters . . . . .	62
3.3.2	Creating DB parameters . . . . .	65
3.4	Logging to server . . . . .	66
3.4.1	Create logs directory . . . . .	66
3.4.2	Logging activation . . . . .	66
3.4.3	Mounting directories with logs when starting services . . . . .	67
3.5	Docker log rotation . . . . .	69
3.6	Set custom InfluxDB settings . . . . .	70
3.7	Use Python Matcher with Python Matcher Proxy . . . . .	72
3.7.1	Python Matcher proxy container launch . . . . .	72
3.8	System scaling . . . . .	73
3.8.1	Launching several containers . . . . .	74
3.9	VLMatch library compilation for Oracle . . . . .	76

## 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 steps for upgrading from LUNA PLATFORM 3 distribution (version 3.3.8) to LUNA PLATFORM 5 with Backport 3 service. See the “Backports” section in administrator manual for information about the Backport 3 service.

The database migration procedures are performed using a script. The script was tested on the LUNA PLATFORM 3 of version 3.3.8. It was not tested on the other LUNA PLATFORM 3 versions. See [“Migration from LUNA PLATFORM 3 to Backport 3”](#).

You should update LUNA PLATFORM to the version 3.3.8 if you have an earlier version.

This instruction describes migration from Aerospike and PostgreSQL (LUNA PLATFORM 3) databases to the PostgreSQL (LUNA PLATFORM 5) databases and full installation of LUNA PLATFORM 5. The instruction provides an example of commands for migrating the PostgreSQL database from version 9.6 running on the server to version 16 running in a Docker container. If necessary, you can migrate to the version 16 running on the server as a service (not described in this documentation).

This document describes migration from LUNA PLATFORM 3.3.8 installed in the default configuration. Note that your LUNA PLATFORM configuration and scaling may differ. In this case, use this manual as an example of the general approach to LUNA PLATFORM migration.

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.

The document describes installation of all the services on a single service.

You should install InfluxDB if monitoring is required.

For a successful upgrade, you need to perform the actions from the sections [“Before upgrade”](#) 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.

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.

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.

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>

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 upgrade

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

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

1. [Create backups.](#)
2. [Delete old symbolic link.](#)
3. [Unpack the distribution of the new version of LUNA PLATFORM.](#)
4. [Create new symbolic link.](#)
5. [Change group and owner for new directories.](#)
6. [Move Image Store buckets.](#)
7. [Configure SELinux and Firewall](#) if not previously configured.
8. [Create log directories for new services](#), if logging to a file was previously used.
9. [Activate license.](#)
10. [Install Docker.](#)
11. [Set up GPU computing](#) if you plan to use GPU.
12. [Login to VisionLabs registry](#) if authorization was not previously performed.

## 1.1 Backups creation

Create backups for all the databases used with LUNA PLATFORM before performing the migration procedures. You can restore your data if any problems occur during the migration.

It is recommended to create backups for Image Store buckets.

Backups creation for databases and buckets is not described in this document.

## 1.2 Delete old symbolic link

Go to the “luna” directory.

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

Delete the “current” symbolic link.

```
rm -f current
```

## 1.3 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.

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
```

## 1.4 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
```

## 1.5 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
```

## 1.6 Move Image Store buckets

LUNA PLATFORM 5 is supposed to store buckets in the root directory /var/lib/luna/ to simplify the process of subsequent updates.

Create a directory to store Image Store buckets.

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

Move the contents of the Image Store bucket directory to the new bucket storage directory.

```
mv /var/lib/luna/luna_v.3.3.8/luna-image-store/luna_image_store/  
local_storage/* /var/lib/luna/image_store
```

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

```
chown -R 1001:0 /var/lib/luna/image_store
```

## 1.7 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.**

## 1.8 Create log directory for new services

**Skip this section if no logs were previously stored on the server.**

In the version of LUNA PLATFORM 5, new services have appeared for which you need to create directories with logs.

See “[Logging to server](#)” section if you have not previously used logging to a file, but want to enable it.

Following are the commands to create directories for all existing services. These commands will create and assign permissions only to missing directories.

```
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.

## 1.9 License activation

To activate/upgrade the license, follow these steps:

- Follow the steps from [license activation manual](#).
- Set settings for [HASP](#) license or [Guardant](#) license before starting Licenses container.

