



VisionLabs LUNA PLATFORM 5

Example of deployment in Kubernetes cluster

v.5.67.0

Contents

Introduction	3
1 Unpacking the distribution	4
2 License activation	5
2.1 Actions from License activation manual	5
3 Prepare user Docker registry for Lambda	6
4 Database configuration	7
4.1 Configuring InfluxDB	7
4.2 VLMatch library compilation	8
4.3 Create user and databases	9
4.4 Add VLMatch functions to Faces and Events databases	10
4.5 Install PostGIS for Events database	10
5 Define LUNA PLATFORM settings	11
5.1 HASP license settings	12
5.2 Guardant License Settings	12
5.3 GPU settings	13
6 Install Helm charts	14
6.1 Helm chart setting	14
6.1.1 GPU setup for Remote SDK	14
6.1.2 Configuring access for Lambda	15
6.2 Start installation of Helm charts	16
7 Additional information	18
7.1 Create Docker registry authentication secret	18
7.2 Use GPU in Minikube	18
7.3 VLMatch library compilation for Oracle	19

Introduction

This document describes the procedure for deploying LUNA PLATFORM to Kubernetes using the Helm charts provided.

The administrator must have a Kubernetes cluster deployed and configured to use Helm charts. 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.5.67.0**, where **v.5.67.0** 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 `c LUNA PLATFORM`.

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

Open the distribution folder.

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

Unzip the files.

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

2 License activation

To activate the license, follow these steps:

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

2.1 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.

3 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 Kaniko executor image 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.0.1.14
```

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

```
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.visionlabs.ru/luna` registry.

```
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
```

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

```
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 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 user and databases for services and assign them the necessary permissions.](#)
- [Add VLMatch functions to Faces and Events databases.](#)

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.

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).

4.1 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.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"
```

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

4.2 VLMatch library compilation

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

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

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

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

```
./make.sh
```

Transfer the generated VLMatchSource.so file to the PostgreSQL DBMS.

4.3 Create user and databases

This section provides sample commands for creating a user and databases 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';"
```

Create databases for all services:

```
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;'
```

Assign privileges to the database user.

```
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;'
```

Allow user authorization on databases:

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

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

4.4 Add VLMatch functions to Faces and Events databases

Define the VLMatch function in the Faces and Events databases:

```
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 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.

5 Define LUNA PLATFORM settings

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

- LICENSE_VENDOR — License settings.
- INFLUX_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_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).

The settings can be specified in the `extras/helms/luna-configurator/files/platform_settings.json` dump file, which is automatically loaded into the Configurator database during the installation of the Configurator service Helm chart. 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 `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/luna_v.5.67.0/extras/helms/luna-configurator/files/platform_settings.json
```

The Helm template `luna-configurator/templates/init-configmap.yaml` uses the `Files.Glob` function to find all JSON files in the `luna-configurator/files` folder in the Helm chart of the Configurator service.

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

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

5.1 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.

5.2 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.

5.3 GPU settings

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

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

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

The Remote SDK service does 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” 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 `extras/helms/luna-configurator/files/platform_settings.json` dump file from the delivery set contains only the “LUNA_REMOTE_SDK_RUNTIME_SETTINGS” section, 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 `extras/conf/luna_platform_<version>_dump.json` as an example.

Note that the “LUNA_REMOTE_SDK_RUNTIME_SETTINGS” section in the dump file has 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 service. An example of passing a tagged setting is commented out in the `values.yaml` file for the Remote SDK service.

6 Install Helm charts

Sample Helm charts for each service are located in the `luna_v.5.67.0/extras/helms/` directory.

6.1 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.

Navigate to the charts directory:

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

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.

6.1.1 GPU setup for Remote SDK

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

GPU usage for the Remote SDK 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"
```

6.1.2 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
```

6.2 Start installation of Helm charts

Navigate to the directory with the Helm charts.

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

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-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
```

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 login: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
```

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

7 Additional information

This section provides the following additional information:

- [Steps to create a Docker registry-authentication-secret.](#)
- [Nuances of using GPU in Minikube.](#)
- [VLMATCH library compilation example for Oracle.](#)

7.1 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.txt --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](#).

7.2 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
```

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

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

```
cd /var/lib/luna/luna_v.5.67.0/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('
1234567890123456789012345678901234567890123456789012345678901234'),
HEXTORAW('
0123456789012345678901234567890123456789012345678901234567890123'), 32)
FROM DUAL;
```

The result should be “0.4765625”.

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