

VisionLabs FaceStream

Installation manual without LP launched

v.5.2.11

Contents

Glossary	4
Introduction	5
1 Before launch	6
1.1 Unpack distribution	7
1.2 Create symbolic link	7
1.3 Install Docker	7
1.4 Install Docker Compose	8
1.5 Choose logging method	9
1.5.1 Logging to stdout	9
1.5.2 Logging to file	9
1.6 Install GPU dependencies	10
1.6.1 Actions to launch FaceStream with GPU through Docker Compose	10
1.7 Login to registry	11
1.8 License activation	12
1.8.1 Ways to specify HASP license settings	12
1.8.1.1 Specify HASP license settings using dump file	12
1.8.2 Ways to specify Guardant license settings	14
1.8.2.1 Specify Guardant license settings using dump file	14
1.9 Launch InfluxDB OSS 2 container	15
1.10 Launch PostgreSQL container	15
1.11 Prepare LUNA Configurator database	16
1.11.1 Create LUNA Configurator database	16
1.11.2 Initialize LUNA Configurator database	16
1.12 Launch LUNA Configurator container	17
1.13 LUNA Licenses service	17
1.13.1 Specify license settings using Configurator	17
1.13.1.1 Specify HASP license settings	18
1.13.1.2 Specify Guardant license settings	18
1.13.2 Launch LUNA Licenses container	18
2 Launch FaceStream	20
2.1 Launch FaceStream manually	21
2.1.1 Upload settings to LUNA Configurator	21
2.1.1.1 Upload LUNA Streams settings	21
2.1.1.2 Upload FaceStream settings	21
2.1.2 Prepare LUNA Streams database	22
2.1.2.1 Create LUNA Streams database	22

2.1.2.2	Initialize LUNA Streams database	23
2.1.3	Launch LUNA Streams container	23
2.1.4	Launch FaceStream container	23
2.1.4.1	Launch FaceStream container using CPU	23
2.1.4.2	Launch FaceStream container using GPU	24
2.2	Launch FaceStream using Docker Compose	26
2.2.1	Launch FaceStream command	26
3	Next actions	28
4	Additional information	29
4.1	Docker commands	30
4.1.1	Show containers	30
4.1.2	Copy files to container	30
4.1.3	Enter container	30
4.1.4	Images names	30
4.1.5	Show container logs	30
4.1.6	Delete image	31
4.1.7	Stop container	31
4.1.8	Delete container	31
4.2	Launching keys	33
4.2.1	Description of container launch parameters	34
4.3	Logging to server	36
4.3.1	Create logs directory	36
4.3.2	Logging activation	36
4.3.2.1	LUNA Streams and other LP services logging activation	36
4.3.2.2	FaceStream logging activation	36
4.3.2.3	Configurator service logging activation	37
4.3.3	Mounting directories with logs when starting services	37
4.4	Docker log rotation	39

Glossary

Term	Meaning
Aspect angle	Head rotation degree (in degrees) on each of the three axes (up/down tilt relative to the horizontal axis; left/right tilt, relative to the vertical axis; a rotation about the vertical axis).
Batch	Group of data processed simultaneously.
Best shot	The frame of the video stream on which the face/body is fixed in the optimal angle for further processing.
Detection	FaceStream entity that contains the coordinates of face or body and the estimated value of the object that determines the best shot.
Descriptor	A set of unique features received from the warp. A descriptor requires much less storage memory in comparison with the sample and is used for comparison of faces.
Event	LUNA PLATFORM entity, which contains information (city, user data, track id, etc.) about one face and/or body. This information is transferred to the LUNA PLATFORM by the FaceStream application. For a complete list of the transferred information, see the OpenAPI LUNA PLATFORM documentation.
LUNA Streams	Service for creating and managing streams that contain policies for processing a video stream/video file/set of images.
Normalized image, warp	Images containing a face or body and corresponding to VisionLabs standard. Used when working with LUNA PLATFORM.
Portrait	Image of face or body that has been transformed to a specific format. The portrait has two types — “warp” (the image is transformed into warp format), “gost” (detection is cut out from the source frame, considering indentation).
Track	Information about object’s position (face of a person) in a sequence of frames. If the object leaves the frame zone, the track doesn’t discontinue right away. For some time, the system expects the object to return and if it does, the track continues.
Tracking	Object (face) tracking function in the frame sequence.

