Importing ONNX Embedding Models
Oracle AI Database 26ai using OML4Py Client in OCI VM

This guide is intended for experimentation and demo purposes. Please refer to the referenced documentation below for best practises and latest information.
The OML4Py Client can also be installed on-premises.

Description

Oracle AI Database 26ai includes an ONNX runtime engine for running embedding models directly inside the database. This section covers the process of setting up the OML4Py Client in an OCI and importing existing pretrained embedding models into Oracle AI Database, including utilising pre-configured models and external models from HuggingFace (subject to compatibility) by converting those models into the ONNX format if they are not already converted.

The OML4Py Client provides the Python package to:

- Download 30+ pre-configured embedding models along with pre-trained embedding models from Hugging Face into your Oracle AI Database.
- Augments the model with pre-processing and post-processing steps and creates a new ONNX model
- Validates the augmented ONNX model
- Loads into the database as a data mining model or optionally exports to a file

If you would like to understand the supported embedding models that can be loaded into Oracle's AI Database across Text, Image and Multi-Modal Embedding, please reference the below documentation.

- Text Embedding Models - https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/onnx-pipeline-models-text-embedding.html
- Image Embedding Models - https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/onnx-pipeline-models-image-embedding.html
- Multi-Modal Embedding Models - https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/onnx-pipeline-models-multi-modal-embedding.html

Assumed Pre-Requisites

- You have the ability to create Networking and Compute resources in OCI to host the OML4Py Client.
- You have already provisioned an Oracle Autonomous AI Database 26ai
    - Downloaded the Wallet File
    - Have a DB User Created with necessary grants
        - `GRANT CREATE MINING MODEL TO <USER>;`

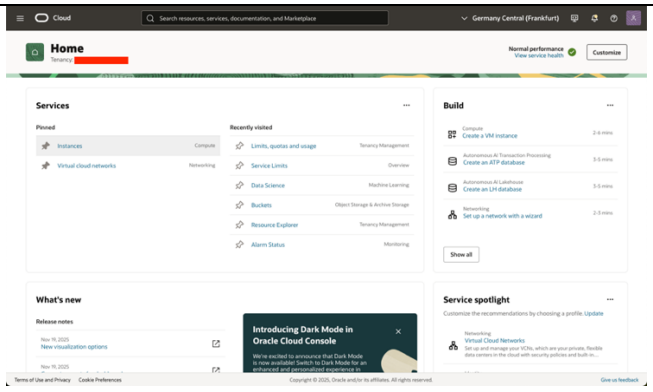
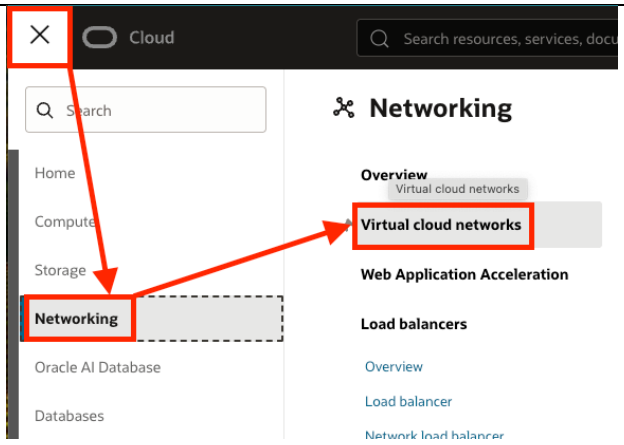
Please Note

- This feature will only work on OML4Py 2.1 client. It is not supported on the OML4Py server or older versions of the OML4Py client.
- In-database embedding models must include tokenization and post-processing. Providing only the core ONNX model to DBMS_VECTOR.LOAD_ONNX_MODEL is insufficient because you need to handle tokenization externally, pass tensors into the SQL operator, and convert output tensors into vectors.
- There are limitations on the size and type of embedding models that can be loaded into the Oracle AI Database. Please refer to the documentation mentioned above which will provide links to the types of models that can be used along with their restrictions.

Referenced Documentation

- Overview - https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/import-pretrained-models-onnx-format-vector-generation-database.html
- End to End Installation - https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/convert-pretrained-models-onnx-model-end-end-instructions.html

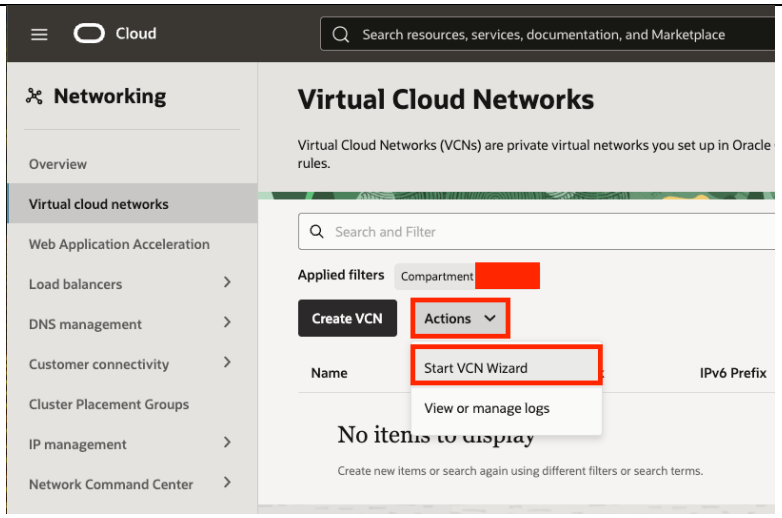
OML4Py 2.1 Client Installation in OCI VM

Step	Screenshot
Login to your OCI Console. https://www.cloud.oracle.com	
First, we will create the VCN for our Compute Instance to live within. Navigate to ***OCI Menu > Networking > Virtual Cloud Networks***	

We will create a basic VCN and Subnet setup with Internet access using the Wizard.

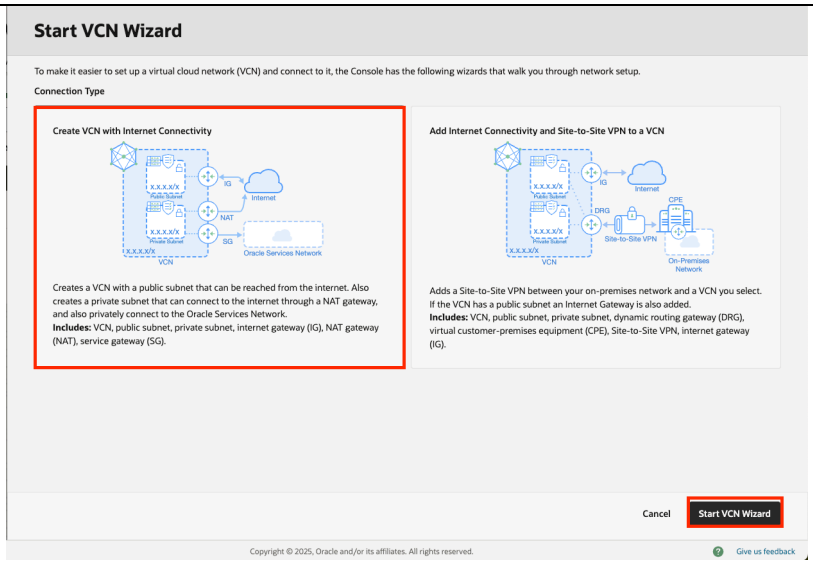
Select ***Actions > Start VCN Wizard***

If you prefer to create your own custom Network setup from scratch, select ***Create VCN***.



Select ***Create VCN with Internet Connectivity***.

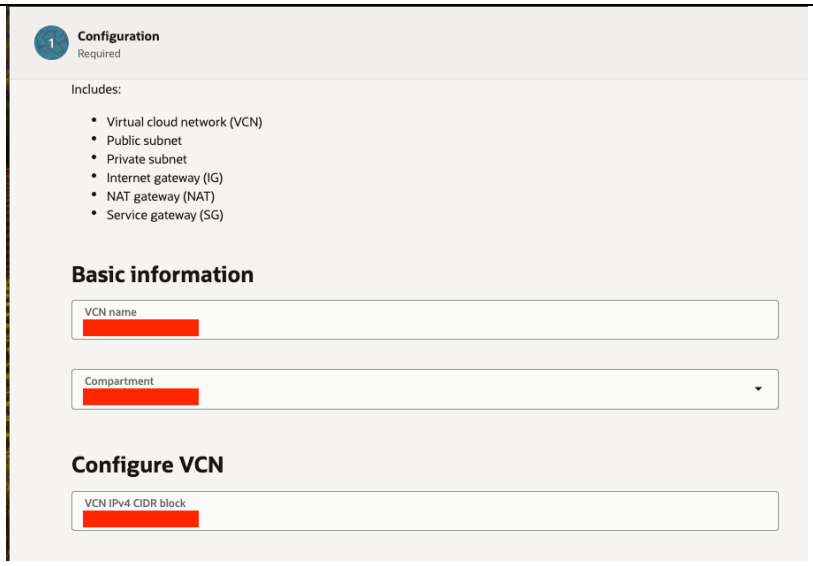
Select ***Start VCN Wizard***.

