



**VisionLabs**  
MACHINES CAN SEE

# **VisionLabs LUNA PLATFORM 5**

**Пример развертывания в кластере Kubernetes**

**v.5.67.0**

## Содержание

<b>Введение</b>	<b>3</b>
<b>1 Распаковка дистрибутива</b>	<b>4</b>
<b>2 Активация лицензии</b>	<b>5</b>
2.1 Действия из руководства по активации лицензии . . . . .	5
<b>3 Подготовка пользовательского Docker реестра для Lambda</b>	<b>6</b>
<b>4 Конфигурация баз данных</b>	<b>7</b>
4.1 Настройка InfluxDB . . . . .	7
4.2 Компиляция библиотеки VLMatch . . . . .	8
4.3 Создание пользователя и баз данных . . . . .	9
4.4 Добавление функций VLMatch в БД Faces и Events . . . . .	10
4.5 Установка PostGIS для БД Events . . . . .	10
<b>5 Задание настроек LUNA PLATFORM</b>	<b>11</b>
5.1 Настройки лицензии HASP . . . . .	12
5.2 Настройки лицензии Guardant . . . . .	12
5.3 Настройки GPU . . . . .	13
<b>6 Установка Helm чартов</b>	<b>14</b>
6.1 Настройка Helm чартов . . . . .	14
6.1.1 Настройка GPU для Remote SDK . . . . .	14
6.1.2 Настройка доступа для Lambda . . . . .	15
6.2 Запуск установки Helm чартов . . . . .	16
<b>7 Дополнительная информация</b>	<b>18</b>
7.1 Создание секрета для авторизации в реестре Docker . . . . .	18
7.2 Использование GPU в Minikube . . . . .	18
7.3 Компиляция библиотеки VLMatch для Oracle . . . . .	19

## Введение

Данный документ описывает порядок действий для разворачивания LUNA PLATFORM в Kubernetes с использованием Helm чартов из комплекта поставки.

Администратор должен иметь развернутый и настроенный кластер Kubernetes для использования Helm чартов. Предполагается, что в пользовательском кластере Kubernetes:

- запущены СУБД PostgreSQL/Oracle и базы данных InfluxDB и Redis
- имеется доступ к объектному хранилищу S3-подобного типа для хранения бакетов

**Важно!** Документация и комплект поставки не включают готовые решения для управления базами данных PostgreSQL/Oracle, InfluxDB и Redis в Kubernetes. Пользователь должен самостоятельно настроить базы данных для обеспечения лучшей отказоустойчивости и масштабируемости. Примеры команд, приведенные в данном документе, предназначены для демонстрации и могут потребовать адаптации под конкретную среду или требования вашего проекта.

Мониторинг в формате отправки данных в InfluxDB и сбор статистики запросов по умолчанию включены. Если доступ к InfluxDB не настроен, то сервисы LUNA PLATFORM не запустятся. Также можно настроить генерацию метрик в формате Prometheus для дальнейшей интеграции с Prometheus, развернутым в пользовательском кластере Kubernetes (см. настройку «LUNA\_SERVICE\_METRICS»).

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

<https://kubernetes.io/docs>

## 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
```

Откройте папку с дистрибутивом.

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

Распакуйте файлы.

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

## 2 Активация лицензии

Для активации лицензии необходимо выполнить следующие действия:

- выполнить действия из [руководства по активации лицензии](#)
- задать настройки лицензирования [HASP](#) или [Guardant](#) на этапе заполнения дамп-файла

### 2.1 Действия из руководства по активации лицензии

Откройте руководство по активации лицензии и выполните необходимые шаги.

В руководстве по активации лицензии приводятся шаги по активации лицензии на конкретном сервере. Тестирование HASP/Guardant в кластере Kubernetes не проводилось.

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

### 3 Подготовка пользовательского Docker реестра для Lambda

**Примечание.** Пропустите данный раздел если не собираетесь использовать сервис Lambda.

Необходимо подготовить пользовательский реестр для хранения образов Lambda. Перенесите базовые образы и образ Kaniko executor в свой реестр с помощью нижеперечисленных команд.

Загрузите образы из удаленного репозитория в локальное хранилище образов:

```
docker pull dockerhub.visionlabs.ru/luna/lpa-lambda-base-fsdk:v.0.1.14
```

```
docker pull dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.1.14
```

```
docker pull dockerhub.visionlabs.ru/luna/kaniko-executor:latest
```

Добавьте новые имена образам, заменив `new-registry` на свои. Имена базовых образов в пользовательском реестре должны быть такими же, как и в реестре `dockerhub.visionlabs.ru/luna`.

