VisionLabs
MACHINES CAN SEE

VISIONLABS ACCESS CONTROL SERVER

Инструкция по установке

ООО «ВижнЛабс»

123458, г. Москва, ул. Твардовского д. 8, стр. 1

+7 (499) 399 3361

info@visionlabs.ru

www.visionlabs.ru

Содержание

Глоссарий

Термин	Определение
Аутентификация	Совокупность мероприятий по проверке лица на принадлежность ему идентификатора (идентификаторов) посредством сопоставления его (их) со сведениями о лице, которыми располагает лицо, проводящее аутентификацию, и установлению правомерности владения лицом идентификатором (идентификаторами) посредством использования аутентифицирующего (аутентифицирующих) признака (признаков) в рамках процедуры аутентификации, в результате чего лицо считается установленным
Бакет	Логическая сущность, которая помогает организовать хранение объектов.
БД	База данных.
Биометрический образец, БО	Фотоизображение, приведенное в формат, который соответствует требованиям Системы.
Биометрический шаблон, БШ	Набор данных в закрытом двоичном формате, подготавливаемый системой на основе анализируемой характеристики. Представляет из себя составной вектор признаков фотоизображения лица человека.
Идентификация	Совокупность мероприятий по установлению сведений о лице и их проверке, и сопоставлению данных сведений с уникальным обозначением (уникальными обозначениями) сведений о лице, необходимым для определения такого лица. В контексте документа – поиск наиболее подходящего БШ путем сравнения полученного БШ с перечнем БШ в базе (сравнение «один ко многим»).
СУБД	Система управления базами данных.

Введение

Настоящий документ описывает процедуру установки пакета программного обеспечения «VisionLabs Access Control Server» (далее – Система).

В данном руководстве представлена следующая информация:

- порты по умолчанию для сервисов;
- взаимодействие сервисов Системы;
- системные требования;
- действия перед установкой;
- запуск сервисов;
- мониторинг сервисов;
- тестирование сервисов.

Установка дополнительных сервисов не обязательна для работы Системы.
Для работы Системы требуется файл лицензии, который поставляется компанией VisionLabs отдельно по запросу.
Выполнять установку компонентов следует в указанном в документе порядке.
Все команды, приведенные в документе, должны выполняться пользователем с правами администратора или пользователем root.

1. Порты по умолчанию для сервисов

Таблица 1. Порты по умолчанию для сервисов

Название сервиса	Порт
Сервис API	5000
Сервис Admin	5010
Сервис Image Store	5020
Сервис Faces	5030
Сервис Events	5040
Сервис Tasks	5050
Сервис Tasks Worker	5051
Сервис Configurator	5070
Сервис Sender	5080
Сервис Handlers	5090
Сервис Python Matcher	5100
Сервис Licenses	5120
Oracle DB	1521
PostgreSQL	5432
Redis DB	6379
InfluxDB	8086

2. Взаимодействие сервисов Системы

В комплект поставки входят основные и дополнительные сервисы.
На изображении ниже (Рисунок 1) для каждого сервиса указаны отдельные БД. Это сделано в целях иллюстрации.
Фактически, можно использовать одну и ту же БД для сервисов, использующих одну и ту же БД, например PostgreSQL. Каждый сервис будет иметь свою собственную таблицу в этой БД. Поэтому нет необходимости устанавливать несколько экземпляров БД PostgreSQL для каждого сервиса.

Сервис API предоставляет интерфейс RESTful для процедур обнаружения лиц, извлечения и сопоставления БШ.

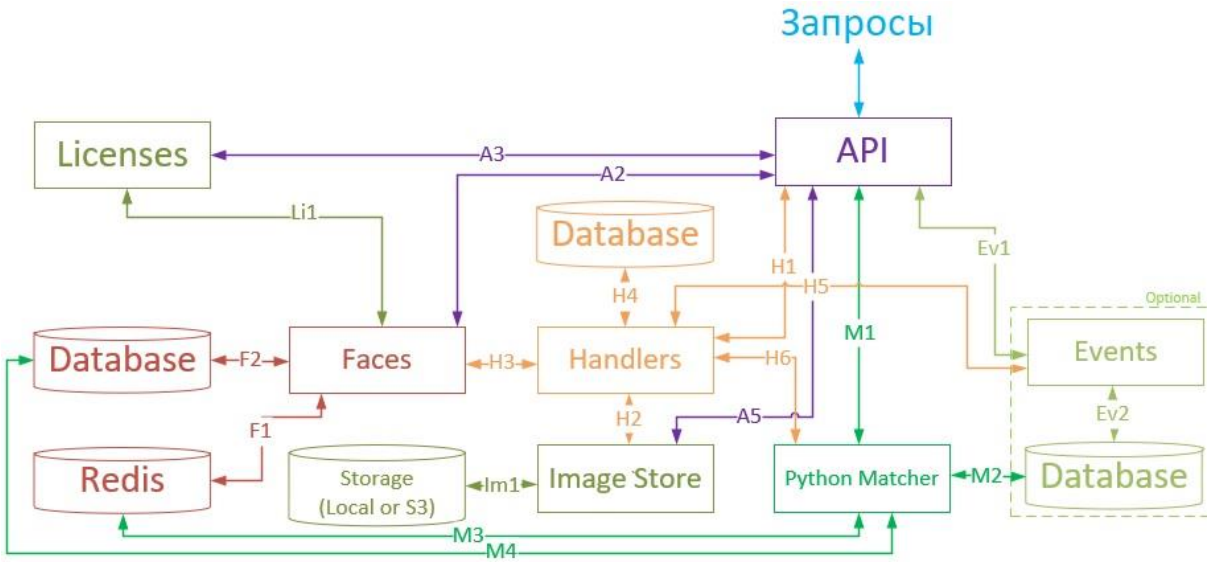

Рисунок 1. Взаимодействие основных сервисов

Запросы отправляются в Систему по протоколу HTTP. Распространенный случай – это, когда внешний сервис отправляет запросы в Систему, получает результаты и обрабатывает их в соответствии с потребностями бизнеса.
API получает запросы, обрабатывает их и отправляет в другие сервисы:

- запросы на обнаружение лиц, оценку свойств лиц и создание образцов отправляются в Handler (**H1**);
- запросы на создание временных атрибутов отправляются в сервис Handler (**H1**);
- запросы на сопоставление дескрипторов могут быть отправлены в сервис Python Matcher (**M1**).