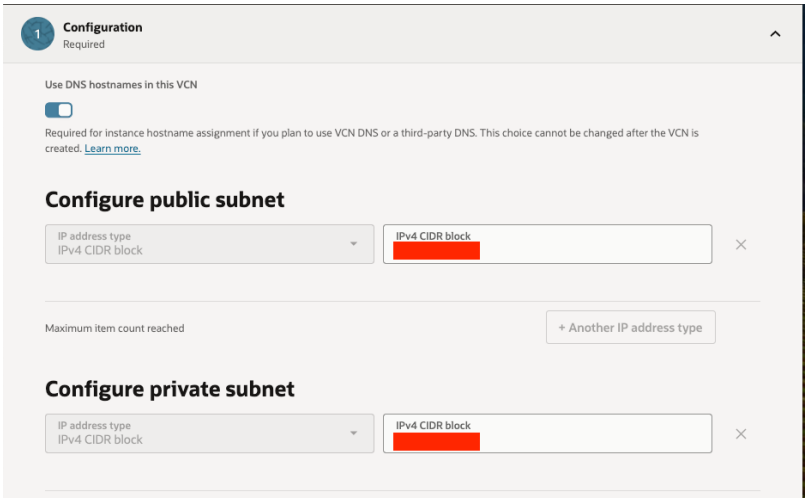
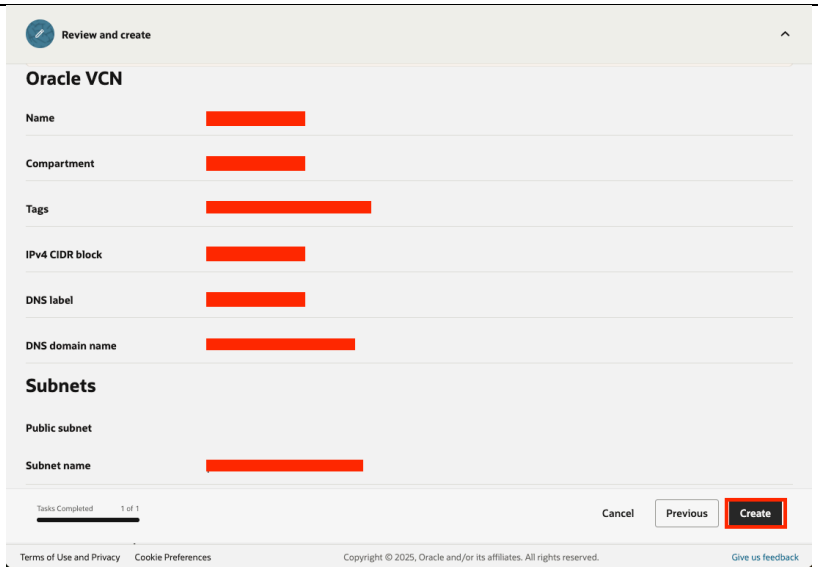
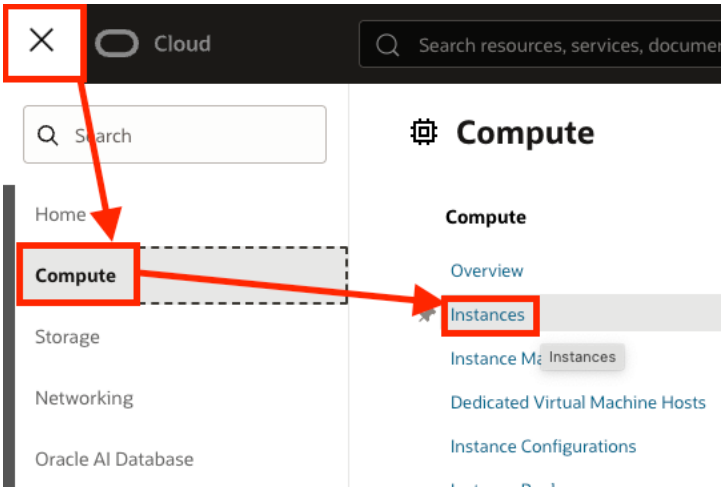


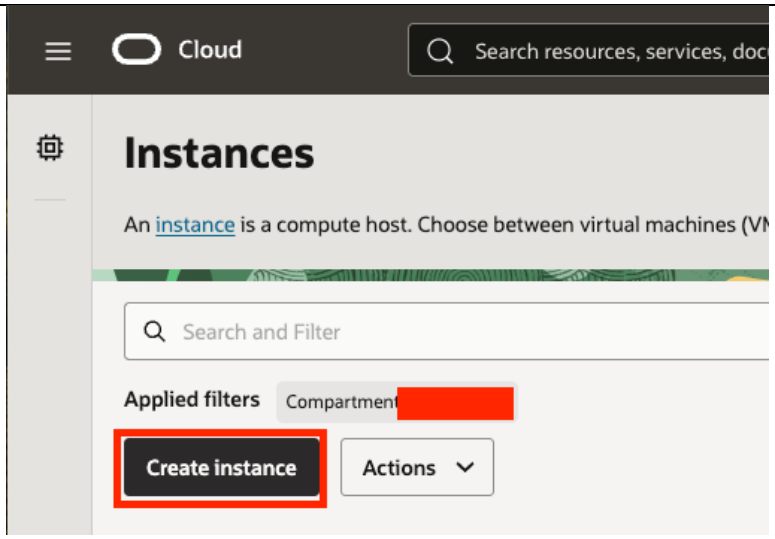
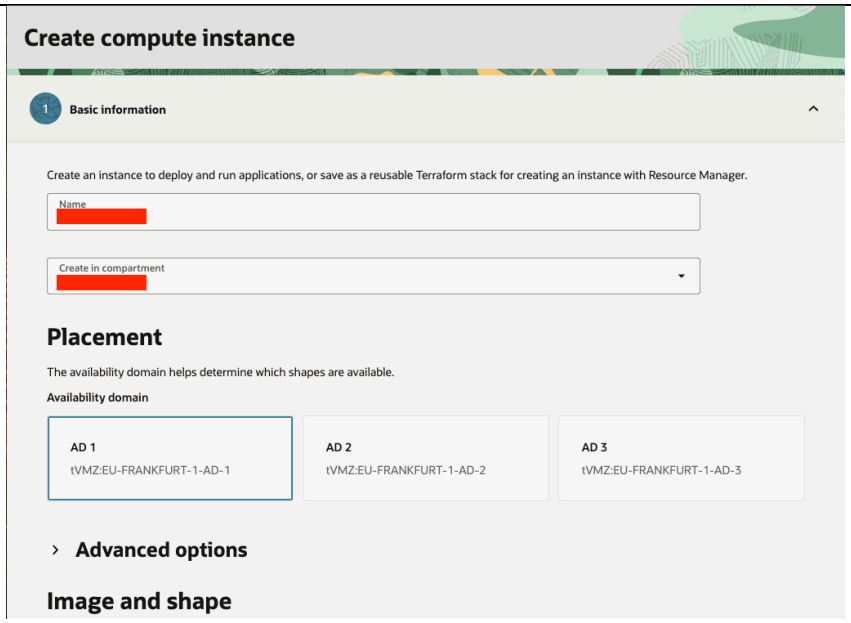
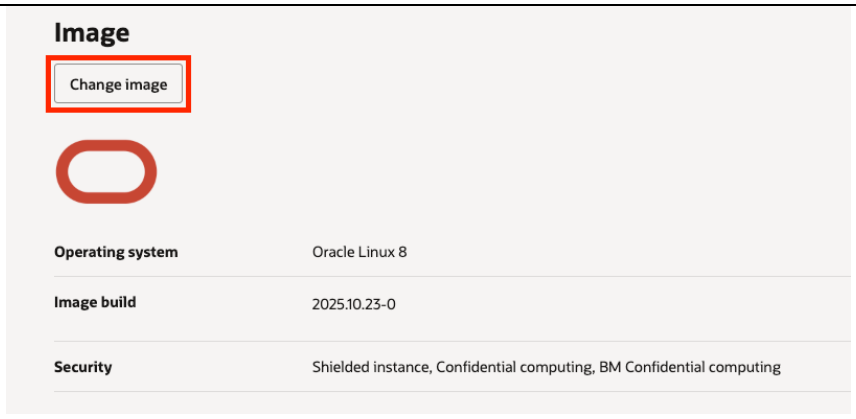
Enter ***VCN Name.***

