Getting Started with the OML4Py 2.1.1 Client Container

Installing, configuring, and using the OML4Py client container to convert pretrained Hugging Face models to ONNX format for Oracle AI Vector Search.

ORACLE

Contents

ORACLE

Introduction

Following the release of Oracle Machine Learning for Python (OML4Py) 2.1.1, a client container is now available that converts pretrained transformer models from Hugging Face to the Open Neural Network Exchange (ONNX) format for AI Vector Search. Previously this meant manually installing Python, installing required third-party packages, and configuring the OML4Py client by hand; the container replaces all of that with a ready-to-use environment where Python and every dependency are preconfigured.

The OML4Py ONNX Pipeline augments pretrained transformer models with the preprocessing (tokenization for text, image preparation for images) and post-processing (pooling, normalization) needed to generate embeddings directly from raw inputs. Embeddings are numeric vectors that capture meaning, letting you search by semantic similarity rather than exact keywords. The OML4Py client container is the producer: you convert a model once, then deploy that same model to either Oracle AI Database 26ai (for use with `VECTOR_EMBEDDING()` in SQL) or the Oracle Private AI Services Container (for embedding generation through a REST API).

This guide covers:

- The software prerequisites for the OML4Py 2.1.1 client container
- How to sign in to Oracle Container Registry and pull the container image
- How to install Podman on Oracle Linux 8, 9, and 10
- How to launch the container with or without an Oracle wallet
- How to convert a pretrained embedding model to ONNX format and load it to the database
- How to copy a converted model out of the container to your local host
- How the converted model can be used with the Oracle Private AI Services Container

Prerequisites

The OML4Py 2.1.1 client container runs on Oracle Linux 8, 9, and 10 (x86_64) and includes Python 3.13.5 along with a preconfigured set of supporting packages including pandas, NumPy, scikit-learn, PyTorch, ONNX Runtime, and Transformers. It is designed to work with OML4Py on both Oracle AI Database 26ai (on-premises and in the cloud) and Oracle Autonomous AI Database 26ai.

A free Oracle account is required to pull the container image from the Oracle Container Registry (OCR). If your host does not have internet access, see the air-gapped installation option in Step 2.

Rootless Podman prerequisites

Rootless Podman requires `/etc/subuid` and `/etc/subgid` files to map user IDs inside the container to unprivileged IDs on the host, ensuring security and isolation. These files must define a range of IDs (usually at least 65,536) for users on the host to create user namespaces without root privileges. Each user must have a unique range in both files (for example, user:100000:65536). If these are not configured, run:

```
sudo usermod --add-subuids 100000-165535 --add-subgids 100000-165535 $USER
```

Then apply the changes:

```
podman system migrate
```

Verify the entries exist:

```
grep $USER /etc/subuid
```

Step 1: Install Podman

The OML4Py client container uses the Podman container runtime. Docker is not supported.

Verify that Podman is installed:

```
$ podman version

Client:       Podman Engine
Version:      4.9.4-rhel
API Version:  4.9.4-rhel
Go Version:   go1.25.7 (Red Hat 1.25.7-1.module+el8.10.0+90804+12f38c29)
Built:        Tue Mar 17 11:33:06 2026
OS/Arch:      linux/amd64
```

If Podman is not installed, install it using:

Oracle Linux 8

```
$ sudo dnf module install -y container-tools:ol8
```

Oracle Linux 9 and 10

```
$ sudo dnf install -y container-tools
```

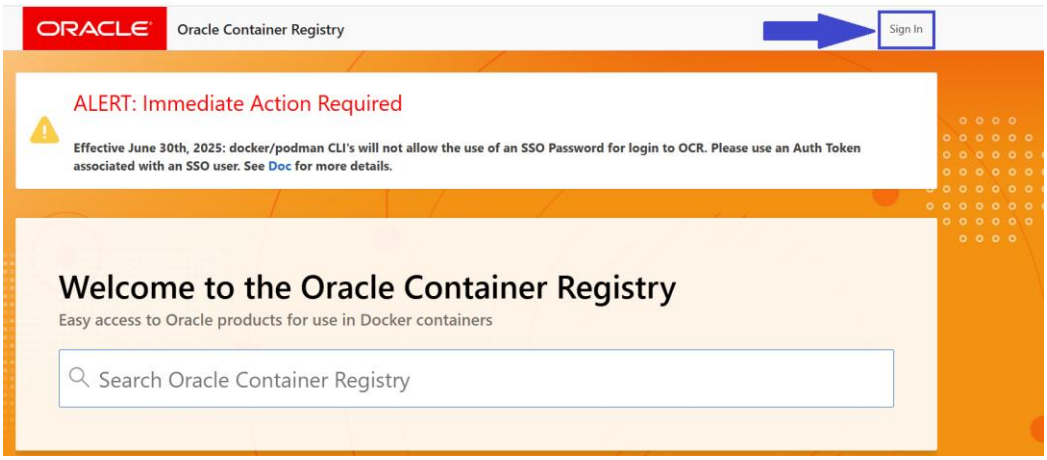
The command resolves and installs the Podman package group along with its dependencies. When it finishes, you will see a list of installed packages ending with “Complete!”.

Step 2: Sign into Oracle Container Registry and pull the image

The OML4Py client container image is available on the Oracle Container Registry (OCR). Follow these steps to sign in and pull the image.

Sign in to OCR

Navigate to Oracle Container Registry and select Sign In. Authenticate using your Oracle SSO username and password. If this is your first time using OCR, you will be prompted to complete a one-time registration that enables your SSO username for OCR access.

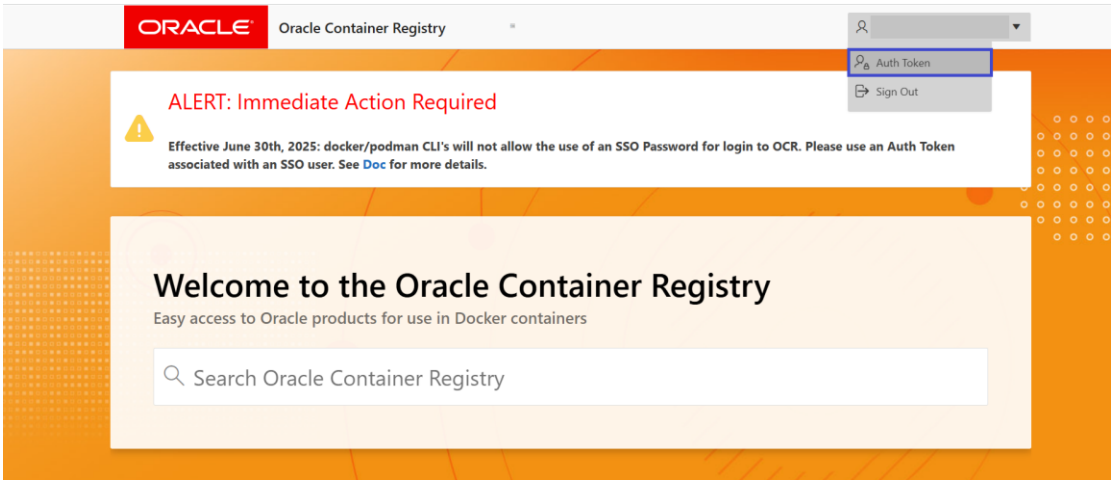


Accept the license agreement

When prompted, review and accept the license agreement for the OML4Py client container. This step is required once per user account.

Generate an OCR authentication token

An authentication token is required as the password when logging in to OCR with Podman. To generate one, click your profile name at the top right of the OCR page and select Auth Token from the dropdown. From the Auth Token page, click Generate Secret Key and copy the token immediately.



Note: Save the authentication token immediately after it is generated. It is displayed only once. If you lose it, you must delete the existing token and generate a new one.