```
docker tag dockerhub.visionlabs.ru/luna/lpa-lambda-base-fsdk:v.0.1.14 new-registry/lpa-lambda-base-fsdk:v.0.1.14
```

```
docker tag dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.1.14 new-registry/lpa-lambda-base:v.0.1.14
```

```
docker tag dockerhub.visionlabs.ru/luna/kaniko-executor:latest new-registry/kaniko-executor:latest
```

Отправьте локальные образы в свой удаленный репозиторий, заменив `new-registry` на свои.

```
docker push new-registry/lpa-lambda-base-fsdk:v.0.1.14
```

```
docker push new-registry/lpa-lambda-base:v.0.1.14
```

```
docker push new-registry/kaniko-executor:latest
```

## 4 Конфигурация баз данных

Для корректной работы LUNA PLATFORM необходимо настроить базы данных следующим образом:

- [настроить InfluxDB](#)
- [скомпилировать библиотеку VLMatch и перенести в СУБД](#)
- [создать пользователя и базы данных для сервисов и присвоить им нужные права](#)
- [добавить функции VLMatch в БД Faces и Events](#)

VLMatch — функция для выполнения вычислений по сравнению биометрических шаблонов. Библиотека VLMatch компилируется для конкретной версии базы данных. Не используйте библиотеку, созданную для другой версии базы данных. Например, библиотека, созданная для PostgreSQL версии 16 нельзя использовать для PostgreSQL версии 12.

В разделах ниже приводятся команды для СУБД PostgreSQL. Для Oracle приводятся только команды по компиляции библиотеки VLMatch (см. раздел [«Компиляция библиотеки VLMatch для Oracle»](#) в разделе «Дополнительная информация»).

### 4.1 Настройка InfluxDB

Если InfluxDB уже развернут в вашем кластере Kubernetes, убедитесь, что следующие данные заданы корректно:

- **Имя пользователя и пароль**
- **Название бакета и организации**
- **Токен администратора**

**Важно!** Вышеописанные данные необходимо [указать в дамп-файле с настройками LUNA PLATFORM](#) для того, чтобы сервисы получили доступ к InfluxDB. Однако настройки сервиса Configurator нельзя задать в дамп-файле, поэтому их нужно задать в Helm чарте сервиса Configurator следующим образом:

```
env:
  - name: VL_SETTINGS.INFLUX_MONITORING.SEND_DATA_FOR_MONITORING
    value: "1"
  - name: VL_SETTINGS.INFLUX_MONITORING.ORGANIZATION
    value: "luna"
  - name: VL_SETTINGS.INFLUX_MONITORING.TOKEN
    value: "12345678"
  - name: VL_SETTINGS.INFLUX_MONITORING.BUCKET
    value: "luna_monitoring"
  - name: VL_SETTINGS.INFLUX_MONITORING.HOST
```

```
value: "influxdb"
- name: VL_SETTINGS.INFLUX_MONITORING.PORT
  value: "8086"
- name: VL_SETTINGS.INFLUX_MONITORING.USE_SSL
  value: "0"
- name: VL_SETTINGS.INFLUX_MONITORING.FLUSHING_PERIOD
  value: "1"
```

Настройки InfluxDB также можно указать в переменных окружения в Helm чарте каждого сервиса.

## 4.2 Компиляция библиотеки VLMatch

**Примечание.** В следующей инструкции приведен пример для СУБД PostgreSQL 16 на CentOS 8.

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

```
/var/lib/luna/luna_v.5.67.0/extras/VLMatch/postgres/
```

Для компиляции функции VLMatch UDx необходимо:

- установить репозиторий RPM:

```
dnf install -y https://download.postgresql.org/pub/repos/yum/repopms/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.

- открыть директорию скрипта make.sh и запустить его:

```
cd /var/lib/luna/luna_v.5.67.0/extras/VLMatch/postgres/
```

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

```
./make.sh
```

Перенесите сгенерированный файл VLMatchSource.so в СУБД PostgreSQL.