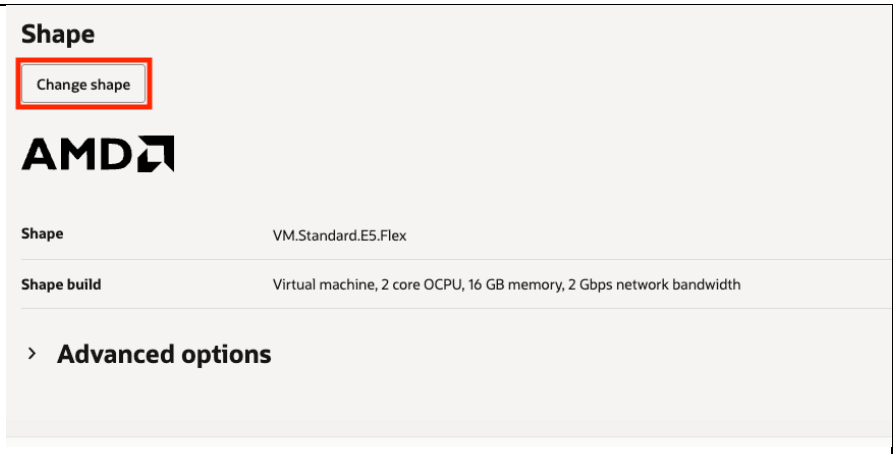
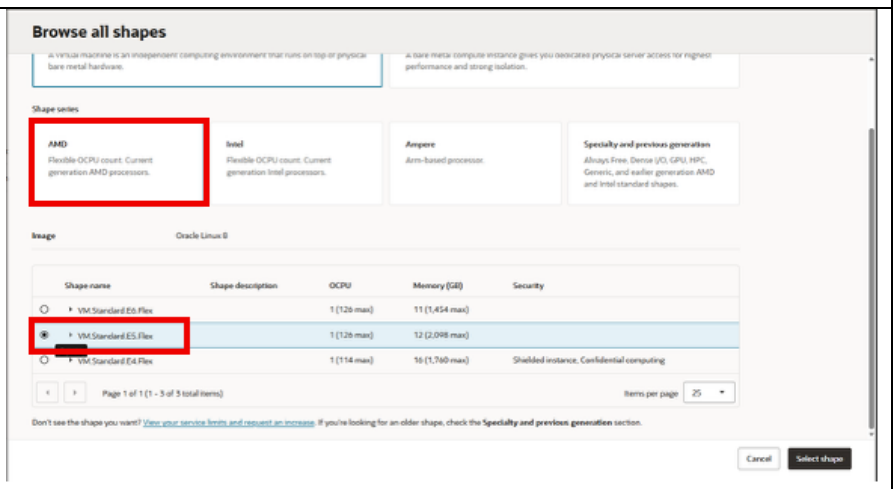
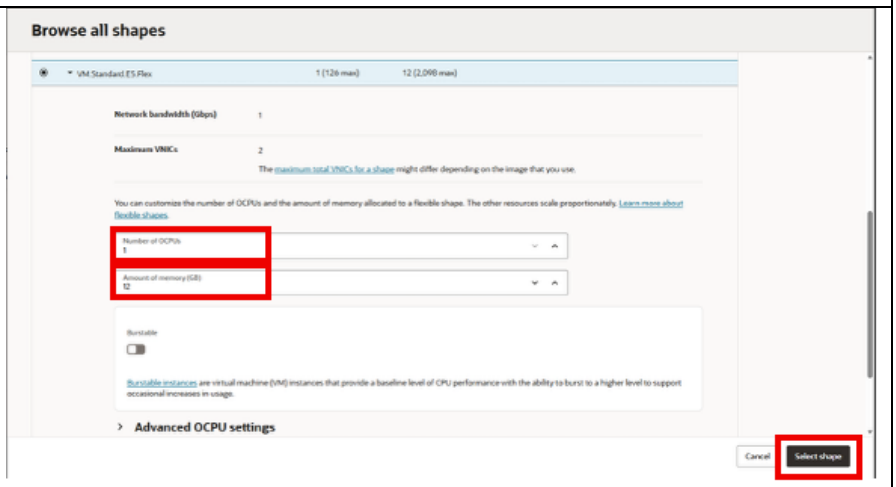
Select ***Compartment*** for your VCN to live in.

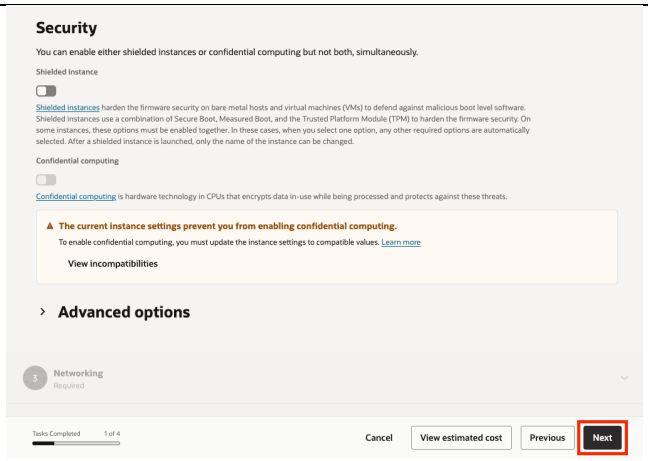
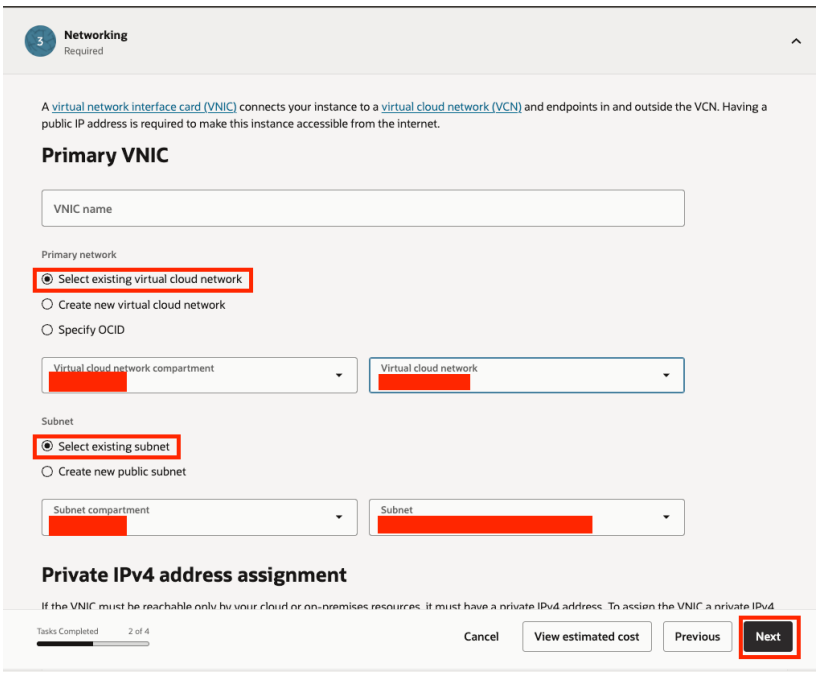
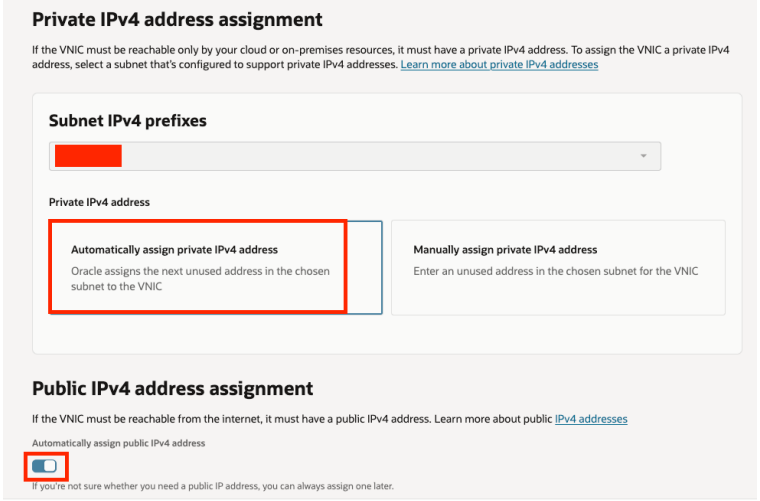
I used the ***default VCN CIDR Block Range***. Feel free to modify to meet the networking requirements of your organisation.