API отправляет запросы в Faces для создания новых лиц, а также для создания списков и управления ими (**A2**).
Сервис API получает информацию о текущих условиях лицензии от сервиса Licenses (**A3**) с использованием промежуточного программного обеспечения.
Сервис API отправляет образцы, полученные от внешних сервисов, в сервис Image Store (**A5**).

Handlers обрабатывает запросы на обнаружение лиц и создание атрибутов. Он оценивает атрибуты лица и свойства лица.
Экземпляр **Handlers** обрабатывает запрос на обнаружение от API (**H1**) и отправляет полученные образцы в сервис Image Store (**H2**).
Экземпляр **Handlers** обрабатывает запрос на создание атрибутов из API (**H1**), запрашивает существующие образцы из сервиса Image Store (**H2**) и отправляет созданные временные атрибуты в сервис Faces, который хранит их в БД Redis (**F1**).
Сервис Handlers создает новые обработчики и сохраняет их в БД Handlers (**H4**).
Запрос на создание нового события принимается API, отправляется в сервис Handlers (**H1**) и обрабатывается в соответствии с Handler, идентификатор которого указан в запросе.
Запрос на создание события может включать следующие политики:

- detect_policy - запрос обрабатывается сервисом Handler аналогично запросу на обнаружение лиц;
- extract_policy - запрос обрабатывается сервисом Handler аналогично запросу на извлечение атрибутов;
- match_policy - запрос отправляется в сервис Python Matcher (**H6**);
- create_face_policy - запрос отправляется в сервис Faces (**H3**);
- link_to_lists_policy - запрос отправляется в сервис Faces (**H3**).

Image Store получает БО из сервиса Handlers, сохраняет их на SSD или в БД S3 и предоставляет к ним доступ (**Im1**).
Licenses. Сервис Licenses получает информацию о количестве созданных лиц с атрибутами из БД Faces (**Li1**).
Faces отвечает за взаимодействие с БД Faces, в которой хранятся: БО, БШ и списки (**F2**). Он обеспечивает доступ к сохраненным данным для сервисов API, Matcher и Tasks.
Faces также хранит созданные временные атрибуты в БД Redis (**F1**).
Python Matcher может получать запросы на сопоставление непосредственно из сервиса API (**M1**). Python Matcher отправляет запросы на сопоставление в БД Faces (**M4**) или БД Events (если она включена) (**M2**). БД сопоставляют дескрипторы и отправляют результаты обратно в сервис API.
Python Matcher также может получать временные атрибуты, необходимые для сопоставления, из БД Redis (**M3**).
Сервис **Events** хранит и предоставляет информацию о событиях. После создания события, сервис Events получает созданное событие от сервиса Handlers (**H5**) и сохраняет его в БД Events (**Ev2**).

> Использование сервиса Events может быть включено/отключено в конфигурационном файле сервиса API.

3. Системные требования

Для установки полного пакета Системы должны выполняться следующие системные требования:

- ОС CentOS Linux release 7.8.2003;
- CPU Intel, не менее 4 физических ядер с тактовой частотой не менее 2,0 ГГц. Требуется поддержка набора инструкций AVX2 для CPU;
- оперативная память DDR3 (рекомендуется DDR4), не менее 8 ГБ;
- свободное место на диске не менее 80 ГБ.

> Рекомендуется использовать SSD для БД и хранилищ Image Store.

- доступ в интернет (для контейнеров и дополнительных загрузок ПО).

> Приведенная выше конфигурация обеспечит минимальную мощность для работы ПО, но для использования системы в продуктивном контуре этого недостаточно. Требования для использования системы в продуктивном контуре рассчитываются в зависимости от предполагаемой нагрузки.

GPU

Для ускорения GPU требуется NVIDIA GPU. Поддерживаются следующие архитектуры:

- Pascal или более новые.

Требуется минимум 6 ГБ оперативной или выделенной видеопамяти. Рекомендуется 8 ГБ VRAM или более.

CUDA версии 11.2 уже установлена в Docker контейнере в сервисе Handlers. Рекомендуемые драйверы NVIDIA: r450, r455.

4. Действия перед установкой

4.1. Распаковка дистрибутива

Все команды, приведенные в документе, должны выполняться пользователем с правами администратора или пользователем root.
Дистрибутив представляет собой архив вида `luna_v.X.Y.ZZ`, где `X.Y.ZZ` – численный идентификатор, обозначающий текущую версию Системы.
Архив содержит все компоненты, необходимые для установки и эксплуатации Системы. Архив не включает зависимости, которые входят в стандартную поставку репозитория CentOS. Зависимости доступны в Интернете.
Перед процессом установки необходимо поместить файлы дистрибутива и лицензии в директорию `/root/`. В данной директории не должно быть других файлов дистрибутива и лицензии кроме целевых, используемых для установки конечного продукта.

