

VisionLabs FaceStream

Руководство по установке без запущенной LP

v.5.1.49

Содержание

Глоссарий	4
Введение	7
1 Подготовка к запуску	8
1.1 Распаковка архива	9
1.2 Создание символической ссылки	9
1.3 Установка Docker	9
1.4 Установка Docker Compose	10
1.5 Выбор способа записи логов	11
1.5.1 Запись логов в stdout	11
1.5.2 Запись логов в файл	11
1.6 Установка зависимостей для GPU	12
1.6.1 Действия для запуска FaceStream с GPU через Docker Compose	13
1.7 Вход в registry	13
1.8 Активация лицензионного ключа	14
1.8.1 Способы задания настроек лицензии HASP	14
1.8.1.1 Задание настроек лицензии HASP с помощью дамп-файла	14
1.8.2 Способы задания настроек лицензии Guardant	16
1.8.2.1 Задание настроек лицензии Guardant помощью дамп-файла	16
1.9 Запуск контейнера InfluxDB OSS 2	17
1.10 Запуск контейнера PostgreSQL	17
1.11 Подготовка БД LUNA Configurator	18
1.11.1 Создание БД LUNA Configurator	18
1.11.2 Инициализация БД Configurator	18
1.12 Запуск контейнера LUNA Configurator	19
1.13 Сервис LUNA Licenses	19
1.13.1 Задание настроек лицензии с помощью Configurator	19
1.13.1.1 Задание настроек лицензии HASP	20
1.13.1.2 Задание настроек лицензии Guardant	20
1.13.2 Запуск контейнера LUNA Licenses	20
2 Запуск FaceStream	22
2.1 Ручной запуск FaceStream	23
2.1.1 Загрузка настроек в LUNA Configurator	23
2.1.1.1 Загрузка настроек LUNA Streams	23
2.1.1.2 Загрузка настроек FaceStream	23
2.1.2 Подготовка БД LUNA Streams	24
2.1.2.1 Создание БД LUNA Streams в контейнере PostgreSQL	24

2.1.2.2	Инициализация БД LUNA Streams	25
2.1.3	Команда запуска контейнера LUNA Streams	25
2.1.4	Команды запуска контейнера FaceStream	25
2.1.4.1	Команда запуска контейнера с использованием CPU	25
2.1.4.2	Команда запуска контейнера с использованием GPU	26
2.2	Запуск FaceStream с помощью Docker Compose	28
2.2.1	Команда запуска FaceStream с помощью Docker Compose	28
3	Дальнейшие действия	30
4	Дополнительная информация	31
4.1	Команды Docker	32
4.1.1	Показать контейнеры	32
4.1.2	Копировать файлы в контейнер	32
4.1.3	Вход в контейнер	32
4.1.4	Имена образов	32
4.1.5	Просмотр логов контейнера	32
4.1.6	Удаление образа	33
4.1.7	Остановка контейнера	33
4.1.8	Удаление контейнера	33
4.2	Ключи запуска	35
4.2.1	Расшифровка параметров запуска контейнера	36
4.3	Запись логов на сервер	38
4.3.1	Создание директории логов	38
4.3.2	Активация записи логов	38
4.3.2.1	Активация записи логов LUNA Streams и других сервисов LP	38
4.3.2.2	Активация записи логов FaceStream	38
4.3.2.3	Активация записи логов сервиса Configurator	39
4.3.3	Монтирование директорий с логами при старте сервисов	39
4.4	Настройка ротации логов Docker	41

Глоссарий

Термин	Значение термина
Батч	Группа данных, обрабатываемых одновременно.
Биометрический образец	Изображения, содержащие лицо или тело и соответствующие стандарту VisionLabs. Используется при работе с LUNA PLATFORM.
Биометрический шаблон	Набор уникальных свойств, получаемых в LUNA PLATFORM из биометрического образца.
Детекция	Сущность FaceStream, содержащая координаты лица или тела и оценочное значение объекта, по которому определяется лучший кадр.
Лучший кадр	Кадр видеопотока, на котором лицо/тело зафиксировано в оптимальном ракурсе для дальнейшей обработки.
Портрет	Изображение лица или тела, трансформированное под определенный формат. Портрет имеет два типа — «warp» (изображение трансформируется в формат биометрического образца), «gost» (из исходного кадра вырезается детекция с учетом отступов).
Ракурс	Степень поворота головы (в градусах) по каждой из трех осей вращения (наклон вверх/вниз относительно горизонтальной оси; наклон влево/вправо относительно вертикальной оси; поворот относительно вертикальной оси).
Событие	Сущность LUNA PLATFORM, которая содержит информацию (город, пользовательские данные, номер трека и т.д.) об одном лице и/или теле. Данная информация передается в LUNA PLATFORM приложением FaceStream. Полный перечень передаваемой информации см. в документации OpenAPI LUNA PLATFORM.
Трек	Информация о положении объекта (лица) одного человека на последовательности кадров. Если объект покидает зону кадра, то трек прерывается не сразу. Некоторое время он ожидает возвращения объекта в кадр. Если объект вернулся, то трек продолжается.
Трекинг	Функция отслеживания объекта (лица) на последовательности кадров.