### 4.3 Создание пользователя и баз данных

В данном разделе приводятся примеры команд для создания пользователя и баз данных на примере СУБД PostgreSQL.

Создайте пользователя базы данных.

```
psql -U postgres -c 'create role luna;'
```

Присвойте пользователю пароль.

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

Создайте базы данных для всех сервисов:

```
psql -U postgres -c 'CREATE DATABASE luna_configurator;'  
psql -U postgres -c 'CREATE DATABASE luna_accounts;'  
psql -U postgres -c 'CREATE DATABASE luna_handlers;'  
psql -U postgres -c 'CREATE DATABASE luna_backport3;'  
psql -U postgres -c 'CREATE DATABASE luna_faces;'  
psql -U postgres -c 'CREATE DATABASE luna_events;'  
psql -U postgres -c 'CREATE DATABASE luna_tasks;'  
psql -U postgres -c 'CREATE DATABASE luna_lambda;'
```

Присвойте привилегии пользователю базам данных.

```
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_configurator TO luna;'
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_accounts TO luna;'
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_handlers TO luna;'
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_backport3 TO luna;'
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_faces TO luna;'
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_events TO luna;'
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_tasks TO luna;'
psql -U postgres -c 'GRANT ALL PRIVILEGES ON DATABASE luna_lambda TO luna;'
```

Разрешите пользователю авторизацию в базах данных:

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

**Примечание.** Обратите внимание, что имя пользователя и пароль указывается [в настройках LUNA PLATFORM](#) для соединения сервисов с БД.

#### 4.4 Добавление функций VLMatch в БД Faces и Events

Определите функцию VLMatch в базах данных Faces и Events:

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

#### 4.5 Установка PostGIS для БД Events

Сервис Events требует расширения PostGIS для работы с координатами.

Поскольку PostGIS является расширением для PostgreSQL, его версия обычно соответствует версии PostgreSQL, с которой оно совместимо.

Самостоятельно установите расширение для используемой версии PostgreSQL, используя [официальную документацию](#).

Для PostgreSQL 16 требуется версия PostGIS 3.4.

## 5 Задание настроек LUNA PLATFORM

Для минимальной работы LUNA PLATFORM необходимо задать следующие настройки:

- LICENSE\_VENDOR — настройки лицензии
- INFLUX\_MONITORING — настройки мониторинга и подключения к БД InfluxDB
- LUNA\_ATTRIBUTES\_DB — адрес БД Redis для хранения временных атрибутов
- TASKS\_REDIS\_DB\_ADDRESS — адрес БД Redis для сервиса Tasks
- LUNA\_<SERVICE>\_DB — настройки подключения к базам данных сервисов
- LUNA\_<SERVICE>\_ADDRESS — настройки с адресами сервисов
- REDIS\_DB\_ADDRESS — адрес БД Redis для сервиса Sender (при использовании сервиса Sender)
- LUNA\_IMAGE\_STORE\_<BUCKET>\_ADDRESS — настройки доступа к бакетам (при использовании сервиса Image Store)
- STORAGE\_TYPE — тип хранилища для хранения бакетов (S3 или локальное, при использовании сервиса Image Store)
- S3 — настройки S3-подобного хранилища для хранения бакетов (при использовании сервиса Image Store и STORAGE\_TYPE = S3)
- LAMBDA\_S3 — настройки S3-подобного хранилища для хранения архивов с модулями (при использовании сервиса Lambda)

Настройки можно задать в дамп-файле `extras/helms/luna-configurator/files/platform_settings.json`, который автоматически загружается в БД Configurator по время установки Helm чарта сервиса Configurator. Дамп-файл содержит шаблон, который необходимо актуализировать, вписав корректные пользовательские данные.

**Важно!** Загружаемый дамп-файл содержит минимально необходимый перечень настроек. При необходимости можно добавить дополнительные настройки, используя в качестве примера полный дамп-файл, расположенный по пути `extras/conf/luna_platform_<version>_dump.json`.