Необходимо создать директорию для распаковки дистрибутива:

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

Следует переместить дистрибутив в созданную директорию:

```
mv /root/luna*.zip /var/lib/luna
```

Нужно установить архиватор unzip, если он не установлен:

```
yum install -y unzip
```

Затем перейти в папку с дистрибутивом:

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

Далее разархивировать файлы:

```
unzip luna*.zip
```

4.2. Создание символьной ссылки

Необходимо создать символьную ссылку, указав вместо `X.Y.ZZ` текущую версию дистрибутива. Ссылка указывает, что именно текущая версия дистрибутива используется для запуска:

```
ln -s luna_v.X.Y.ZZ current
```

4.3. Изменение группы и владельца директорий

Сервисы Системы запускаются внутри контейнеров пользователем `luna`. Поэтому для этого пользователя необходимо установить права доступа на использование смонтированных томов.
Необходимо перейти в директорию Системы `example-docker`:

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

Следует установить права доступа для пользователя с `UID 1001` и группой `0` для использования смонтированных директорий:

```
mkdir luna_configurator/used_dumps
```

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

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

4.4. Создание директории логов

Необходимо пропустить этот раздел, если не требуется сохранять логи на сервере.
Следует создать директорию логов, если собираетесь записывать логи на сервер.
Все логи сервисов будут скопированы в эту директорию:

```
mkdir /tmp/logs
```

```
chown -R 1001:0 /tmp/logs
```

Необходимо вручную создать все необходимые директории для логов и установить права доступа:

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

4.5. SELinux и Firewall

Необходимо настроить SELinux и Firewall, чтобы они не блокировали работу сервисов Системы.

> Настройка SELinux и Firewall не описана в данном документе.

Если SELinux and Firewall не настроены, установка не может быть выполнена.

4.6. Активация лицензионного ключа

Сервис HASP используется для лицензирования Системы. Без лицензии не будет возможности запускать и использовать сервисы Системы.
Для Системы используется следующий ключ:

- общий ключ HASP, который позволяет использовать Систему. Он использует библиотеку поставщика haspvlib_111186.so.

Библиотеки поставщика находятся в каталоге `/var/hasplm/`.

> Лицензионные ключи предоставляются VisionLabs отдельно по запросу.

Для использования Системы в контейнерах Docker требуется сетевая лицензия.

> Лицензионные ключи предоставляются VisionLabs отдельно по запросу. Лицензионный ключ создается с помощью отпечатка (fingerprint), который создается на основе информации об аппаратных характеристиках сервера. Таким образом, полученный лицензионный ключ будет работать только на том же сервере, на котором был получен отпечаток (fingerprint). Существует вероятность того, что при выполнении любых изменений на сервере лицензии потребуется новый лицензионный ключ.

Необходимо выполнить следующие действия:

- установить утилиту HASP на используемый сервер. Утилита HASP обычно устанавливается на отдельном сервере;
- запустить утилиту HASP;
- создать отпечаток (fingerprint) используемого сервера и отправить его в VisionLabs;
- активировать ключ, полученный от VisionLabs;
- указать адрес используемого сервера HASP в специальном файле.

На вкладке «Sentinel Keys» пользовательского интерфейса (`<server_host_address>:1947`) отображаются активированные ключи.

4.6.1. Лицензия Системы

4.6.1.1. Установка утилиты HASP для Системы

Необходимо перейти в каталог HASP:

```
cd /var/lib/luna/current/extras/hasp/
```

Следует установить утилиту HASP на используемый сервер:

```
yum -y install /var/lib/luna/current/extras/hasp/haspd-*.rpm
```

Далее необходимо запустить утилиту HASP:

```
systemctl daemon-reload
```

```
systemctl start aksusbd
```

```
systemctl enable aksusbd
```

```
systemctl status aksusbd
```

4.6.1.2. Настройка утилиты HASP

Можно настроить утилиту HASP с помощью файла `/etc/hasplm/hasplm.ini`.

Файл настроен по умолчанию. Его параметры не описаны в данном документе.

4.6.1.3. Создание отпечатка (fingerprint) для Системы

Необходимо перейти в каталог HASP:

```
cd /var/lib/luna/current/extras/hasp/
```

Следует добавить права доступа в скрипт:

```
chmod +x LicenseAssist
```

Затем запустить скрипт:

```
./LicenseAssist fingerprint fingerprint_111186.c2v
```

Отпечаток (fingerprint) сохраняется в файле `fingerprint_111186.c2v`.
Необходимо отправить файл в VisionLabs. Лицензионный ключ будет создан с использованием данного отпечатка (fingerprint).

4.6.1.4. Добавление файла лицензии вручную через пользовательский интерфейс

Для того, чтобы добавить файл лицензии вручную через пользовательский интерфейс необходимо выполнить следующие действия (Рисунок 2):

- перейти по адресу `<host_address>:1947` (если доступ запрещен, необходимо проверить настройки Firewall / SELinux;
- выбрать вкладку «Update/Attach» на панели слева;
- нажать кнопку «Browse» и выбрать файл(ы) лицензии в открывшемся окне;
- нажать кнопку «Apply file».

