VisionLabs
MACHINES CAN SEE

VisionLabs LUNA PLATFORM 5

Пример развертывания в кластере Kubernetes

v.5.94.0

Содержание

Введение

Данный документ описывает:

- подготовку окружения LUNA PLATFORM в Kubernetes с использованием утилиты Storages и соответствующих манифестов из комплекта поставки.
- запуск сервисов LUNA PLATFORM в Kubernetes с использованием Helm чартов из комплекта поставки.

> В основном разделе документа приводится пример запуска и подготовки окружения для всех сервисов кроме сервиса Lambda и сервисов Backport 3, Backport 4, UI 3 и UI 4. Информация о подготовке окружения для данных сервисов описана в разделах «Запуск Lambda» и «Backports» в дополнительной информации.

С помощью документа можно как установить LUNA PLATFORM с нуля, так и выполнить обновление или даунгрейд.

> См. подробную информацию про Storages в руководстве по утилите Storages.

Администратор должен иметь развернутый и настроенный кластер Kubernetes для использования Helm чартов и манифестов. 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.5.94.0**, где **v.5.94.0** 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.5.94.0.zip /var/lib/luna
```

Откройте папку с дистрибутивом.

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

Распакуйте файлы.

```
unzip luna_v.5.94.0.zip
```

2 Создание символической ссылки

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

```
ln -s luna_v.5.94.0 current
```

Перейдите в рабочую директорию с файлами Kubernetes для дальнейшей работы:

```
cd /var/lib/luna/current/extras/k8s/
```

3 Примечания перед обновлением/даунгрейдом

Примечание. Пропустите данный раздел если разворачиваете LUNA PLATFORM с нуля.

Во время обновления/даунгрейда необходимо выполнить следующие действия:

- спланировать воздействие на запущенные сервисы, использующие базы данных. См. подробную информацию в разделе «Рекомендации по поведению сервисов во время подготовки окружения» руководства по утилите Storages.
- при даунгрейде указать нужную версию LUNA PLATFORM в переменной `--platform-version` в следующих файлах:
  - `storages/overlays/check/overlay-check.yaml`
  - `storages/overlays/prepare/overlay-prepare.yaml`

  См. раздел «Подготовка окружения».
- удалить старые Helm чарты и манифесты подготовки окружения перед установкой новых. См. раздел «Удаление старых Helm чартов и манифестов».

4 Удаление старых Helm чартов и манифестов

Примечание. Пропустите данный раздел если разворачиваете LUNA PLATFORM с нуля.

Удалите старые Helm чарты всех сервисов LUNA PLATFORM с помощью команды `helm uninstall`. Например, `helm uninstall luna-configurator`.

Удалите старые джобы и configmap'ы, которые использовались при подготовке предыдущего окружения, если они еще не были удалены:

```
kubectl delete configmap storages-config
kubectl delete configmap storages-dump
kubectl delete job storages-prepare
kubectl delete job storages-load-dump
```

5 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

5.1 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.

6 Включение GPU

Примечание. Пропустите данный раздел если не собираетесь использовать GPU.

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

Для включения GPU необходимо выполнить следующие действия:

- настроить GPU в пользовательском дампе
- настроить GPU в Helm-чарте

Выполните вышеописанные действия на соответствующих этапах настройки.

7 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 БД
- установить Postgis для БД 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, библиотеку, созданная для PostgreSQL версии 16 нельзя использовать для PostgreSQL версии 12.

> Storages автоматически добавит функции VLMatch в СУБД PostgreSQL и активирует Postgis.

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»).

7.1 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.LUNA_MONITORING.STORAGE_TYPE
      value: "influx"
    - name: VL_SETTINGS.LUNA_MONITORING.SEND_DATA_FOR_MONITORING
      value: "1"
    - name: VL_SETTINGS.LUNA_MONITORING.ORGANIZATION
      value: "luna"
    - name: VL_SETTINGS.LUNA_MONITORING.TOKEN
```

```
      value: "12345678"
    - name: VL_SETTINGS.LUNA_MONITORING.BUCKET
      value: "luna_monitoring"
    - name: VL_SETTINGS.LUNA_MONITORING.HOST
      value: "influxdb"
    - name: VL_SETTINGS.LUNA_MONITORING.PORT
      value: "8086"
    - name: VL_SETTINGS.LUNA_MONITORING.USE_SSL
      value: "0"
    - name: VL_SETTINGS.LUNA_MONITORING.FLUSHING_PERIOD
      value: "1"
```