Термин	Значение термина
LUNA Streams	Сервис для создания и управления потоками, которые содержат политики обработки видеопотока/видеофайла/набора изображений.

Аббревиатура	Расшифровка
БД, DB	База данных
LP	LUNA PLATFORM

Введение

Данный документ описывает:

- запуск минимально необходимых сервисов для запуска FaceStream (LUNA Configurator, LUNA Licenses, PostgreSQL и InfluxDB)
- активацию лицензионного ключа LUNA PLATFORM для возможности создания потоков;
- ручной процесс запуска приложения FaceStream и сервиса LUNA Streams с помощью Docker;
- автоматическое разворачивание приложения FaceStream вместе с сервисом LUNA Streams с помощью Docker Compose.

Скрипт Docker Compose из данного дистрибутива используется для разворачивания LUNA Streams и FaceStream на одном сервере.

Считается, что запуск выполняется на сервере с операционной системой CentOS, где FaceStream не был установлен.

Администратор должен вручную настроить Firewall и SELinux на сервере. В данном документе не описывается их настройка.

Данный документ не включает руководство по использованию Docker. Пожалуйста, обратитесь к документации Docker для более подробной информации:

<https://docs.docker.com>

В данном документе приведены примеры разворачивания FaceStream в минимальной рабочей конфигурации для использования в демонстрационных целях. Данная конфигурация не является достаточной для реальной эксплуатации системы в продуктивном контуре.

Все описываемые команды необходимо исполнять в оболочке Bash (когда команды запускаются напрямую на сервере) или в программе для работы с сетевыми протоколами (в случае удаленного подключения к серверу), например, Putty.

Подробную информацию по общей работе и настройках приложения см. в руководстве администратора FaceStream.

1 Подготовка к запуску

Убедитесь в том, что вы являетесь **root**-пользователем перед тем, как начать запуск!

Перед запуском FaceStream необходимо выполнить следующие действия:

1. [Распаковать дистрибутив FaceStream](#)
2. [Создать символическую ссылку](#)
3. [Выполнить установку Docker](#)
4. [Выполнить установку Docker Compose](#), если планируется запускать FaceStream с помощью автоматического запуска через скрипт Docker Compose
5. [Выбрать способ записи логов](#)
6. [Настроить вычисления с помощью GPU](#), если планируется использовать GPU
7. [Авторизоваться в registry VisonLabs](#)
8. [Активировать лицензию](#)
9. [Запустить контейнер Influx OSS 2](#)
10. [Запустить контейнер PostgreSQL](#)
11. [Создать и инициализировать БД для LUNA Configurator](#)
12. [Запустить контейнер LUNA Configurator](#)
13. [Запустить контейнер LUNA Licenses](#)

После выполненных действий можно приступить к ручному или автоматическому запуску LUNA Streams и FaceStream.

1.1 Распаковка архива

Рекомендуется переместить архив в предварительно созданную директорию для FaceStream и распаковать архив в этой директории.

Указанные команды следует выполнять под пользователем root.

Создайте директорию для FaceStream.

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

Переместите архив в созданную директорию. Предполагается, что архив сохранён на сервере в директории «/root».

```
mv /root/facestream_docker_v.5.1.49.zip /var/lib/fs/
```

Перейдите в директорию.

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

Установите утилиту unzip, если она ещё не установлена.

```
yum install unzip
```

Распакуйте архив.

```
unzip facestream_docker_v.5.1.49.zip
```

1.2 Создание символической ссылки

Создайте символическую ссылку. Символическая ссылка указывает на директорию, в которой хранятся файлы для запуска нужной версии программного продукта.

```
ln -s facestream_docker_v.5.1.49 fs-current
```

1.3 Установка Docker

Docker требуется для запуска контейнера FaceStream.

Установка Docker описана в официальной документации:

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

Если на сервере уже установлен Docker версии 20.10.8, то выполнять повторную установку не требуется. Не гарантируется работа с более высокими версиями Docker.

Ниже перечислены команды для быстрой установки:

При возникновении проблем с установкой обратитесь к официальной документации Docker.

Установите зависимости.

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

Добавьте репозиторий.

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

Установите Docker.

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

Запустите Docker.

```
systemctl start docker
```

```
systemctl enable docker
```

Проверьте, статус запуска Docker.

```
systemctl status docker
```

1.4 Установка Docker Compose

Примечание. Выполняйте установку Docker Compose только если собираетесь использовать скрипт автоматического запуска FaceStream.