Introduction

This document describes:

- Launching the minimum required services to launch FaceStream (LUNA Configurator, LUNA Licenses, PostgreSQL and InfluxDB).
- Activation of the license key LUNA PLATFORM for the possibility of creating streams.
- Manual process of launching the FaceStream application and the LUNA Streams service using Docker.
- Automatic deployment of the FaceStream application with the LUNA Streams service using Docker Compose.

Before launching, it is necessary to explore the general information and sequence of actions.

FaceStream licensing is managed by a special parameter of the LUNA PLATFORM 5 license key, which determines the number of streams for LUNA Streams operation. Thus, a LUNA PLATFORM 5 license is required for FaceStream to work.

This manual provides an example of activating the LUNA PLATFORM license.

The Docker Compose scenario from this distribution is used for deploying the LUNA Streams and FaceStream on a single server.

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

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

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

<https://docs.docker.com>

This document includes examples of FaceStream deployment in a minimal 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.

For more information on general operation and application settings, see the FaceStream administrator manual.

1 Before launch

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

Before launch FaceStream, you need to do the following:

1. [Unpack the FaceStream distribution.](#)
2. [Create symbolic link.](#)
3. [Install Docker.](#)
4. [Install Docker Compose](#) if you plan to start FaceStream using Docker Compose script.
5. [Choose logging method.](#)
6. [Set up GPU computing](#) if you plan to use GPU.
7. [Login to VisionLabs registry.](#)
8. [Activate license key.](#)
9. [Launch Influx OSS 2 container.](#)
10. [Launch PostgreSQL container.](#)
11. [Create and initialize database for LUNA Configurator.](#)
12. [Launch LUNA Configurator container.](#)
13. [Launch LUNA Licenses container.](#)

After the steps have been performed, you can start manually or automatically launching LUNA Streams and FaceStream.

1.1 Unpack distribution

It is recommended to move the archive to a pre-created directory for FaceStream and unpack the archive there.

The following commands should be performed under the root user.

Create a directory for FaceStream.

```
mkdir -p /var/lib/fs
```

Move the archive to the created directory. It is considered that the archive is saved to the “/root” directory.

```
mv /root/facestream_standalone_eng_v.5.2.11.zip /var/lib/fs/
```

Go to the directory.

```
cd /var/lib/fs/
```

Install the unzip utility if it is not installed.

```
yum install unzip
```

Unpack the archive.

```
unzip facestream_standalone_eng_v.5.2.11.zip
```

1.2 Create symbolic link

Create a symbolic link. The link indicates that the current version of the distribution file is used to run the software package.

```
ln -s facestream_standalone_eng_v.5.2.11 fs-current
```

1.3 Install Docker

Docker is required for launching of the FaceStream container.

The Docker installation is described in the official documentation:

<https://docs.docker.com/engine/install/centos/>.

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
```

1.4 Install Docker Compose

Note. Install Docker Compose only if you are going to use the FaceStream automatic launching script.

Install Docker Compose.

```
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```



```
chmod +x /usr/local/bin/docker-compose
```

```
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

See the official documentation for details:

<https://docs.docker.com/compose/install/>

1.5 Choose logging method

There are two methods to output logs

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

Log output settings for LUNA PLATFORM services and LUNA Streams service are set in section <SERVICE_NAME>_LOGGER of LUNA Configurator service.

Log output settings for FaceStream are set in the settings logging of section FACE_STREAM_CONFIG of LUNA Configurator service.

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

1.5.1 Logging to stdout

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

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

1.5.2 Logging to file

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

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

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

In the Docker Compose script, synchronization of directories with folders **is not configured**. You need to manually add folder mounting to the `docker-compose.yml` file.

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

1.6 Install GPU dependencies

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 FaceStream distribution and should be downloaded from the Internet):

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

See the documentation for additional information:

<https://github.com/NVIDIA/nvidia-docker#centos-7x8x-docker-ce-rhel-7x8x-docker-ce-amazon-linux-12>.

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.

1.6.1 Actions to launch FaceStream with GPU through Docker Compose

To launch FaceStream with GPU through Docker Compose, it is necessary, in addition to the above actions, to add the `deploy` section in the `facestream` field to the `docker-compose.yml` file.

Before starting the FaceStream container with GPU, it is required to **enable GPU** for calculations in the FaceStream settings using the “enable_gpu_processing” parameter (see the “FaceStream configuration” section in the administrator manual).

