

VisionLabs LUNA PLATFORM 5

Руководство по установке

v.5.67.0

Содержание

Порты сервисов по умолчанию	5
Названия сервисов в Configurator	6
Введение	7
1 Подготовка к запуску	9
1.1 Распаковка дистрибутива	10
1.2 Создание символической ссылки	10
1.3 Изменение группы и владельца для директорий	11
1.4 SELinux и Firewall	11
1.5 Активация лицензии	12
1.5.1 Действия из руководства по активации лицензии	12
1.5.2 Способы задания настроек лицензии HASP	12
1.5.3 Способы задания настроек лицензии Guardant	13
1.6 Авторизация в registry	15
1.7 Установка Docker	15
1.8 Выбор способа записи логов	16
1.8.1 Запись логов в stdout	16
1.8.2 Запись логов в файл	16
1.9 Вычисления с помощью GPU	17
2 Запуск сервисов	19
2.1 Сторонние сервисы	21
2.1.1 InfluxDB	21
2.1.2 PostgreSQL	22
2.1.3 Redis	22
2.2 Configurator	24
2.2.1 Использование необязательных сервисов	24
2.2.2 Создание таблиц базы данных Configurator	24
2.2.3 Запуск контейнера Configurator	25
2.3 Image Store	26
2.3.1 Запуск контейнера Image Store	26
2.3.2 Создание бакетов	26
2.4 Accounts	29
2.4.1 Создание базы данных Accounts	29
2.4.2 Запуск контейнера Accounts	29
2.5 Licenses	30
2.5.1 Задание настроек лицензии с помощью Configurator	30

2.5.2	Запуск контейнера Licenses	31
2.6	Faces	32
2.6.1	Создание таблиц базы данных Faces	32
2.6.2	Запуск контейнера Faces	32
2.7	Events	33
2.7.1	Создание таблиц базы данных Events	33
2.7.2	Запуск контейнера Events	33
2.8	Сервисы Python Matcher	34
2.8.1	Использование Python Matcher без Python Matcher Proxy	34
2.8.2	Запуск контейнера Python Matcher	34
2.9	Remote SDK	36
2.9.1	Запуск контейнера Remote SDK	36
2.10	Handlers	41
2.10.1	Создание таблиц базы данных Handlers	41
2.10.2	Запуск контейнера Handlers	41
2.11	Tasks	42
2.11.1	Создание таблиц базы данных Tasks	42
2.11.2	Запуск контейнеров Tasks и Tasks Worker	42
2.12	Sender	44
2.12.1	Запуск контейнера Sender	44
2.13	API	45
2.13.1	Запуск контейнера API	45
2.13.2	Создание аккаунта	45
2.13.3	Создание расписания задачи GC	47
2.14	Admin	48
2.14.1	Запуск контейнера Admin	48
2.15	Backport 3	49
2.15.1	Создание бакета Backport 3	49
2.15.2	Создание таблиц базы данных Backport 3	49
2.15.3	Запуск контейнера Backport 3	49
2.15.4	User Interface 3	51
2.16	Backport 4	52
2.16.1	Запуск контейнера Backport 4	52
2.16.2	User Interface 4	53
3	Дополнительная информация	54
3.1	Визуализация мониторинга и логов с помощью Grafana	55
3.1.1	LUNA Dashboards	55
3.1.2	Grafana Loki	55

3.2	Команды Docker	57
3.2.1	Показать контейнеры	57
3.2.2	Копировать файлы в контейнер	57
3.2.3	Вход в контейнер	57
3.2.4	Имена образов	57
3.2.5	Удаление образа	57
3.2.6	Остановка контейнера	58
3.2.7	Удаление контейнера	58
3.3	Описание параметров запуска	60
3.3.1	Параметры запуска сервисов	60
3.3.2	Параметры создания баз данных	63
3.4	Способы изменения настроек сервисов	65
3.5	Запись логов на сервер	67
3.5.1	Создание директории логов	67
3.5.2	Активация записи логов	67
3.5.3	Монтирование директорий с логами при старте сервисов	68
3.6	Настройка ротации логов Docker	70
3.6.1	Задание пользовательских настроек InfluxDB	71
3.7	Использование Python Matcher с Python Matcher Proxy	73
3.7.1	Запуск контейнера Python Matcher Proxy	73
3.8	Масштабирование системы	74
3.8.1	Nginx	75
3.8.2	Запуск нескольких контейнеров	76
3.9	Повышение производительности сервисов в Docker-контейнерах	78
3.10	Конфигурация внешней PostgreSQL	80
3.10.1	Создание пользователя PostgreSQL	80
3.10.2	Создание базы данных Configurator	80
3.10.3	Создание базы данных Accounts	81
3.10.4	Создание базы данных Handlers	81
3.10.5	Создание базы данных Backport 3	82
3.10.6	Создание базы данных Faces	82
3.10.7	Создание базы данных Events	82
3.10.8	Создание базы данных Tasks	83
3.10.9	Компиляция VLMATCH в PostgreSQL	83
3.10.10	Добавление функции VLMATCH для БД Faces в PostgreSQL	84
3.10.11	Добавление функции VLMATCH для БД Events в PostgreSQL	85
3.10.12	Установка PostGIS для БД Events	86
3.11	VLMATCH для Oracle	86

Порты сервисов по умолчанию

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

Названия сервисов в Configurator

Таблица ниже включает в себя названия сервисов в сервисе Configurator. Данные параметры используются для конфигурации сервисов.

Сервис	Название сервиса в Configurator
API	luna-api
Licenses	luna-licenses
Faces	luna-faces
Image Store	luna-image-store
Accounts	luna-accounts
Tasks	luna-tasks
Events	luna-events
Sender	luna-sender
Admin	luna-admin
Remote SDK	luna-remote-sdk
Handlers	luna-handlers
Lambda	luna-lambda
Python Matcher	luna-python-matcher
Backport 3	luna-backport3
Backport 4	luna-backport4

Настройки для сервиса Configurator устанавливаются в его конфигурационном файле.

Введение

Данный документ описывает общий подход для развертывания LUNA PLATFORM в Docker-контейнерах.

Данный документ включает в себя пример развертывания LUNA PLATFORM. LUNA PLATFORM разворачивается в минимальной рабочей конфигурации для использования в демонстрационных целях. Данная конфигурация не является достаточной для реальной эксплуатации системы в продуктивном контуре.

Считается, что развертывание выполняется на сервере с CentOS, где LP не была установлена.

Для развертывания LUNA PLATFORM нужно выполнить действия из следующих разделов:

- «Подготовка к запуску» — действия по распаковке архивов, подготовке директорий, настройке лицензии и пр. Некоторые действия могут быть опциональными.
- «Запуск сервисов» — запуск Docker-контейнеров с сервисами LUNA PLATFORM.

В разделе «Дополнительная информация» приводится полезная информация по описанию параметров запуска сервисов, командах Docker, включении Grafana для визуализации мониторинга и пр.

Для использования LUNA PLATFORM в Docker-контейнерах требуется сетевая лицензия. Лицензия предоставляется компанией VisionLabs по запросу отдельно от поставки. Лицензионный ключ создается с помощью отпечатка системы. Этот отпечаток создается на базе информации об аппаратных характеристиках сервера. Таким образом, полученный лицензионный ключ будет работать только на том же сервере, с которого был получен отпечаток системы. LUNA PLATFORM можно активировать с помощью одной из двух утилит — HASP или Guardant. В разделе «Активация лицензии» приведены полезные ссылки на инструкции по активации лицензионного ключа для каждого способа.

Администратор должен вручную сконфигурировать Firewall и SELinux на сервере. Их конфигурация не описывается в данном документе.

В данной установке не предполагается резервное копирование каких-либо данных LP или копирование баз данных.

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

<https://docs.docker.com>

Для коммерческого использования LP рекомендуется оркестрация сервисов. Их использование не описано в данном документе.

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.

Для активации LUNA PLATFORM требуется файл лицензии. Этот файл предоставляется компанией VisionLabs по запросу.

Все действия, описанные в данном руководстве, должны выполняться пользователем **root**. В данном документе не описывается создание пользователя с привилегиями администратора и последующая установка, выполняемая этим пользователем.

1 Подготовка к запуску

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

Перед запуском LUNA PLATFORM необходимо выполнить следующие действия:

1. [Распаковать дистрибутив LUNA PLATFORM](#)
2. [Создать символическую ссылку](#)
3. [Изменить группу и владельца для директорий](#)
4. [Настроить SELinux и Firewall](#)
5. [Активировать лицензию](#)
6. [Авторизоваться в registry VisonLabs](#)
7. [Выполнить установку Docker](#)
8. [Выбрать способ записи логов](#)
9. [Настроить вычисления с помощью GPU](#), если планируется использовать GPU

1.1 Распаковка дистрибутива

Дистрибутив представляет собой архив **luna_v.5.67.0**, где **v.5.67.0** это числовой идентификатор, обозначающий версию LUNA PLATFORM.

Архив включает в себя конфигурационные файлы, требуемые для установки и использования. Он не включает в себя Docker образы сервисов, их требуется скачать из Интернета отдельно.