Установите 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
```

Для получения более подробной информации обратитесь к официальной документации:

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

1.5 Выбор способа записи логов

Существует два способа вывода логов:

- стандартный вывод логов (stdout);
- вывод логов в файл.

Настройки вывода логов сервисов LUNA PLATFORM и сервиса LUNA Streams задаются в секции <SERVICE_NAME>_LOGGER сервиса Configurator.

Настройки вывода логов FaceStream задаются в настройках logging секции FACE_STREAM_CONFIG сервиса Configurator.

При необходимости можно использовать оба способа вывода логов.

1.5.1 Запись логов в stdout

Данный способ используется по умолчанию и для него не требуется выполнять дополнительных действий.

Рекомендуется настроить ротацию логов Docker для ограничения их размеров (см. раздел «[Настройка ротации логов Docker](#)»).

1.5.2 Запись логов в файл

Примечание. При включении сохранения логов в файле необходимо помнить о том, что логи занимают определенное место в хранилище, а процесс логирования в файл негативно влияет на производительность системы.

Для использования данного способа необходимо выполнить следующие дополнительные действия:

- **перед запуском сервисов:** создать директории для логов на сервере;
- **после запуска сервисов:** активировать запись логов и задать расположение хранения логов внутри контейнеров сервисов LP;

- **во время запуска сервисов:** настроить синхронизацию директорий логов в контейнере с логами на сервере с помощью аргумента `volume` при старте каждого контейнера.

В скрипте Docker Compose **не настроена** синхронизация директорий с папками. Необходимо вручную добавить монтирование папок в файл `docker-compose.yml`.

См. инструкцию по включению записи логов в файлы в разделе «[Запись логов на сервер](#)».

1.6 Установка зависимостей для GPU

Пропустите данный раздел если не собираетесь использовать FaceStream с GPU.

Для использования GPU с Docker контейнерами необходимо установить NVIDIA Container Toolkit.

Пример установки приведен ниже.

```
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
```

Проверьте работу NVIDIA Container toolkit, запустив базовый контейнер CUDA (он не входит в дистрибутив FaceStream, его необходимо загрузить из Интернета):

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

Для дополнительной информации см. следующую документацию:

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

Извлечение атрибутов на GPU разработано для максимальной пропускной способности. Выполняется пакетная обработка входящих изображений. Это снижает затраты на вычисления для изображения, но не обеспечивает минимальную задержку для каждого изображения.

GPU-ускорение разработано для приложений с высокой нагрузкой, где количество запросов в секунду достигает тысяч. Нецелесообразно использовать ускорение GPU в сценариях

с небольшой нагрузкой, когда задержка начала обработки имеет значение.

1.6.1 Действия для запуска FaceStream с GPU через Docker Compose

Для запуска FaceStream с GPU через Docker Compose необходимо, кроме вышеописанных действий, добавить секцию `deploy` в поле `facestream` в файл `docker-compose.yml`.

Перед запуском контейнера FaceStream с GPU требуется **включить использование GPU** для вычислений в настройках FaceStream с помощью параметра «`enable_gpu_processing`» (см. раздел «Настройки FaceStream» в руководстве администратора).

```
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
```

`driver` — в данном поле указывается драйвер для зарезервированного устройства(устройств);

`count` — в данном поле задается количество графических процессоров, которые должны быть зарезервированы (при условии, что хост содержит такое количество графических процессоров);

`capabilities` — данное поле выражает как общие, так и специфические возможности драйвера. Его необходимо задать, иначе будет возвращена ошибка при развертывании сервиса.

Для дополнительной информации см. следующую документацию:

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

1.7 Вход в registry

При запуске контейнеров необходимо указать ссылку на образ, необходимый для запуска контейнера. Этот образ загружается из VisionLabs registry. Перед этим необходима авторизация.

Логин и пароль можно запросить у представителя VisionLabs.

Введите логин <username>.

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

После выполнения команды будет запрошен ввод пароля. Введите пароль.

В команде `docker login` можно вводить логин и пароль одновременно, однако это не гарантирует безопасность, т.к. пароль можно будет увидеть в истории команд.

1.8 Активация лицензионного ключа

Для активации/обновления лицензии выполните действия из руководства по активации лицензии в комплекте поставки.

После этого выполните действия, описанные ниже.

1.8.1 Способы задания настроек лицензии HASP

Для HASP-ключа нужно задать IP-адрес сервера лицензирования. Его можно задать одним из двух способов:

- в дамп-файле «platform_settings.json» (см. ниже). Содержимое стандартных настроек будет перезаписано содержимым этого файла на этапе запуска сервиса Configurator.
- в настройках сервиса Licenses в пользовательском интерфейсе Configurator (см. раздел «[Задание настроек лицензии HASP](#)»).

Выберите наиболее удобный способ и выполните действия, описанные в соответствующих разделах.

1.8.1.1 Задание настроек лицензии HASP с помощью дамп-файла

Откройте файл «platform_settings.json»:

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

Задайте IP-адрес сервера с вашим ключом HASP в поле «server_address»:

```
{
  "value": {
    "vendor": "hasp",
```

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

Сохраните файл.

Обратите внимание, что если лицензия активируется с помощью ключа HASP, то должно быть указано два параметра «vendor» и «server_address». Если вы хотите изменить защиту HASP на Guardant, то необходимо добавить поле «license_id».

1.8.2 Способы задания настроек лицензии Guardant

Для Guardant-ключа нужно задать IP-адрес сервера лицензирования и идентификатор лицензии. Настройки можно задать одним из двух способов:

- в дамп-файле «platform_settings.json» (см. ниже). Содержимое стандартных настроек будет перезаписано содержимым этого файла на этапе запуска сервиса Configurator.
- в настройках сервиса Licenses в пользовательском интерфейсе Configurator (см. раздел «[Задание настроек лицензии с помощью Configurator](#)»).

Выберите наиболее удобный способ и выполните действия, описанные в соответствующих разделах.

1.8.2.1 Задание настроек лицензии Guardant помощью дамп-файла

Откройте файл «platform_settings.json»:

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

Задайте следующие данные:

- IP-адрес сервера с вашим ключом Guardant в поле «server_address»
- идентификатор лицензии в формате 0x<your_license_id>, полученный в разделе «Сохранение идентификатора лицензии» руководства по активации лицензии, в поле «license_id»:

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

Сохраните файл.

Обратите внимание, что если лицензия активируется с помощью ключа Guardant, то должно быть указано три параметра «vendor», «server_address» и «license_id». Если вы хотите изменить защиту Guardant на HASP, то необходимо удалить поле «license_id».

1.9 Запуск контейнера InfluxDB OSS 2

InfluxDB 2.0.8-alpine требуется для мониторинга минимально необходимых сервисов LP (дополнительную информацию см. в разделе «Мониторинг» в руководстве администратора LUNA PLATFORM 5).

Примечание! Если у вас уже установлен InfluxDB 2.0.8-alpine, пропустите этот шаг.

Используйте команду `docker run` со следующими параметрами:

```
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 Запуск контейнера PostgreSQL

Используйте следующую команду для запуска PostgreSQL.

Убедитесь, что старый контейнер PostgreSQL удален.

```
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
```

`-v /var/lib/luna/current/example-docker/postgresql/data:/var/lib/postgresql/data/` — эта команда позволяет монтировать директорию «data» в контейнер PostgreSQL. Директория на сервере и директория в контейнере будут синхронизированы. Данные PostgreSQL из контейнера будут сохраняться в эту директорию.

`--network=host` — при необходимости изменить порт для PostgreSQL, следует изменить эту строку на `-p 5440:5432`. Здесь первый порт 5440 — локальный, а 5432 — порт в контейнере.

1.11 Подготовка БД LUNA Configurator

1.11.1 Создание БД LUNA Configurator

Создайте базу данных для LUNA Configurator в контейнере Postgres:

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

Дайте возможность пользователю авторизоваться в БД:

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

1.11.2 Инициализация БД Configurator

Используйте команду `docker run` со следующими параметрами для создания таблиц базы данных Configurator.

```
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.2.2.14 \  

```

```
-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"
```

/var/lib/fs/fs-current/extras/conf/configurator_configs/platform_settings.json
— позволяет задавать путь к файлу с конфигурациями LP.

./base_scripts/db_create.py; — создает структуру базы данных.

python3 -m configs.migrate head; — выполняет миграции настроек в базе данных Configurator и устанавливает ревизию для миграции.

--dump-file /srv/luna_configurator/used_dumps/platform_settings.json — обновляет настройки в базе данных Configurator значениями из предоставленного файла.

1.12 Запуск контейнера LUNA Configurator

Используйте следующую команду для запуска LUNA 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.2.2.14
```

1.13 Сервис LUNA Licenses

1.13.1 Задание настроек лицензии с помощью Configurator

Выполните действия по заданию настроек для [HASP-ключа](#) или [Guardant-ключа](#).

Примечание. Не выполняйте нижеописанные действия если вы уже указали настройки лицензии в разделах «[Задание настроек лицензии HASP с помощью дамп-файла](#)» или «[Задание настроек лицензии Guardant с помощью дамп-файла](#)».

1.13.1.1 Задание настроек лицензии HASP

Примечание. Выполняйте данные действия только если используется лицензия HASP. См. раздел «Задание настроек лицензии Guardant», если используется ключ Guardant.

Для задания адреса сервера лицензирования нужно выполнить следующие действия:

- перейдите в интерфейс сервиса Configurator `http://<configurator_server_ip>:5070/`
- введите в поле «Setting name» значение «LICENSE_VENDOR» и нажмите «Apply Filters»
- задайте IP-адрес сервера с вашим ключом HASP в поле «server_address»
- нажмите «Save»

Обратите внимание, что если лицензия активируется с помощью ключа HASP, то должно быть указано два параметра «vendor» и «server_address». Если вы хотите изменить защиту HASP на Guardant, то необходимо добавить поле «license_id».

1.13.1.2 Задание настроек лицензии Guardant

Примечание. Выполняйте данные действия только если используется ключ Guardant. См. раздел «Задание настроек лицензии HASP», если используется ключ HASP.

Для задания адреса сервера лицензирования нужно выполнить следующие действия:

- перейдите в интерфейс сервиса Configurator `http://<configurator_server_ip>:5070/`
- введите в поле «Setting name» значение «LICENSE_VENDOR» и нажмите «Apply Filters»
- задайте IP-адрес сервера с вашим ключом Guardant в поле «server_address»
- задайте идентификатор лицензии в формате `0x<your_license_id>`, полученный в разделе «Сохранение идентификатора лицензии» руководства по активации лицензии, в поле «license_id»
- нажмите «Save»

Обратите внимание, что если лицензия активируется с помощью ключа Guardant, то должно быть указано три параметра «vendor», «server_address» и «license_id». Если вы хотите изменить защиту Guardant на HASP, то необходимо удалить поле «license_id».

1.13.2 Запуск контейнера LUNA Licenses

Используйте следующую команду для запуска сервиса:

```
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.0.10.14
```

2 Запуск FaceStream

Существует два способа запуска FaceStream — ручной и автоматический с помощью скрипта Docker Compose.

Используйте гиперссылки ниже для перехода к инструкции по необходимому способу запуска:

- [ручной способ запуска FaceStream](#)
- [автоматический способ запуска FaceStream](#)

2.1 Ручной запуск FaceStream

Примечание. Выполняйте данные действия только если собираетесь запускать FaceStream вручную. Если собираетесь запускать FaceStream с помощью скрипта Docker Compose перейдите в раздел «Запуск FaceStream с помощью Docker Compose».

2.1.1 Загрузка настроек в LUNA Configurator

Основные настройки LUNA Streams и FaceStream должны быть заданы в сервисе Configurator после его запуска. Исключением являются настройки FaceEngine, которые задаются в конфигурационном файле «faceengine.conf» и передаются во время запуска контейнера FaceStream.

При необходимости можно использовать вместо настроек сервиса Configurator конфигурационные файлы и передавать их во время запуска контейнера (для дополнительной информации см. раздел «Использование FaceStream с конфигурационными файлами» руководства администратора).

Настройки FaceStream и LUNA Streams загружаются в Configurator по-разному.

2.1.1.1 Загрузка настроек LUNA Streams

Для загрузки настроек LUNA Streams в сервис Configurator, необходимо использовать механизм миграции конфигураций.

```
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.1.0.8 \
-c "python3 -m streams_configs.migrate head --config_db_url postgres://luna:
luna@127.0.0.1:5432/luna_configurator"
```

--config_db_url postgres://luna:luna@127.0.0.1:5432/luna_configurator — флаг указания адреса БД luna_configurator

2.1.1.2 Загрузка настроек FaceStream

Настройки FaceStream расположены в специальном файле «facestream_dump.json». Для загрузки этих настроек в сервис LUNA Configurator, необходимо использовать скрипт «load_dump.py».

Для использования скрипта «load_dump.py» требуется наличие Python версии 2.x или 3.x. Если установлена версия 2.x, то скрипт нужно запускать с помощью команды `python`. Если же установлена версия 3.x, то скрипт нужно запускать с помощью команды `python3`.

- Перейдите в директорию со скриптом и файлом с настройками:

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

- Запустите скрипт для загрузки настроек FaceStream в сервис LUNA Configurator, указав установленную версию Python (команда ниже приводит пример запуска скрипта для Python версии 3.x):

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

Все необходимые параметры будут автоматически добавлены в сервис LUNA Configurator.

2.1.2 Подготовка БД LUNA Streams

Для запуска FaceStream необходимо запустить сервис LUNA Streams, создав и инициализировав для него БД. Данный сервис не входит в поставку LUNA PLATFORM 5, поэтому должен быть запущен отдельно.

2.1.2.1 Создание БД LUNA Streams в контейнере PostgreSQL

Создайте базу данных для LUNA Streams:

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

Дайте возможность пользователю авторизоваться в БД:

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

Активируйте PostGIS:

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


2.1.2.2 Инициализация БД LUNA Streams

Инициализируйте данные в БД LUNA Streams:

```
docker run -v /etc/localtime:/etc/localtime:ro \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/luna-streams:v.1.0.8 \
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.1.3 Команда запуска контейнера LUNA Streams

Запуск контейнера осуществляется следующей командой:

```
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.1.0.8
```

Для проверки корректности запуска сервиса можно выполнить GET-запрос `http://127.0.0.1:5160/version`. В ответе должна вернуться версия LUNA Streams v.1.0.8.

2.1.4 Команды запуска контейнера FaceStream

2.1.4.1 Команда запуска контейнера с использованием CPU

Запуск контейнера осуществляется следующим образом:

```
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.5.1.49
```

Описание остальных параметров и ключей запуска см. в разделе «[Описание ключей запуска](#)».

Для проверки корректности запуска можно выполнить GET-запрос `http://127.0.0.1:34569/version`. В ответе должна вернуться версия FaceStream.

2.1.4.2 Команда запуска контейнера с использованием GPU

Примечание. Используйте данную команду только если собираетесь использовать FaceStream с GPU.

Перед запуском FaceStream в режиме GPU необходимо установить дополнительные зависимости (см. раздел «[Установка зависимостей для GPU](#)»).

