



VisionLabs
MACHINES CAN SEE

VisionLabs LUNA PLATFORM 5

Example of deployment in Kubernetes cluster

v.5.95.0

Contents

Introduction	4
1 Unpacking the distribution	5
2 Symbolic link creation	6
3 Notes before upgrading/downgrading	7
4 Uninstall old Helm charts and manifests	8
5 License activation	9
5.1 Actions from License activation manual	9
6 Enabling GPU	10
7 Database configuration	11
7.1 Configuring InfluxDB	11
7.2 VLMatch library compilation	12
7.3 Create database user	13
7.4 Install PostGIS for Events database	13
8 Define LUNA PLATFORM settings	15
8.1 HASP license settings	16
8.2 Guardant license settings	16
8.3 GPU settings	17
9 Configuring Storages	18
10 Prepare environment	19
11 Install Helm charts	21
11.1 Helm chart setting	21
11.1.1 GPU setup for Remote SDK and Video Agent	21
11.2 Start installation of Helm charts	22
12 Additional information	23
12.1 Create Docker registry authentication secret	23
12.2 Backports	24
12.2.1 Prepare environment	24
12.3 Start installation of Helm charts	24
12.4 Launch Lambda	24
12.4.1 Prepare user Docker registry for Lambda	25

12.4.2	Configuring access for Lambda	26
12.4.3	Start installation of Helm chart for Lambda	27
12.5	Use GPU in Minikube	27
12.6	VLMATCH library compilation for Oracle	28

Introduction

This document describes:

- Preparing LUNA PLATFORM environment in Kubernetes using Storages utility and corresponding manifests from the distribution package.
- Running LUNA PLATFORM services in Kubernetes using Helm charts from the distribution package.

The main section of the document provides an example of launching and preparing the environment for all services except the Lambda service and the Backport 3, Backport 4, UI 3, and UI 4 services. Information on preparing the environment for these services is described in the “Launch Lambda” and “Backports” sections of the additional information.

Using the document, you can either install LUNA PLATFORM from scratch, or perform an update or downgrade.

See detailed information about Storages in the Storages utility manual.

The administrator must have a Kubernetes cluster deployed and configured to use Helm charts and manifests. It is assumed that the user’s Kubernetes cluster:

- PostgreSQL/Oracle DBMS and InfluxDB and Redis databases are running.
- There is access to S3-like object storage for storing buckets.

Important: The documentation and distribution package do not include out-of-the-box solutions for managing PostgreSQL/Oracle, InfluxDB and Redis databases in Kubernetes. The user should configure the databases themselves for better fault tolerance and scalability. The sample commands in this document are for demonstration purposes and may need to be customized for your project’s specific environment or requirements.

Monitoring in the format of sending data to InfluxDB and collecting query statistics is enabled by default. If access to InfluxDB is not configured, LUNA PLATFORM services will not start. You can also configure Prometheus metrics generation for further integration with Prometheus deployed in a custom Kubernetes cluster (see the “LUNA_SERVICE_METRICS” setting).

This document does not include guidance on how to use Kubernetes. Please refer to the Kubernetes documentation for more details:

<https://kubernetes.io/docs>

1 Unpacking the distribution

The distribution is an archive **luna_v.5.95.0**, where **v.5.95.0** is a numeric identifier denoting the version of LUNA PLATFORM.

The archive includes the configuration files required for installation and use. It does not include the Docker service images, these need to be downloaded from the Internet separately.

Move the distribution to a directory on your server before installing. For example, move the files to the `/root/` directory. It should not contain any other distribution or license files other than the target files.

Create a directory to unzip the distribution file.

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

Move the distribution to the directory with LUNA PLATFORM.

```
mv /root/luna_v.5.95.0.zip /var/lib/luna
```

Open the distribution folder.

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

Unzip the files.

```
unzip luna_v.5.95.0.zip
```

2 Symbolic link creation

Create a symbolic link. The link indicates that the current version of the distribution file is used to run LUNA PLATFORM.

```
ln -s luna_v.5.95.0 current
```

Go to the working directory with Kubernetes files for further work:

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

3 Notes before upgrading/downgrading

Note: Skip this section if you are deploying LUNA PLATFORM from scratch.