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

7.2 Компиляция библиотеки VLMatch

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

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/reporpms/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/current/extras/VLMatch/postgres/
```

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

```
./make.sh
```

Перенесите сгенерированный файл `VLMatchSource.so` в СУБД PostgreSQL в директорию `\srv`.

7.3 Создание пользователя БД

В данном разделе приводятся примеры команд для создания пользователя 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';"
```

Примечание. Обратите внимание, что имя пользователя и пароль указывается в настройках LUNA PLATFORM для соединения сервисов с БД.

Storages автоматически создаст нужные базы данных и выдаст все необходимые права в соответствии с именем пользователя из конфигурационного файла Storages.

7.4 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.

8 Задание настроек LUNA PLATFORM

Для минимальной работы LUNA PLATFORM необходимо задать следующие настройки:

- `LICENSE_VENDOR` — настройки лицензии
- `LUNA_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_RETRANSLATOR_DB_ADDRESS` — адрес БД Redis для сервиса Streams Retranslator (при использовании сервиса Streams Retranslator)
- `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)

> Обратите внимание, что для включения опциональных сервисов нужно также обновить настройку `ADDITIONAL_SERVICE_USAGE`.

> Запуск сервиса Lambda описан в разделе «Запуск Lambda» в дополнительной информации.

Настройки можно задать в дамп-файле `storages/files/platform_settings.json`, который автоматически загружается в БД Configurator во время выполнения команды `load_dump`. 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 `/var/lib/luna/current/extras/conf/luna_platform_<version>_dump.json`.

Актуализируйте загружаемый дамп-файл с помощью следующей команды:

```
vi /var/lib/luna/current/extras/k8s/storages/files/platform_settings.json
```

Настройки лицензирования HASP и Guardant задаются по-разному. Выберите раздел ниже для на-

стройки лицензии исходя из необходимого механизма защиты:

- HASP
- Guardant

8.1 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.

8.2 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.

8.3 Настройки GPU

Примечание. Пропустите данный раздел если не собираетесь использовать GPU.

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

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

> Запуск сервиса Lambda описан в разделе «Запуск Lambda» в дополнительной информации.

Сервисы Remote SDK и Video Agent 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» или «LUNA_VIDEO_AGENT_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 `storages/files/platform_settings.json` из комплекта поставки содержатся только секции «LUNA_REMOTE_SDK_RUNTIME_SETTINGS» и «LUNA_VIDEO_AGENT_-RUNTIME_SETTINGS», позволяющие 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 `/var/lib/luna/current/extras/conf/luna_platform_<version>_dump.json`.

> Обратите внимание, что для секций «LUNA_REMOTE_SDK_RUNTIME_SETTINGS» и «LUNA_-VIDEO_AGENT_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 чартах сервисов Remote SDK и Video Agent. Пример передачи тегированной настройки закомментирован в файле `values.yaml` для сервисов Remote SDK и Video Agent.

9 Задание настроек Storages

Для корректной подготовки окружения необходимо настроить конфигурацию Storages так, чтобы утилита имела возможность взаимодействовать с различными сервисами, базами данных, бакетами и другими ресурсами. Необходимые настройки Storages можно указать с помощью:

- конфигурационного файла сервиса Storages
- запущенного сервиса Configurator (при обновлении/даунгрейде)

В манифестах в комплекте поставки настройки передаются с помощью указания конфигурационного файла, который необходимо заполнить.