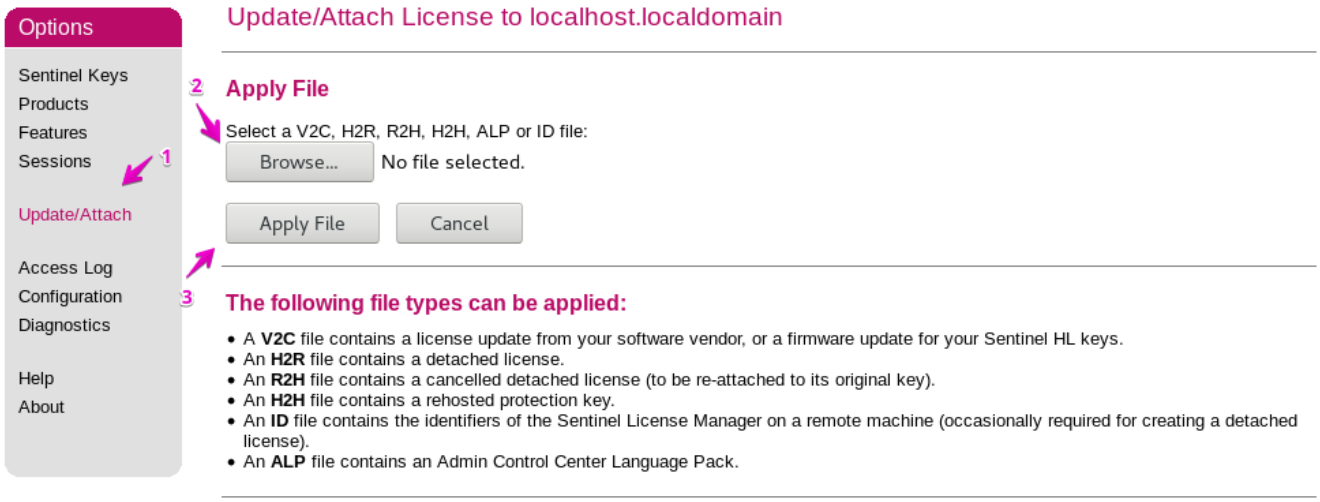

Рисунок 2. Ручное добавление файла лицензии

4.6.1.5. Адрес сервера лицензии для Системы

Необходимо указать свой IP-адрес сервиса лицензирования в конфигурационном файле в каталоге `/var/lib/luna/current/example-docker/hasp_redirect/`. Следует изменить адрес для сервера HASP в следующих документах:

```
vi /var/lib/luna/current/example-docker/hasp_redirect/hasp_111186.ini
```

Необходимо изменить адрес сервера в файле `hasp_111186.ini`.

```
serveraddr = <HASP_server_address>
```

Файл `hasp_111186.ini` используется сервисом Licenses при запуске контейнера.
`HASP_server_address` - IP-адрес сервера с полученным ключом HASP. Необходимо использовать IP-адрес, а не имя сервера.

5. Запуск сервисов

В данном разделе приведены примеры для:

- создания БД;
- создания бакетов;
- запуска контейнеров.

> Команды даются в том порядке, в каком они должны быть выполнены.

> Рекомендуется запускать контейнеры один за другим и ждать, пока статус контейнера станет «up» (следует использовать команду `docker ps`).

5.1. Общие параметры запуска контейнеров

При запуске Docker контейнера для сервиса Системы следует указать дополнительные параметры, необходимые для запуска сервиса.

> Параметры, характерные для конкретного контейнера, описаны в разделе о запуске этого контейнера.

Все параметры, приведенные в примере запуска сервиса, необходимы для правильного запуска и использования сервиса.

5.1.1. Доступ к реестру

При запуске контейнеров необходимо указать ссылку на изображение, необходимое для запуска контейнера.
Это изображение будет загружено из реестра VisionLabs. Перед этим следует войти в реестр.

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

Необходимо ввести логин `<username>` и пароль `<password>`, которые выданы VisionLabs.

5.1.2. Общие параметры запуска

Общие параметры для запуска контейнеров описаны в данном разделе.

```
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>`
```

`docker run` - команда для запуска выбранного образа в качестве нового контейнера.
`dockerhub.visionlabs.ru/luna/<service-name>:<version>` - параметр указывает образ, необходимый для запуска контейнера.

> Ссылки для загрузки необходимых изображений контейнеров приведены в описании соответствующего запуска контейнера.

`--network=host` - параметр указывает, что сеть не моделируется и используется серверная сеть. Если необходимо изменить порт для сторонних контейнеров, следует изменить эту строку на `p 5440:5432`. Где первый порт `5440` является локальным портом, а `5432` - портом, используемым внутри контейнера. Пример приведен для PostgreSQL.
`--env=` - параметр указывает переменные среды, необходимые для запуска контейнера. Указаны следующие общие значения:

- `CONFIGURATOR_HOST=127.0.0.1` - хост, на котором запущен сервис Configurator. Локальный хост устанавливается в том случае, когда контейнер запускается на одном сервере с сервисом Configurator;
- `CONFIGURATOR_PORT=5070` – порт сервиса Configurator. По умолчанию используется порт 5070;
- `PORT=<Port_of_the_service>` - порт сервиса для «прослушивания» сервера;
- `WORKER_COUNT` указывает количество рабочих процессов для сервиса;
- `RELOAD_CONFIG` включает автоматическую перезагрузку конфигураций для сервиса, если установлено значение `1`;
- `RELOAD_CONFIG_INTERVAL` указывает период проверки конфигурации (по умолчанию 10 секунд).

`-v` - параметр тома позволяет смонтировать содержимое папки сервера в том в контейнере. Таким образом, их содержимое будет синхронизироваться. Монтируются следующие общие данные:

- `/etc/localtime:/etc/localtime:ro` устанавливает текущий часовой пояс, используемый системой в контейнере;
- `/tmp/logs/<service>:/srv/logs/` - позволяет копировать папку с логами сервиса на используемый сервер в каталог `/tmp/logs/<service>`. Можно изменить каталог, в котором будут сохраняться логи, в соответствии с потребностями.

`--name=<service_container_name>` - параметр указывает имя запущенного контейнера. Имя должно быть уникальным. Если существует контейнер с таким же именем, произойдет ошибка.
`--restart=always` - параметр указывает политику перезапуска. Демон всегда будет перезапускать контейнер, независимо от статуса выхода.
`--detach=true` запускает контейнер в фоновом режиме.

5.1.3. Параметры создания БД

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /tmp/logs/<service>:/srv/logs/ \
--rm \
--network=host \
dockerhub.visionlabs.ru/luna/<service-name>:<version> \
python3.9 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

Следующие параметры используются при запуске контейнеров для операций переноса и создания БД.
`--rm` - параметр указывает, будет ли контейнер удален после того, как все указанные скрипты завершат обработку.
`python3.9 ./base_scripts/db_create.py` - параметр указывает версию Python и скрипт `db_create.py`, запущенный в контейнере. Скрипт используется для создания структуры БД.
`--luna-config http://localhost:5070/1` - параметр указывает, где запущенный скрипт должен получать конфигурации. По умолчанию сервис запрашивает конфигурации у сервиса Configurator.

5.2. Конфигурация мониторинга

5.2.1. InfluxDB

InfluxDB необходима для целей мониторинга Системы.