During an update/downgrade, you must perform the following steps:

- Plan the impact on running services that use databases. See the “Recommendations services behavior during environment preparation” section of the Storages utility manual for details.
- When downgrading, specify the desired version of LUNA PLATFORM in the `--platform-version` variable in the following files:
 - `storages/overlays/check/overlay-check.yaml`
 - `storages/overlays/prepare/overlay-prepare.yaml`

See [“Prepare environment”](#).

- Uninstall old Helm charts and environment preparation manifests before installing new ones. See [“Uninstall old Helm charts and manifests”](#).

4 Uninstall old Helm charts and manifests

Note: Skip this section if you are deploying LUNA PLATFORM from scratch.

Uninstall old Helm charts of all LUNA PLATFORM services using the `helm uninstall` command. For example, `helm uninstall luna-configurator`.

Delete old jobs and configmaps that were used to prepare the previous environment, if they have not already been deleted:

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


5 License activation

To activate the license, follow these steps:

- Follow the steps from [license activation manual](#).
- Set settings for [HASP](#) license or [Guardant](#) license.

5.1 Actions from License activation manual

Open the license activation manual and follow the necessary steps.

The license activation guide provides steps to activate the license on a specific server. HASP/Guardant has not been tested in a Kubernetes cluster.

Note: This action is mandatory. The license will not work without following the steps to activate the license from the corresponding manual.

6 Enabling GPU

Note: Skip this section if you do not intend to use the GPU.

GPU can be enabled for Remote SDK services, Video Agent, and for individual Lambda instances.

To enable GPU, you need to do the following:

- [configure GPU in user dump](#)
- [configure GPU in Helm chart](#)

Follow the steps above for the appropriate configuration steps.

7 Database configuration

For LUNA PLATFORM to work correctly, you must configure the databases as follows:

- [Configure InfluxDB.](#)
- [Compile the VLMatch library and transfer it to the DBMS.](#)
- [Create database user.](#)
- [Install PostGIS for Events database.](#)

VLMatch is a function for performing descriptor matching calculations. The VLMatch library is compiled for a specific version of the database. Do not use a library created for a different version of the database. For example, a library created for PostgreSQL version 16 cannot be used for PostgreSQL version 12.

Storages will automatically add VLMatch functions to the PostgreSQL DBMS and activate Postgis.

The sections below provide commands for PostgreSQL. For Oracle, only the VLMatch library compilation commands are given (see [“VLMatch library compilation for Oracle”](#) in the “Additional information” section).

7.1 Configuring InfluxDB

If InfluxDB is already deployed in your Kubernetes cluster, make sure the following information is set correctly:

- **Username and password**
- **Bucket and organization name**
- **Administrator Token**

Important: The above data must be [specified in the LUNA PLATFORM settings dump file](#) in order for services to access InfluxDB. However, Configurator service settings cannot be specified in the dump file, so they must be specified in the Configurator service Helm chart as follows:

```
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"
```

InfluxDB settings can also be specified in environment variables in the Helm chart of each service.

7.2 VLMatch library compilation

Note: The following instructions provide an example for PostgreSQL 16 DBMS on Almalinux 8.

All files required to compile the user-defined extension (UDx) into VLMatch can be found in the following directory:

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

To compile the VLMatch UDx function, you need to:

- Install the RPM repository:

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

- Install PostgreSQL:

```
dnf install postgresql16-server
```

- Install development environment:

```
dnf install postgresql16-devel
```

- Install the gcc package:

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

- Install CMAKE. Version 3.5 or higher is required.

- Open the `make.sh` script in a text editor. It includes the paths to the currently used PostgreSQL version. Change the following values (if necessary):

`SDK_HOME` specifies the path to the PostgreSQL home directory. The default is `/usr/pgsql-16/include/server`.

`LIB_ROOT` specifies the path to the PostgreSQL library root directory. The default is `/usr/pgsql-16/lib`.