Переместите дистрибутив в директорию на вашем сервере перед установкой. Например, переместите файлы в директорию `/root/`. В ней не должно быть никакого другого дистрибутива или файлов лицензии кроме целевых.

Создайте директорию для распаковки файла дистрибутива.

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

Переместите дистрибутив в директорию с LUNA PLATFORM.

```
mv /root/luna_v.5.67.0.zip /var/lib/luna
```

Установите приложение для распаковки архива при необходимости

```
yum install -y unzip
```

Откройте папку с дистрибутивом

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

Распакуйте файлы

```
unzip luna_v.5.67.0.zip
```

1.2 Создание символической ссылки

Создайте символическую ссылку. Она показывает, что актуальная версия файла дистрибутива используется для запуска LUNA PLATFORM.

```
ln -s luna_v.5.67.0 current
```

1.3 Изменение группы и владельца для директорий

Сервисы LP запускаются внутри контейнеров пользователем «luna». Таким образом, требуется установить разрешения для данного пользователя на работу с примонтированными директориями.

Откройте директорию LP «example-docker»:

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

Создайте директорию для хранения настроек:

```
mkdir luna_configurator/used_dumps
```

Установите для пользователя с UID 1001 и группой 0 разрешения на работу с примонтированными директориями.

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

Откройте корневую директорию LP:

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

Создайте директорию для хранения бакетов Image Store:

```
mkdir image_store
```

Установите для пользователя с UID 1001 и группой 0 разрешения на работу с примонтированными директориями.

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

1.4 SELinux и Firewall

SELinux и Firewall необходимо настроить так, чтобы они не блокировали сервисы LUNA PLATFORM.

Конфигурация SELinux и Firewall не описываются в данном руководстве.

Если SELinux и Firewall не настроены, дальнейшая установка невозможна.

1.5 Активация лицензии

Для активации лицензии необходимо выполнить следующие действия:

- выполнить действия из [руководства по активации лицензии](#)
- задать настройки лицензирования [HASP](#) или [Guardant](#)

1.5.1 Действия из руководства по активации лицензии

Откройте руководство по активации лицензии и выполните необходимые шаги.

Примечание. Это действие является обязательным. Лицензия не будет работать без выполнения шагов по активации лицензии из соответствующего руководства.

1.5.2 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.

1.5.2.1 Задание настроек лицензии HASP с помощью дамп-файла

Откройте файл «platform_settings.json»:

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

1.5.3 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.

1.5.3.1 Задание настроек лицензии Guardant помощью дамп-файла

Откройте файл «platform_settings.json»:

```
vi /var/lib/luna/current/extras/conf/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» в руководстве по активацию 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».

1.6 Авторизация 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.

1.7 Установка Docker

Установка Docker описана в [официальной документации](#).

Примечание. При тестировании данной инструкции использовался Docker версии 25.0.3. Не гарантируется работа с более высокими версиями Docker.

Команды для быстрой установки приведены ниже.

Проверьте официальную документацию на наличие обновлений при возникновении каких-либо проблем с установкой.

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 Docker:

```
systemctl status docker
```

1.8 Выбор способа записи логов

В LUNA PLATFORM существует два способа вывода логов:

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

Настройки вывода логов задаются в настройках каждого сервиса в секции <SERVICE_NAME>_LOGGER.

При необходимости можно использовать оба способа вывода логов.

Для более подробной информации о системе логирования LUNA PLATFORM см. раздел «Логирование информации» в руководстве администратора.

1.8.1 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](#)»).

1.8.2 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.

Примеры команд запуска контейнеров в данной документации содержат аргументы для синхронизации директорий логов.

Обратите внимание, что вышеперечисленные действия должны выполняться перед, во время и после запуска сервисов. Запись логов в файл не будет работать если выполнять все действия после запуска контейнеров.

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

1.9 Вычисления с помощью GPU

Для основных вычислений, выполняемых сервисом Remote SDK, можно использовать GPU.

Пропустите данный раздел, если не собираетесь использовать GPU для вычислений.

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 LP, его необходимо загрузить из Интернета):

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

См. [документацию NVIDIA](#) для дополнительной информации.

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.

2 Запуск сервисов

В данном разделе приведены примеры:

- Создания таблиц баз данных
- Создания бакетов
- Запуска контейнеров

Сервисы LUNA PLATFORM должны запускаться в следующем порядке:

- Базы данных, балансировщики, HASP сервис и прочие сторонние сервисы
- [Configurator](#)
- [Image Store](#)
- [Accounts](#)
- [Licenses](#)
- [Faces](#)
- [Events](#)
- [Python Matcher](#)
- [Python Matcher Proxy](#). Сервис отключен по умолчанию.
- [Remote SDK](#)
- [Handlers](#)
- [Tasks](#)
- [Sender](#)
- [API](#)
- [Admin](#)

Следующие сервисы используются, когда требуется обеспечить совместимость с запросами формата LUNA PLATFORM 3:

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

Следующие сервисы используются, когда требуется обеспечить совместимость с запросами формата LUNA PLATFORM 4:

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

Рекомендуется запускать контейнеры один за другим и ожидать отображения статуса контейнера «up» (команда `docker ps`).

Некоторые из этих сервисов не являются обязательными к запуску и можно отключить их использование. Рекомендуется использовать сервисы Events, Tasks, Sender и Admin по умолчанию. См. раздел [«Использование необязательных сервисов»](#) для более подробной информации.

При запуске каждого сервиса используются определенные параметры, например, `--detach`, `--`

network и др. См. раздел «[Описание параметров запуска](#)» для получения более подробной информации о всех параметрах запуска сервисов LUNA PLATFORM и баз данных.

См. раздел «[Команды Docker](#)» для получения более подробной информации о работе с контейнерами.

2.1 Сторонние сервисы

В данном разделе описывается запуск баз данных в Docker-контейнерах. Они должны быть запущены перед сервисами LP.

2.1.1 InfluxDB

Для мониторинга сервисов LUNA PLATFORM требуется наличие запущенной базы данных Influx 2.0.8-alpine. Ниже приведены команды по запуску контейнера InfluxDB.

Дополнительную информацию см. в разделе «Мониторинг» в руководстве администратора.

При необходимости можно настроить визуализацию данных мониторинга с помощью сервиса LUNA Dashboards, включающего в себя настроенную систему визуализации данных Grafana. Кроме того, можно запустить инструмент для расширенной работы с логами Grafana Loki. См. инструкцию по запуску LUNA Dashboards и Grafana Loki в разделе [«Визуализация мониторинга и логов с помощью Grafana»](#).

Примечание. При необходимости можно использовать внешнюю БД InfluxDB 2.0.8-alpine. В таком случае можно пропустить команду ниже, однако вам придется задать [пользовательские настройки](#) для каждого сервиса LUNA PLATFORM.

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/luna/influx:/var/lib/influxdb2 \
--restart=always \
--detach=true \
--network=host \
--name influxdb \
dockerhub.visionlabs.ru/luna/influxdb:2.0.8-alpine
```

2.1.2 PostgreSQL

Примечание. При необходимости можно использовать внешнюю СУБД PostgreSQL. В таком случае можно пропустить команду ниже и выполнить действия из раздела [«Конфигурация внешней PostgreSQL»](#).

Используйте следующую команду для запуска PostgreSQL.

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

`-v /var/lib/luna/current/example-docker/postgresql/entrypoint-initdb.d:/docker-entrypoint-initdb.d/ \` — скрипт `«docker-entrypoint-initdb.d»` включает в себя команды для создания баз данных сервисов.

`-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.

Все базы данных для сервисов LP следует создавать вручную, если используется уже установленный PostgreSQL.

2.1.3 Redis

Примечание. Если у вас уже установлен Redis, пропустите этот шаг.

Используйте следующую команду для запуска Redis.

```
docker run \
```

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

2.2 Configurator

2.2.1 Использование необязательных сервисов

Следующие сервисы необязательны для LP:

- Events
- Image Store
- Tasks
- Sender
- Handlers
- Python Matcher Proxy (отключен по умолчанию)
- Lambda (отключен по умолчанию)

Работа с сервисом Lambda возможна только при разворачивании сервисов LUNA PLATFORM в Kubernetes. См. подробную информацию в руководстве по разворачиванию LP в Kubernetes.

Эти сервисы можно отключить при отсутствии необходимости в них.

Используйте секцию «ADDITIONAL_SERVICES_USAGE» в настройках сервиса API в сервисе Configurator, чтобы отключить ненужные сервисы.

Можно использовать файл сброса, предоставленный в комплекте поставки, для включения/отключения сервисов перед запуском сервиса Configurator.

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

Отключение какого-либо из сервисов имеет определенные последствия. См. подробную информацию в разделе «Отключаемые сервисы» руководства администратора.