### 1.9.1 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.

## 1.10 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
```

## 1.11 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.



## 1.12 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.

## 2 Services launch

This section gives examples for:

- Databases tables migration.
- 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.

Next, you need to launch the Backport 3 service and its user interface:

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

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 Migration from version 1

If necessary, you can upgrade from the InfluxDB OSS 1 version.

The process of migrating InfluxDB from version 1 is not described in this documentation. InfluxDB provides built-in tools for migrating from version 1 to version 2. See the documentation:

<https://docs.influxdata.com/influxdb/v2.0/upgrade/v1-to-v2/docker/>

### 2.1.2 InfluxDB OSS 2

You can use InfluxDB OSS 2 as a service, or run it in a Docker container.

If you plan to use InfluxDB OSS 2 as a service, skip this step and make sure you have migrated from InfluxDB OSS 1.

To run InfluxDB OSS 2 in a Docker container, follow the steps below:

- Stop the InfluxDB service.

```
systemctl stop influxdb.service
```

- Run InfluxDB OSS 2 in Docker container.

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

#### 2.2.1.1 Migrate PostgreSQL 9.6 to PostgreSQL 16

In LUNA PLATFORM 5, the VisionLabs image for PostgreSQL has been updated from version 9.6 to version 16.

If this image was previously used, then you need to perform the migration yourself according to [official documentation](#). If necessary, you can continue using PostgreSQL 9.6.

Mounting PostgreSQL 9.6 data into a container for PostgreSQL 16 will result in an error.

#### 2.2.1.2 Launch PostgreSQL

**Note:** Make sure that the old PostgreSQL is deleted.

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-vlmatch: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 Migration from LUNA PLATFORM 3 to Backport 3

This section describes accounts, descriptors, and persons migration from LUNA PLATFORM 3 databases to LUNA PLATFORM 5 databases.

### 2.4.1 Edit configuration file

You need to set up the following configuration file before starting the migration:

```
vi /var/lib/luna/current/extras/conf/migration_config.conf
```

Enter the following information:

- API, Faces, Broker services general parameters (LUNA PLATFORM 3).
- Backport 3 and Faces services general parameters (LUNA PLATFORM 5).
- Luna Image Store general parameters.
- Logging parameters.

### 2.4.2 Before migration

1. The server where the migrations script is launched should have a connection to all the specified databases and the Broker service.
2. Make sure that the Broker service (LUNA PLATFORM 3) is launched. It is used to get descriptors from the database.

It is not necessary to launch the LUNA PLATFORM 5 services to perform migrations.

The Faces (LUNA PLATFORM 3) and Faces (LUNA PLATFORM 5) database names are similar by default (luna\_faces). You should fix it using one of the following ways:

- You can use another PostgreSQL for creating a new database for the Faces service (LUNA PLATFORM 5) hence the “luna\_faces” name will not be changed. In this case, you should specify the new address of the PostgreSQL database in the “LUNA\_FACES\_DB” section of the Faces configuration file/Configurator.
- You can create a new database for LUNA PLATFORM 5 (for example, “luna\_faces\_5”) and change the default name for the Faces DB in the “LUNA\_FACES\_DB” section of the Faces/Configurator configuration file.

The second method is described in this manual below.

Make sure that the databases for the Backport 3 service (LUNA PLATFORM 5) and the Faces service (LUNA PLATFORM 5) are empty (there are no entries) before starting the migration.

Do not launch the creation of database tables for the Faces service before changing the database name/PostgreSQL address in the “LUNA\_FACES\_DB” section of the Faces service configuration file/Configurator. Otherwise, you can lose the data stored in the Faces database of LUNA PLATFORM 3.

### 2.4.3 Faces DB creation for LUNA PLATFORM 5

Create a new database.

```
docker exec -it postgres psql -U luna -c "CREATE DATABASE luna_faces_5;"
```

Grant privileges to the database user.

```
docker exec -it postgres psql -U luna -c "GRANT ALL PRIVILEGES ON DATABASE  
luna_faces_5 TO luna;"
```

Allow user to authorize in the DB.

```
docker exec -it postgres psql -U luna -c "ALTER ROLE luna WITH LOGIN;"
```

Add VLMatch function to perform matching.