- Open the `make.sh` script directory and run it:

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

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

```
./make.sh
```

Move the generated `VLMatchSource.so` file to the PostgreSQL DBMS in the `\srv` directory.

7.3 Create database user

This section provides sample commands for creating a user using the PostgreSQL DBMS as an example.

Create a database user.

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

Assign a password to the user.

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

Note: Note that the username and password are specified in [LUNA PLATFORM settings](#) to connect services to the database.

Storages will automatically create the necessary databases and grant all necessary rights according to the user name from the Storages configuration file.

7.4 Install PostGIS for Events database

The Events service requires a PostGIS extension to work with coordinates.

Since PostGIS is an extension for PostgreSQL, its version usually corresponds to the version of PostgreSQL with which it is compatible.

Install the extension yourself for the PostgreSQL version you are using, using [official documentation](#).

PostgreSQL 16 requires PostGIS version 3.4.

8 Define LUNA PLATFORM settings

The following settings must be set for LUNA PLATFORM to work minimally:

- `LICENSE_VENDOR` — License settings.
- `LUNA_MONITORING` — Settings for monitoring and connection to the InfluxDB database.
- `LUNA_ATTRIBUTES_DB` — Redis database address for storing temporary attributes.
- `TASKS_REDIS_DB_ADDRESS` — Redis database address for the Tasks service.
- `LUNA_<SERVICE>_DB` — Settings of connection to service databases.
- `LUNA_<SERVICE>_ADDRESS` — Settings with service addresses.
- `REDIS_DB_ADDRESS` — Redis database address for Sender service (when using Sender service).
- `LUNA_RETRANSLATOR_DB_ADDRESS` — Redis database address for Streams Retranslator service (when using Streams Retranslator service).
- `LUNA_IMAGE_STORE_<BUCKET>_ADDRESS` — Settings for access to bucket (when using Image Store service).
- `STORAGE_TYPE` — Type of storage for bucket storage (S3 or local, when using Image Store service).
- `S3` — Settings of S3-like storage for storing bucket (when using Image Store service and `STORAGE_TYPE = S3`).
- `LAMBDA_S3` — Settings of S3-like storage for storing archives with modules (when using Lambda service).

To enable optional services you also need to update the `ADDITIONAL_SERVICE_USAGE` setting.

Launching the Lambda service is described in the [“Launch Lambda”](#) section in the additional information.

The settings can be specified in the `storages/files/platform_settings.json` dump file, which is automatically loaded into the Configurator database during the `load_dump` command execution. The dump file contains a template that must be updated by entering the correct user data.

Important: The downloaded dump file contains the minimum required list of settings. If necessary, you can add additional settings using the full dump file located at `/var/lib/luna/current/extras/conf/luna_platform_<version>_dump.json` as an example.

Update the dump file to be loaded using the following command:

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

HASP and Guardant license settings are set differently. Select the section below to configure the license based on the required protection mechanism:

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

8.1 HASP license settings

Note: Follow the steps in this section only if you are activating the license with HASP. If you need to activate a Guardant license, follow the steps in [“Guardant license settings”](#).

Specify the IP address of the server with your HASP key in the “server_address” field:

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

Save the file.

8.2 Guardant license settings

Note: Follow the steps in this section only if you are activating the license with Guardant. If you need to activate a HASP license, follow the steps in [“HASP license settings”](#).

Set the following details:

- IP address of the server with your Guardant key in the “server_address” field.
- License ID in the format 0x<your_license_id> obtained in the section “Saving the license ID” in the license activation guide, in the field “license_id”:

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

Save file.

8.3 GPU settings

Note: Skip this section if you do not intend to use the GPU.

GPU can be enabled for Remote SDK and Video Agent services and for individual Lambda instances.

GPU settings for individual Lambda instances are set at creation time (see the “create lambda” request).

Launching the Lambda service is described in the “[Launch Lambda](#)” section in the additional information.

The Remote SDK and Video Agent services do not use the GPU by default.