```
vi /var/lib/fs/fs-current/example-docker/docker-compose.yml
```

```
facestream:
  image: ${REGISTRY_ADDRESS}:${DOCKER_REGISTRY_PORT}/facestream:${FS_VER}
  deploy:
    resources:
      reservations:
        devices:
          - driver: nvidia
            count: all
            capabilities: [gpu]
  restart: always
  environment:
    CONFIGURATOR_HOST: ${HOST_CONFIGURATOR}
    CONFIGURATOR_PORT: 5070
```

Here:

- `driver` — Specifies the driver for the reserved device(s).
- `count` — Specifies the number of GPU devices that should be reserved (providing the host holds that number of GPUs).
- `capabilities` — Expresses both generic and driver specific capabilities. It must be set, otherwise, an error will be returned when deploying the service.

See the documentation for additional information:

<https://docs.docker.com/compose/gpu-support/#enabling-gpu-access-to-service-containers>.

1.7 Login to registry

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

Login and password can be requested from the VisionLabs representative.

Enter login <username>.

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

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

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

1.8 License activation

To activate/upgrade a license, follow the steps in the license activation manual included in the distribution package.

After that, follow the steps described below.

1.8.1 Ways to specify HASP license settings

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

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

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

1.8.1.1 Specify HASP license settings using dump file

Open the “platform_settings.json” file:

```
vi /var/lib/fs/fs-current/extras/conf/configurator_configs/platform_settings.json
```

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

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

Save the file.

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

1.8.2 Ways to specify Guardant license settings

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

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

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

1.8.2.1 Specify Guardant license settings using dump file

Open the file “platform_settings.json”.

```
vi /var/lib/fs/fs-current/extras/conf/configurator_configs/platform_settings.json
```

Enter the following data:

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

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

Save the file.

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

1.9 Launch InfluxDB OSS 2 container

InfluxDB 2.0.8-alpine is required to monitor the minimum required LP services (for more information, see the “Monitoring” section in the LUNA PLATFORM administrator manual).

Note! If you already have InfluxDB 2.0.8-alpine installed, skip this step.

Use the following command to launch InfluxDB:

```
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=
  kofqt4Pfqn6o0RBtMDQqVoJLgHoxxDUmmhiAZ7JS6VmEnrqZXQhxDhad8AX9tmiJH6CjM7Y1U8p5eSEocG
  == \
-v /etc/localtime:/etc/localtime:ro \
-v /var/lib/fs/fs-current/example-docker/influx:/var/lib/influxdb2 \
--restart=always \
--detach=true \
--network=host \
--name influxdb \
dockerhub.visionlabs.ru/luna/influxdb:2.0.8-alpine
```

1.10 Launch PostgreSQL container

Note. Make sure that the old PostgreSQL container 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/fs/fs-current/example-docker/postgresql/data:/var/lib/
  postgresql/data/ \
-v /var/lib/fs/fs-current/example-docker/postgresql/entrypoint-initdb.d:/
  docker-entrypoint-initdb.d/ \
-v /etc/localtime:/etc/localtime:ro \
--name=postgres \
--restart=always \
--detach=true \
```

```
--network=host \  
dockerhub.visionlabs.ru/luna/postgis-vmatch:16
```

Here:

- `-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.

1.11 Prepare LUNA Configurator database

1.11.1 Create LUNA Configurator database

Create a database for LUNA Configurator:

```
docker exec -i postgres psql -U luna -c "CREATE DATABASE luna_configurator;"
```

Allow the user to log in to the database:

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

1.11.2 Initialize LUNA Configurator database

Use the following command to launch to create the Configurator database tables.

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /var/lib/fs/fs-current/extras/conf/configurator_configs/  
luna_configurator_postgres.conf:/srv/luna_configurator/configs/config.  
conf \  
-v /var/lib/fs/fs-current/extras/conf/configurator_configs/platform_settings  
.json:/srv/luna_configurator/used_dumps/platform_settings.json \  
--network=host \  
--rm \  
--entrypoint bash \  
dockerhub.visionlabs.ru/luna/luna-configurator:v.2.2.76 \  

```



```
-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/fs/fs-current/extras/conf/configurator_configs/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.
- --dump-file /srv/luna_configurator/used_dumps/platform_settings.json — Updates settings in the Configurator DB with values from the provided file.