Важно! Особенно необходимо обратить внимание на нижеописанные настройки:

- `LUNA_MONITORING` — настройки мониторинга и подключения к БД InfluxDB
- `S3` — настройки S3-подобного хранилища для хранения бакетов

Примечание. Необходимо включить использование S3-подобного хранилища в настройке «STORAGE_TYPE» в пользовательском дамп-файле.

Примечание. Если параметр `SEND_DATA_FOR_MONITORING` в группе параметров `LUNA_MONITORING` отключен, то подготовка окружения для бакетов InfluxDB выполнена не будет.

Актуализируйте конфигурационный файл Storages с помощью следующей команды:

```
vi /var/lib/luna/current/extras/k8s/storages/files/storages_config.conf
```

10 Подготовка окружения

Окружение готовится с использованием утилиты Storages.

Убедитесь, что вы находитесь в рабочей директории с файлами Kubernetes:

```
cd /var/lib/luna/current/extras/k8s/
```

С помощью утилиты можно подготовить окружение для установки с нуля, обновления или даунгрейда. По умолчанию для аргумента `--platform-version` установлено значение актуальной версии LUNA PLATFORM. Для даунгрейда необходимо указать соответствующую версию LUNA PLATFORM.

Важно! В ходе подготовки окружения при обновлении или даунгрейде важно тщательно планировать воздействие на запущенные сервисы, использующие базы данных. См. подробную информацию в разделе «Рекомендации по поведению сервисов во время подготовки окружения» руководства по утилите Storages.

Необходимо задать следующие настройки в соответствии с пользовательской логикой:

- `storages/files/platform_settings.json` — дамп-файл, позволяющий переопределить дефолтные настройки с помощью команды `load_dump`.
- `storages/files/storages_config.conf` — настройки Storages, позволяющие утилите ходить в БД, S3 и пр.
- `storages/overlays/<command_name>` — позиционные аргументы Storages.

Подготовка окружения выполняется с помощью инструмента Kustomize без использования Helm. В файле `storages/base/base.yaml` определен шаблон, на который накладываются слои из директории `storages/overlays`. Доступно 4 слоя - `check`, `list`, `load_dump`, `prepare`, каждый из которых соответствует определенной команде позиционного аргумента `luna_prepare`.

Для загрузки образа из регистри VisionLabs необходимо также заполнить параметр `imagePullSecrets` в манифесте `storages/base/base.yaml` (см. «Создание секрета для авторизации в реестре Docker» в разделе «Дополнительная информация»).

Для подготовки окружения нужно выполнить следующие команды:

1) Создать Configmap для загрузки настроек Storages и дамп-файла:

```
kubectl create configmap storages-config --from-file=storages/files/
   storages_config.conf
kubectl create configmap storages-dump --from-file=storages/files/
   platform_settings.json
```

2) Выполнить подготовку окружения с помощью команды `prepare`:

```
kubectl apply -k storages/overlays/prepare
kubectl get pods
kubectl logs storages-prepare-xxxxx
```

> По умолчанию готовится окружение для всех сущностей (позиционный аргумент `all_entites`). Для подготовки отдельных сущностей необходимо отредактировать файл `storages/overlays/prepare/overlay-prepare.yaml` соответствующим образом.

3) Загрузить пользовательский дамп-файл с помощью команды `load_dump`:

```
kubectl apply -k storages/overlays/load_dump
kubectl get pods
kubectl logs storages-load-dump-xxxxx
```

> После выполнения поды будут иметь статус `Completed` и не будут автоматически удалены. Для корректного удаления манифеста необходимо выполнить команду `kubectl delete -k storages/overlays/<storages_command>`.

При необходимости можно использовать команды `list` и `check` аналогично вышеописанным примерам.