> Если уже установлена InfluxDB, следует пропустить этот шаг.

5.2.1.1. Запуск контейнера InfluxDB

Необходимо использовать следующую команду для запуска InfluxDB.
Необходимо использовать команду `docker run` со следующими параметрами:

```
docker run \
-e INFLUXDB_DB=luna_monitoring \
-e INFLUXDB_USER=luna \
-e INFLUXDB_USER_PASSWORD=luna \
-v /etc/localtime:/etc/localtime:ro \
-v /var/lib/luna/current/example-docker/influx:/var/lib/influxdb \
--name=influxdb \
--restart=always \
--detach=true \
--network=host \
dockerhub.visionlabs.ru/luna/influxdb:1.7.10-alpine
```

5.2.2. Настройка параметров мониторинга

Необходимо включить мониторинг сервисов с помощью Influx DB.
Необходимо настроить доступ к Influx DB в конфигурациях сервисов Python.

> Мониторинг сервисов включен по умолчанию.

Таблица 2. Имена сервисов в Configurator

Сервис	Имя сервиса в Configurator
API	luna-api
Faces	luna-faces
Image Store	luna-image-store
Tasks	luna-tasks
Events	luna-events
Sender	luna-sender
Admin	luna-admin
Licenses	luna-licenses
Python Matcher	luna-python-matcher
Handlers	luna-handlers

Сам сервис Configurator настраивается только в файле конфигурации: `/var/lib/luna/current/example-docker/luna_configurator/configs/luna_configurator_postgres.conf`
Можно указать следующие параметры:

`SEND_DATA_FOR_MONITORING` включает мониторинг для сервиса.

`MONITORING_USER_NAME` - имя пользователя InfluxDB.

`MONITORING_PASSWORD` - пароль InfluxDB.

`MONITORING_DB_NAME` - имя БД InfluxDB.

`MONITORING_HOST` - IP-адрес InfluxDB.

`MONITORING_PORT` – порт InfluxDB.

`MONITORING_USE_SSL` - позволяет использовать протокол HTTPS для подключения к InfluxDB (0 – не использовать, 1 – использовать).

`MONITORING_FLUSHING_PERIOD` – частота отправки данных мониторинга в InfluxDB.

5.3. Запуск сторонних сервисов

В этом разделе описывается запуск БД и очередей сообщений в Docker контейнерах. Они должны быть запущены до сервисов Системы.

5.3.1. PostgreSQL

Если уже установлен PostgreSQL, следует пропустить этот шаг.
Необходимо использовать следующую команду для запуска PostgreSQL.

```
docker run \
--env=POSTGRES_USER=luna \
--env=POSTGRES_PASSWORD=luna \
```

```
    --shm-size=1g \
    -v /var/lib/luna/current/example-docker/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:12
```

`-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 БД сервисов.

`-v /var/lib/luna/current/example-docker/postgresql/data/:/var/lib/postgresql/data/` - команда тома позволяет подключить папку `data` к контейнеру PostgreSQL. Папка на сервере и папка в контейнере будут синхронизированы. Данные PostgreSQL из контейнера будут сохранены в этом каталоге.

`--network=host` - если необходимо изменить порт для Postgresql, следует изменить эту строку на `-p 5440:5432`. Где первый порт `5440` является локальным портом, а `5432` - портом, используемым внутри контейнера.

> Следует создать все БД для сервисов Системы вручную, если планируется использование уже установленной PostgreSQL.

5.3.2. Redis

Если уже установлен Redis, следует пропустить этот шаг.
Необходимо использовать 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:5.0.6-alpine3.10
```

5.4. Сервис Configurator

5.4.1. Использование дополнительных сервисов

Перечисленные сервисы не являются обязательными для Системы. Можно их отключить, если их функциональность не требуется.
Необходимо использовать сервис Configurator, чтобы отключить ненужные сервисы для всех сервисов сразу.

Сервис	Зависимые сервисы
Events	API, Admin, Handlers, Tasks.
Tasks	API, Admin, Handlers
Sender	API, Admin, Handlers
Python Matcher Proxy	API, Admin, Handlers, Tasks

Можно использовать дамп-файл, предоставленный в дистрибутиве, для включения и отключения сервисов перед запуском Configurator.

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

> Требуется перезапустить зависимый сервис после изменения его параметров, если сервис уже был запущен.

5.4.2. Создание таблиц БД Configurator

Необходимо использовать команду `docker run` с данными параметрами для создания БД 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.1.2.3 \
-c "python3.9 ./base_scripts/db_create.py; cd /srv/luna_configurator/configs
/configs/; python3.9 -m configs.migrate --profile platform head; cd /srv;
python3.9 ./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 указать путь к дамп-файлу с конфигурациями Системы.

`./base_scripts/db_create.py;` - создает структуру БД.

`python3.9 -m configs.migrate --profile platform head;` - выполняет перенос настроек в БД Configurator и устанавливает ревизию для переноса. Ревизия потребуется во время обновления до новой сборки Системы.

`--dump-file /srv/luna_configurator/used_dumps/platform_settings.json` - обновляет настройки в БД Configurator значениями из предоставленного файла.

5.4.3. Запуск контейнера Configurator

Необходимо использовать команду `docker run` с указанными 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.1.2.3
```

5.4.4. Изменение настроек

Существует три основных способа изменения настроек, хранящихся в БД Configurator:

- использование GUI в Configurator;
- использование запросов API в Configurator;
- использование дамп-файла.

Настройки могут быть изменены после запуска сервиса Configurator.
Настройки самого Configurator хранятся в файле `luna_configurator_postgres.conf`, который загружается в контейнер Configurator во время его запуска.

Интерфейс Configurator

Можно войти в Configurator GUI и изменить настройки. По умолчанию на локальном хосте используется следующий адрес: `<Configurator_server_address>:5070.`

Configurator API

Можно использовать Configurator API для обновления настроек.