2.2.2 Создание таблиц базы данных 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/luna/current/example-docker/luna_configurator/configs/  
  luna_configurator_postgres.conf:/srv/luna_configurator/configs/config.  
  conf \  
-v /var/lib/luna/current/extras/conf/platform_settings.json:/srv/  
  luna_configurator/used_dumps/platform_settings.json \  
--network=host \  
-v /tmp/logs/configurator:/srv/logs \  

```



```
--rm \
--entrypoint bash \
dockerhub.visionlabs.ru/luna/luna-configurator:v.2.2.18 \
-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/luna/current/extras/conf/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. Ревизия потребуется в процессе обновления на новую сборку LP5.

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

2.2.3 Запуск контейнера Configurator

Используйте команду `docker run` со следующими параметрами для запуска Configurator:

```
docker run \
--env=PORT=5070 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /etc/localtime:/etc/localtime:ro \
-v /var/lib/luna/current/example-docker/luna_configurator/configs/
luna_configurator_postgres.conf:/srv/luna_configurator/configs/config.
conf \
-v /tmp/logs/configurator:/srv/logs \
--name=luna-configurator \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-configurator:v.2.2.18
```

На данном этапе можно активировать запись логов в файл, если необходимо сохранять их на сервере (см. раздел «Запись логов на сервер»).

2.3 Image Store

2.3.1 Запуск контейнера Image Store

Примечание. Если вы не собираетесь использовать сервис Image Store, не запускайте этот контейнер и отключите использование сервиса в сервисе Configurator. См. раздел [«Использование необязательных сервисов»](#).

Используйте следующую команду для запуска сервиса Image Store:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5020 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /var/lib/luna/image_store:/srv/local_storage/ \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/image-store:/srv/logs \
--name=luna-image-store \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-image-store:v.3.14.2
```

-v /var/lib/luna/image_store:/srv/local_storage/ — данные из указанной директории добавляются в Docker-контейнер, когда он запущен. Все данные из указанной директории Docker-контейнера сохраняются в данную директорию.

Если директория с бакетами LP уже создана, укажите ее вместо /var/lib/luna/image_store/.

2.3.2 Создание бакетов

Бакеты используются для хранения данных в Image Store. Сервис Image Store должен быть запущен перед выполнением команд.

При обновлении с предыдущей версии рекомендуется запустить команды создания бакетов еще раз. Это гарантирует создание всех необходимых бакетов.

Если в процессе запуска приведенных выше команд появляется ошибка со статус-кодом 13006, это означает, что бакет уже создан.

Бакеты в LP можно создавать двумя способами:

- с помощью скрипта `lis_bucket_create.py`, расположенного в контейнере сервиса Image Store
- с помощью прямых запросов к сервису Image Store

Скрипт создания бакетов

Запустите данный скрипт для создания основных бакетов:

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/api:/srv/logs \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/luna-api:v.6.29.0 \
python3 ./base_scripts/lis_bucket_create.py -ii --luna-config http://
localhost:5070/1
```

Если необходимо задать время хранения объекта в бакете, то можно дополнительно указать количество дней с помощью аргумента `--bucket-ttl`. См. подробную информацию в разделе «Жизненный цикл объектов» руководства администратора.

Если вы собираетесь использовать сервис Tasks, используйте следующую команду, чтобы дополнительно создать «task-result» в сервисе Image Store:

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/tasks:/srv/logs \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.22.1 \
python3 ./base_scripts/lis_bucket_create.py -ii --luna-config http://
localhost:5070/1
```

Если вы собираетесь использовать портреты, используйте следующую команду, чтобы дополнительно создать «portraits».

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/api:/srv/logs \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/luna-backport3:v.0.11.9 \
python3 ./base_scripts/lis_bucket_create.py -ii --luna-config http://
localhost:5070/1
```

Запросы к сервису Image Store

Для следующих запросов требуется утилита curl.

Бакет «visionlabs-samples» используется для хранения биометрических образцов лиц. Этот бакет требуется для использования LP.

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

Если необходимо задать время хранения объекта в бакете, то можно дополнительно указать количество дней с параметра запроса `ttl`. См. подробную информацию в разделе «Жизненный цикл объектов» руководства администратора.

Бакет «portraits» используется для хранения портретов. Этот бакет требуется для использования Backport 3.

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

Бакет «visionlabs-bodies-samples» используется для хранения биометрических образцов тел. Этот бакет требуется для использования LP.

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

Бакет «visionlabs-image-origin» используется для хранения исходных изображений. Этот бакет требуется для использования LP.

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

Бакет «visionlabs-objects» используется для хранения объектов. Этот бакет требуется для использования LP.

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

Бакет «task-result» для сервиса Tasks. Не используйте его, если не собираетесь использовать этот сервис.

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

2.4 Accounts

2.4.1 Создание базы данных Accounts

Используйте следующую команду для создания таблиц базы данных Accounts:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/accounts:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-accounts:v.0.3.9 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.4.2 Запуск контейнера Accounts

Используйте следующую команду для запуска сервиса Accounts:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5170 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/accounts:/srv/logs \  
--name=luna-accounts \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-accounts:v.0.3.9
```

2.5 Licenses

Примечание. Для использования триальной лицензии необходимо запускать сервис Licenses на том же сервере, на котором она используется.

2.5.1 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](#)».

2.5.1.1 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» в формате «127.0.0.1».
- 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».

2.5.1.2 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».

2.5.2 Запуск контейнера 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.0.10.18
```

2.6 Faces

2.6.1 Создание таблиц базы данных Faces

Используйте следующую команду для создания таблиц базы данных Faces:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/faces:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-faces:v.4.11.9 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.6.2 Запуск контейнера Faces

Используйте следующую команду для запуска сервиса:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5030 \  
--env=WORKER_COUNT=2 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/faces:/srv/logs \  
--name=luna-faces \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-faces:v.4.11.9
```


2.7 Events

2.7.1 Создание таблиц базы данных Events

Примечание. Если вы не собираетесь использовать сервис Events, не запускайте этот контейнер и отключите использование сервиса в сервисе Configurator. См. раздел «[Использование необязательных сервисов](#)».

Используйте следующую команду для создания таблиц базы данных Events:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/events:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-events:v.4.12.9 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.7.2 Запуск контейнера Events

Примечание. Если вы не собираетесь использовать сервис Events, не запускайте этот контейнер и отключите использование сервиса в сервисе Configurator. См. раздел «[Использование необязательных сервисов](#)».

Используйте следующую команду для запуска сервиса:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5040 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/events:/srv/logs \  
--name=luna-events \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-events:v.4.12.9
```

2.8 Сервисы Python Matcher

Для задач сравнения можно использовать либо только сервис Python Matcher, либо дополнительно использовать сервис Python Matcher Proxy, который перенаправляет запросы сравнения либо сервису Python Matcher либо плагинам сравнения. В данном разделе описывается использование Python Matcher без Python Matcher Proxy.

Необходимо использовать сервис Python Matcher Proxy только если собираетесь использовать плагины сравнения. Использование Python Matcher Proxy и запуск соответствующего docker-контейнера описаны в разделе [«Использование Python Matcher с Python Matcher Proxy»](#).

См. описание и использование плагинов сравнения в руководстве администратора.

2.8.1 Использование Python Matcher без Python Matcher Proxy

Сервис Python Matcher со сравнением посредством базы данных Faces включен по умолчанию при запуске.

Сервис Python Matcher со сравнением посредством Events также включен по умолчанию. Его можно отключить, указав «USE_LUNA_EVENTS = 0» в разделе «ADDITIONAL_SERVICES_USAGE» настроек Configurator (см. раздел [«Использование необязательных сервисов»](#)). Таким образом, сервис Events не будет использоваться для LUNA PLATFORM.

Python Matcher, который производит сравнение с помощью библиотеки сравнений, включается когда «CACHE_ENABLED» установлен как «true» в настройке «DESCRIPTORS_CACHE».

Для сервисов Python Matcher и Python Matcher Proxy загружается одно изображение.

2.8.2 Запуск контейнера Python Matcher

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5100 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/python-matcher:/srv/logs \
--name=luna-python-matcher \
--restart=always \
```

```
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-python-matcher:v.1.9.9
```

2.9 Remote SDK

2.9.1 Запуск контейнера Remote SDK

Вы можете запустить сервис Remote SDK, используя CPU (задано по умолчанию) или GPU.

По умолчанию сервис Remote SDK запускается со всеми включенными эстиматорами и детекторами. При необходимости можно отключить использование некоторых эстиматоров или детекторов при запуске контейнера Remote SDK. Отключение ненужных эстиматоров позволяет экономить оперативную память или память GPU, поскольку при старте сервиса Remote SDK выполняется проверка возможности выполнения указанных оценок и загрузка нейронных сетей в память. При отключении эстиматора или детектора можно также удалить его нейронную сеть из контейнера Remote SDK. См. подробную информацию в разделе «Включение/отключение некоторых эстиматоров и детекторов» руководства администратора.

Запустите сервис Remote SDK, используя одну из следующих команд в соответствии с используемым процессором.

2.9.1.1 Запуск Remote SDK с использованием CPU

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5220 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/remote-sdk:/srv/logs \
--network=host \
--name=luna-remote-sdk \
--restart=always \
--detach=true \
dockerhub.visionlabs.ru/luna/luna-remote-sdk:v.0.6.0
```

2.9.1.2 Запуск Remote SDK с использованием GPU

Сервис Remote SDK не использует GPU по умолчанию. Если вы собираетесь использовать GPU, то следует включить его использование для сервиса Remote SDK в сервисе Configurator.

