

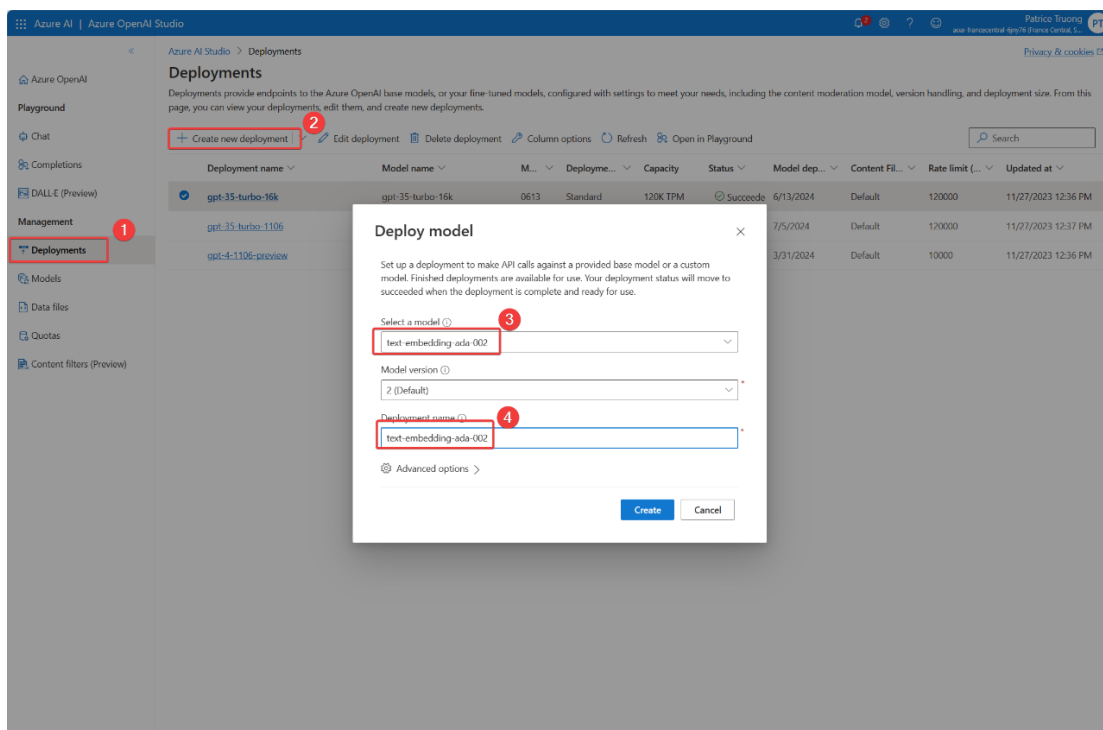
# Lab 1: Prepare environment

This document explains how to create databases and containers that support vector search in a Cosmos DB for NoSQL account

## Deploy Azure OpenAI models

Follow these steps to deploy the Azure OpenAI models (GPT 3.5 Turbo and Text-embedding-ada-002)

- Login to the Azure Portal
- Connect to the Azure OpenAI account
- In the left menu, select “Model deployments”
- Click “Manage deployments”
- Select the “Deployments” section
- Click on the “Create deployment” button
- Select the “text-embedding-ada-002” model in the dropdown list
- In the deployment name, type text-embedding-ada-002
- Click on the “Create” button to deploy the model



- Repeat the previous steps to deploy the gpt-3.5-turbo model

## Get Azure OpenAI key

- Connect to Azure Portal
- Connect to Azure OpenAI account
- In the left navigation menu, select “Keys and Endpoint”
- Key 1 and endpoint are the values you will need to add to your .env file

The screenshot displays the Microsoft Azure portal interface. At the top, the header shows 'Microsoft Azure' and a search bar. Below the header, the breadcrumb navigation indicates 'Home > team13openai'. The main title of the page is 'team13openai | Keys and Endpoint'. On the left side, there is a navigation menu with various categories: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource Management, Monitoring, Automation, and Help. The 'Keys and Endpoint' option under 'Resource Management' is highlighted with a red box. The main content area shows a warning message about key security, followed by a 'Show Keys' button. Below this, there are two key entries: 'KEY 1' and 'KEY 2', each with a text input field containing a masked key and a copy icon. The 'Endpoint' is displayed as 'https://team13openai.openai.azure.com/' with a copy icon. The 'Location/Region' is set to 'westus'.