1.12 Launch LUNA Configurator container

Use the following command 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/fs/fs-current/extras/conf/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.2.2.76
```

1.13 LUNA Licenses service

1.13.1 Specify license settings using Configurator

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

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

dump file”.

1.13.1.1 Specify HASP license settings

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

To set the license server address, follow these steps:

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

1.13.1.2 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.

1.13.2 Launch LUNA Licenses container

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.0.12.1
```

2 Launch FaceStream

There are two ways to launch FaceStream — manual and automatic using a Docker Compose script.

Use the hyperlinks below to go to the instructions for the required launch method:

- [manual way to launch FaceStream](#)
- [automatic way to launch FaceStream](#)

2.1 Launch FaceStream manually

Note. Perform these actions only if you are going to launch FaceStream manually. If you are going to launch FaceStream using the Docker Compose script, go to [“Launch FaceStream using Docker Compose”](#).

2.1.1 Upload settings to LUNA Configurator

The main settings of LUNA Streams and FaceStream should be set in the Configurator service after its launch. The exception is the FaceEngine settings, which are set in the configuration file “faceengine.conf” and transferred during the launch of the FaceStream container.

If necessary, you can use configuration files instead of the Configurator service settings and transfer them during container launching (for more information, see the “Use FaceStream with configuration files” section of the administrator manual).

FaceStream and LUNA Streams settings are uploaded into the Configurator in different ways.

2.1.1.1 Upload LUNA Streams settings

To upload LUNA Streams settings into the Configurator service, you should use the configuration migration mechanism.

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/streams:/srv/logs \
--entrypoint=/bin/bash \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/streams-configs:v.1.1.24 \
-c "python3 -m streams_configs.migrate head --config_db_url postgres://luna:
luna@127.0.0.1:5432/luna_configurator"
```

Here:

- `python3 -m streams_configs.migrate` — Migration script.
- `--config_db_url postgres://luna:luna@127.0.0.1:5432/luna_configurator` — The “luna_configurator” database address flag.

2.1.1.2 Upload FaceStream settings

FaceStream settings are located in a special file “facestream_dump.json”. To upload these settings into the Configurator service, you should use the “load_dump.py” script.

The “load_dump.py” script requires Python version 2.x or 3.x. If version 2.x is installed, then the script should be run using the `python` command. If version 3.x is installed, then the script should be run using the `python3` command.

- Go to the directory with script and dump file:

```
cd /var/lib/fs/fs-current/example-docker/luna_configurator/dumps/
```

- Run the script to upload FaceStream settings into the Configurator service, specifying the installed version of Python (the command below gives an example of running the script for Python version 3.x):

```
python3 -m load_dump --dump-file=facestream_dump.json --luna-config=http://127.0.0.1:5070/1
```

All necessary parameters will be automatically added to the Configurator service.

2.1.2 Prepare LUNA Streams database

To launch FaceStream, you need to launch the LUNA Streams service by creating and initializing a database for it. This service is not included in the LUNA PLATFORM 5 distribution, so it should be launched separately.

2.1.2.1 Create LUNA Streams database

Create a database for LUNA Streams:

```
docker exec -i postgres psql -U luna -c "CREATE DATABASE luna_streams;"
```

Allow the user to log in to the database:

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

Activate PostGIS:

```
docker exec -i postgres psql -U luna luna_streams -c "CREATE EXTENSION postgis;"
```

2.1.2.2 Initialize LUNA Streams database

Initialize the data in the LUNA Streams database:

```
docker run -v /etc/localtime:/etc/localtime:ro \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/luna-streams:v.1.1.24 \
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.1.3 Launch LUNA Streams container