Если необходимо использовать GPU сразу для всех эстиматоров и детекторов, то необходимо использовать параметр «global_device_class» в секции «LUNA_REMOTE_SDK_RUNTIME_SETTINGS».

Все эstimаторы и детекторы будут использовать значение данного параметра, если в параметре «device_class» их собственных настроек выставлено значение «global» (по умолчанию).

Если необходимо использовать GPU для определенного эstimатора или детектора, то необходимо использовать параметр «device_class» в секциях вида "LUNA_REMOTE_SDK_<estimator-or-detector-name>_SETTINGS.runtime_settings".

См. раздел «[Вычисления с помощью GPU](#)» для получения дополнительных требований к использованию GPU.

Используйте следующую команду для запуска сервиса Remote SDK с помощью GPU:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5220 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
--gpus device=0 \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/remote-sdk:/srv/logs \
--network=host \
--name=luna-remote-sdk \
--restart=always \
--detach=true \
dockerhub.visionlabs.ru/luna/luna-remote-sdk:v.0.6.0
```

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

2.9.1.3 Запуск облегченной версии Remote SDK

Можно запустить облегченную версию сервиса Remote SDK, содержащую только конфигурационные файлы без нейронных сетей. Предполагается, что пользователь сам добавит в контейнер необходимые ему нейронные сети.

Запуск облегченной версии сервиса Remote SDK предназначен для продвинутых пользователей.

Для успешного запуска контейнера Remote SDK с пользовательским набором нейронных сетей нужно выполнить следующие действия:

- запросить у VisionLabs требуемые нейронные сети
- поместить нейронные сети в папку с установленной LUNA PLATFORM

- присвоить соответствующие права для файлов нейронных сетей
- смонтировать файлы нейронных сетей в папку /srv/fsdk/data контейнера Remote SDK
- с помощью аргументов переменной «EXTEND_CMD» явно указать какие из нейронных сетей должны использоваться

Обратите внимание, что с помощью флага «enable-all-estimators-by-default» для переменной «EXTEND_CMD» можно выключить по умолчанию использование всех нейронных сетей (эстиматоров), а затем с помощью специальных флагов явно указывать какие нейронные сети должны быть использованы. Если не указывать данный флаг или выставить значение «-enable-all-estimators-by-default=1», то сервис Remote SDK будет пытаться найти в контейнере все нейронные сети. Если какая-то из нейронных сетей не будет найдена, то сервис Remote SDK не запустится.

Список доступных аргументов для запуска:

Аргумент	Описание
--enable-all-estimators-by-default	включить все эстиматоры по умолчанию
--enable-human-detector	одновременный детектор
--enable-face-detector	детектор лиц
--enable-body-detector	детектор тел
--enable-face-landmarks5-estimator	эстиматор 5 контрольных точек лица
--enable-face-landmarks68-estimator	эстиматор 68 контрольных точек лица
--enable-head-pose-estimator	эстиматор положения головы
--enable-liveness-estimator	эстиматор OneShotLiveness
--enable-fisheye-estimator	эстиматор бочообразной дисторсии (эффекта FishEye)
--enable-face-detection-background-estimator	эстиматор фона изображения
--enable-face-warp-estimator	эстиматор биометрического образца лица
--enable-body-warp-estimator	эстиматор биометрического образца тела
--enable-quality-estimator	эстиматор качества изображения
--enable-image-color-type-estimator	эстиматор типа цвета по лицу
--enable-face-natural-light-estimator	эстиматор естественности освещения
--enable-eyes-estimator	эстиматор глаз
--enable-gaze-estimator	эстиматор направления взгляда
--enable-mouth-attributes-estimator	эстиматор атрибутов рта

Аргумент	Описание
--enable-emotions-estimator	эстиматор эмоций
--enable-mask-estimator	эстиматор маски
--enable-glasses-estimator	эстиматор очков
--enable-eyebrow-expression-estimator	эстиматор бровей
--enable-red-eyes-estimator	эстиматор красных глаз
--enable-headwear-estimator	эстиматор головного убора
--enable-basic-attributes-estimator	эстиматор базовых атрибутов
--enable-face-descriptor-estimator	эстиматор извлечения биометрического шаблона лица
--enable-body-descriptor-estimator	эстиматор извлечения биометрического шаблона тела
--enable-body-attributes-estimator	эстиматор атрибутов тел
--enable-people-count-estimator	эстиматор количества людей
--enable-deepfake-estimator	эстиматор Deepfake

См. подробную информацию включения и выключения определенных эстиматоров в разделе «Включение/отключение некоторых эстиматоров и детекторов» руководства администратора.

Ниже приведен пример команды для присвоения прав файлу нейронной сети:

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

Пример команды запуска контейнера Remote SDK с монтированием нейронных сетей для детекции лиц и извлечения биометрических шаблонов лиц:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5220 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
--env=EXTEND_CMD="--enable-all-estimators-by-default=0 --enable-face-
detector=1 --enable-face-descriptor-estimator=1" \
```

```
-v /var/lib/luna/current/cnn59b_cpu-avx2.plan:/srv/fsdk/data/cnn59b_cpu-avx2
.plan \
-v /var/lib/luna/current/FaceDet_v3_a1_cpu-avx2.plan:/srv/fsdk/data/
FaceDet_v3_a1_cpu-avx2.plan \
-v /var/lib/luna/current/FaceDet_v3_redetect_v3_cpu-avx2.plan:/srv/fsdk/data
/FaceDet_v3_redetect_v3_cpu-avx2.plan \
-v /var/lib/luna/current/slnet_v3_cpu-avx2.plan:/srv/fsdk/data/slnet_v3_cpu-
avx2.plan \
-v /var/lib/luna/current/LNet_precise_v2_cpu-avx2.plan:/srv/fsdk/data/
LNet_precise_v2_cpu-avx2.plan \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/remote-sdk:/srv/logs \
--network=host \
--name=luna-remote-sdk \
--restart=always \
--detach=true \
dockerhub.visionlabs.ru/luna/luna-remote-sdk:v.0.6.0
```


2.10 Handlers

Примечание. Если вы не собираетесь использовать сервис Handlers, не запускайте этот контейнер и отключите использование сервиса в сервисе Configurator. См. раздел [«Использование необязательных сервисов»](#).

2.10.1 Создание таблиц базы данных Handlers

Используйте следующую команду для создания таблиц базы данных Handlers:

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/handlers:/srv/logs \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/luna-handlers:v.3.6.0 \
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.10.2 Запуск контейнера Handlers

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5090 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/handlers:/srv/logs \
--name=luna-handlers \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-handlers:v.3.6.0
```

2.11 Tasks

Примечание. Если вы не собираетесь использовать сервис Tasks, не запускайте контейнер Tasks и контейнер Tasks Worker. Отключите сервис Tasks в сервисе Configurator. См. раздел [«Использование необязательных сервисов»](#).

2.11.1 Создание таблиц базы данных Tasks

Используйте следующую команду для создания таблиц БД Tasks:

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/tasks:/srv/logs \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.22.1 \
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.11.2 Запуск контейнеров Tasks и Tasks Worker

Образ сервиса Tasks включает в себя сервисы Tasks и Tasks Worker («рабочие процессы сервиса Tasks»). Они оба должны быть запущены.

Необходимо создать бакет «task-result» для сервиса Tasks перед запуском сервиса. Создание бакетов описано в разделе [«Создание бакетов»](#).

Если необходимо использовать задачу Estimator с использованием сетевого диска, то необходимо предварительно смонтировать директорию с изображениями с сетевого диска в специальные директории контейнеров Tasks и Tasks Worker. См. подробную информацию в разделе «Задача Estimator» в руководстве администратора.

2.11.2.1 Запуск контейнера Tasks worker

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5051 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
--env=SERVICE_TYPE="tasks_worker" \
```

```
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/tasks-worker:/srv/logs \  
--name=luna-tasks-worker \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.22.1
```

2.11.2.2 Запуск контейнера Tasks

Используйте следующую команду для запуска сервиса:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5050 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/tasks:/srv/logs \  
--name=luna-tasks \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-tasks:v.3.22.1
```

2.12 Sender

2.12.1 Запуск контейнера Sender

Примечание. Если вы не собираетесь использовать сервис Sender, не запускайте этот контейнер и отключите этот сервис в Configurator. См. раздел «[Использование необязательных сервисов](#)».

Используйте следующую команду для запуска сервиса:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5080 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/sender:/srv/logs \  
--name=luna-sender \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-sender:v.2.11.9
```