Log in to OCR from Podman and pull the image

```
podman login container-registry.oracle.com
```

When prompted, enter your Oracle SSO username and the authentication token as the password. Then pull the OML4Py client image:

```
$ podman pull container-registry.oracle.com/database/oml4py-client:2.1.1.0.0
Trying to pull container-registry.oracle.com/database/oml4py-client:2.1.1.0.0...
Getting image source signatures
Copying blob 3d99478b45ad done   |
Copying blob 5e861cb8ee23 done   |
Copying config 03f866a6cb done   |
Writing manifest to image destination
03f866a6cb75acad97fe08dc096083954708ad84290059947b92835e84c1e054
```

Verify the image was pulled successfully:

```
$ podman images

REPOSITORY                                                TAG         IMAGE ID      CREATED       SIZE

container-registry.oracle.com/database/oml4py-client  2.1.1.0.0   03f866a6cb75  3 weeks ago  5.74 GB
localhost/oml4py-client                                latest      03f866a6cb75  3 weeks ago  5.74 GB
```

Confirm that container-registry.oracle.com/database/oml4py-client:2.1.1.0.0 appears in the output.

Air-gapped installation

If your target host does not have internet access, save the image to a tar file on an internet connected machine, transfer it, and load it on the air-gapped host. On the connected host, save the image:

```
podman save -o oml4py-client_2.1.1.0.0.tar container-registry.oracle.com/database/oml4py-client:2.1.1.0.0
```

After copying the tar file to the air-gapped host, load it:

```
podman load -i oml4py-client_2.1.1.0.0.tar
```

Verify the image is available:

```
podman images
```

ORACLE

Step 3: Launch the OML4Py client container

How you launch the container depends on whether you are connecting to the database using an Oracle wallet. Choose the option that matches your environment.

Option A: Run without an Oracle wallet

If you are connecting to the database without a wallet, run:

```
$ podman run --name <container_name> -it localhost/oml4py-client:latest
```

Note: The `--userns=keep-id` flag can be used to ensure the container user UID matches your host user UID. The --user flag is not supported.

The `/u01/oml_home` directory is the designated writable directory inside the container. If you need to write output files from the container to your host, mount a host directory to it:

```
podman run -it --name <container_name> \
  -v <host_output_folder>:/u01/oml_home/:U,Z \
  container-registry.oracle.com/database/oml4py-client:2.1.1.0.0
```

Option B: Run with an Oracle wallet

If your database connection requires a wallet, first upload the wallet to the host and extract it to a dedicated subdirectory. For example:

```
unzip <wallet_name>.zip -d /home/opc/wallet
```

Note: The wallet files must be in their own dedicated subdirectory, for example /home/opc/wallet. Mounting a parent directory such as `/home/opc` directly will cause SELinux relabeling and chown errors:

```
Error: SELinux relabeling of /home/opc is not allowed
Error: chowning host path "/home/opc" is not allowed
```

Update sqlnet.ora in the wallet directory to point to the wallet location inside the container:

```
sed -i.bak 's|?/network/admin|/u01/oml_home/wallet|g' /home/opc/wallet/sqlnet.ora
```

Verify that sqlnet.ora now contains:

```
WALLET_LOCATION =
   (SOURCE =
   (METHOD = file)
   (METHOD_DATA =
   (DIRECTORY="/u01/oml_home/wallet")))
```

Then launch the container with the wallet directory mounted:

```
podman run -it --name <container_name> \
  -v /home/opc/wallet:/u01/oml_home/wallet:U,Z \
  -e TNS_ADMIN=/u01/oml_home/wallet \
  container-registry.oracle.com/database/oml4py-client:2.1.1.0.0
```

Note: If a container with the same name already exists from a previous attempt, the command will fail with Error: the container name "oml4py_container" is already in use. Remove the old container first with `podman rm oml4py_container`, or add the `--replace` flag to your `podman run` command to handle this in one step.