Перед запуском контейнера FaceStream с GPU требуется **включить использование GPU** для вычислений в настройках FaceStream с помощью параметра «`enable_gpu_processing`» (см. раздел «Настройка FaceStream» в руководстве администратора).

```
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.5.1.49
```

`--gpus device=0` — параметр указывает используемое устройство GPU и позволяет использовать GPU. Один GPU используется для одного экземпляра FaceStream. Использование множества GPU для одного экземпляра невозможно.

Описание остальных параметров и ключей запуска см. в разделе [«Описание ключей запуска»](#).

Для проверки корректности запуска можно выполнить GET-запрос `http://127.0.0.1:34569/version`. В ответе должна вернуться версия FaceStream.

2.2 Запуск FaceStream с помощью Docker Compose

Скрипт Docker Compose:

- загружает настройки LUNA Streams и FaceStream в Configurator
- создает и инициализирует БД LUNA Streams
- запускает LUNA Streams и FaceStream
- тестируется с использованием настроек LUNA Streams и FaceStream по умолчанию.
- не предназначен для использования в целях масштабирования FaceStream:
 - Не используется для развертывания FaceStream на нескольких серверах.
 - Не используется для развертывания и балансирования нескольких экземпляров FaceStream на одном сервере.
- поддерживает использование GPU для вычислений FaceStream.
- не обеспечивает возможность использования внешней базы данных для LUNA Streams, уже установленной на сервере.
- не выполняет миграции из предыдущих версий FaceStream и обновления предыдущих сборок FaceStream.

См. файл «docker-compose.yml» и другие файлы в директории «example-docker» для получения дополнительной информации.

Можно написать собственный скрипт, который разворачивает и конфигурирует все необходимые сервисы LP и FaceStream. Данный документ не включает информацию о создании скриптов и не обучает использованию Docker. Обратитесь к документации Docker для получения подробной информации о Docker и Docker Compose:

<https://docs.docker.com>

2.2.1 Команда запуска FaceStream с помощью Docker Compose

Откройте директорию Docker Compose:

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

Убедитесь в том, что контейнер FS не запущен до выполнения скрипта. Попытка запустить контейнер с таким же именем, как существующий контейнер, приведет к ошибке. Если контейнер запущен, необходимо остановить его с помощью команды `docker container rm -f <container_name>`. Чтобы остановить все контейнеры, используйте `docker container rm -f $(docker container ls -aq)`.

Для запуска FaceStream с GPU с помощью Docker Compose необходимо выполнить действия, описанные в разделе [«Установка зависимостей для GPU»](#).

Запуск FaceStream через Docker Compose:

Необходимо выполнить вход в VisionLabs registry (см. раздел [«Вход в registry»](#))

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

--configurator_address=127.0.0.1 — адрес сервиса Configurator

Проверьте статус всех запущенных Docker контейнеров.

```
docker ps
```

Список потоков доступен по адресу <http://127.0.0.1:34569/api/1/streams/>. Просмотр потока в браузере доступен по адресу http://127.0.0.1:34569/api/1/streams/preview/<stream_id>.

3 Дальнейшие действия

Для дальнейшей работы с FaceStream необходимо создать поток с помощью POST-запроса «create stream» к сервису LUNA Streams. Поток содержит политики обработки видеопотока/видеофайла/-набора изображений. Если поток создается со статусом «pending» (по умолчанию), то «рабочий процесс» FaceStream автоматически начнет обработку потока.

См. подробную информацию работе с потоками и LUNA Streams в разделе «Взаимодействие FaceStream с LUNA Streams» руководства администратора.

Список обрабатываемых потоков доступен по адресу <http://127.0.0.1:34569/api/1/streams/>. Просмотр потока в браузере доступен по адресу http://127.0.0.1:34569/api/1/streams/preview/<stream_id>.

4 Дополнительная информация

В данном разделе приводится следующая дополнительная информация:

- [Полезные команды для работы с Docker](#)
- [Описание ключей запуска](#)
- [Действия по включению сохранения логов в файлы](#)
- [Настройка ротации логов Docker](#)

4.1 Команды Docker

4.1.1 Показать контейнеры

Чтобы показать список запущенных Docker контейнеров, используйте команду:

```
docker ps
```

Чтобы показать все имеющиеся Docker контейнеры, используйте команду:

```
docker ps -a
```

4.1.2 Копировать файлы в контейнер

Можно переносить файлы в контейнер. Используйте команду `docker cp` для копирования файла в контейнер.

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

4.1.3 Вход в контейнер

Можно входить в отдельные контейнеры с помощью следующей команды:

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

Для выхода из контейнера используйте следующую команду:

```
exit
```

4.1.4 Имена образов

Можно увидеть все имена образов с помощью команды

```
docker images
```

4.1.5 Просмотр логов контейнера

Просмотреть логи контейнера можно с помощью следующей команды:


```
docker logs <container_name>
```

4.1.6 Удаление образа

Если требуется удаление образа:

- запустите команду `docker images`
- найдите требуемый образ, например: `dockerhub.visionlabs.ru/luna/facestream:v.5.1.49`
- скопируйте соответствующий ID образа из IMAGE ID, например, «61860d036d8c»
- укажите его в команде удаления:

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

Удаление всех существующих образов:

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

4.1.7 Остановка контейнера

Контейнер можно остановить с помощью следующей команды:

```
docker stop <container_name>
```

Остановка всех контейнеров:

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

4.1.8 Удаление контейнера

Если необходимо удалить контейнер:

- запустите команду «`docker ps`»
- остановите контейнер (см. [Остановка контейнера](#))
- найдите требуемый образ, например: `dockerhub.visionlabs.ru/luna/facestream:v.5.1.49`
- скопируйте соответствующий ID контейнера из столбца CONTAINER ID, например, «23f555be8f3a»
- укажите его в команде удаления:

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

Удаление всех контейнеров:

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

4.2 Ключи запуска

Ключи запуска задаются с помощью переменных окружения:

- `--env=` — этот параметр задает переменные окружения, требуемые для запуска контейнера.

Указываются следующие основные значения:

- `CONFIGURATOR_HOST` — хост, на котором запущен сервис Configurator. Локальный хост задается в случае, если контейнер запущен на том же сервере, где работает Configurator.
- `CONFIGURATOR_PORT` — порт прослушивания для сервиса Configurator. По умолчанию используется порт 5070.
- `PORT` — порт, где FaceStream будет слушать.
- `STREAMS_ID` — в теге задается список идентификаторов потоков, которые будут запрошены из LUNA Streams для обработки. Остальные потоки будут отфильтрованы. Параметр `stream_id` выдается в ответе на запрос `create stream`.

Если значение равно « » или тег `STREAMS_ID` не задан, то FaceStream будет брать из очереди все существующие `stream_id`.

Если задано несуществующее значение, то при запуске FaceStream будет указана ошибка о некорректном UUID.

По умолчанию значение равно « ».

Для использования ключа должны быть указаны переменные `CONFIGURATOR_HOST` и `CONFIGURATOR_PORT`.

- `STREAMS_NAME` — в теге задается список имен потоков. Имена потоков задаются с помощью параметра `name` во время их создания (запрос `create streams`). Потоки с данными именами будут запрошены из LUNA Streams для обработки. Остальные потоки будут отфильтрованы.

В остальном принцип работы схож с тегом `STREAMS_ID`.

- `GROUPS_ID` и `GROUPS_NAME` — в тегах задаются список идентификаторов групп или список имён групп. Параметры `group_id` или `group_name` задаются во время создания потока (запрос `create stream`). Потоки с данными параметрами будут запрошены из LUNA Streams для обработки. Остальные потоки будут отфильтрованы.

Если значение равно « » или теги `GROUPS_ID`/`GROUPS_NAME` не заданы, то FaceStream не будет фильтровать потоки по группам.

Если задано несуществующее значение, то при запуске FaceStream будет указана ошибка о некорректном UUID.

По умолчанию значение равно « ».

Для использования ключей должны быть указаны переменные «CONFIGURATOR_HOST» и «CONFIGURATOR_PORT».

Для тегов «STREAMS_NAME», «STREAMS_ID», «GROUPS_NAME» и «GROUPS_ID» можно задать несколько значений. Пример синтаксиса: `-env=STREAMS_ID=«037f3196-c874-4eca-9d7c-91fd8dfc9593 4caf7cf7-dd0d-4ad5-a35e-b263e742e28a»`

- CONFIGS_ID — тег LUNA Configurator, который связан с основными конфигурациями для работы FaceStream. Единый тег должен быть задан для «TRACK_ENGINE_CONFIG» и «FACE_STREAM_CONFIG».

Если задано значение « », то будут использованы записи по умолчанию «TRACK_ENGINE_CONFIG» и «FACE_STREAM_CONFIG» из LUNA Configurator. Если запись по умолчанию не существует или содержит недопустимый для JSON синтаксис, то будет использован конфигурационный файл из комплекта поставки.

По умолчанию значение равно « ».

Для использования ключа должны быть указаны переменные «CONFIGURATOR_HOST» и «CONFIGURATOR_PORT».

- CONFIG_RELOAD — тег, включающий проверку наличия изменений в секции «FACE_STREAM_CONFIG» сервиса LUNA Configurator и принимающий следующие значения:
 - * «1» — отслеживание изменений включено, при наличии изменений в конфигурации будут автоматически перезапущены все контейнеры FaceStream;
 - * «0» — отслеживание изменений отключено.

По умолчанию значение равно «1».

- PULLING_TIME — тег, задающий период получения новых параметров из секции «FACE_STREAM_CONFIG» сервиса LUNA Configurator в диапазоне [1...3600] сек. Используется совместно с тегом «CONFIG-RELOAD».

По умолчанию значение равно «10».

- `--device=` — данный параметр необходим для указания адреса USB устройства. Адрес должен быть указан в источнике потока при его создании. Пример: `--device=/dev/video0`.

См. принцип работы FaceStream с LUNA Configurator в разделе «Использование FaceStream с LUNA Configurator» руководства администратора.

4.2.1 Расшифровка параметров запуска контейнера

Ниже приведена расшифровка параметров запуска контейнера:

- `docker run` — команда для запуска выбранного образа в качестве нового контейнера.