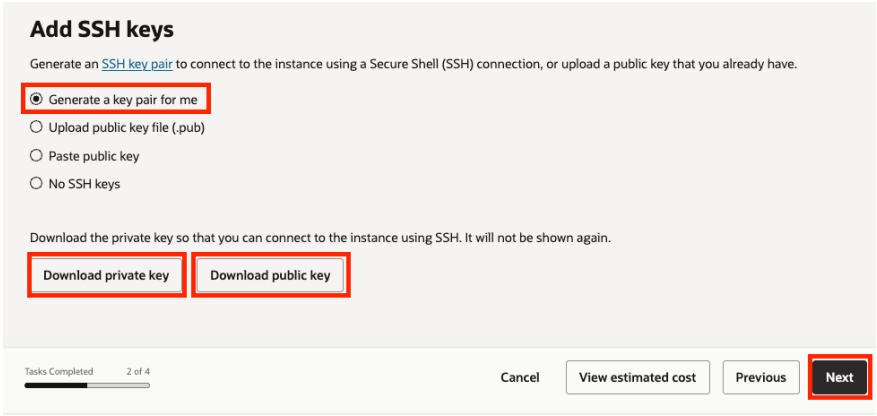
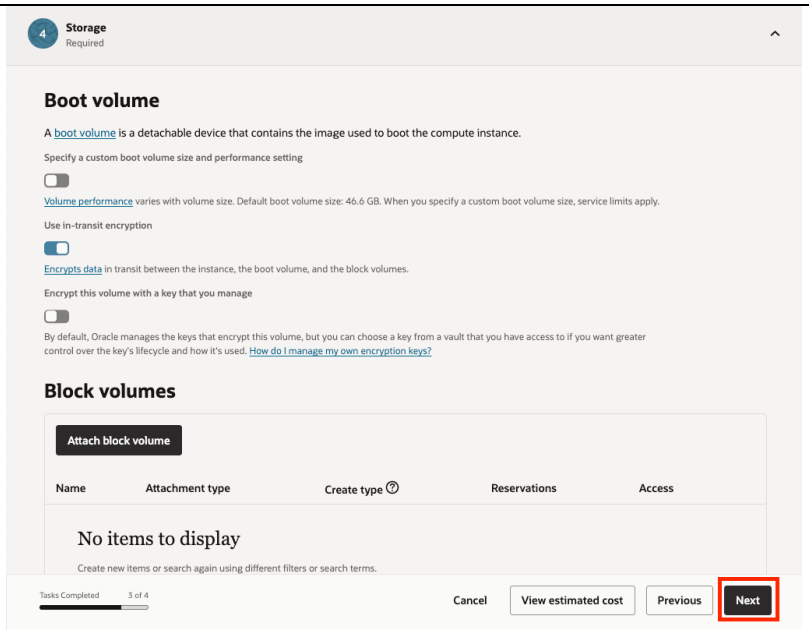
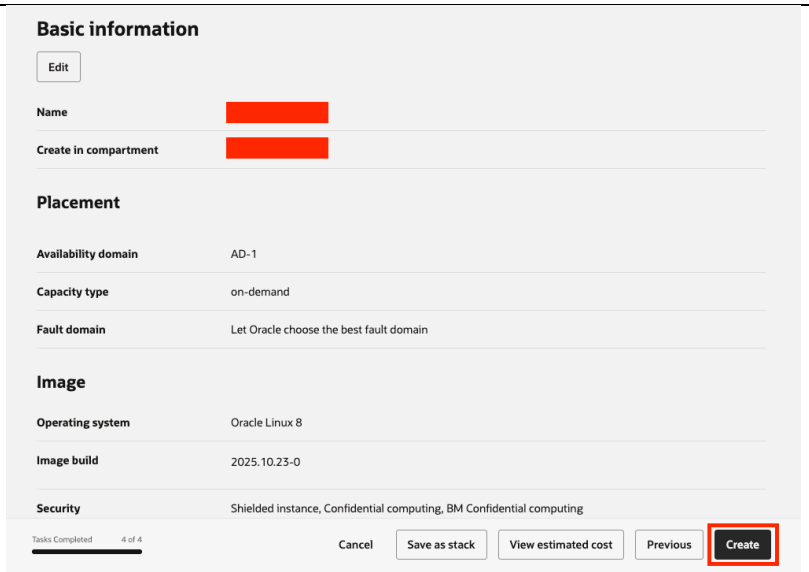


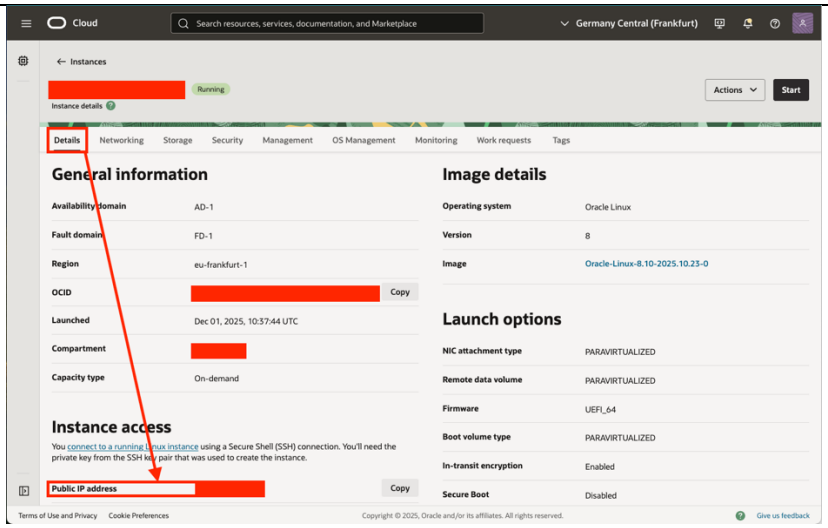
I accepted the ***default IP CIDR Block Ranges for the Public and Private Subnets*** that will be created. Feel free to modify to meet the networking requirements of your organisation. Click ***Next***.	
Review the configurations for your VCN setup. Click ***Create***.	
Now we will create the Compute Instance to host the OML4Py Client which is responsible for downloading embedding models to be used in database. Navigate to ***OCI Menu > Compute > Instances.***	

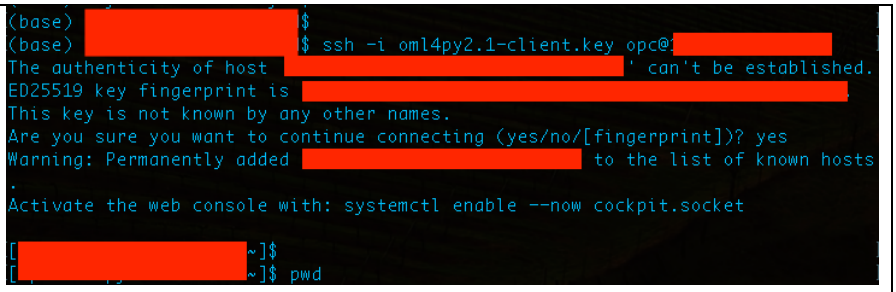
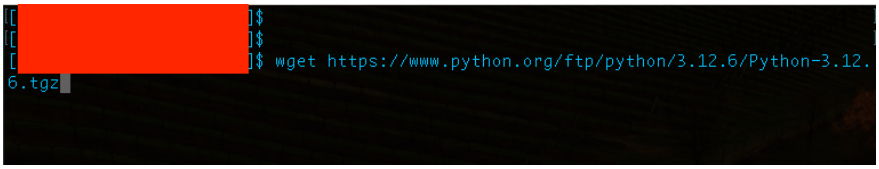
Click ***Create Instance***.	
Enter a ***Name*** for your Instance. Select the ***Compartment*** for the Instance to live in. I kept the ***Availability Domain as Default***.	
Click on ***Change Image*** to select the OS of the Instance. I selected ***Oracle Linux 8*** which supports the OML4Py Client.	

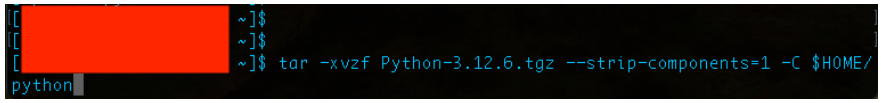
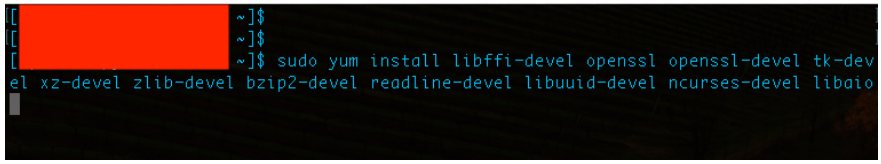
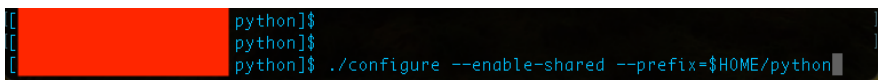
Click on ***Change Shape***. I will select a shape of: • ***VM.Standard.E5.Flex*** • ***2 OCPUs and 16GB Memory***	
Select the Shape Series as ***AMD.*** Click on the drop down next to the ***VM.Standard.E5.Flex*** Shape.	
Set your number of OCPUs and Memory. I selected a Compute Shape of: • ***2 OCPUs and 16GB Memory*** Click ***Select Shape.*** Click ***Next.***	

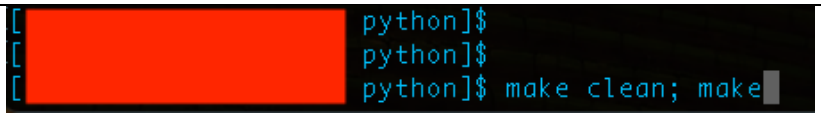
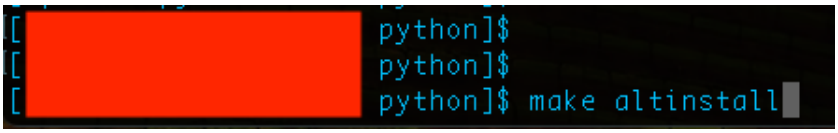
I left all the Security options as ***Default***. Click ***Next***.	
Select Primary Network as ***Select existing virtual cloud network***. Select the ***VCN we created earlier in our Compartment***. Select Subnet as ***Select existing subnet***. Select the ***Subnet we created earlier in our Compartment.*** Click ***Next***.	
Select ***Automatically assign private IPv4 address***. Ensure the toggle for ***Automatically assign public IPv4 address*** is selected.	