Набор манифестов не включает в себя пример использования команды `logs`.

11 Установка Helm чартов

Убедитесь, что вы находитесь в рабочей директории с файлами Kubernetes:

```
cd /var/lib/luna/current/extras/k8s/
```

11.1 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.

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.

11.1.1 Настройка GPU для Remote SDK и Video Agent

Примечание. Пропустите данный раздел если не собираетесь использовать GPU.

Использование GPU для сервисов Remote SDK и Video Agent 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"
```

11.2 Запуск установки Helm чартов

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-video-manager ./luna-video-manager
helm install --wait --timeout 10m luna-video-agent ./luna-video-agent
helm install --wait --timeout 10m luna-streams-retranslator ./luna-streams-
   retranslator
helm install --wait --timeout 10m luna-api ./luna-api
helm install --wait --timeout 10m luna-admin ./luna-admin
```

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

12 Дополнительная информация

В данном разделе приводится следующая дополнительная информация:

- шаги по созданию секрета для авторизации в реестре Docker
- шаги по подготовке окружения и запуску сервисов Backpors
- шаги по запуску Lambda
- нюансы использования GPU в Minikube
- пример компиляции библиотеки VLMatch для Oracle

12.1 Создание секрета для авторизации в реестре Docker

Чтобы скачивать образы с сервисами LUNA PLATFORM нужно авторизироваться в реестре Docker.

Создайте файл с учетными данными, например, `vlabs-credentials.json`, 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.json --type=kubernetes.io/
   dockerconfigjson
```

Если вы ранее уже авторизировались 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.

12.2 Backports

12.2.1 Подготовка окружения

Для подготовки окружения необходимо включить профиль `backports` в файле `storages/overlays/prepare/overlay-prepare.yaml`:

```
  command: ["/bin/bash", "-c", "luna_prepare prepare all_entities \
    --s3-buckets \
    --ignore-integrity \
    --platform_version={{.LUNA_PLATFORM_TAG}} \
    --profile=backports \
    --config=/srv/storages_config.conf"]
```

Далее необходимо подготовить окружение аналогично описанию из раздела «Подготовка окружения».

12.2.2 Запуск установки Helm чартов

Перейдите в директорию с Helm чартами.

```
  cd /var/lib/luna/current/extras/k8s
```

Запустите установку Helm чартов для сервиса Lambda с помощью следующих команд:

```
  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 `user@mail.com: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
```

12.3 Запуск Lambda

Для запуска Lambda необходимо выполнить некоторые дополнительные действия.

12.3.1 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 инструмента для сборки контейнеров 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.0.7.0
```

```
docker pull dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.7.0
```

Загрузите используемый образ инструмента для сборки контейнеров:

```
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.0.7.0 new-
    registry/lpa-lambda-base-fsdk:v.0.7.0
```

```
docker tag dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.7.0 new-registry
    /lpa-lambda-base:v.0.7.0
```

```
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.0.7.0
```

```
docker push new-registry/lpa-lambda-base:v.0.7.0
```

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

12.3.2 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
```

12.3.3 Запуск установки Helm чарта для Lambda

Перейдите в директорию с Helm чартами.

```
cd /var/lib/luna/current/extras/k8s
```

Запустите установку Helm чартов для сервиса Lambda с помощью следующих команд:

```
helm install --wait --timeout 10m luna-lambda ./luna-lambda
```

12.4 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 локальной 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
```

12.5 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/current/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/current/extras/VLMatch/oracle/make.sh
```

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

```
cd /var/lib/luna/current/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('
   12345678901234567890123456789012345678901234567890123456789012345678901234'),
   HEXTORAW('
   01234567890123456789012345678901234567890123456789012345678901234567890123'), 32)
   FROM DUAL;
```

Результат должен быть равен «0.4765625».

Перенесите сгенерированный файл `VLMatchSource.so` в СУБД Oracle.