```
docker exec -it postgres psql -U luna -d luna_faces_5 -c "CREATE OR REPLACE  
FUNCTION VLMatch(bytea, bytea, int) RETURNS float8 AS '/srv/VLMatchSource  
.so', 'VLMatch' LANGUAGE C PARALLEL SAFE;"
```

### 2.4.4 Change the utilized DB

Now you should specify the “luna\_faces\_5” DB name in the settings of the Faces service.

- Go to the Configurator service user interface ([http://<server\\_name>:5070](http://<server_name>:5070)).
- Select “luna-faces” in the “Service name” filter.
- Find the “LUNA\_FACES\_DB” group of settings.
- Set the "db\_name": "luna\_faces\_5" parameter.
- Press the [Save] button.

### 2.4.5 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
```

#### 2.4.6 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
```

#### 2.4.7 Accounts DB tables 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
```

#### 2.4.8 Migration launch

All the migration procedures are performed using the “start\_migration.py” script. See the “Migration script description” section below for additional information about the script.

Run the script using the Backport 3 container:

```
docker run \
--rm -t \
-v /tmp/logs/backport3:/srv/logs \
-v /var/lib/luna/current/extras/conf/migration_config.conf:/srv/base_scripts/migrate_backport3/config/config.conf \
-v /var/lib/luna/image_store:/local_storage \
--network=host \
--entrypoint bash dockerhub.visionlabs.ru/luna/luna-backport3:v.0.10.2 -c "
    cd ./base_scripts/migrate_backport3 && pip3 install -r requirements.txt
    && python3 ./start_migration.py"
```

Here:

- `-v /var/lib/luna/image_store:/local_storage` - Local storage with Image Store samples is added to the container. It is required for the samples migration.
- `cd ./base_scripts/migrate_backport3 && pip3 install -r requirements.txt && python3 ./start_migration.py` - The “requirements.txt” file is required to run the “start\_migration.py” script with a set of specified dependencies.
- `-v /tmp/logs/backport3:/srv/logs` - The script is saving logs to the /srv/logs directory by default. Then the logs are moved to the /tmp/logs/backport3 directory on your server.

The configuration file `/var/lib/luna/current/extras/conf/migration_config.conf` is added to the container for the script launching.

You can optionally use the `--skip_missing_descriptors` parameter, which will enable you to ignore missing descriptors in the LP 3 database.

### 2.4.9 Migration script description

Data transfer will be performed in the following order:

- Database migration from API (LUNA PLATFORM 3) service to Backport 3 (LUNA PLATFORM 5).
- Database migration from Faces (LUNA PLATFORM 3) to Backport 3 (LUNA PLATFORM 5).
- Database migration from Faces (LUNA PLATFORM 3) to Faces (LUNA PLATFORM 5).
- Updating descriptors in the Faces database (LUNA PLATFORM 5) from the CORE (LUNA PLATFORM 3) database.
- Add accounts to samples and optionally portraits in the Luna Image Store storage.

When migrating API and Faces service databases, all LP 3 accounts will be migrated and stored in the database of the Accounts service. All migrated accounts will be of type “user”. The fields “password”, “email” and “organization\_name” will be transferred to the “account” table of the Accounts database under new names - “password”, “login” and “description” respectively. Tokens

will stored in the Backport3 database, but their identifier will also be entered in the Accounts database, where the necessary permissions will be automatically set.

When the script is launched, the log files “luna-backport3\_ERROR\_migration.txt” and “luna-backport3\_WARNING\_migration.txt” are created. The files include information about all the errors and warnings that occurred during the migration process.

The files are saved to “/srv/logs/” directory of the container.

Run the following script to get help:

```
docker run --rm -t \
--network=host \
--entrypoint bash dockerhub.visionlabs.ru/luna/luna-backport3:v.0.10.2 -c "
    cd ./base_scripts/migrate_backport3 && pip3 install -r requirements.txt
    && python3 ./start_migration.py --help"
```

To start individual migration steps, pass the --migrate command line argument.