2.13 API

2.13.1 Запуск контейнера API

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5000 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
--name=luna-api \
--restart=always \
--detach=true \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/api:/srv/logs \
--network=host \
dockerhub.visionlabs.ru/luna/luna-api:v.6.29.0
```

2.13.2 Создание аккаунта

Аккаунт создается с помощью HTTP-запроса к ресурсу «create account».

Аккаунт также можно создать с помощью сервиса Admin. Данный способ требует наличия существующих логина и пароль (или логина и пароля по умолчанию) и позволяет создать аккаунты типа «admin». См. подробную информацию в разделе «Сервис Admin» руководства администратора.

Для создания аккаунта с помощью запроса к сервису API необходимо указать следующие обязательные данные:

- login — электронный адрес
- password — пароль
- account_type — тип аккаунта («user» или «advanced_user»)

Создайте аккаунт, используя свои аутентификационные данные.

Пример CURL-запроса к ресурсу «create account»:

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

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

Необходимо заменить аутентификационные данные из примера на свои.

См. подробную информацию об аккаунтах в разделе «Аккаунты и типы авторизации» руководства администратора.

Для работы с токенами необходимо наличие аккаунта.

2.13.3 Создание расписания задачи GC

Перед началом работы с LUNA PLATFORM можно создать расписание для задачи Garbage collection.

Для этого следует выполнить запрос «create tasks schedule» к сервису API, указав необходимые правила для расписания.

Пример команды создания расписания для аккаунта из раздела «Создание аккаунта», приведен ниже.

В примере задается расписание для задачи Garbage collection для событий старше 30 дней с удалением БО и исходных изображений. Задача будет повторяться **один раз в сутки в 05:30 утра**.

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

При необходимости можно создать расписание без его автоматической активации. Для этого нужно указать параметр «create_stopped»: «true». В таком случае после создания расписания его необходимо активировать вручную с помощью параметра «action» = «start» запроса «patch tasks schedule».

См. подробную информацию в разделе «Запуск задач по расписанию» руководства администратора.

2.14 Admin

2.14.1 Запуск контейнера Admin

Примечание. Если вы не собираетесь использовать сервис Admin, не запускайте этот контейнер.

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5010 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/admin:/srv/logs \
--name=luna-admin \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-admin:v.5.7.1
```

Данные о количестве выполненных запросов сохраняются в бакете luna-admin базы данных Influx. Для включения сохранения этих данных требуется выполнить следующую команду:

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


2.15 Backport 3

В данном разделе описывается запуск сервиса Backport 3.

Сервис не обязателен для использования LP5 и требуется только для эмуляции LP 3 API.

2.15.1 Создание бакета Backport 3

Используйте следующую команду для создания пакета «portraits» для Backport 3:

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

2.15.2 Создание таблиц базы данных Backport 3

Используйте следующую команду для создания таблиц базы данных для Backport 3:

```
docker run \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/backport3:/srv/logs \  
--rm \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-backport3:v.0.11.9 \  
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

2.15.3 Запуск контейнера Backport 3

Используйте следующую команду для запуска сервиса:

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

`dockerhub.visionlabs.ru/luna/luna-backport3:v.0.11.9`

2.15.4 User Interface 3

User Interface 3 используется только с сервисом Backport 3.

2.15.4.1 Запуск контейнера User Interface 3

Используйте следующую команду для запуска сервиса:

```
docker run \  
--env=PORT=4100 \  
--env=LUNA_API_URL=http://127.0.0.1:5140 \  
--name=luna-ui-3 \  
--restart=always \  
--detach=true \  
--network=host \  
-v /etc/localtime:/etc/localtime:ro \  
dockerhub.visionlabs.ru/luna/luna3-ui:v.0.5.13
```

--env=LUNA_API_URL — URL сервиса Backport 3.

--env=PORT — порт сервиса User Interface 3.

2.16 Backport 4

В данном разделе описывается запуск сервиса Backport 4.

Этот сервис необязателен для использования LP5 и требуется только для эмуляции LP 4 API.

2.16.1 Запуск контейнера Backport 4

Используйте следующую команду для запуска сервиса:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5130 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--name=luna-backport4 \  
--restart=always \  
--detach=true \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/backport4:/srv/logs \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-backport4:v.1.5.9
```

2.16.2 User Interface 4

User Interface 4 используется только с сервисом Backport 4.

2.16.2.1 Запуск контейнера User Interface 4

Примечание. Перед запуском контейнера User Interface 4 необходимо наличие аккаунта типа **user**. Его логин и пароль в формате Base64 будут использованы для работы с пользовательским интерфейсом.

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=PORT=4200 \
--env=LUNA_API_URL=http://<server_external_ip>:5130 \
--env=BASIC_AUTH=dXNlckBtYWlsLmNvbTpwYXNzd29yZA== \
--name=luna-ui-4 \
--restart=always \
--detach=true \
--network=host \
-v /etc/localtime:/etc/localtime:ro \
dockerhub.visionlabs.ru/luna/luna4-ui:v.0.1.8
```

--env=PORT — задает порт для запуска User Interface 4.

--env=BASIC_AUTH — задает авторизацию типа Basic для аккаунта, данные которого отображаются в пользовательском интерфейсе. Необходимо перевести login:password, созданные согласно разделу «Создание аккаунта» в формат Base64. Для аккаунта должен быть задан тип **user**.

--env=LUNA_API_URL — задает URL сервиса Backport 4.

- Необходимо использовать внешний IP сервиса, а не локальный хост.
- Необходимо указать порт сервиса Backport 4 (5130 задан по умолчанию).

3 Дополнительная информация

В данном разделе приводится следующая дополнительная информация:

- [Визуализация мониторинга и логов с помощью Grafana](#)
- [Полезные команды для работы с Docker](#)
- [Описание параметров запуска сервисов LUNA PLATFORM и создания баз данных](#)
- [Способы изменения настроек сервисов](#)
- [Действия по включению сохранения логов сервисов LP в файлы](#)
- [Настройка ротации логов Docker](#)
- [Задание пользовательских настроек InfluxDB](#)
- [Использование сервиса Python Matcher с сервисом Python Matcher Proxy](#)
- [Масштабирование системы](#)
- [Повышение производительности сервисов в Docker-контейнерах](#)
- [Ручное создание баз данных при использовании внешней СУБД PostgreSQL \(в т.ч. создание функции VLMatch для БД Faces и Events\)](#)
- [Компиляция библиотеки VLMatch для Oracle](#)

3.1 Визуализация мониторинга и логов с помощью Grafana

Визуализация мониторинга выполняется за счет сервиса LUNA Dashboards, который содержит в себе платформу для визуализации данных мониторинга Grafana с настроенными дашбордами LUNA PLATFORM.

При необходимости можно отдельно установить настроенные дашборды для Grafana. См. дополнительную информацию в разделе «LUNA Dashboards» в руководстве администратора.

Вместе с Grafana можно использовать систему агрегации логов Grafana Loki, позволяющую гибко работать с логами LUNA PLATFORM. Для доставки логов LUNA PLATFORM в Grafana Loki используется агент Promtail (дополнительную информацию см. в разделе «Grafana Loki» в руководстве администратора).

3.1.1 LUNA Dashboards

Примечание. Для работы с Grafana необходимо использовать InfluxDB версии 2.

3.1.1.1 Запуск контейнера LUNA Dashboards

Используйте команду `docker run` со следующими параметрами для запуска LUNA Dashboards:

```
docker run \
--restart=always \
--detach=true \
--network=host \
--name=grafana \
-v /etc/localtime:/etc/localtime:ro \
dockerhub.visionlabs.ru/luna/luna-dashboards:v.0.1.0
```

Для использования веб-интерфейса Grafana нужно перейти по адресу «`http://IP_ADDRESS:3000`», при условии, что контейнеры LUNA Dashboards и InfluxDB были запущены.

3.1.2 Grafana Loki

Примечание. Для запуска Grafana Loki требуется наличие запущенного сервиса LUNA Dashboards.

3.1.2.1 Запуск контейнера Grafana Loki

Используйте команду `docker run` со следующими параметрами для запуска Grafana Loki:

```
docker run \
--name=loki \
```

```
--restart=always \  
--detach=true \  
--network=host \  
-v /etc/localtime:/etc/localtime:ro \  
dockerhub.visionlabs.ru/luna/loki:2.7.1
```

3.1.2.2 Запуск контейнера Promtail

Используйте команду `docker run` со следующими параметрами для запуска Promtail:

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

`-v /var/lib/luna/current/example-docker/logging/promtail.yml:/etc/promtail/
luna.yml` — монтирование конфигурационного файла в контейнер Promtail

`-config.file=/etc/promtail/luna.yml` — флаг с адресом конфигурационного файла

`-client.url=http://127.0.0.1:3100/loki/api/v1/push` — флаг с адресом развернутой Grafana Loki

`-client.external-labels=job=containerlogs,pipeline_id=,job_id=,version=` — статические метки для добавления ко всем логам, отправляемым в Grafana Loki

3.2 Команды Docker

3.2.1 Показать контейнеры

Чтобы показать список запущенных Docker-контейнеров, используйте команду:

```
docker ps
```

Чтобы показать все имеющиеся Docker-контейнеры, используйте команду:

```
docker ps -a
```

3.2.2 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>
```