- `-v` — параметр `volume` позволяет загружать содержимое серверной папки в объем контейнера. Таким образом содержимое синхронизируется.
- `-v /var/lib/fs/fs-current/extras/conf/configs/faceengine.conf:/srv/facestream/data/faceengine.conf \` — этот параметр позволяет использовать настройки FaceEngine из конфигурационного файла «faceengine.conf».
- `-v /var/lib/fs/fs-current/extras/conf/configs/runtime.conf:/srv/facestream/data/runtime.conf \` — этот параметр позволяет монтировать конфигурационный файл среды выполнения в контейнер FaceStream. Перед изменением стандартных настроек требуется проконсультироваться со специалистами VisionLabs.
- `--network=host` — этот параметр указывает, что отсутствует симуляция сети и используется серверная сеть. При необходимости изменить порт для сторонних контейнеров следует заменить эту строку на `-p 5440:5432`. Здесь первый порт 5440 — это локальный порт, а 5432 — это порт, используемый в контейнере.
- `/etc/localtime:/etc/localtime:ro` — задает текущий часовой пояс, используемый системой контейнера.
- `--name=facestream` — этот параметр задает имя запускаемого контейнера. Имя должно быть уникальным. Если уже существует контейнер с таким же именем, произойдет ошибка.
- `--restart=always` — этот параметр определяет политику перезагрузки. Демон всегда перезагружает контейнер вне зависимости от кода завершения.
- `--detach=true` — запуск контейнера в фоновом режиме.

4.3 Запись логов на сервер

Чтобы включить сохранение логов на сервер, необходимо:

- создать директории для логов на сервере;
- активировать запись логов и задать расположение хранения логов внутри контейнеров;
- настроить синхронизацию директорий логов в контейнере с логами на сервере с помощью аргумента `volume` при старте каждого контейнера.

В скрипте Docker Compose **не настроена** синхронизация директорий с папками. Необходимо вручную добавить монтирование папок в файл `docker-compose.yml`.

4.3.1 Создание директории логов

Ниже приведены примеры команд для создания директорий для хранения логов и присвоения им прав для FaceStream и LUNA Streams.

```
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 Активация записи логов

4.3.2.1 Активация записи логов LUNA Streams и других сервисов LP

Для активации записи в файл необходимо задать настройки `log_to_file` и `folder_with_logs` в секции `<SERVICE_NAME>_LOGGER` настроек каждого сервиса.

Перейдите в интерфейс сервиса Configurator (127.0.0.1:5070) и задайте путь расположения логов в контейнере в параметре `folder_with_logs` для всех сервисов, чьи логи необходимо сохранить. Например, можно использовать путь `/srv/logs`.

Установите параметр `log_to_file` как `true` чтобы активировать запись логов в файл.

4.3.2.2 Активация записи логов FaceStream

Для активации записи логов в файл необходимо задать значение настройки `logging > mode` в секции `FACE_STREAM_CONFIG` на `l2f` (выводить логи только в файл) или `l2b` (выводить логи и файл и в консоль).

Перейдите в интерфейс сервиса Configurator (127.0.0.1:5070) и укажите необходимое значение настройки. Путь расположения логов в контейнере FaceStream невозможно изменить. Необходимо указывать путь /srv/logs при монтировании.

По умолчанию в логах FaceStream выводятся только предупреждения системы. С помощью настройки параметра «severity» можно включить вывод ошибок (см. описание параметра в руководстве администратора).

4.3.2.3 Активация записи логов сервиса Configurator

Настроек сервиса Configurator нет в пользовательском интерфейсе Configurator, они расположены в следующем файле:

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

Следует изменить параметры логирования в этом файле перед запуском сервиса Configurator или перезапустить его после внесения изменений.

Задайте путь расположения логов в контейнере в параметре FOLDER_WITH_LOGS = ./ файла. Например, FOLDER_WITH_LOGS = /srv/logs.

Установите параметр log_to_file как true чтобы активировать запись логов в файл.

4.3.3 Монтирование директорий с логами при старте сервисов

Директория с логами монтируется с помощью следующего аргумента при старте контейнера:

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

где <server_logs_folder> директория, созданная на этапе [создания директории логов](#), а <container_logs_folder> директория, созданная на этапе [активации записи логов](#).

Пример команды запуска FaceStream с монтированием директории с логами:

```
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.5.1.49
```

Примеры команд запуска контейнеров ручным способом содержат эти аргументы.

4.4 Настройка ротации логов Docker

Чтобы ограничить размер логов, генерируемых Docker, можно настроить автоматическую ротацию логов. Для этого необходимо добавить в файл `/etc/docker/daemon.json` следующие данные:

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

Это позволит Docker хранить до 5 файлов логов на контейнер, при этом каждый файл будет ограничен 100 МБ.

После изменения файла необходимо перезапустить Docker:

```
systemctl reload docker
```

Вышеописанные изменения являются значениями по умолчанию для любого вновь созданного контейнера, они не применяются к уже созданным контейнерам.