Актуализируйте загружаемый дамп-файл с помощью следующей команды:

```
vi /var/lib/luna/extras/helms/luna-configurator/files/platform_settings.json
```

В шаблоне Helm `luna-configurator/templates/init-configmap.yaml` используется функция `Files.Glob` для поиска всех файлов JSON в папке `luna-configurator/files` в Helm чарте сервиса Configurator.

Настройки лицензирования HASP и Guardant задаются по-разному. Выберите раздел ниже для настройки лицензии исходя из необходимого механизма защиты:

- [HASP](#)
- [Guardant](#)

## 5.1 Настройки лицензии HASP

**Примечание.** Выполняйте действия из данного раздела только если активируете лицензию с помощью HASP. Если нужно активировать лицензию Guardant, выполните действия из раздела «Настройки лицензии Guardant».

Задайте IP-адрес сервера с вашим ключом HASP в поле «server\_address»:

```
{
  "value": {
    "vendor": "hasp",
    "server_address": "<your-server-address>"
  },
  "description": "License vendor config",
  "name": "LICENSE_VENDOR",
  "tags": []
},
```

Сохраните файл.

## 5.2 Настройки лицензии Guardant

**Примечание.** Выполняйте действия из данного раздела только если активируете лицензию с помощью Guardant. Если нужно активировать лицензию HASP, выполните действия из раздела «Настройки лицензии HASP».

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": "<your-server-address>",
    "license_id": "0x92683BEA"
  },
  "description": "License vendor config",
  "name": "LICENSE_VENDOR",
  "tags": []
},
```

Сохраните файл.

### 5.3 Настройки GPU

**Примечание.** Пропустите данный раздел если не собираетесь использовать GPU.

GPU можно включить для сервисов Remote SDK и для отдельных экземпляров Lambda.

Настройки GPU для отдельных экземпляров Lambda задаются во время их создания (см. запрос «create lambda»).

Сервис Remote SDK не использует GPU по умолчанию.

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 эстиматоры 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 эстиматора или детектора, то необходимо использовать параметр «device\_class» в секциях вида "LUNA\_REMOTE\_SDK\_<estimator-or-detector-name>\_SETTINGS.runtime\_settings".

**Примечание.** В дамп-файле `extras/helms/luna-configurator/files/platform_settings.json` из комплекта поставки содержится только секция «LUNA\_REMOTE\_SDK\_RUNTIME\_SETTINGS», позволяющая включить GPU сразу для всех эстиматоров и детекторов. При необходимости можно самостоятельно добавить в дамп-файл настройки для необходимого эстиматора или детектора, используя в качестве примера полный дамп-файл, расположенный по пути `extras/conf/luna_platform_<version>_dump.json`.

Обратите внимание, что для секции «LUNA\_REMOTE\_SDK\_RUNTIME\_SETTINGS» в дамп-файле указан тег «gpu». Для использования настроек из данной секции нужно передать тегированную секцию с помощью переменной окружения «EXTEND\_CMD» в Helm чарте сервиса Remote SDK. Пример передачи тегированной настройки закомментирован в файле `values.yaml` для сервиса Remote SDK.

## 6 Установка Helm чартов

Примеры Helm чартов для каждого сервиса расположены в директории `extras/helms/`.

### 6.1 Настройка Helm чартов

Helm чарты из комплекта поставки не подходят для полноценной работы в продуктивном контуре. Необходимо настроить чарты в соответствии со своей бизнес логикой перед их установкой.

Перейдите в директорию с чартами:

```
cd /var/lib/luna/luna_v.5.67.0/extras/helms/
```

Настройте в файлах `luna-<service-name>/values.yaml` все необходимые параметры, особенно обращая внимание на:

- секцию `resources` для задания ресурсов (например, CPU и память) для контейнеров сервиса
- секцию `ingress` для настройки маршрутизации входящего трафика к сервису
- параметр `pullSecrets` в секции `image` для указания секрета, который будет использоваться при извлечении образа контейнера из реестра (см. [«Создание секрета для авторизации в реестре Docker»](#) в разделе «Дополнительная информация»).

**Примечание.** Рекомендуется настроить аннотацию `nginx.ingress.kubernetes.io/proxy-body-size` к сервису API (или к любому другому сервису, к которому отправляются запросы с изображениями) в зависимости от требований к размеру передаваемых изображений. В Helm чарте сервиса API дан пример использования данной аннотации.

Эти параметры играют важную роль в обеспечении производительности и доступности вашего приложения в продуктивной среде.

#### 6.1.1 Настройка GPU для Remote SDK

**Примечание.** Пропустите данный раздел если не собираетесь использовать GPU.

Использование GPU для сервиса Remote SDK включается с помощью передачи соответствующего ключа в секции `resources` в файле `values.yaml` соответствующего Helm чарта.

Например, можно настроить доступ к одному графическому процессору следующим образом:

```
resources:
  limits:
    cpu: 5000m
    memory: 10Gi
```

```
nvidia.com/gpu: 1
requests:
  cpu: 5000m
  memory: 10Gi
  nvidia.com/gpu: 1
```

**Примечание.** Также для включения эстимаций/детекций на GPU необходимо задать необходимые настройки (см. «[Настройки GPU](#)»). При необходимости можно использовать переменную EXTEND\_CMD для передачи тегированных настроек.

```
env:
  - name: EXTEND_CMD
    value: " --LUNA_REMOTE_SDK_RUNTIME_SETTINGS gpu"
```

### 6.1.2 Настройка доступа для Lambda

**Примечание.** Пропустите данный раздел если не собираетесь использовать сервис Lambda.

Для корректной работы сервиса Lambda необходимо правильно настроить доступ к ресурсам Kubernetes, чтобы обеспечить безопасность и эффективное управление сервисом. Это можно сделать, например, путем определения ролей и привязок ролей с помощью механизма управления доступом на основе ролей (RBAC).

Приведенный ниже пример показывает, как настроить доступы с использованием RBAC в Kubernetes для сервиса Lambda:

- Определите объект типа ServiceAccount, который представляет собой идентификатор, используемый сервисом для взаимодействия с Kubernetes API сервером:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: lambda-user
```

- Определите тип объекта Role, который определяет набор разрешений для ресурсов, с которыми ваш сервис будет работать:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: production
  name: lambda-admin-role
rules:
```

```
- apiGroups: [ "", "apps", "networking.k8s.io" ]
  resources: [ "deployments", "pods", "pods/log", "pods/status", "services",
    "services/proxy", "ingresses" ]
  verbs: [ "get", "watch", "list", "create", "delete", "patch" ]
```

Здесь `services/proxy` означает возможность отправки запросов к ресурсу `/lambdas/{lambda_id}/proxy` сервиса Lambda.

- Определите тип объекта `RoleBinding`, который связывает роль с созданным типом `ServiceAccount`, определяя, какие ресурсы и операции доступны сервису Lambda:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: admin-lambda
  namespace: production
subjects:
- kind: ServiceAccount
  name: lambda-user
  namespace: production
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: lambda-admin-role
```

## 6.2 Запуск установки Helm чартов

Перейдите в директорию с Helm чартами.

```
cd /var/lib/luna/luna_v.5.67.0/extras/helms
```

Запустите установку Helm чартов для необходимых сервисов с помощью следующих команд:

```
helm install --wait --timeout 10m luna-configurator ./luna-configurator
helm install --wait --timeout 10m luna-image-store ./luna-image-store
helm install --wait --timeout 10m luna-licenses ./luna-licenses
helm install --wait --timeout 10m luna-faces ./luna-faces
helm install --wait --timeout 10m luna-events ./luna-events
helm install --wait --timeout 10m luna-python-matcher ./luna-python-matcher
helm install --wait --timeout 10m luna-remote-sdk ./luna-remote-sdk
helm install --wait --timeout 10m luna-handlers ./luna-handlers
helm install --wait --timeout 10m luna-sender ./luna-sender
```



```
helm install --wait --timeout 10m luna-tasks-worker ./luna-tasks-worker
helm install --wait --timeout 10m luna-tasks ./luna-tasks
helm install --wait --timeout 10m luna-accounts ./luna-accounts
helm install --wait --timeout 10m luna-lambda ./luna-lambda
helm install --wait --timeout 10m luna-api ./luna-api
helm install --wait --timeout 10m luna-admin ./luna-admin
helm install --wait --timeout 10m luna-backport3 ./luna-backport3
helm install --wait --timeout 10m luna-backport4 ./luna-backport4
```

Перед запуском сервисов UI 4 и UI 3 необходимо выполнить дополнительные действия в Helm чартах:

- актуализировать параметр LUNA\_API\_URL для обоих Helm чартов, который является внутренним адресом Backport 3 и Backport 4 соответственно
- актуализировать параметр BASIC\_AUTH для Helm чарта UI 4, указав данные авторизации для аккаунта типа **user** в формате login:password, закодированным в Base64

Необходимо предварительно создать аккаунт типа «user» с помощью запроса «create account» к сервису API или с помощью сервиса Admin.