3.2.3 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
```

3.2.4 Имена образов

Можно увидеть все имена образов с помощью команды

```
docker images
```

3.2.5 Удаление образа

Если требуется удаление образа:

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

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

Удалите все существующие образы:

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

3.2.6 Остановка контейнера

Контейнер можно остановить с помощью следующей команды:

```
docker stop <container_name>
```

Остановить все контейнеры:

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

3.2.7 Удаление контейнера

Если необходимо удалить контейнер:

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

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

Удалить все контейнеры:

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

3.2.7.1 Проверка логов сервисов

Чтобы показать логи сервиса, используйте команду:

```
docker logs <container_name>
```

3.3 Описание параметров запуска

При запуске Docker-контейнера для какого-либо из сервисов LUNA PLATFORM необходимо задать дополнительные параметры, требуемые для работы этого сервиса.

Параметры, требуемые для конкретного контейнера, описаны в разделе, посвященном запуску этого контейнера.

Все параметры, приведенные в примере запуска сервиса, необходимы для корректного запуска и работы сервиса.

3.3.1 Параметры запуска сервисов

Пример команды запуска контейнеров сервисов LP:

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

Следующие параметры используются при запуске контейнеров сервисов LP:

- `docker run` — команда для запуска выбранного образа в качестве нового контейнера.
- `dockerhub.visionlabs.ru/luna/<service-name>:<version>` — позволяет задать образ, требуемый для запуска контейнера.

Ссылки для загрузки требуемых образов контейнера доступны в описании запуска соответствующего контейнера.

- `--network=host` — 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. Пример приведен для PostgreSQL.

- `--env=` — задает переменные окружения, требуемые для запуска контейнера (см. раздел «Аргументы сервисов»).
- `--name=<service_container_name>` — задает имя запускаемого контейнера. Имя должно быть уникальным. Если уже существует контейнер с таким же именем, произойдет ошибка.
- `--restart=always` — определяет политику перезагрузки. Демон всегда перезагружает контейнер вне зависимости от кода завершения.
- `--detach=true` — позволяет запустить контейнер в фоновом режиме.
- `-v` — позволяет загружать содержимое серверной папки в объем контейнера. Таким образом содержимое синхронизируется. Загружаются следующие общие данные:
- `/etc/localtime:/etc/localtime:ro` — задает текущий часовой пояс, используемый системой контейнера.
- `/tmp/logs/<service>:/srv/logs/` — позволяет копировать папку с записями (логами) сервиса на сервер в директорию `/tmp/logs/<service>`. Директорию для хранения логов можно изменить при желании.

3.3.1.1 Аргументы сервисов

Каждый сервис в LUNA PLATFORM имеет свои собственные аргументы запуска. Эти аргументы можно передать через:

- задание флага для скрипта запуска (`run.py`) соответствующего сервиса
- установку отдельных переменных окружения (`--env`) в командной строке Docker

Например, с использованием флага `--help` можно получить список всех доступных аргументов. Пример передачи аргумента для сервиса API может выглядеть следующим образом:

```
docker run --rm dockerhub.visionlabs.ru/luna/luna-api:v.6.29.0 python3 /srv/luna_api/run.py --help
```

Список основных аргументов:

Флаг в строке запуска	Переменная окружения	Описание
<code>--port</code>	PORT	Порт, на котором сервис будет ожидать подключений.
<code>--workers</code>	WORKER_COUNT	Количество «рабочих процессов» для сервиса.
<code>--log_suffix</code>	LOG_SUFFIX	Суффикс, добавляемый к именам файлов логов (при включенном параметре записи логов в файл).

<code>--config-reload</code>	RELOAD_CONFIG	Включение автоматической перезагрузки конфигураций. См. раздел «Автоматическая перезагрузка конфигураций» в руководстве администратора LUNA PLATFORM 5.
<code>--pulling-time</code>	RELOAD_CONFIG_INTERVAL	Период проверки конфигураций (по умолчанию 10 секунд). См. раздел «Автоматическая перезагрузка конфигураций» в руководстве администратора LUNA PLATFORM 5.
<code>--luna-config</code>	CONFIGURATOR_HOST, CONFIGURATOR_PORT	Адрес сервиса Configurator для загрузки настроек. Для <code>--luna-config</code> передается в формате <code>http://localhost:5070/1</code> . Для переменных окружения хост и порт задаются явно. Если аргумент не задан, то будет использован конфигурационный файл по умолчанию.
<code>--config</code>	Нет	Путь до конфигурационного файла с настройками сервиса.
<code>--<config_name></code>	Нет	Тег указанной настройки в Configurator. При задании данной настройки будет использовано значение тегированной настройки. Пример: <code>--INFLUX_MONITORING TAG_1</code> . Примечание. Необходимо заранее присвоить тег соответствующим настройкам в Configurator. Примечание. Работает только с флагом <code>--luna-config</code> .
<code>--tls-cert</code>	Нет	Путь к SSL-сертификату для запуска сервиса с использованием протокола HTTPS.

<code>--tls_key</code>	Нет	Путь к SSL-закрытому ключу для запуска сервиса с использованием протокола HTTPS.
<code>--tls_key_pass</code>	Нет	Пароль для SSL-закрытого ключа для запуска сервиса с использованием протокола HTTPS.

Перечень аргументов может отличаться в зависимости от сервиса.

Также доступна возможность переопределить настройки сервисов при их старте с помощью переменных окружения.

Для переопределения настроек используется префикс `VL_SETTINGS`. Примеры:

- `--env=VL_SETTINGS.INFLUX_MONITORING.SEND_DATA_FOR_MONITORING=0`. Использование переменной окружения из данного примера установит значение настройки «`SEND_DATA_FOR_MONITORING`» для секции «`INFLUX_MONITORING`» равным «0».
- `--env=VL_SETTINGS.OTHER.STORAGE_TIME=LOCAL`. Для несоставных настроек (настроек, которые расположены в секции «`OTHER`» в конфигурационном файле) необходимо указать префикс «`OTHER`». Использование переменной окружения из данного примера установит значение настройки «`STORAGE_TIME`» (если сервис использует данную настройку) на значение «`LOCAL`».

Передача флагов с использованием переменной окружения

Флаги, для которых явно не выделена переменная окружения, можно передать с помощью переменной окружения `EXTEND_CMD`.

Например, можно передать тег настроек следующим способом:

```
--env=EXTEND_CMD="--INFLUX_MONITORING=TAG_1 --LUNA_EVENTS_DB=TAG_2"
```

3.3.2 Параметры создания баз данных

Пример команды запуска контейнеров для миграции баз данных или их создания:

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/<service>:/srv/logs/ \
--rm \
--network=host \
```

```
dockerhub.visionlabs.ru/luna/<service-name>:<version> \
python3 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

Следующие параметры используются при запуске контейнеров для миграции баз данных или их создания:

`--rm` — этот параметр указывает, удаляется ли контейнер после завершения обработки всех заданных скриптов.

`python3 ./base_scripts/db_create.py` — этот параметр содержит версию Python и скрипт `db_create.py`, запускаемый в контейнере. Этот скрипт используется для создания структуры базы данных.

`--luna-config http://localhost:5070/1` — этот параметр указывает, откуда запущенный скрипт должен получать конфигурации. По умолчанию конфигурации запрашиваются сервисами от сервиса Configurator.

3.4 Способы изменения настроек сервисов

Существует три основных способа изменения настроек, хранящихся в базе данных сервиса Configurator:

- с помощью пользовательского интерфейса сервиса Configurator,
- с помощью запросов API сервиса Configurator,
- с помощью файла с настройками.

Настройки можно изменить после запуска сервиса Configurator.

Настройки самого сервиса Configurator хранятся в файле «luna_configurator_postgres.conf», который загружается в контейнер сервиса Configurator в процессе его запуска.

Интерфейс сервиса Configurator

Можно войти в пользовательский интерфейс сервиса Configurator и изменить требуемую настройку. По умолчанию используется следующий адрес на локальном хосте: <Configurator_server_address>:5070.

API сервиса Configurator

Для обновления настроек можно использовать API сервиса Configurator.

См. спецификацию OpenAPI сервиса Configurator.

Файл сброса

Можно получить файл сброса со всеми настройками сервисов LP. Используйте одну из следующих команд:

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

или

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

Необходимо задать корректный адрес и порт сервиса Configurator.

Необходимо удалить из файла раздел «limitations». В противном случае будет невозможно применить обновление файла сброса.

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

Отредактируйте параметры в разделе «settings».

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

Скопируйте файл с настройками в контейнер сервиса Configurator.

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

Используйте полученный файл:

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

3.5 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 сервисов LP;
- настроить синхронизацию директорий логов в контейнере с логами на сервере с помощью аргумента `volume` при старте каждого контейнера.

3.5.1 Создание директории логов

Ниже приведены примеры команд для создания директорий для хранения логов и присвоения им прав для всех сервисов LUNA PLATFORM.

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

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

Если необходимо использовать сервис Python Matcher Proxy, то нужно дополнительно создать директорию `/tmp/logs/python-matcher-proxy` и установить ей разрешения.

3.5.2 Активация записи логов

3.5.2.1 Активация записи логов сервисов LP

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

Автоматический способ (перед/после запуска Configurator)

Для обновления настроек ведения логов можно использовать файл настроек `logging.json`, предоставленный в комплекте поставки.

Выполните следующую команду после запуска сервиса Configurator:

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

Обновите настройки записи логов с помощью скопированного файла.

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

Ручной способ (после запуска Configurator)

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.

3.5.2.2 Активация записи логов сервиса Configurator (перед/после запуска 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.

3.5.3 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 сервиса API с монтированием директории с логами:

```
docker run \  
--env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5000 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
--name=luna-api \  
--restart=always \  
--detach=true \  
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/api:/srv/logs \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-api:v.6.29.0
```

Примеры команд запуска контейнеров в данной документации содержат эти аргументы.

3.6 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.

3.6.1 Задание пользовательских настроек InfluxDB

Для InfluxDB OSS 2 доступны следующие настройки:

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

Можно обновить настройки InfluxDB для сервисов LP в сервисе Configurator, выполнив следующие действия:

- откройте следующий файл:

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

- задайте необходимые данные;
- сохраните изменения;
- скопируйте файл в контейнер InfluxDB:

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

- обновите настройки в сервисе Configurator.

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

Также можно вручную обновить настройки в пользовательском интерфейсе сервиса Configurator.

Настройки сервиса Configurator задаются отдельно.

- откройте файл с настройками Configurator:

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

- задайте необходимые данные;
- сохраните изменения;
- перезапустите Configurator:

```
docker restart luna-configurator
```


3.7 Использование Python Matcher с Python Matcher Proxy

Как было сказано ранее, вместе с сервисом Python Matcher можно дополнительно использовать сервис Python Matcher Proxy, который будет перенаправлять запросы сравнения либо сервису Python Matcher, либо плагинам сравнения. Использование плагинов может значительно ускорить выполнение запросов на сравнение. Например, с помощью плагинов возможно организовать хранение необходимых для выполнения операций сравнения данных и дополнительных полей объектов в отдельном хранилище, что позволит ускорить доступ к данным по сравнению с использованием стандартной БД LUNA PLATFORM.

Для использования сервиса Python Matcher с Python Matcher Proxy необходимо дополнительно запустить соответствующий контейнер, а затем выставить определенную настройку в сервисе Configurator. Выполняйте нижеперечисленные действия только если собираетесь использовать плагины сравнения.

См. описание и использование плагинов сравнения в руководстве администратора.

3.7.1 Запуск контейнера Python Matcher Proxy

Используйте следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5110 \
--env=WORKER_COUNT=1 \
--env=RELOAD_CONFIG=1 \
--env=RELOAD_CONFIG_INTERVAL=10 \
--env=SERVICE_TYPE="proxy" \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/python-matcher-proxy:/srv/logs \
--name=luna-python-matcher-proxy \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/luna-python-matcher:v.1.9.9
```

После запуска контейнера необходимо выставить следующее значение в сервисе Configurator.

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

3.8 Масштабирование системы

Все сервисы LP линейно масштабируемы и могут располагаться на нескольких серверах.

Можно запустить дополнительные контейнеры с сервисами LP для повышения производительности и отказоустойчивости. Количество сервисов и характеристики серверов зависят от решаемой задачи.

Для повышения производительности можно либо увеличить производительность одного сервера, либо увеличить количество используемых серверов, распределив наиболее ресурсозатратные компоненты системы.

Для распределения запросов среди запущенных экземпляров сервисов используются балансировщики. Этот подход обеспечивает требуемую скорость обработки и требуемый уровень отказоустойчивости для конкретных задач клиента. В случае отказа узла система не остановится: запросы перераспределятся на другой узел.

На рисунке ниже показано, как два экземпляра сервиса Faces балансируются посредством Nginx. Nginx получает запросы на порту 5030 и направляет их экземплярам Faces. Сервисы Faces запущены на портах 5031 и 5032.



Рис. 1: Балансирование сервиса Faces

Крайне рекомендуется регулярно создавать резервные копии баз данных на отдельном сервере независимо от уровня отказоустойчивости системы. Это защитит от потери данных в непредвиденных случаях.

Очереди сообщений, базы данных и балансировщики, используемые LUNA PLATFORM — это продукты сторонних разработчиков. Их необходимо конфигурировать в соответствии с рекомендациями их поставщиков.

Сервисы Remote SDK и Python Matcher выполняют наиболее ресурсозатратные операции.

Сервис Remote SDK выполняет математическое преобразование изображений и извлечение биометрических шаблонов. Эти операции требуют значительных вычислительных мощностей. Для вычислений можно использовать как CPU, так и GPU.

Использование GPU предпочтительно, т.к. обработка запросов происходит эффективнее. Однако, поддерживаются не все типы видеокарт.

Сервис Python Matcher выполняет сравнение по спискам. Сравнение требует ресурсов CPU, однако также следует выделить максимально возможный объем оперативной памяти под каждый экземпляр Python Matcher. RAM используется для хранения биометрических шаблонов, полученных из базы данных. Таким образом, сервису Python Matcher не требуется запрашивать каждый БШ из базы данных.

При распределении экземпляров на нескольких серверах следует учитывать производительность каждого сервера. Например, если крупная задача выполняется несколькими экземплярами Python Matcher, а один из них находится на сервере с низкой производительностью, выполнение всей задачи в целом может замедлиться.

Обратите внимание, что для каждого экземпляра сервиса можно задать количество «рабочих процессов». Чем больше количество «рабочих процессов», тем больше ресурсов и памяти потребляется экземпляром сервиса. См. подробную информацию в разделе «Рабочие процессы» в руководстве администратора LUNA PLATFORM.

3.8.1 Nginx

Nginx требуется при использовании нескольких экземпляров сервисов LUNA PLATFORM.

3.8.1.1 Конфигурация Nginx

Конфигурационный файл Nginx включает в себя параметры для балансировки запросов в API, Faces, Image Store и Events.

Проверьте конфигурационный файл «/var/lib/luna/current/extras/conf/nginx.conf» перед запуском NGINX.

Nginx слушает запросы на указанных портах и отправляет запросы доступным экземплярам сервисов. В конфигурационном файле Nginx указаны следующие порты и серверы.

Имя сервиса	Порт в Nginx	Порты для запущенных сервисов
API	5000	5001-5004
Faces	5030	5031-5034
Image Store	5020	5021-5024

Имя сервиса	Порт в Nginx	Порты для запущенных сервисов
Events	5040	5041-5044

3.8.1.2 Запуск контейнера Nginx

Примечание. Требуется сконфигурировать файл для использования с запущенными сервисами.

Используйте команду `docker run` со следующими параметрами для запуска контейнера Nginx:

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

`-v /var/lib/luna/current/extras/conf/nginx.conf:/etc/nginx/nginx.conf` — конфигурационный файл Nginx, используемый для запуска Nginx.

См. пример конфигурационного файла Nginx для запуска большего количества экземпляров сервисов здесь:

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

3.8.2 Запуск нескольких контейнеров

Для запуска нескольких экземпляров одного и того же сервиса LP необходимо выполнить два шага.

1. Запустить несколько контейнеров этого сервиса

Необходимо запустить требуемое количество сервисов, используя соответствующую команду для каждого сервиса.

Например, для сервиса API необходимо запустить следующую команду с обновленными параметрами.

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=<port> \
```

```
-v /etc/localtime:/etc/localtime:ro \  
-v /tmp/logs/<folder_name>:/srv/logs \  
--name=<name> \  
--restart=always \  
--detach=true \  
--network=host \  
dockerhub.visionlabs.ru/luna/luna-api:v.6.29.0
```

При запуске нескольких схожих контейнеров должны различаться следующие параметры контейнеров:

--env=PORT=<port> — порты, указываемые для схожих контейнеров, должны быть разными. Необходимо задать доступный порт для экземпляра. Например, «5001», «5002». Порт «5000» будет задан для балансировщика Nginx.

/tmp/logs/<folder_name>:/srv/logs — папки для логов должны быть с разными именами, чтобы не смешивались записи разных экземпляров сервиса.

--name=<container_name> — запущенный контейнер должен иметь другое имя, т.к. нельзя запускать два контейнера с одним и тем же именем. Например, «api_1», «api_2».

--gpus device=0 — сервисы CORE обычно используют разные устройства GPU. Таким образом необходимо указывать разные номера устройств.

2. Настройте балансировщик (например, Nginx) для маршрутизации запросов по сервисам.

Для каждого масштабированного сервиса LP необходимо задать порт, на котором Nginx будет слушать запросы сервисам и реальные порты каждого экземпляра сервиса, куда Nginx будет перенаправлять запросы.

Пример конфигурационного файла Nginx можно посмотреть здесь:

«/var/lib/luna/current/extras/conf/nginx.conf».

Можно использовать другой балансировщик, но его использование не описано в данном документе.

3.9 Повышение производительности сервисов в Docker-контейнерах

Следуйте этим шагам, если есть необходимость увеличить производительность LUNA PLATFORM в Docker-контейнерах.

Политики безопасности Docker накладывают ограничения на сетевое взаимодействие сервисов LP. При выполнении действий, приведенных ниже, и отключении этих политик новое сетевое соединение Docker-контейнеров будет устанавливаться приблизительно на 20% быстрее.

Изменение настроек может отрицательно сказаться на безопасности. Не изменяйте настройки, если плохо разбираетесь в работе Docker.

Перед выполнением приведенных ниже действий необходимо установить Docker. См. [«Установка Docker»](#).

- 1) Добавьте параметры `--network=host --security-opt seccomp=unconfined` к запуску каждого контейнера сервисов LP. Команды запуска описаны в разделе [«Запуск сервисов»](#)

Ниже приведен пример для сервиса Faces.