Дамп-файл

Можно получить дамп-файл со всеми настройками сервисов Системы. Необходимо использовать 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
```

Требуется указать правильный адрес и порт сервиса Configurator.
Следует удалить раздел `"limitations"` из файла. Не получится применить обновленный дамп-файл, если раздел останется:

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

Необходимо изменить параметры в разделе `"settings"`:

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

Следует скопировать дамп-файл в контейнер Configurator:

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

Далее применить файл:

```
docker exec luna-configurator python3.9 ./base_scripts/db_create.py --dumpfile
/srv/settings_dump.json
```

5.4.5. Сохранение логов в директорию на сервере

Если необходимо сохранить логи сервиса в одной директории на используемом сервере, следует обновить конфигурации сервисов.

> По умолчанию логи не сохраняются в директории сервера.

Система по умолчанию не сохраняет логи на используемом сервере. Чтобы включить сохранение логов на сервере, необходимо выполнить следующие действия:

- создать каталог для логов;
- включить сохранение логов и изменить директорию логов в сервисе Configurator.

Когда логирование в файл включено и настроено, логи сервиса записываются в директорию `/srv/logs` в контейнере сервиса и в указанную директорию сервиса на сервере (например, `/tmp/logs`).

5.4.5.1. Изменение директории логов вручную перед запуском Configurator

Настройки сервиса Configurator хранятся в файле `/var/lib/luna/current/example-docker/luna_configurator/configs/luna_configurator_postgres.conf`.
Следует изменить его параметры логирования в этом файле перед запуском Configurator или перезапустить Configurator после их изменения.
Необходимо изменить этот параметр логирования `FOLDER_WITH_LOGS = ./` на `FOLDER_WITH_LOGS = /srv/logs`.
Следует установить для параметра `"log_to_file"` значение `True`, чтобы включить логирование в файл.

5.4.5.2. Изменение директории логов вручную после запуска Configurator

Необходимо задать значение параметра `"folder_with_logs"` в `"/srv/logs"` для всех сервисов, в которые следует записывать логи. Команды `docker run` уже настроены для сохранения логов на используемом сервере в этой директории. После запуска контейнера сервиса, сервис будет записывать логи в директорию `"/srv/logs"`.

Следует установить для параметра `"log_to_file"` значение `True`, чтобы включить логирование в файлы.

5.4.5.3. Изменение директории логов с помощью дамп-файла

Можно использовать дамп-файл, предоставленный в дистрибутиве, для обновления настроек логирования `/var/lib/luna/current/extras/conf/logging.json`.

```
docker run \
-v /etc/localtime:/etc/localtime:ro \
-v /var/lib/luna/current/extras/conf/logging.json:/srv/luna_configurator/
used_limitations/logging.json \
--network=host \
-v /tmp/logs/configurator:/srv/logs \
--rm \
--entrypoint=python3.9 \
dockerhub.visionlabs.ru/luna/luna-configurator:v.1.2.3 \
./base_scripts/db_create.py --dump-file /srv/luna_configurator/
used_limitations/logging.json
```

5.5. Сервис Image Store

5.5.1. Запуск контейнера Image Store

Необходимо использовать 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/current/example-docker/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.3.3.3
```

`-v /var/lib/luna/current/example-docker/image_store/:/srv/local_storage/` - данные из указанной папки добавляются в Docker контейнер при его запуске. Все данные из указанной папки Docker контейнера сохраняются в этот каталог.

> Если уже есть каталог с бакетами Системы, следует указать его вместо `/var/lib/luna/current/example-docker/image_store/`.

5.5.2. Создание бакетов

Для хранения данных в Image Store требуются Бакеты. Сервис Image Store должен быть запущен до выполнения команд.

При обновлении с предыдущей версии рекомендуется еще раз запустить команды создания бакета для того, чтобы убедиться, что все необходимые бакеты были созданы.
Если при запуске перечисленных выше команд появляется ошибка `“error_code”:13006,“desc”:“Unique constraint error”,“detail”:“Bucket with name ‘task-result’ already exist”}`, то бакет уже создан.
Существует два способа создания бакетов в Системе:
1. Можно запустить перечисленные выше скрипты для создания бакетов.
Необходимо запустить этот скрипт для создания общих 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.5.5.1 \
python3.9 ./base_scripts/lis_bucket_create.py -ii --luna-config http://
localhost:5070/1
```

Если планируется использовать сервис Tasks, необходимо использовать 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.3.3.1 \
python3.9 ./base_scripts/lis_bucket_create.py -ii --luna-config http://
localhost:5070/1
```

2. Можно использовать прямые запросы для создания необходимых бакетов.

> Утилита `curl` требуется для следующих запросов.

Бакет `visionlabs-samples` используется для хранения образцов лица. Бакет требуется для утилизации Системы.

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

Бакет `visionlabs-bodies-samples` используется для хранения образцов тел. Бакет требуется для утилизации Системы.

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

Бакет `visionlabs-image-origin` используется для хранения исходных изображений. Бакет требуется для утилизации Системы.

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

Бакет `task-result` - для сервиса Tasks. Не следует использовать его, если не планируется использование сервиса Tasks.