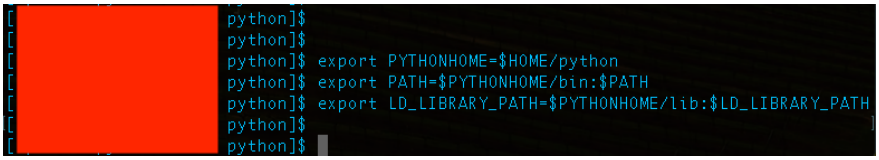
We will generate new SSH Keys to login to our instance. Select ***Generate a key pair for me***. ***Download the Private and Public Key.*** Click ***Next***.	
I have left the Boot Volume and Block Volume settings as ***defaults***. Click ***Next***.	
View the summary of the instance creation. Click ***Create***. This will take a couple minutes to provision.	

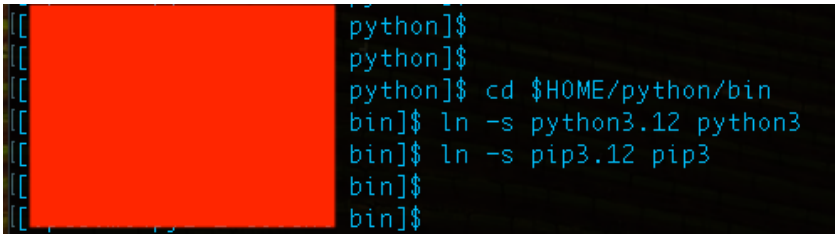
Once provisioned, navigate to the ***Details Tab*** and locate your ***Public IP Address***. Make a note of it, we will need this soon.	
If on a Mac/Linux machine open a Terminal, navigate to your Downloads folder and change the permission on your Private Key file we downloaded earlier. ***chmod 600 private-key-file.key***	 ***If using a Windows:*** 1:- Right-click on the target file and select `properties` then select `Security` Tab 2:- Click `Advanced` and then make sure inheritance is disabled. 3:- Click `apply` and then click `Edit` in the security menu 4:- `Remove` all users except yours/admin user, which should have full control. Admin account should have all checkboxes checked on `Allow` column except special permission. 5:- Click `Apply` and then click OK.

On a Mac/Linux Machine, within the Terminal, SSH into our Compute Instance. ***ssh -i <private-key-file.key> opc@<public-ip-address>*** The above assumes you are in the same directory as your private key file, otherwise specify the full path. If prompted to continue, type ***yes***.	 If using Windows, you can use your favourite SSH tool such as PuTTY or Terminus to SSH into your compute instance in the same way. Here is a YouTube video on how use PuTTY on Windows - https://www.youtube.com/watch?v=4jakCV5JYx0
Once logged in, let's download Python 3.12.6. `wget https://www.python.org/ftp/python/3.12.6/Python-3.12.6.tgz`	
Create a new directory to install Python. `mkdir -p $HOME/python`	

Unzip the Python installation into our new directory. `tar -xvzf Python-3.12.6.tgz --strip-components=1 -C $HOME/python`	
Install the additional libraries to the Linux Machine. `sudo yum install libffi-devel openssl openssl-devel tk-devel xz-devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel libaio`	
Navigate into the Python Installation Directory. `cd $HOME/python`	
Configure the installation. `./configure --enable-shared --prefix=$HOME/python`	

Make installation. `make clean; make`	
Make installation. `make altinstall`	

Set environment variables.
 	
`export PYTHONHOME=$HOME/python`
 	
`export PATH=$PYTHONHOME/bin:$PATH`
 	
`export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH`
	

Navigate to the Python Bin directory and create symbolic links.
 	
`cd $HOME/python/bin`
 	
`ln -s python3.12 python3`
 	
`ln -s pip3.12 pip3`
	

We will now set up the Oracle Instant Client so that we can save downloaded embedding models directly to our Autonomous AI Database.

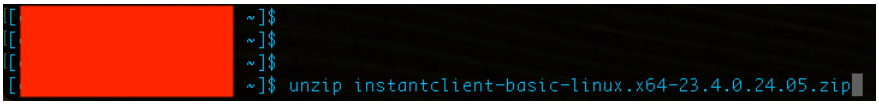
Navigate to home directory and download the Oracle Instant Client.

```
cd $HOME
```

```
wget https://download.oracle.com/otn_software/linux/instantclient/2340000/instantclient-basic-linux.x64-23.4.0.24.05.zip
```

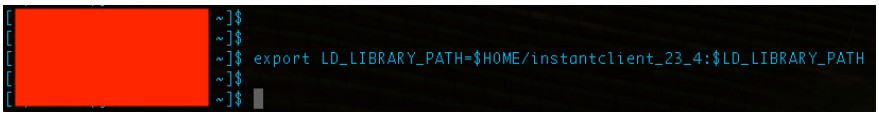
Unzip the Oracle Instant Client Download.

```
unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
```



Set the LD Library Path.

```
export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH
```



To ensure that the environment variables get set every time we start the compute instance, lets add them to the .bashrc file.

```
cd ~
```

```
vi .bashrc
```

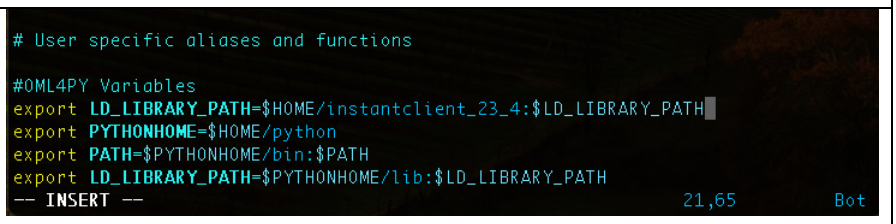
Press ***i*** on your keyboard to insert text.

Use your ***keyboard arrows*** to navigate to the bottom of the file.

Copy and paste the text on the right-hand side, ensuring instant client version is the same →

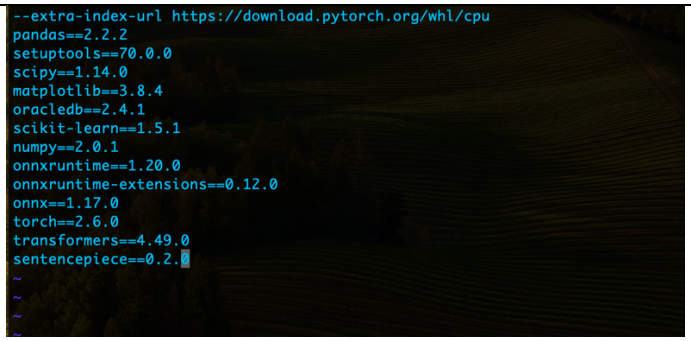
Press the ***'Esc'*** key on your keyboard to exit insert mode.