Запустите установку Helm чартов UI 4 и UI 3 с помощью следующих команд:

```
helm install --wait --timeout 10m luna3-ui ./luna3-ui
helm install --wait --timeout 10m luna4-ui ./luna4-ui
```

После установки Helm чартов рекомендуется провести тщательное тестирование LUNA PLATFORM в среде, которая соответствует вашим требованиям по производительности и безопасности.

## 7 Дополнительная информация

В данном разделе приводится следующая дополнительная информация:

- [шаги по созданию секрета для авторизации в реестре Docker](#)
- [нюансы использования GPU в Minikube](#)
- [пример компиляции библиотеки VLMATCH для Oracle](#)

### 7.1 Создание секрета для авторизации в реестре Docker

Чтобы скачивать образы с сервисами LUNA PLATFORM нужно авторизоваться в реестре Docker. Создайте файл с учетными данными, например, `vlabs-credentials.txt`, содержащий логин и пароль:

```
{
  "auths": {
    "dockerhub.visionlabs.ru": {
      "username": "your_username",
      "password": "your_password"
    }
  }
}
```

Предоставьте Kubernetes доступ к реестру с образами Docker.

```
kubectl create secret generic my-dockerhub-secret --from-file=.
dockerconfigjson=vlabs-credentials.txt --type=kubernetes.io/
dockerconfigjson
```

Если вы ранее уже авторизовались через команду `docker login`, то можно предоставить доступ Kubernetes с помощью следующей команды:

```
kubectl create secret generic my-dockerhub-secret --from-file=.
dockerconfigjson=$HOME/.docker/config.json --type=kubernetes.io/
dockerconfigjson
```

Секрет можно указать во время [настройки Helm чартов](#).

### 7.2 Использование GPU в Minikube

Minikube — это инструмент для локальной установки и управления кластером Kubernetes. Он используется разработчиками и тестировщиками для создания и тестирования приложений в ло-

кальной среде перед их развертыванием в более крупных кластерах Kubernetes.

Использование GPU в Minikube поддерживается только с версии 1.32.

Каждый сервис LUNA PLATFORM, поддерживающий работу на GPU, автоматически создает процессы на GPU, независимо от того, какие ресурсы (CPU или GPU) установлены. Если запускается более одного сервиса на GPU, то ресурсы графического процессора необходимо делить между ними чтобы избежать возможных ошибок, вызванных конфликтом доступа к видеокарте.

См. [официальную документацию NVIDIA](#) для более подробной информации о разделении ресурсов GPU.

Для изоляции сервисов от GPU и предотвращения создания ими дополнительных процессов следует установить переменную окружения `CUDA_VISIBLE_DEVICES/NVIDIA_VISIBLE_DEVICES` на `none` для тех сервисов, которые используют GPU и не должны использоваться.

```
env:
  - name: CUDA_VISIBLE_DEVICES
    value: none
```

### 7.3 Компиляция библиотеки 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/luna_v.5.67.0/extras/VLMatch/oracle
```

Для компиляции функции VLMatch UDx необходимо:

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

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

- 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/luna_v.5.67.0/extras/VLMatch/oracle/make.sh
```

- Откройте директорию и запустите файл «`make.sh`».

```
cd /var/lib/luna/luna_v.5.67.0/extras/VLMatch/oracle
```

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

```
./make.sh
```

- 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);
```

- Протестируйте функцию посредством вызова (из консоли базы данных):

```
SELECT VLMatch(HEXTORAW('123456789012345678901234567890123456789012345678901234'  
    ,  
    HEXTORAW('012345678901234567890123456789012345678901234567890123'  
    ), 32)  
FROM DUAL;
```

Результат должен быть равен «0.4765625».

Перенесите сгенерированный файл VLMatchSource.so в СУБД Oracle.