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

5.6. Сервис Faces

5.6.1. Создание таблиц БД Faces

Необходимо использовать следующую команду для создания БД 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.4.2.3 \
python3.9 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

5.6.2. Запуск контейнера Faces

Необходимо использовать следующую команду для запуска сервиса:

```
docker run \
--env=CONFIGURATOR_HOST=127.0.0.1 \
--env=CONFIGURATOR_PORT=5070 \
--env=PORT=5030 \
--env=WORKER_COUNT=1 \
--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.4.2.3
```

5.7. Запуск сервиса Licenses

5.7.1. Запуск контейнера сервиса Licenses

Необходимо убедиться, что указан адрес сервера лицензий в файле `hasp_111186.ini`.
Следует добавить право доступа для пользователя `luna` в каталог `hasp_redirect`.

```
chown -R 1001:0 /var/lib/luna/current/example-docker/hasp_redirect/
```

Необходимо использовать 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 \
    -v /var/lib/luna/current/example-docker/hasp_redirect/hasp_111186.ini:/home/
    luna/.hasplm/hasp_111186.ini \
    --name=luna-licenses \
    --restart=always \
    --detach=true \
    --network=host \
    dockerhub.visionlabs.ru/luna/luna-licenses:v.0.1.36
```

5.8. Сервис Events

5.8.1. Создание таблиц БД Events

> Если не планируется использование сервиса Events, не следует запускать этот контейнер. Следует отключить использование сервиса в Configurator (См. п. 5.4.1).

Необходимо использовать следующую команду для создания БД 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.2.2.4 \
    python3.9 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

5.8.2. Запуск контейнера сервиса Events

> Если не планируется использование сервиса Events, не следует запускать этот контейнер. Следует отключить использование сервиса в Configurator (См. п. 5.4.1).

Необходимо использовать 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.2.2.4
```

5.9. Сервисы Matcher

5.9.1. Использование Python Matcher

Сервис Python Matcher с сопоставлением БШ по БД Faces включен по умолчанию после запуска. Сервис Python Matcher с сопоставлением БШ по Events также включен по умолчанию. Можно отключить его, указав `USE_LUNA_EVENTS = 0` в разделе `ADDITIONAL_SERVICES_USAGE`. Таким образом, сервис Events не будет использоваться для Системы.
Python Matcher, который сопоставляет БШ, используя библиотеку сопоставления, активен, когда в настройках `DESCRIPTORS_CACHE` параметр `CACHE_ENABLED` имеет значение `true`.

5.10. Сервис Python Matcher

> Для сервиса Python Matcher и сервиса Python Matcher Proxy загружается одно изображение.

5.10.1. Запуск контейнера сервиса Python Matcher

Необходимо использовать 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.0.4.7
```

5.11. Сервис Handlers

5.11.1. Создание таблиц БД Handlers

Необходимо использовать следующую команду для создания БД 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.1.4.3 \
python3.9 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

5.11.2. Запуск контейнера Handlers

Можно запустить сервис Handlers, используя CPU (установленный по умолчанию) или GPU. Необходимо запустить сервис Handlers, используя одну из следующих команд в соответствии с используемым процессором.

5.11.2.1. Запуск Handlers с помощью CPU

Необходимо использовать следующую команду для запуска сервиса Handlers с помощью CPU:

```
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.1.4.3
```

5.11.2.2. Запуск Handlers с помощью GPU

Сервис Handlers по умолчанию не использует GPU. Следует включить использование GPU для сервиса Handlers в сервисе Configurator.
Следует рассмотреть параметр `device_class` следующих настроек:

- LUNA_HANDLERS_WARP_ESTIMATOR_RUNTIME_SETTINGS;
- LUNA_HANDLERS_HUMAN_EXTRACTOR_RUNTIME_SETTINGS;
- LUNA_HANDLERS_HUMAN_DETECTOR_RUNTIME_SETTINGS;
- LUNA_HANDLERS_EXTRACTOR_RUNTIME_SETTINGS;
- LUNA_HANDLERS_DETECTOR_RUNTIME_SETTINGS.

Необходимо использовать следующую команду для запуска Сервис Handlers с помощью GPU:

```
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 \
--gpus device=0 \
-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.1.4.3
```

`--gpus device=0` - параметр указывает используемое устройство GPU и позволяет использовать GPU.
Для каждого экземпляра Handlers можно использовать один GPU. Использование нескольких GPU на один экземпляр недоступно.

5.12. Сервис Tasks

5.12.1. Создание таблиц БД Tasks

Необходимо использовать следующую команду для создания 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.3.3.1 \
    python3.9 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

5.12.2. Запуск контейнеров Tasks and Tasks Worker

Образ Luna Task Docker включает в себя сервис Tasks и Tasks Worker, которые должны быть запущены.

> Если не планируется использование сервиса Tasks, не следует запускать контейнер Tasks и контейнер Tasks Worker. Следует отключить использование сервиса в Configurator (См. п. 5.4.1).

Бакет `task-result` должен быть создан для сервиса Tasks до запуска сервиса. Создание бакетов описано п. 5.5.2.

5.12.2.1. Запуск сервиса Tasks Worker

Необходимо использовать 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.3.3.1
```

5.12.2.2. Запуск сервиса Tasks

Необходимо использовать 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.3.3.1
```

5.13. Сервис Sender

5.13.1. Запуск контейнера сервиса Sender

> Если не планируется использование сервиса Sender, не следует запускать этот контейнер. Следует отключить использование сервиса в Configurator (См. п. 5.4.1).

Необходимо использовать 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.2.1.8
```

5.14. Сервис API

5.14.1. Запуск контейнера сервиса API

Необходимо использовать 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.5.5.1
```

5.15. Сервис Admin

5.15.1. Создание таблиц БД Admin

Необходимо использовать следующую команду для создания БД Admin:

```
    docker run \
    -v /etc/localtime:/etc/localtime:ro \
    -v /tmp/logs/admin:/srv/logs \
    --rm \
    --network=host \
    dockerhub.visionlabs.ru/luna/luna-admin:v.4.2.7 \
    python3.7 ./base_scripts/db_create.py --luna-config http://localhost:5070/1
```

5.15.2. Запуск контейнера сервиса Admin

Необходимо использовать 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.4.2.7
```

6. Мониторинг сервисов

Запуск контейнера InfluxDB и настройка параметров мониторинга выполняются в соответствии с подразделом 5.2.
Компоненты Системы отправляют в InfluxDB метрики по каждому запросу. Например, сервис Faces для запросов отправляет следующие метрики:

- service_name – наименование сервиса;
- account_id – идентификатор аккаунта, с которого выполнен запрос;
- route – объединение метода запроса и ресурса запроса (POST: / faces);
- status_code – HTTP статус код ответа;
- request_id – идентификатор запроса;
- execution_time – время выполнения запроса.