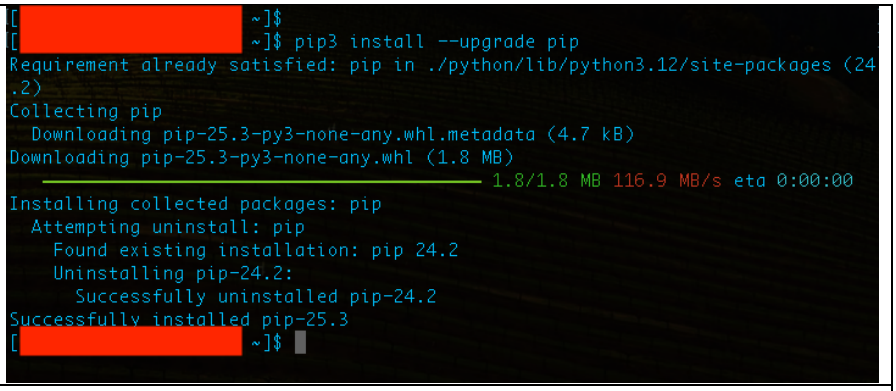
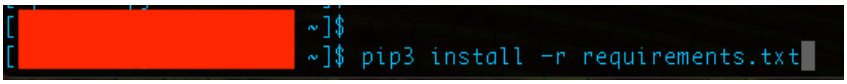
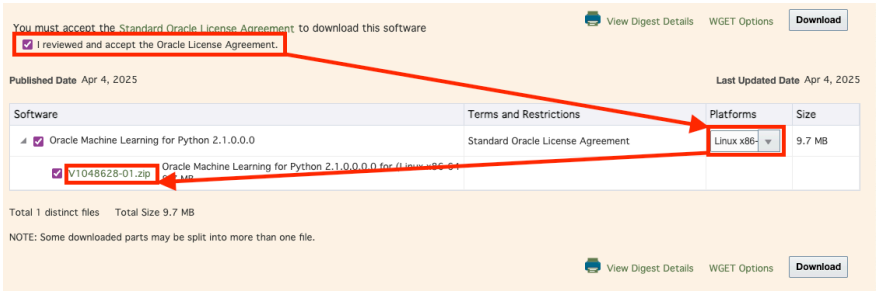
Type the following and press enter: ***:wq***

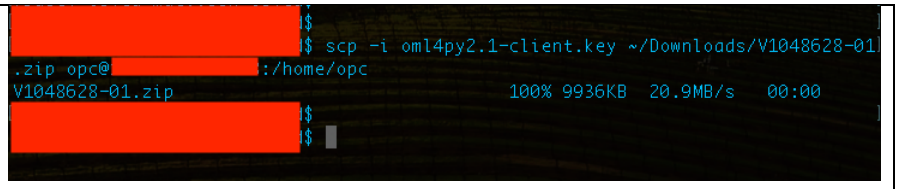
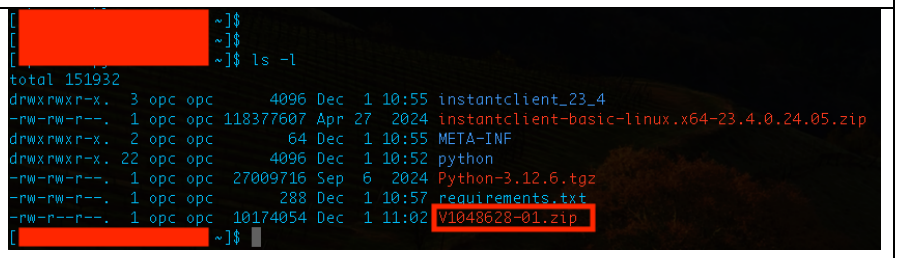
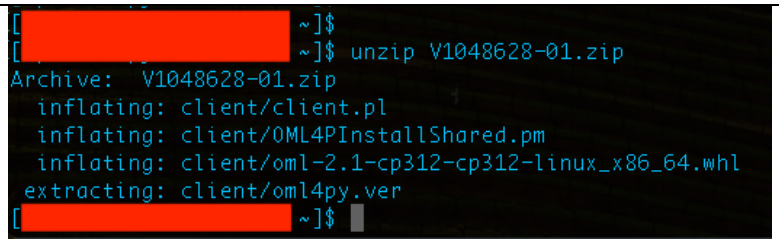
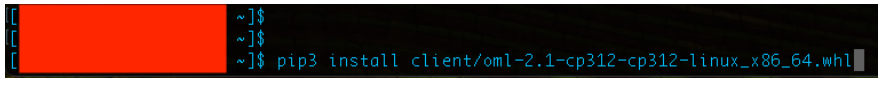



```
# Environment variables for Python

export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH

export PYTHONHOME=$HOME/python

export PATH=$PYTHONHOME/bin:$PATH

export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

Reload the .bashrc file. `source .bashrc` Now every time you start the compute instance, the environment variables will automatically be set.	
Create a Python requirements file for libraries to be installed. `vi requirements.txt`	

Press ***i*** on your keyboard to insert text.
 	
Use your ***keyboard arrows*** to navigate to the bottom of the file.
 	
Copy and paste the text on the right-hand side. →
 	
Press the ***'Esc'*** key on your keyboard to exit insert mode.
 	
Type the following and press enter: ***:wq***
	
	
	`--extra-index-url https://download.pytorch.org/whl/cpu`
	`pandas==2.2.2`
	`setuptools==70.0.0`
	`scipy==1.14.0`
	`matplotlib==3.8.4`
	`oracledb==2.4.1`
	`scikit-learn==1.5.1`
	`numpy==2.0.1`
	`onnxruntime==1.20.0`
	`onnxruntime-extensions==0.12.0`
	`onnx==1.17.0`
	`torch==2.6.0`
	`transformers==4.49.0`
	`sentencepiece==0.2.0`

Upgrade pip. `pip3 install --upgrade pip`	
Install Python requirements file. `pip3 install -r requirements.txt`	
In your local browser download OML4Py 2.1.0. Visit https://www.oracle.com/database/technologies/oml4py-downloads.html and download the Client for OML4Py 2.1.0 (Database 23/26ai).	
Read and accept the Oracle License Agreement. Select the Platform as Linux. Then click on the ***V1048628-01.zip*** to download the ZIP file.	

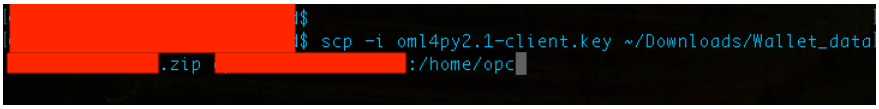
Copy the downloaded .zip file to your compute instance. As I am using a Mac, I can use the scp command in my local Terminal. `scp -i <private-key-file> <location-of-downloaded-zip> opc@<public-ip>:/home/opc`	 If you are using a Windows, you can do this using Windows PowerShell with the same command or download and use a tool such as WinSCP - https://winscp.net/eng/index.php
Back in our compute instance, we can now see the ZIP file copied across within our compute instance. `ls -la`	
Unzip the file. `unzip V1048628-01.zip`	
Install the OML4Py python library. `pip3 install client/oml-2.1-cp312-cp312-linux_x86_64.whl`	

We will now configure Oracle Instant Client to communicate with our Autonomous AI Database so we can save models directly to the DB.

Copy your Autonomous AI Database Wallet file to your compute instance.

As I am using a Mac, I can use the scp command in my local Terminal.

```
scp -i <private-key-file> <location-of-wallet-file> opc@<public-ip>:/home/opc
```

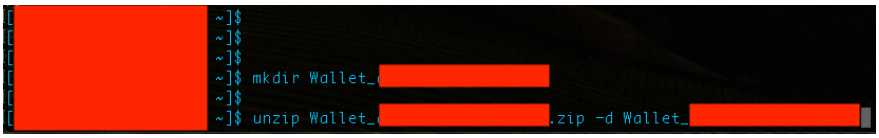


If you are using a Windows, you can do this using Windows PowerShell with the same command or download and use a tool such as WinSCP - https://winscp.net/eng/index.php

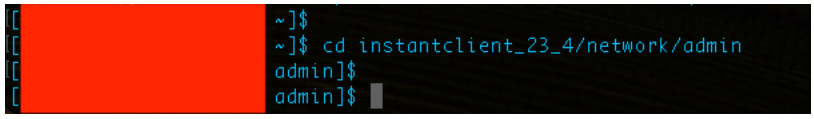
Back in my compute instance I can create a new empty directory to unzip my Wallet into.

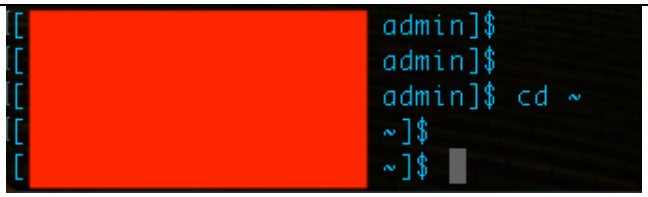
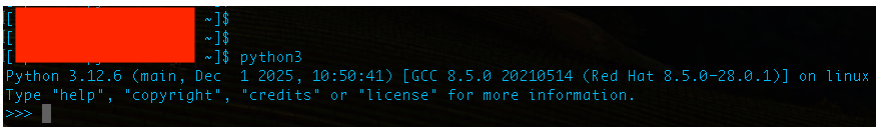
```
mkdir <new-wallet-directory>
```