`podman run` creates a new container each time. To restart an existing container in a subsequent session, use:

```
podman start -ai <container_name>
```

Step 4: Convert a pretrained model to ONNX and load it to the database

Once inside the container, start a Python session and use the OML4Py ONNX utilities to convert a preconfigured embedding model.

```
$ python3

>>> import oml
>>> from oml.utils import ONNXPipeline, ONNXPipelineConfig

# Display available preconfigured models
>>> ONNXPipelineConfig.show_preconfigured()
[
    'sentence-transformers/all-mpnet-base-v2',
    'sentence-transformers/all-MiniLM-L6-v2',
    'sentence-transformers/multi-qa-MiniLM-L6-cos-v1',
    'sentence-transformers/distiluse-base-multilingual-cased-v2',
    'sentence-transformers/all-MiniLM-L12-v2',
    'BAAI/bge-small-en-v1.5',
    'BAAI/bge-base-en-v1.5',
    'taylorAI/bge-micro-v2',
    'intfloat/e5-small-v2',
    'intfloat/e5-base-v2',
    'thenlper/gte-base',
    'thenlper/gte-small',
    'TaylorAI/gte-tiny',
    'sentence-transformers/paraphrase-multilingual-mpnet-base-v2',
    'intfloat/multilingual-e5-base',
    'intfloat/multilingual-e5-small',
    'sentence-transformers/stsb-xlm-r-multilingual',
    'Snowflake/snowflake-arctic-embed-xs',
    'Snowflake/snowflake-arctic-embed-s',
    'Snowflake/snowflake-arctic-embed-m',
    'mixedbread-ai/mxbai-embed-large-v1',
    'openai/clip-vit-large-patch14',
    'google/vit-base-patch16-224',
    'microsoft/resnet-18',
    'microsoft/resnet-50',
    'WinKawaks/vit-tiny-patch16-224',
    'Falconsai/nsfw_image_detection',
    'WinKawaks/vit-small-patch16-224',
    'nateraw/vit-age-classifier',
    'rizvandwiki/gender-classification',
    'AdamCodd/vit-base-nsfw-detector',
    'trpakov/vit-face-expression',
    'BAAI/bge-reranker-base',
    'BAAI/bge-large-en-v1.5',
    'ibm-granite/granite-embedding-278m-multilingual',
    'thenlper/gte-large',
```

```
    'Snowflake/snowflake-arctic-embed-l',
    'WhereIsAI/UAE-Large-V1'
]

# Create a pipeline for a selected model
>>> pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2")
```

Load the model to the database

Connect to the database and load the converted model directly. Using a username and password:

```
oml.connect(user="<user>", password="<password>", dsn="<service_name>")
pipeline.export2db("<YOUR_MODEL_NAME>")
```

Or, if you launched the container with a wallet located at /home/opc/wallet on the host, connect using the TNS alias from your wallet's tnsnames.ora, no username or password needed:

```
oml.connect(user="", password="", dsn="<tns_alias>")
pipeline.export2db("<YOUR_MODEL_NAME>")
```

The model is now stored in the database and available for use with VECTOR_EMBEDDING() in SQL. Confirm the model is loaded to the database and produces vectors:

```
$ sqlplus <user>/<password>@<service_name>;

SQL> SELECT MODEL_NAME, ALGORITHM, MINING_FUNCTION FROM USER_MINING_MODELS WHERE
MODEL_NAME='<YOUR_MODEL_NAME>';

MODEL_NAME
------------------------------------------------------------
ALGORITHM                      MINING_FUNCTION
------------------------------ ------------------------------
<YOUR_MODEL_NAME>
ONNX                           EMBEDDING

SQL> SELECT VECTOR_EMBEDDING(<YOUR_MODEL_NAME> USING 'May the vectors be with you' as DATA) AS
embedding;

EMBEDDING

---------------------------------------------------------------------------
[-1.60649009E-002,-4.39648442E-002,-3.46924202E-003,-1.17296912E-002, …
```

Export to a local file

If you prefer to export the model, use export2file instead:

```
>>> pipeline.export2file("<YOUR_MODEL_NAME>")
```

This writes the model to the current working directory inside the container (`/u01/oml_home` by default) - a single YOUR_MODEL_NAME.onnx file, or a zip archive if the model uses external data (models over 1 GB automatically, or smaller models configured with use_external_data = True). You can then load the model into the database using `DBMS_VECTOR.LOAD_ONNX_MODEL`, or copy it to the Oracle Private AI Services Container for embedding generation through its REST API.