В первую очередь это необходимо для отслеживания ошибок и времени выполнения запросов.
Для использования этой функции необходимо включить мониторинг при первичной настройке. Для этого необходимо отредактировать параметры в файле конфигурации `luna_configurator_postgres` в разделе `INFLUX_MONITORING`.

7. Тестирование сервисов

7.1. Общая информация о тестировании

Можно запустить автоматические тестирования для сервисов Системы.

> Необходимо запускать тестирование только в пустой БД. Ошибки будут возникать, если БД уже содержит данные.

Тестирование сервиса может быть пропущено во время установки, но рекомендуется выполнить тестирование. Тестирование позволяют убедиться, что сервис, прошедший тестирование, и все связанные сервисы работают должным образом.
Можно протестировать большинство сервисов Python после их запуска. Сервисы API и Admin должны быть протестированы только после запуска всех других сервисов.
Результат тестирования выводится на консоль. Пример тестирования приведен ниже:

```
[root@localhost]# docker exec luna-faces python3.9 -m unittest tests.
   unittests_main

.....................................................................
.....................................................................
.....................................................................
.....................................................................s
.....................................................................
.............................s...........................

----------------------------------------------------------------------

Ran 401 tests in 249.803s



OK (skipped=2)
```

> Фактическое количество тестов может отличаться, так как наборы тестов постоянно обновляются и добавляются новые тесты.

Символ «.» обозначает успешно завершенные тесты.
Символ «s» обозначает пропущенные тесты. Тесты по какой-то причине были отключены.
Символ «F» обозначает неудачные тесты. Эти тесты были выполнены, но результат теста неверен. Описание проблемы выводится на консоль после завершения всех тестов.

```
FAIL: test_emit_events_source_and_tags (tests.unittests_emit_events.
   TestEmitEvents) (param='tags')

----------------------------------------------------------------------

Traceback (most recent call last):

 File "/var/lib/luna/luna_11470331eaf91f4c30864ce45d4af6a0ce70ba97/luna-api/
    tests/unittests_emit_events.py", line 150, in
```

```
        test_emit_events_source_and_tags

      self.assertEqual(201, reply.statusCode, reply.text)

   AssertionError: 201 != 500 : {"error_code":2104,"desc":"Queue driver call
      failed","detail":"RabbitMQ MessageReturnedException: Message returned.
      Reply code: 312 NO_ROUTE correlation id: 1,13f23ce3-88ae-4723-b407-7
      ff372554986"}
```

Символ «E» обозначает ошибки во время выполнения. Они показывают, что тест не может быть завершен по некоторым причинам.
Описания ошибок выводятся на консоль после завершения всех тестов.

```
   ======================================================================

   ERROR: test_bad_event_time (tests.unittests_handler_events.TestHandlerEvents
      )

   ----------------------------------------------------------------------

   Traceback (most recent call last):

    File "/var/lib/luna/luna_11470331eaf91f4c30864ce45d4af6a0ce70ba97/luna-api/
       tests/unittests_handler_events.py", line 100, in test_bad_event_time

     self.attributes.append(reply.json['events'][0]['attributes']['attribute_id
        '])

   KeyError: 'events'
```

Ошибки могут возникать по разным причинам:

- неверные настройки сервиса;
- один или несколько необходимых сервисов или БД не были запущены;
- доступ к одному или нескольким серверам с необходимыми сервисами запрещен;
- отсутствует лицензионный ключ;
- конфигурационные файлы тестирования сервиса содержат недопустимые параметры;
- и т.д.

7.2. Запуск тестирований

Данный подраздел включает команды запуска для тестирования всех сервисов Системы.

> Необходимо запустить тестирование для сервисов, которые были запущены и настроены для работы.

> Тестирование запускается на пустых БД.

7.2.1. Команды для запуска тестирования

Необходимо запускать тестирование только для запущенных сервисов. Нужно убедиться, что использование не запущенных сервисов отключено в сервисе Configurator (при необходимости).

7.2.1.1. Тестирование API

Необходимо отключить тестирование Python Matcher Proxy (по умолчанию сервис не используется):

```
docker exec -it luna-api sed -i 's/USE_MATCHER_PROXY.*/USE_MATCHER_PROXY =
   0/' tests/config.py
```

Необходимо запустить тестирование:

```
docker exec luna-api python3.9 -m unittest tests.unittests_main
```

7.2.1.2. Тестирование Handlers

Необходимо запустить тестирование:

```
docker exec luna-handlers python3.9 -m unittest tests.unittests_main
```

7.2.1.3. Тестирование Faces

Необходимо запустить тестирование:

```
docker exec luna-faces python3.9 -m unittest tests.unittests_main
```

7.2.1.4. Тестирование Python Matcher

Необходимо запустить тестирование:

```
docker exec luna-python-matcher python3.7 -m unittest tests.unittests_main
```

7.2.1.5. Тестирование Admin

Необходимо запустить тестирование:

```
docker exec luna-admin python3.7 -m unittest tests.unittests_main
```

7.2.1.6. Тестирование Image Store

Необходимо запустить тестирование:

```
docker exec luna-image-store python3.9 -m unittest tests.unittests_main
```

7.2.1.7. Тестирование Tasks

Необходимо запустить тестирование:

```
docker exec luna-tasks python3.9 -m unittest tests.unittests_main
```

7.2.1.8. Тестирование Sender

Необходимо запустить тестирование:

```
docker exec luna-sender python3.9 -m unittest tests.unittests_main
```

7.2.1.9. Тестирование Events

Необходимо запустить тестирование:

```
docker exec luna-events python3.9 -m unittest tests.unittests_main
```

7.2.1.10. Тестирование Licenses

Необходимо запустить тестирование:

```
docker exec luna-licenses python3.9 -m unittest tests.unittests_main
```