```
unzip <zipped-wallet> -d <new-wallet-directory>
```



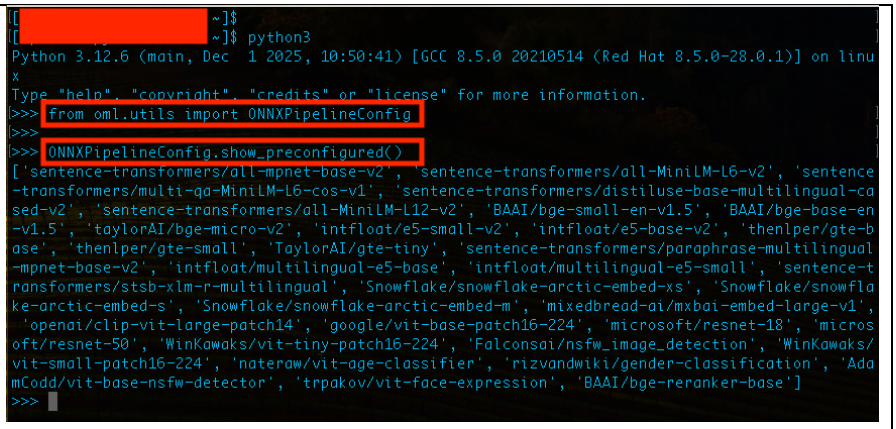
Make a copy of your ***tnsnames.ora*** and ***sqlnet.ora*** files into the Oracle Instant Client network/admin directory.
 	
`cp <new-wallet-directory>/tnsnames.ora instantclient_23_4/network/admin`
 	
`cp <new-wallet-directory>/sqlnet.ora instantclient_23_4/network/admin`
	

Navigate to the Network Admin directory. `cd instantclient_23_4/network/admin`	
Edit the sqlnet.ora file to update your Unzipped Wallet Location. `vi sqlnet.ora`	

Update the Wallet Directory Location. Press ***i*** on your keyboard to insert text. Use your ***keyboard arrows*** to ***DIRECTORY*** parameter and update this to the location of your unzipped Wallet. Press the ***'Esc'*** key on your keyboard to exit insert mode. Type the following and press enter: ***:wq***	
Navigate back to the Home Directory. `cd ~`	
Enter the Python Terminal. `python3`	

Import the ONNXPipeline Libraries and display the pre-configured models.

```
from oml.utils import ONNXPipelineConfig

ONNXPipelineConfig.show_preconfigured()
```



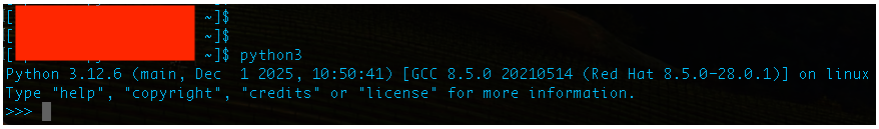
If you are able to list the 30+ preconfigured ONNX Embedding Models, you have verified the OML4Py 2.1.0 Client Installation.

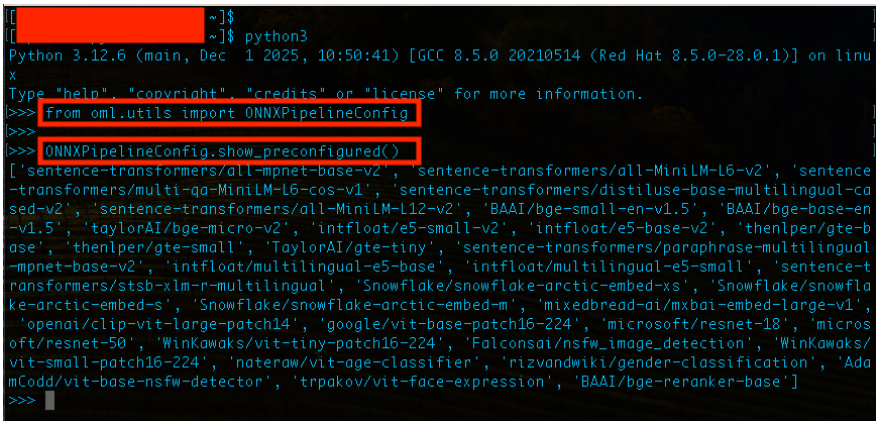
Below I will go through four scenarios for downloading ONNX Embedding Models.

1. Downloading a pre-configured text embedding model locally
2. Loading a pre-configured text embedding model directly to our Autonomous DB
3. Downloading a custom multi-modal embedding model from HuggingFace locally
4. Loading a custom multi-modal embedding model from HuggingFace to our Autonomous DB

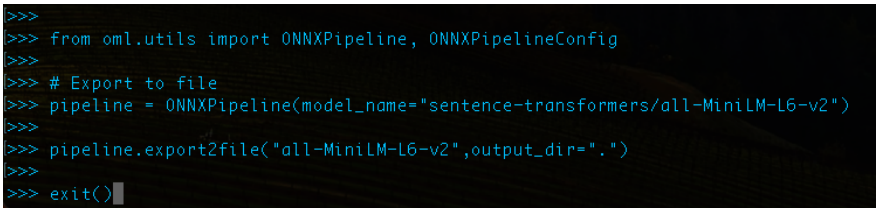
Scenario 1: Downloading a pre-configured text embedding model locally

Referenced Documentation: https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/onnx-pipeline-models-text-embedding.html

Step	Screenshot
Within your OML4Py Client Installation, Open a Python session. `python3`	

First, we can display the pre-configured models.
 	
These are a list of preconfigured ONNX embedding models that have been verified by Oracle which can be utilised as is.
 	
Code snippet below screenshot →
	
	
	`>> from oml.utils import ONNXPipelineConfig`
	
	`>> ONNXPipelineConfig.show_preconfigured()`

We can then save the ***sentence-transformers/all-MiniLM-L6-v2*** model locally.



Code snippet below screenshot →

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

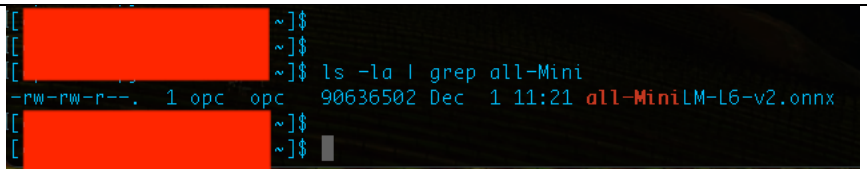
```
>> pipeline = ONNXPipeline(model_name="sentence-
transformers/all-MiniLM-L6-v2")
```

```
>> pipeline.export2file("all-MiniLM-L6-
v2",output_dir=".")
```

```
>> exit()
```

Outside of the Python session, we can then search our local directory for the file.