The argument takes the following parameters:

- stage\_1 - Migrate from API (LUNA PLATFORM 3) and Faces (LUNA PLATFORM 3) databases to Backport 3 (LUNA PLATFORM 5) database.
- stage\_2 - Migrate from Faces (LUNA PLATFORM 3) database to Faces (LUNA PLATFORM 5) database.
- stage\_3 - Update descriptors in the Faces (LUNA PLATFORM 5) from the CORE (LUNA PLATFORM 3) database.
- stage\_4 - Add accounts for samples and optionally portraits (--migrate\_portraits flag) in the Image Store service.
- all - Perform all the above steps.

Code example:

```
docker run \
--rm -t \
-v /tmp/logs/backport3:/srv/logs \
-v /var/lib/luna/current/extras/conf/migration_config.conf:/srv/base_scripts
/migrate_backport3/config/config.conf \
-v /var/lib/luna/image_store:/local_storage \
--network=host \
--entrypoint bash dockerhub.visionlabs.ru/luna/luna-backport3:v.0.10.2 -c "
    cd ./base_scripts/migrate_backport3 && pip3 install -r requirements.txt
    && python3 ./start_migration.py --migrate stage_1"
```

If you are migrating from `stage_3`, check that the “face” and “attribute” tables in the Faces (LUNA PLATFORM 5) database have entries.

If something went wrong during `stage_3`, use the argument `--lower_boundary` to specify the last failed face ID to continue migration. The face ID can be found in migration logs.

For example:

```
docker run --rm -t -v \  
/tmp/logs/backport3:/srv/logs \  
-v /var/lib/luna/current/extras/conf/migration_config.conf:/srv/base_scripts  
/migrate_backport3/config/config.conf \  
--network=host \  
--entrypoint bash dockerhub.visionlabs.ru/luna/luna-backport3:v.0.10.2 -c "  
  cd ./base_scripts/migrate_backport3 && pip3 install -r requirements.txt  
  && python3 ./start_migration.py --lower_boundary 02e7b0db-b3c3-4446-bbdd  
  -0f0d9a566058"
```

#### 2.4.10 Stop LUNA PLATFORM 3 services

Stop and disable all the LUNA PLATFORM 3 services.

```
systemctl stop luna-image-store luna-faces luna-broker luna-extractor@1 luna  
-matcher@1 luna-stat-lpse.service luna-stat-sm.service luna-api luna-  
admin_back luna-admin_tasks aerospike
```

```
systemctl disable luna-image-store luna-faces luna-broker luna-extractor@1  
luna-matcher@1 luna-stat-lpse.service luna-stat-sm.service luna-api luna-  
admin_back luna-admin_tasks aerospike
```

```
systemctl status luna-image-store luna-faces luna-broker luna-extractor@1  
luna-matcher@1 luna-stat-lpse.service luna-stat-sm.service luna-api luna-  
admin_back luna-admin_tasks aerospike
```

## 2.5 Image Store

### 2.5.1 Samples migrations

Samples migration is required to add an account for each sample.

Create a backup of all the samples buckets before launching the following script.

Implied that Image Store from LUNA PLATFORM 3 will be used with LUNA PLATFORM 5. Storage transfer not provided during the migration.

You should use existing buckets during the LUNA PLATFORM 5 Image Store launching.

Change the default bucket used for the samples storage to “visionlabs-warps”.

By default, the samples bucket in LUNA PLATFORM 3 was called “visionlabs-warps”.

- Go to the Configurator service user interface ([http://<server\\_name>:5070](http://<server_name>:5070)).
- Enter “LUNA\_IMAGE\_STORE\_FACES\_SAMPLES\_ADDRESS” in the “Setting name” filter.
- Specify the “bucket” name:

```
"bucket": "visionlabs-warps"
```

- Press the [Save] button.

### 2.5.2 Portraits migration

Backport uses samples as portraits by default. Thus it is not required to store portraits and samples simultaneously.

If samples will not be used as portraits, its required to migrate portraits (run migration with **--migrate\_portraits** flag).

If portraits are required, you should turn off the USE\_SAMPLES\_AS\_PORTRAITS setting of Backport 3.