Unlike `export2db`, which loaded the model directly, `DBMS_VECTOR.LOAD_ONNX_MODEL` loads it from a database directory object - so first move the ONNX file somewhere the database server can read. As SYS, create the directory and grant your OML user access, then load the model:

```
SQL> CREATE DIRECTORY ONNX_IMPORT AS '/u01/oml_home';
SQL> GRANT READ ON DIRECTORY ONNX_IMPORT TO YOUR_USER;
SQL> GRANT CREATE MINING MODEL TO YOUR_USER;

BEGIN
  DBMS_VECTOR.LOAD_ONNX_MODEL(
    directory  => 'ONNX_IMPORT',
    file_name  => 'YOUR_MODEL_NAME.onnx',
    model_name => 'YOUR_MODEL_NAME');
END;
/
```

Confirm the load with the same `USER_MINING_MODELS` and `VECTOR_EMBEDDING()` queries shown above.

Step 5: Copy the ONNX model file out of the container

Use `podman cp` to copy the model file from the container to your local host. Run the following command from the local host:

```
podman cp <container_name>:<path_to_file> <host_output_folder>
```

For example:

```
podman cp oml4py-container:/u01/oml_home/YOUR_MODEL_NAME.onnx /home/opc/models/
```

ORACLE

Using the converted model with the Private AI Services Container

The same augmented ONNX model that gets loaded into Oracle AI Database 26ai can also be used by the Oracle Private AI Services Container to generate vector embeddings outside the database through an OpenAI-compatible REST API, without any internet access required. The same converted model has two deployment targets:

Oracle AI Database 26ai, for in-database embedding generation with `VECTOR_EMBEDDING()` in SQL. Load the model directly with `pipeline.export2db("YOUR_MODEL_NAME")` from the OML4Py client, or use `pipeline.export2file("YOUR_MODEL_NAME")` and load the resulting ONNX file from any database session with DBMS_VECTOR.LOAD_ONNX_MODEL.

The Private AI Services Container, for embedding generation through the container's `/v1/embeddings` REST endpoint. Use `pipeline.export2file("YOUR_MODEL_NAME")` and copy the resulting ONNX file to the container's model directory on the host machine.

Troubleshooting

subuid / subgid not configured

If rootless Podman fails to start a container, confirm your user has entries in `/etc/subuid` and `/etc/subgid` with `grep $USER /etc/subuid`, and run the `usermod` command in the Prerequisites section if they are missing.

Authentication token rejected at podman login

The OCR password is the generated auth token, not your Oracle SSO password. If login fails, regenerate the token (it is shown only once) and ensure you accepted the container license agreement.

SELinux relabeling or chown error on container start

Mount the wallet from its own dedicated subdirectory (for example `/home/opc/wallet`), not a parent directory such as `/home/opc`.

Container name already in use

Remove the existing container with `podman rm <container_name>`, or add `--replace` to your `podman run` command.

ORACLE

sed -i fails with couldn't open temporary file, Permission denied

Check the ownership of the wallet directory with `ls -ld <path_to_wallet>`. If it is owned by a high-numbered ID such as 100999 rather than your host user, the wallet has already been mounted into a container with the `:U` flag, which rechowns the host files to the mapped container UID. The `sed -i` command cannot write its temporary file because your host user no longer owns the directory. Reclaim ownership with `sudo chown -R <user>:<user> <path_to_wallet>`, then re-run the sed command.

ORACLE

blogs.oracle.com facebook.com/oracle twitter.com/oracle