```
ls -la | grep
all-Mini
```

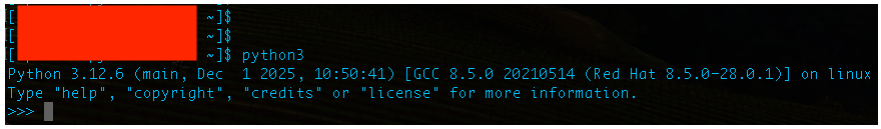


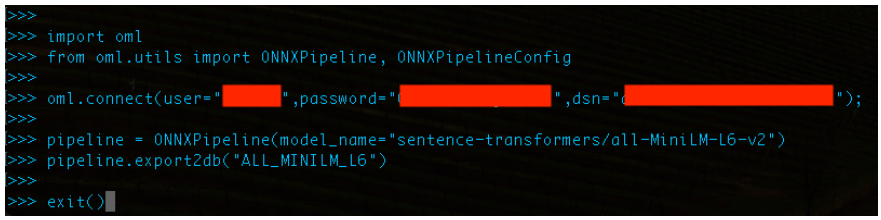
This can now be copied across to the Oracle AI Database via the DB Server or via Object Storage to be loaded in as a Model.

The next scenario will show you how to do that automatically through our OML4Py and Instant Client configuration we did above.

Scenario 2: Loading a pre-configured text embedding model directly to Autonomous Database

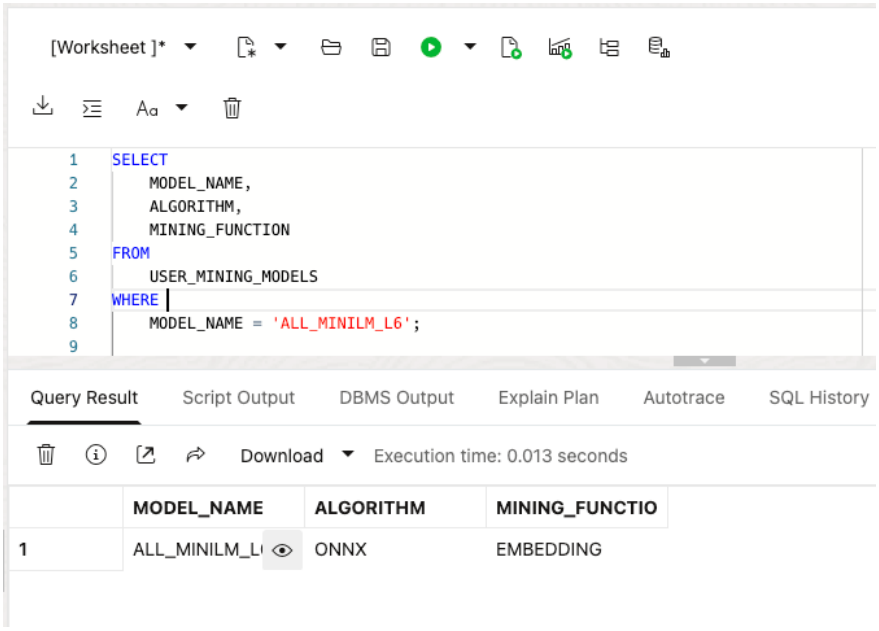
Referenced Documentation: https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/onnx-pipeline-models-text-embedding.html

Step	Screenshot
Within your OML4Py Client Installation, Open a Python session. `python3`	

Import the libraries.
 	
Connect to our Autonomous DB
 	
(Replace you're your connection details, where ***dsn*** is our Autonomous DB service name.)
 	
Define our ONNX pipeline for a pre-configured model
 	
Export to Database.
 	
Code snippet below screenshot →
	
	
	`>> import oml`
	`>> from oml.utils import ONNXPipeline, ONNXPipelineConfig`
	
	`>> oml.connect(user="username", password="password", dsn="dsn");`
	
	`>> pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2")`
	
	`>> pipeline.export2db("ALL_MINILM_L6")`
	
	`>> exit()`

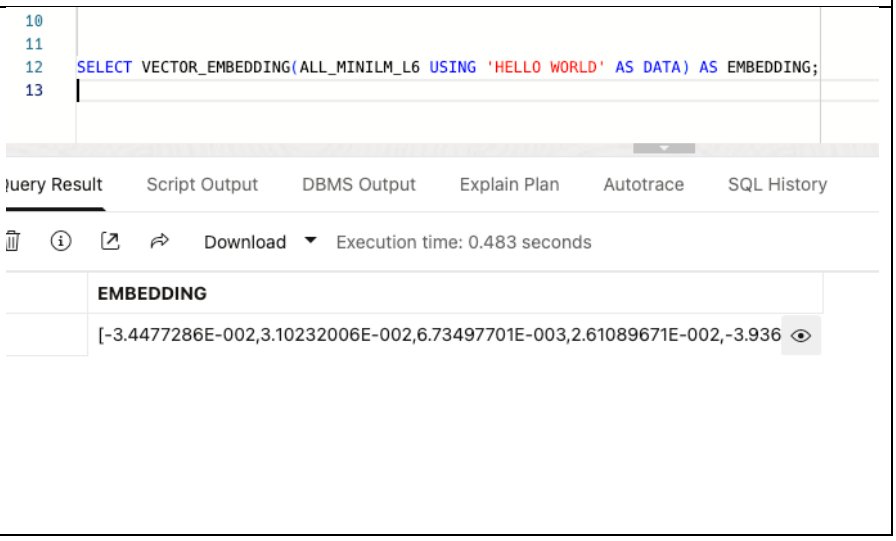
We can then check via SQL Developer whether the model was loaded in successfully.

```
SELECT
MODEL_NAME,
ALGORITHM,
MINING_FUNCTION
FROM
USER_MINING_MODEL
S
WHERE
MODEL_NAME='ALL_M
INILM_L6';
```



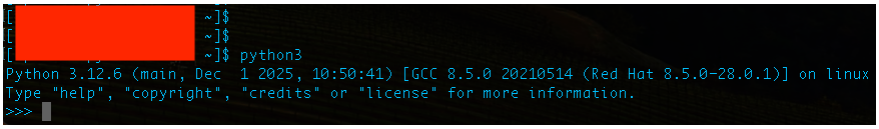
We can then test the embedding model works.

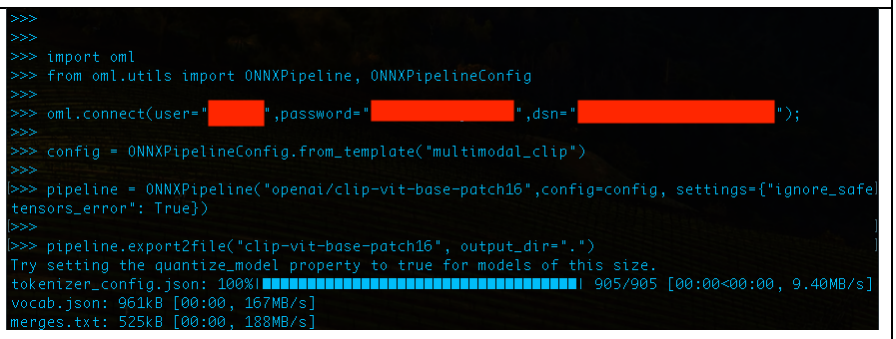
```
SELECT
VECTOR_EMBEDDING(
ALL_MINILM_L6
USING 'HELLO
WORLD' AS DATA)
AS EMBEDDING;
```



Scenario 3: Downloading a custom multi-modal embedding model from HuggingFace locally

Referenced Documentation: https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/onnx-pipeline-models-multi-modal-embedding.html

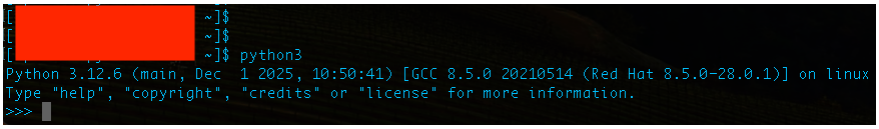
Step	Screenshot
Within your OML4Py Client Installation, Open a Python session. `python3`	

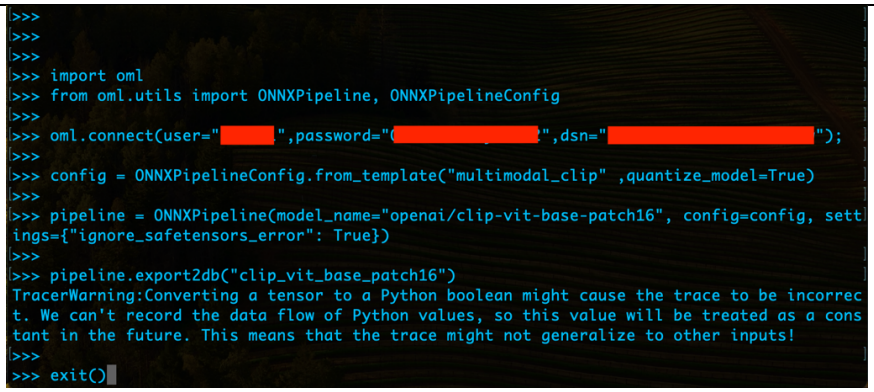
We will look at downloading the ***openai/clip-vit-base-patch16*** model from HuggingFace using the ONNX Pipeline Templates.
 	
Import the Libraries
 	
Define the multimodal CLIP Template.
 	
Define the ONNX Pipeline.
 	
Export to file.
 	
Code snippet below screenshot →
	
	
	`>> from oml.utils import ONNXPipeline, ONNXPipelineConfig`
	
	`>> config = ONNXPipelineConfig.from_template("multimodal_clip")`
	
	`>> pipeline = ONNXPipeline(model_name="openai/clip-vit-base-patch16", config=config, settings={"ignore_safetensors_error": True})`
	
	`>> pipeline.export2file("clip-vit-base-patch16",output_dir=".")`
	
	`>> exit()`

Outside of the Python session, we can then search our local directory for the file. `ls -la | grep clip`	

Scenario 4: Loading a custom multi-modal embedding model directly to Autonomous Database

Referenced Documentation: https://docs.oracle.com/en/database/oracle/machine-learning/oml4py/2-23ai/mlpug/onnx-pipeline-models-multi-modal-embedding.html

Step	Screenshot
Within your OML4Py Client Installation, Open a Python session. `python3`	

We will look at downloading the ***openai/clip-vit-base-patch16*** model from HuggingFace using the ONNX Pipeline Templates.
 	
Import the Libraries
 	
Connect to our Autonomous DB
 	
(Replace you're your connection details, where ***dsn*** is our Autonomous DB service name.)
 	
Define an empty multimodal CLIP Template.
 	
Define the ONNX Pipeline
 	
Export to Database.
 	
Code snippet below screenshot →
	
	
	`>> import oml`
	`>> from oml.utils import ONNXPipeline, ONNXPipelineConfig`
	
	`>> oml.connect(user="username", password="password", dsn="dsn");`
	
	`>> config = ONNXPipelineConfig.from_template("multimodal_clip")`
	
	`>> pipeline = ONNXPipeline(model_name="openai/clip-vit-base-patch16", config=config, settings={"ignore_safetensors_error": True})`
	
	`>> pipeline.export2db("clip_vit_base_patch16")`
	
	`>> exit()`

We can then check via SQL Developer whether the model was loaded in successfully.

```
SELECT
MODEL_NAME,
ALGORITHM,
MINING_FUNCTION
FROM
USER_MINING_MODEL
S
WHERE MODEL_NAME
LIKE 'CLIP%';
```