You should follow one of these steps if you are going to use portraits:

- If you store portraits in the Image Store service (the “SEND\_TO\_LUNA\_IMAGE\_STORE” setting of the API service in LUNA PLATFORM 3 was enabled), you should set up the corresponding settings in the “LUNA\_IMAGE\_STORE\_PORTRAITS\_ADDRESS” section in the Backport 3 settings. You can find old values in the “LUNA\_IMAGE\_STORE\_ORIGIN” and “LUNA\_IMAGE\_STORE\_BUCKET” settings in the configuration file of the API service (LUNA PLATFORM 3)
- If you had used a plugin to store portraits, you should move the existing portraits to the Image Store service of LUNA PLATFORM 5. See the “Image Store” section in the administrator manual for details about creating a bucket and saving images to the service.

You can configure all the listed settings in the Configurator service of LUNA PLATFORM 5 or configuration files of the corresponding services (if the Configurator service is not utilized).

### 2.5.3 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/`.

### 2.5.4 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
```

## 2.6 Accounts

### 2.6.1 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
```

## 2.7 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.

### 2.7.1 Specify license settings using Configurator

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

#### 2.7.1.1 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/](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.

### 2.7.1.2 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.

## 2.7.2 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
```

## 2.8 Faces

### 2.8.1 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
```

## 2.9 Events

### 2.9.1 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
```

### 2.9.2 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
```

## 2.10 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.

### 2.10.1 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.

### 2.10.2 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
```

## 2.11 Remote SDK

### 2.11.1 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.

#### 2.11.1.1 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
```

#### 2.11.1.2 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.

### 2.11.1.3 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
```

## 2.12 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”](#).

### 2.12.1 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
```

### 2.12.2 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
```

## 2.13 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”](#).

### 2.13.1 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
```

### 2.13.2 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.

#### 2.13.2.1 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
```

#### 2.13.2.2 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
```

## 2.14 Sender

### 2.14.1 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
```



## 2.15 API

### 2.15.1 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
```

## 2.16 Admin

### 2.16.1 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
```

## 2.17 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.

### 2.17.1 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
```

## 2.17.2 User Interface 3

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

### 2.17.2.1 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.

## 2.18 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 Create Lambda database

Use the following command to create a Lambda database in PostgreSQL:

```
docker exec -it postgres psql -U luna -c "CREATE DATABASE luna_lambda;"
```

### 2.18.3 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
```

### 2.18.4 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.](#)
- [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.](#)
- [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.

**Note:** Before updating, make sure that the old LUNA Dashboards container is deleted.

##### 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.

**Note:** Before updating, make sure that the old Grafana Loki and Promtail containers are removed.

##### 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](https://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](https://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>--&lt;config_name&gt;</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> <b>Note:</b> You must pre-tag the appropriate configuration in. Configurator. <b>Note:</b> 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 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.

#### 3.4.1 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.

#### 3.4.2 Logging activation

##### 3.4.2.1 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.

#### 3.4.2.2 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.

### 3.4.3 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.

### 3.5 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.

### 3.6 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
```

### 3.7 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.

#### 3.7.1 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
```



### 3.8 System scaling

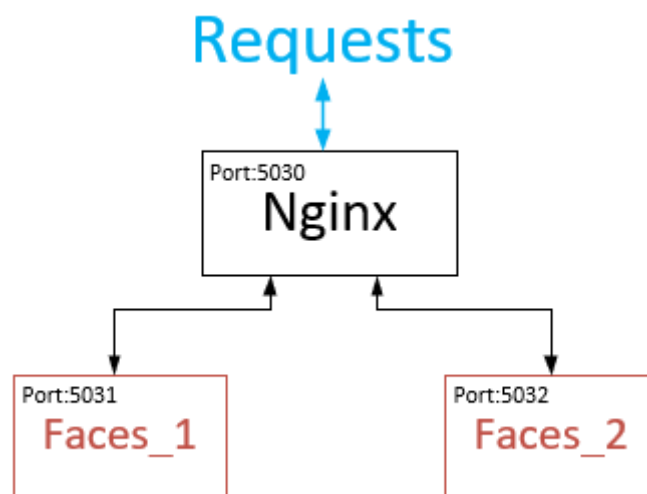
All LP services are linearly scalable and can be located on several services.

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.

### 3.8.1 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.

### 3.9 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('
123456789012345678901234567890123456789012345678901234'),
HEXTORAW('
012345678901234567890123456789012345678901234567890123'), 32)
FROM DUAL;
```

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