The container is launched with the following command:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5160 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/streams:/srv/logs \
--name=luna-streams \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-streams:v.1.1.24
```

To check if the service started correctly, you can perform a GET request <http://127.0.0.1:5160/> version. The response should return the LUNA Streams version v.1.1.24.

2.1.4 Launch FaceStream container

2.1.4.1 Launch FaceStream container using CPU

The container is launched as follows:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
-v /var/lib/fs/fs-current/extras/conf/configs/faceengine.conf:/srv/
facestream/data/faceengine.conf \
```

```
-v /var/lib/fs/fs-current/extras/conf/configs/runtime.conf:/srv/facestream/
  data/runtime.conf \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/facestream:/srv/logs \
--env=PORT=34569 \
--detach=true \
--restart=always \
--name=facestream \
--network=host \
dockerhub.visionlabs.ru/luna/facestream:v.5.2.11
```

For a description of the remaining parameters and launching keys, see the [“Launching keys”](#) section.

To verify that the application was launched correctly, you can perform a GET request `http://127.0.0.1:34569/version`. The response should return the FaceStream v.5.2.11.

2.1.4.2 Launch FaceStream container using GPU

Note. Use this command only if you are going to use FaceStream with GPU.

Before launching FaceStream in GPU mode, additional dependencies should be installed (see [“Install GPU dependencies”](#) section).

Before starting the FaceStream container with GPU, it is required to **enable GPU** for calculations in the FaceStream settings using the “enable_gpu_processing” parameter (see the “FaceStream configuration” section in the administrator manual).

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
-v /var/lib/fs/fs-current/extras/conf/configs/faceengine.conf:/srv/
  facestream/data/faceengine.conf \
-v /var/lib/fs/fs-current/extras/conf/configs/runtime.conf:/srv/facestream/
  data/runtime.conf \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/facestream:/srv/logs \
--env=PORT=34569 \
--gpus device=0 \
--detach=true \
--restart=always \
--name=facestream \
--network=host \
dockerhub.visionlabs.ru/luna/facestream:v.5.2.11
```

Here `--gpus device=0` is the parameter specifies the used GPU device and enables GPU utilization. A

single GPU can be utilized per FaceStream instance. Multiple GPU utilization per instance is not available.

For a description of the remaining parameters and launching keys, see the [“Launching keys”](#) section.

To verify that the application was launched correctly, you can perform a GET request `http://127.0.0.1:34569/version`. The response should return the FaceStream v.5.2.11.

2.2 Launch FaceStream using Docker Compose

The Docker Compose script:

- Loads LUNA Streams and FaceStream settings into Configurator.
- Creates and initializes the LUNA Streams database.
- Launches the LUNA Streams and FaceStream.
- Is tested using the default LUNA Streams and FaceStream configurations.
- Is not intended to be used for FaceStream scaling:
 - It is not used for the deployment of FaceStream services on several servers.
 - It is not used for deployment and balancing of several FaceStream services on a single service.
- Supports GPU utilization for FaceStream calculations.
- Does not provide the possibility to use external database for LUNA Streams already installed on the server.
- Does not perform migrations from previous FaceStream versions and updates from the previous FaceStream build.

See the “docker-compose.yml” file and other files in the “example-docker” directory for the information about launched services and performed actions.

You can write your scenario that deploys and configures all the required services. This document does not include information about scenario creation or tutorial for Docker usage. Please refer to the Docker documentation to find more information about Docker and Docker Compose:

<https://docs.docker.com>

2.2.1 Launch FaceStream command

Go to the Docker Compose folder:

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

Make sure that FS container are not launched before executing the script. An error will occur if you try to run a container with the same name as an existing container. If one or several LP containers are launched, you should stop them using the `docker container rm -f <container_name>` command. To stop all the containers, use `docker container rm -f $(docker container ls -aq)`.

To launch FaceStream with GPU using Docker Compose, you need to follow the steps in [“Install GPU dependencies”](#).

Launch Docker Compose:

You should be logged in the VisionLabs registry (see section [“Login to registry”](#))

```
./start_facestream.sh --configurator_address=127.0.0.1
```

--configurator_address=127.0.0.1 — LUNA Configurator service address

Check the state of launched Docker containers.