```
docker run --env=CONFIGURATOR_HOST=127.0.0.1 \  
--env=CONFIGURATOR_PORT=5070 \  
--env=PORT=5031 \  
--env=WORKER_COUNT=1 \  
--env=RELOAD_CONFIG=1 \  
--env=RELOAD_CONFIG_INTERVAL=10 \  
-v /etc/localtime:/etc/localtime:ro \  
--name=luna-faces1 \  
--restart=always \  
--detach=true \  
--network=host \  
--security-opt seccomp=unconfined \  
dockerhub.visionlabs.ru/luna/luna-faces:v.4.11.9
```

- 2) Добавьте «userland-proxy»: false в конфигурацию Docker.

Создайте конфигурационный файл для Docker.

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

Создайте файл.

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

Добавьте следующую опцию в открытый файл.

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

Перезапустите Docker.

```
systemctl restart docker
```

3.10 Конфигурация внешней PostgreSQL

В данном разделе описаны команды, необходимые для конфигурации внешней PostgreSQL для работы с LP. Внешней означает, что уже существует рабочая база данных и её необходимо использовать в LP.

Необходимо указать внешнюю базу данных и очередь сообщений в конфигурациях сервисов LP.

Пропустите этот раздел, если используете PostgreSQL из Docker-контейнера, поставляемого VisionLabs.

Пользователь базы данных и другие параметры могут различаться. Учитывайте это при выполнении команд.

В разделах ниже описано создание баз данных. Используйте скрипт «db_create.py» для создания структуры таблиц после создания самих таблиц. Запуск скрипта из контейнера описан для каждого сервиса в разделах выше. Например, см. раздел [«Создание таблиц базы данных Faces»](#).

3.10.1 Создание пользователя PostgreSQL

Создайте пользователя базы данных.

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

Присвойте пользователю пароль.

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

3.10.2 Создание базы данных Configurator

Предполагается, что пользователь базы данных уже создан.

Создайте базу данных для сервиса Configurator.

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

Присвойте привилегии пользователю базы данных.

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


Разрешите пользователю авторизацию в базе данных.

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

3.10.3 Создание базы данных Accounts

Предполагается, что пользователь базы данных уже создан.

Создайте базу данных для сервиса Accounts.

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

Присвойте привилегии пользователю базы данных.

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

Разрешите пользователю авторизацию в базе данных.

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

3.10.4 Создание базы данных Handlers

Предполагается, что пользователь базы данных уже создан.

Создайте базу данных для сервиса Handlers.

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

Присвойте привилегии пользователю базы данных.

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

Разрешите пользователю авторизацию в базе данных.

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

3.10.5 Создание базы данных Backport 3

Предполагается, что пользователь базы данных уже создан.

Создайте базу данных для сервиса Backport 3.

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

Присвойте привилегии пользователю базы данных.

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

Разрешите пользователю авторизацию в базе данных.

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

3.10.6 Создание базы данных Faces

Предполагается, что пользователь базы данных уже создан.

Создайте базу данных для сервиса Faces.

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

Присвойте привилегии пользователю базы данных.

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

Разрешите пользователю авторизацию в базе данных.

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

3.10.7 Создание базы данных Events

Предполагается, что пользователь базы данных уже создан.

Создайте базу данных для сервиса Events.

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

Присвойте привилегии пользователю базы данных.

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

Разрешите пользователю авторизацию в базе данных.

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

3.10.8 Создание базы данных Tasks

Предполагается, что пользователь базы данных уже создан.

Создайте базу данных для сервиса Tasks.

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

Присвойте привилегии пользователю базы данных.

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

Разрешите пользователю авторизацию в базе данных.

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

3.10.9 Компиляция VLMatch в PostgreSQL

Примечание. В следующей инструкции описана установка для PostgreSQL 16.

Все файлы, требуемые для компиляции расширения, заданного пользователем (UDx), в VLMatch, можно найти в следующей директории:

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

Для компиляции функции VLMatch UDx необходимо:

- установить репозиторий RPM:

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

- установить PostgreSQL:

```
dnf install postgresql16-server
```

- установить окружение для разработки:

```
dnf install postgresql16-devel
```

- установить пакет gcc:

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

- установить CMAKE. Необходима версия 3.5 или выше.
- открыть скрипт make.sh в текстовом редакторе. Он включает в себя пути к используемой на данный момент версии PostgreSQL. Измените следующие значения (при необходимости):

SDK_HOME задает путь к домашней директории PostgreSQL. По умолчанию это /usr/pgsql-16/include/server;

LIB_ROOT задает путь к библиотечной корневой директории PostgreSQL. По умолчанию это /usr/pgsql-16/lib.

Откройте директорию скрипта make.sh и запустите его:

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

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

```
./make.sh
```

3.10.10 Добавление функции VLMatch для БД Faces в PostgreSQL

Сервис Faces требует добавления дополнительной функции VLMatch к используемой базе данных. LUNA PLATFORM не может выполнять вычисления по сравнению биометрических шаблонов без этой функции.

Библиотека VLMatch компилируется для конкретной версии базы данных.

Не используйте библиотеку, созданную для другой версии базы данных. Например, библиотеку, созданную для PostgreSQL версии 12 нельзя использовать для PostgreSQL версии 16.

В данном разделе описывается создание функции для PostgreSQL. Библиотека VLMatch должна быть скомпилирована и перенесена в PostgreSQL. См. раздел [«Компиляция VLMatch для PostgreSQL»](#).

3.10.10.1 Добавление функции VLMatch в базу данных Faces

Функцию VLMatch необходимо применить в базу данных PostgreSQL.

Определите функцию в базе данных Faces:

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

Протестируйте функцию, отправив следующий запрос в базу данных сервиса:

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

База данных должна вернуть результат «0.4765625».

3.10.11 Добавление функции VLMatch для БД Events в PostgreSQL

Сервис Events требует добавления дополнительной функции VLMatch к используемой базе данных. LUNA PLATFORM не может выполнять вычисления по сравнению биометрических шаблонов без этой функции.

Библиотека VLMatch компилируется для конкретной версии базы данных.

Не используйте библиотеку, созданную для другой версии базы данных. Например, библиотеку, созданную для PostgreSQL версии 12 нельзя использовать для PostgreSQL версии 16.

В данном разделе описывается создание функции для PostgreSQL. Библиотека VLMatch должна быть скомпилирована и перенесена в PostgreSQL. См. раздел [«Компиляция VLMatch для PostgreSQL»](#).

3.10.11.1 Добавление функции VLMatch в базу данных Events

Функцию VLMatch необходимо применить в базе данных PostgreSQL.

Определите функцию в базе данных Events.

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

Протестируйте функцию.

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

База данных должна вернуть результат «0.4765625».

3.10.12 Установка PostGIS для БД Events

Сервис Events требует PostGIS для работы с координатами.

В данной инструкции описывается установка PostGIS для СУБД PostgreSQL 16. Версия PostGIS зависит от версии PostgreSQL.

Установите epel-release для доступа к расширенному репозиторию.

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

Установите PostGIS.

```
yum -y install postgis34_16
```

Активируйте PostGIS в базе данных.

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

См. дополнительную информацию о PostGIS на сайте <https://postgis.net/>.

3.11 VLMatch для Oracle

Примечание. В следующей инструкции описана установка для Oracle 21c.

Все файлы, требуемые для компиляции расширения, заданного пользователем (UDx), в VLMatch, можно найти в следующей директории:

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

Для компиляции функции VLMatch UDx необходимо:

- Установить требуемое окружение, см. [требования](#):

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

2. Поменяйте переменную SDK_HOME — oracle sdk root (по умолчанию \$ORACLE_HOME/bin, проверьте, что переменная окружения \$ORACLE_HOME задана) в makefile.

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

3. Откройте директорию и запустите файл «make.sh».

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

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

```
./make.sh
```

4. Определите библиотеку и функцию внутри базы данных (из консоли базы данных):

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

5. Протестируйте функцию посредством вызова (из консоли базы данных):

```
SELECT VLMatch(HEXTORAW('123456789012345678901234567890123456789012345678901234'),
```

```
HEXTORAW('0123456789012345678901234567890123456789012345678901234567890123', 32)  
FROM DUAL;
```

Результат должен быть равен «0.4765625».