If you want to use the GPU for all estimators and detectors at once, you must use the “global_device_class” parameter in the “LUNA_REMOTE_SDK_RUNTIME_SETTINGS” or “LUNA_VIDEO_AGENT_RUNTIME_SETTINGS” section. All estimators and detectors will use the value of this parameter if the “device_class” parameter of their own settings is set to “global” (default).

If you want to use the GPU for a specific estimator or detector, you must use the “device_class” parameter in sections like “LUNA_REMOTE_SDK_estimator-or-detector-name_SETTINGS.runtime_settings”.

Note: The storages/files/platform_settings.json dump file from the delivery set contains only the “LUNA_REMOTE_SDK_RUNTIME_SETTINGS” and “LUNA_VIDEO_AGENT_RUNTIME_SETTINGS” sections, which allows enabling GPU for all estimators and detectors at once. If necessary, you can add settings for the required estimator or detector to the dump file yourself, using the full dump file located at the path /var/lib/luna/current/extras/conf/luna_platform_<version>.dump.json as an example.

Note that the “LUNA_REMOTE_SDK_RUNTIME_SETTINGS” and “LUNA_VIDEO_AGENT_RUNTIME_SETTINGS” sections in the dump file have the “gpu” tag specified. To use the settings from this section, you need to transfer the tagged section using the “EXTEND_CMD” environment variable in the Helm chart of the Remote SDK or Video Agent service. An example of passing a tagged setting is commented out in the values.yaml file for the Remote SDK and Video Agent services.

9 Configuring Storages

To prepare the environment correctly, you must configure the Storages configuration so that the utility is able to interact with various services, databases, bucket and other resources. The necessary Storages settings can be specified using:

- Configuration file of the Storages service
- Running Configurator service (when upgrading/downgrading).

In the manifests in the distribution package, the settings are transferred by specifying a configuration file to be filled in.

Important: It is especially important to pay attention to the following settings:

- LUNA_MONITORING — Settings for monitoring and connecting to the InfluxDB.
- S3 — Settings for an S3-like storage for storing buckets.

Note. It is necessary to enable the use of an S3-like storage in the “[STORAGE_TYPE](#)” setting in the custom dump file.

Update the Storages configuration file using the following command:

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

If the `SEND_DATA_FOR_MONITORING` parameter in the Storages configuration is disabled, the environment preparation for InfluxDB buckets will not be performed.

10 Prepare environment

The environment is prepared using the Storages utility.

Make sure you are in the working directory with the Kubernetes files:

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

Using the utility, you can prepare the environment for installation from scratch, upgrade or downgrade. By default, the `--platform-version` argument is set to the current version of LUNA PLATFORM. To downgrade, you must specify the appropriate version of LUNA PLATFORM.

Important: When preparing your environment for an upgrade or downgrade, it is important to carefully plan the impact on running services that use databases. See the “Recommendations services behavior during environment preparation” section of the Storages utility manual for details.

The following settings must be set according to the user logic:

- `storages/files/platform_settings.json` - [Dump file](#) to override the default settings using the `load_dump` command.
- `storages/files/storages_config.conf` - [Storages settings](#), allowing the utility to go to the database, S3, etc.
- `storages/overlays/<command_name>` - Storages positional arguments.

Environment preparation is performed with the Kustomize tool without using Helm. The `storages/base/base.yaml` file defines a template on which layers from the `storages/overlays` directory are overlaid. There are 4 layers available - `check`, `list`, `load_dump`, `prepare`, each corresponding to a specific command of the positional argument `luna_prepare`.

To load an image from a VisionLabs registry, you must also fill in the `imagePullSecrets` parameter in the `storages/base/base.yaml` manifest (see “[Create Docker Registry authentication secret](#)” section).