```
docker ps
```

The list of streams is available at <http://127.0.0.1:34569/api/1/streams/>. Viewing the stream in the browser is available at http://127.0.0.1:34569/api/1/streams/preview/<stream_id>.

3 Next actions

To work further with FaceStream, you need to create a stream using a POST request “create stream” to the LUNA Streams service. The stream contains policies for processing a video stream/video file/set of images. If the stream is created with the status “pending” (by default), then the FaceStream worker will automatically start processing the stream.

See “Interaction of FaceStream with LUNA Streams” in the administrator manual for details on working with streams and LUNA Streams.

The list of streams to process is available at <http://127.0.0.1:34569/api/1/streams/>. Previewing a stream in a browser is available at http://127.0.0.1:34569/api/1/streams/preview/<stream_id>.

4 Additional information

This section provides the following additional information:

- [Useful commands for working with Docker.](#)
- [Launching keys description.](#)
- [Actions to enable saving logs to files.](#)
- [Configuring Docker log rotation.](#)

4.1 Docker commands

4.1.1 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
```

4.1.2 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>
```

4.1.3 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
```

4.1.4 Images names

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

```
docker images
```

4.1.5 Show container logs

You can view the container logs with the following command:

```
docker logs <container_name>
```

4.1.6 Delete image

If you need to delete an image:

- Run the `docker images` command.
- Find the required image, for example: `dockerhub.visionlabs.ru/luna/v.5.2.11`.
- 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)
```

4.1.7 Stop container

You can stop the container using the command:

```
docker stop <container_name>
```

Stop all the containers:

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

4.1.8 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/v.5.2.11`.
- 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)
```


4.2 Launching keys

To launch FaceStream with Configurator, the keys are set using environment variables:

- `--env=` — this parameter sets the environment variables required to start the container. The following basic values are specified:
 - `CONFIGURATOR_HOST` — Host on which the Configurator service is running. The local host is set if the container is running on the same server where the Configurator is running.
 - `CONFIGURATOR_PORT` — Listening port for the Configurator service. By default, port 5070 is used.
 - `PORT` — Port where FaceStream will listen.
 - `STREAMS_ID` — Tag specifies a list of stream IDs that will be requested from LUNA Streams for processing. Other streams will be filtered. The “stream_id” parameter is given in response to the “create stream” request.

If the value is “ ” or the “STREAMS_ID” tag is not set, then FaceStream will take all existing “stream_id” from the queue.

If a non-existent value is set, an error about an incorrect UUID will be indicated when launching FaceStream.

By default, the value equals “ ”.

To use the key, the “CONFIGURATOR_HOST” and “CONFIGURATOR_PORT” variables should be specified.

- `STREAMS_NAME` — List of streams names sets in this tag. Streams names are set using the “name” parameter at the time of their creation (“create streams” request). Streams with these names will be requested from LUNA Streams for processing. Other streams will be filtered.

Otherwise, the principle of operation is similar to the “STREAMS_ID” tag.

- `GROUPS_ID` and `GROUPS_NAME` — Tags specify a list of group IDs or a list of group names. The parameters “group_id” or “group_name” are set during stream creation (“create stream” request). Streams with these parameters will be requested from LUNA Streams for processing. Other streams will be filtered.

If the value is “ ” or the “GROUPS_ID”/“GROUPS_NAME” tags are not set, then FaceStream will not filter streams by groups.

If a non-existent value is set, an error about an incorrect UUID will be indicated when launching FaceStream.

By default, the value equals “ ”.

To use the keys, the “CONFIGURATOR_HOST” and “CONFIGURATOR_PORT” variables should be specified.

You can set multiple values for “STREAMS_NAME”, “STREAMS_ID”, “GROUPS_NAME” and “GROUPS_ID” tags. Syntax example: `--env=STREAMS_ID="037f3196-c874-4eca-9d7c-91fd8dfc9593 4caf7cf7-dd0d-4ad5-a35e-b263e742e28a"`

- `CONFIGS_ID=""` — Tag is used to set a LUNA Configurator tag, which relates to the FaceStream main configurations. The same tag should be set for “TRACK_ENGINE_CONFIG” and “FACE_STREAM_CONFIG”.

If the value is set to “ ” then the “TRACK_ENGINE_CONFIG” and “FACE_STREAM_CONFIG” records will be used by default. If the record by default does not exist or has an invalid JSON syntax, the configuration file from the distribution package will be used.

By default, the value equals “ ”.

To use the key, the “CONFIGURATOR_HOST” and “CONFIGURATOR_PORT” variables should be specified.