To prepare the environment, run the following commands:

- 1) Create a Configmap to load the Storages settings and dump file:

```
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) Perform environment preparation using the `prepare` command:

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

```
kubectl logs storages-prepare-xxxxx
```

By default, the environment for all entities is prepared (positional argument `all_entities`). To prepare individual entities, edit the `storages/overlays/prepare/overlay-prepare.yaml` file accordingly.

3) Load the user dump file using the `load_dump` command:

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

Once executed, pods will have a `Completed` status and will not be automatically deleted. To correctly delete the manifest, you must run the command `kubectl delete -k storages/overlays/<storages_command>`.

If necessary, you can use the `list` and `check` commands similar to the above examples.

The manifest set does not include an example of using the `logs` command.

11 Install Helm charts

Make sure you are in the working directory with the Kubernetes files:

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

11.1 Helm chart setting

The supplied Helm charts are not suitable for full operation in the production loop. You need to customize the charts according to your business logic before installing them.

Configure in the `luna-<service-name>/values.yaml` files all the necessary parameters, especially paying attention to:

- `resources` section for specifying resources (e.g. CPU and memory) for the service containers.
- `ingress` section to configure routing of incoming traffic to the service.
- `pullSecrets` parameter in the `image` section to specify the secret to be used when extracting the container image from the registry (see [“Create Docker registry authentication secret”](#) in the “Additional information” section).

Note: It is recommended to configure the `nginx.ingress.kubernetes.io/proxy-body-size` annotation to the API service (or any other service to which image requests are sent) depending on the size requirements of the images being transmitted. The API service Helm chart gives an example of how to use this annotation.

These settings play an important role in ensuring the performance and availability of your application in a productive environment.

11.1.1 GPU setup for Remote SDK and Video Agent

Note: Skip this section if you do not intend to use the GPU.

GPU usage for the Remote SDK and Video Agent service is enabled by passing the appropriate key in the `resources` section of the `values.yaml` file of the corresponding Helm chart.

For example, you can configure access to a single GPU as follows:

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

```
memory: 10Gi
nvidia.com/gpu: 1
```

Note: Also, to enable estimations/detections on the GPU, the necessary settings must be set (see [“GPU settings”](#)). If necessary, you can use the EXTEND_CMD variable to pass the tagged settings.

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

11.2 Start installation of Helm charts

Run the Helm charts installation for the required services using the following commands:

```
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
```

After installing Helm charts, it is recommended that you thoroughly test LUNA PLATFORM in an environment that meets your performance and security requirements.

12 Additional information

This section provides the following additional information:

- [Steps to create a Docker registry-authentication-secret.](#)
- [Steps to prepare the environment and start Backports services.](#)
- [Steps to launch Lambda.](#)
- [Nuances of using GPU in Minikube.](#)
- [VLMatch library compilation example for Oracle.](#)

12.1 Create Docker registry authentication secret

To download images with LUNA PLATFORM services you need to authorize in the Docker registry.

Create a credentials file, such as `vlabs-credentials.json`, containing the login and password:

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

Grant Kubernetes access to the registry with Docker images.

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

If you have previously authorized via the `docker login` command, you can grant Kubernetes access using the following command:

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

The secret can be specified during [Helm chart setting](#).

12.2 Backports

12.2.1 Prepare environment

To prepare the environment, you need to enable the backports profile in the `storages/overlays/prepare/overlay-prepare.yaml` file:

```
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"]
```

Next, you need to prepare the environment similarly to the description in the [“Prepare environment”](#) section.

12.3 Start installation of Helm charts

Run the Helm charts installation for the required services using the following commands:

```
helm install --wait --timeout 10m luna-backport3 ./luna-backport3
helm install --wait --timeout 10m luna-backport4 ./luna-backport4
```

Before starting the UI 4 and UI 3 services, you must perform additional actions in the Helm charts:

- Update the `LUNA_API_URL` parameter for both Helm charts, which is the internal address of Backport 3 and Backport 4 respectively.
- Update the `BASIC_AUTH` parameter for Helm chart UI 4, specifying the authorization data for an account of **user** type in `user@mail.com:password` format encoded in Base64.

It is necessary to create an account of type “user” using the “create account” request to the API service or using the Admin service.

Run the Helm charts installation for the UI 4 and UI 3 services using the following commands:

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

12.4 Launch Lambda

To run Lambda, you need to do some additional steps.

12.4.1 Prepare user Docker registry for Lambda

Note: Skip this section if you are not going to use the Lambda service.

You need to prepare the user registry for storing Lambda images. Transfer the base images and the container assembly tool to your registry using the commands below.

Upload the images from the remote repository to the local image repository:

```
docker pull dockerhub.visionlabs.ru/luna/lpa-lambda-base-fsdk:v.0.8.0
```

```
docker pull dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.8.0
```

Upload the used container build tool image:

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

Add new names to the images by replacing `new-registry` with your own. The names of the base images in the custom registry should be the same as in the `dockerhub.visionlabs.ru/luna` registry.

```
docker tag dockerhub.visionlabs.ru/luna/lpa-lambda-base-fsdk:v.0.8.0 new-registry/lpa-lambda-base-fsdk:v.0.8.0
```

```
docker tag dockerhub.visionlabs.ru/luna/lpa-lambda-base:v.0.8.0 new-registry/lpa-lambda-base:v.0.8.0
```

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

Send local images to your remote repository, replacing `new-registry` with your own.

```
docker push new-registry/lpa-lambda-base-fsdk:v.0.8.0
```

```
docker push new-registry/lpa-lambda-base:v.0.8.0
```

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

12.4.2 Configuring access for Lambda

Note: Skip this section if you are not going to use the Lambda service.

For the Lambda service to work properly, access to Kubernetes resources must be properly configured to ensure the security and efficient management of the service. This can be done, for example, by defining roles and role bindings using the Role Based Access Control (RBAC) mechanism.

The example below shows how to configure accesses using RBAC in Kubernetes for the Lambda service:

- Define an object of type `ServiceAccount`, which represents the identifier used by the service to interact with the Kubernetes API server:

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

- Define a `Role` object type that defines a set of permissions for the resources your service will work with:

```
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" ]
```

Here, `services/proxy` means the ability to send requests to the `/lambdas/{lambda_id}/proxy` resource of the Lambda service.

- Define a `RoleBinding` object type that binds a role to the created `ServiceAccount` type, determining which resources and operations are available to the Lambda service:

```
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
```

12.4.3 Start installation of Helm chart for Lambda

Navigate to the directory with the Helm charts.

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

Run the Helm charts installation for the required services using the following commands:

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

12.5 Use GPU in Minikube

Minikube is a tool for locally installing and managing a Kubernetes cluster. It is used by developers and testers to build and test applications in a local environment before deploying them to larger Kubernetes clusters.

The use of GPUs in Minikube is only supported from version 1.32.

Each LUNA PLATFORM service that supports GPU running automatically creates GPU processes, regardless of which resources (CPU or GPU) are installed. If more than one GPU service is running, the GPU resources must be shared between them to avoid possible errors caused by video card access conflicts.

See [official NVIDIA documentation](#) for more information about GPU resource sharing.

To isolate services from the GPU and prevent them from creating additional processes, set the `CUDA_VISIBLE_DEVICES/NVIDIA_VISIBLE_DEVICES` environment variable to `none` for those services that use the GPU and should not be used.

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

12.6 VLMatch library compilation for Oracle

Note: The following instructions describe the installation for Oracle 21c.

All files required to compile a user-defined extension (UDx) into VLMatch can be found in the following directory:

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

To compile the VLMatch UDx function you need to:

- Install the required environment, see. [requirements](#):

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

- Change the SDK_HOME — oracle sdk root variable (default is \$ORACLE_HOME/bin, check that the \$ORACLE_HOME environment variable is set) in the makefile.

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

- Open the directory and run the “make.sh” file.

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

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

```
./make.sh
```

- Define the library and function inside the database (from the database console):

```
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);
```

- Test the function by calling (from the database console):

```
SELECT VLMATCH(HEXTORAW('12345678901234567890123456789012345678901234'),
HEXTORAW('012345678901234567890123456789012345678901234567890123'), 32)
FROM DUAL;
```

The result should be “0.4765625”.

Transfer the generated VLMATCHSource .so file to the Oracle DBMS.