- `CONFIG_RELOAD = 1` — Tag that enables checking for changes in the “FACE_STREAM_CONFIG” section of the LUNA Configurator service and takes the following values: — “1” — Change tracking is enabled, if there are changes in the configuration, all FaceStream containers will be automatically restarted. — “0” — Change tracking is disabled.

By default, the value equals “1”.

- `PULLING_TIME = 10` — Tag that sets the period for receiving new parameters from the “FACE_STREAM_CONFIG” section of the LUNA Configurator service in the range [1...3600] sec. Used in conjunction with the CONFIG-RELOAD tag.

By default, the value equals “10”.

- `--device=` — This parameter is required to specify the address to the USB device. The address must be specified in the stream source when it is created. Example: `--device=/dev/video0`.

See how FaceStream works with LUNA Configurator in the section “Use FaceStream with LUNA Configurator” of the administrator manual.

4.2.1 Description of container launch parameters

Below is a description of the container launch commands:

- `docker run` — Command to launch the selected image as a new container.
- `-v` — Enables you to load the contents of the server folder into the volume of the container. This way the content is synchronized.
- `-v /var/lib/fs/fs-current/extras/conf/configs/faceengine.conf:/srv/facestream/data/faceengine.conf \` — This parameter enables you to use the FaceEngine settings from the configuration file “faceengine.conf”.

- `-v /var/lib/fs/fs-current/extras/conf/configs/runtime.conf:/srv/facestream/data/runtime.conf \` — This parameter enables you to mount the runtime configuration file into the FaceStream container. Before changing the default settings, you need to consult with VisionLabs specialists.
- `--network=host` — This parameter specifies that there is no network simulation and a server network is used. If you need to change the port for third-party containers, replace this line with `-p 5440:5432`. Here, the first port 5440 is the local port, and 5432 is the port used in the container.
- `/etc/localtime:/etc/localtime:ro` — Sets the current time zone used by the container system.
- `--name=facestream` — This parameter specifies the name of the container to be launched. The name must be unique. If a container with the same name already exists, an error will occur.
- `--restart=always` — This parameter defines the restart policy. Daemon always restarts the container regardless of the completion code.
- `--detach=true` — Running the container in the background.

4.3 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 containers.
- Configure synchronization of log directories in the container with logs on the server using the `volume` argument at the start of each container.

In the Docker Compose script, synchronization of directories with folders **is not configured**. You need to manually add folder mounting to the `docker-compose.yml` file.

4.3.1 Create logs directory

Below are examples of commands for creating directories for saving logs and assigning rights to them for LUNA Streams and FaceStream.

```
mkdir -p /tmp/logs/configurator /tmp/logs/licenses /tmp/logs/facestream /tmp/logs/streams
```

```
chown -R 1001:0 /tmp/logs/configurator /tmp/logs/licenses /tmp/logs/facestream /tmp/logs/streams
```

4.3.2 Logging activation

4.3.2.1 LUNA Streams and other 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.

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 file.

4.3.2.2 FaceStream logging activation

To enable logging to file, you need to set the value of the `logging > mode` setting in the `FACE_STREAM_CONFIG` section to `l2f` (output logs only to file) or `l2b` (output logs both to file and to console).

Go to the Configurator service interface (127.0.0.1:5070) and specify the required setting value. The location path of logs in the FaceStream container cannot be changed. It is necessary to specify the path `/srv/logs` when mounting.

By default, only system warnings are displayed in the FaceStream logs. By setting the “severity” parameter, you can enable error output (see the parameter description in the administrator manual).

4.3.2.3 Configurator service logging activation

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.

4.3.3 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 FaceStream with mounting a directory with logs:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
-v /var/lib/fs/fs-current/extras/conf/configs/faceengine.conf:/srv/  
facestream/data/faceengine.conf \  
-v /var/lib/fs/fs-current/extras/conf/configs/runtime.conf:/srv/facestream/  
data/runtime.conf \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/facestream:/srv/logs \  
--env=PORT=34569 \  
--detach=true \  
--restart=always \  

```

```
--name=facestream \  
--network=host \  
dockerhub.visionlabs.ru/luna/facestream:v.5.2.11
```

Examples of manual container launch commands contain these arguments.

4.4 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.