Microsoft Azure

Search resources, services, and docs (G+/I)

Home > team13openai

team13openai | Keys and Endpoint

Search

Regenerate Key1 Regenerate Key2

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Resource Management

Keys and Endpoint

Model deployments

Encryption

Pricing tier

Networking

Identity

Cost analysis

Properties

Locks

Monitoring

Alerts

Metrics

Diagnostic settings

Logs

Automation

Tasks (preview)

Export template

Help

Resource health

Support + Troubleshooting

These keys are used to access your Azure AI service API. Do not share your keys. Store them securely- for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

Show Keys

KEY 1

KEY 2

Location/Region

westus

Endpoint

https://team13openai.openai.azure.com/

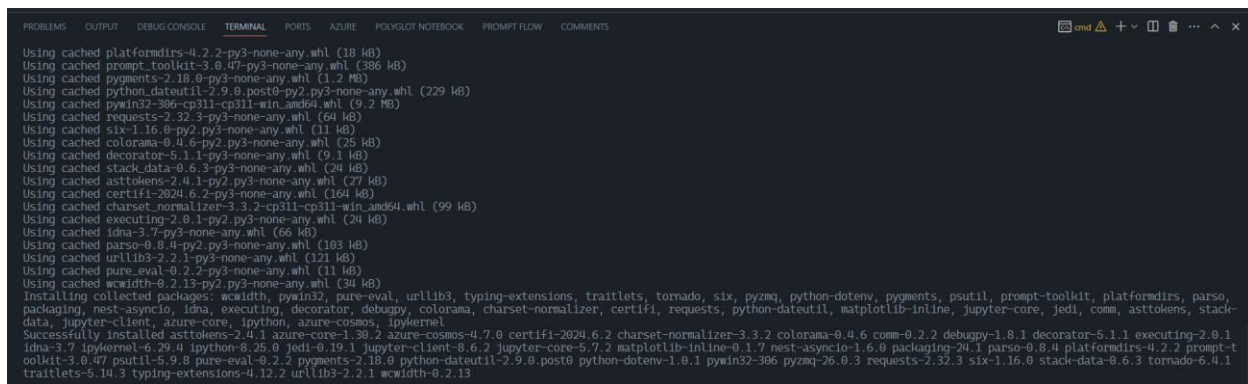


## Create virtual environment

1. Create a `requirements.txt` file at the root of the `"labs"` folder
2. Add the following libraries

```
python-dotenv
tenacity
ipykernel
matplotlib
plotly
scikit-learn
openai
azure-cosmos
streamlit
pymongo
tiktoken
azure-identity
```

3. Open a command prompt and navigate to the `"labs"` folder
4. Create a virtual environment with this command: `python -m venv .venv`
5. Activate the virtual environment with `.venv\scripts\activate`
6. Install the required libraries with `pip install -r requirements.txt`



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE POLYGLOT NOTEBOOK PROMPT FLOW COMMENTS
Using cached platformdirs-4.2.2-py3-none-any.whl (18 kB)
Using cached prompt_toolkit-3.0.47-py3-none-any.whl (386 kB)
Using cached pygments-2.18.0-py3-none-any.whl (1.2 MB)
Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Using cached pywin32-306-cp311-cp311-win_amd64.whl (9.2 MB)
Using cached requests-2.32.3-py3-none-any.whl (64 kB)
Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Using cached decorator-5.1.1-py3-none-any.whl (9.1 kB)
Using cached stack_data-0.6.3-py3-none-any.whl (24 kB)
Using cached asttokens-2.4.1-py2.py3-none-any.whl (27 kB)
Using cached certifi-2024.6.2-py3-none-any.whl (164 kB)
Using cached charset_normalizer-3.3.2-cp311-cp311-win_amd64.whl (99 kB)
Using cached executing-2.0.1-py2.py3-none-any.whl (24 kB)
Using cached idna-3.7-py3-none-any.whl (66 kB)
Using cached parso-0.8.4-py2.py3-none-any.whl (103 kB)
Using cached urllib3-2.2.1-py3-none-any.whl (121 kB)
Using cached pure_eval-0.2.2-py3-none-any.whl (11 kB)
Using cached wcwidth-0.2.13-py2.py3-none-any.whl (34 kB)
Installing collected packages: wcwidth, pywin32, pure-eval, urllib3, typing-extensions, traitlets, tornado, six, pyzmq, python-dotenv, pygments, psutil, prompt-toolkit, platformdirs, parso, packaging, nest-asyncio, idna, executing, decorator, debugpy, colorama, charset-normalizer, certifi, requests, python-dateutil, matplotlib-inline, jupyter-core, jedi, comm, asttokens, stack-data, jupyter-client, azure-core, ipython, azure-cosmos, ipykernel
Successfully installed asttokens-2.4.1 azure-core-1.30.2 azure-cosmos-4.7.0 certifi-2024.6.2 charset-normalizer-3.3.2 colorama-0.4.6 comm-0.2.2 debugpy-1.8.1 decorator-5.1.1 executing-2.0.1 idna-3.7 ipykernel-6.29.4 ipython-8.25.0 jedi-0.19.1 jupyter-client-8.6.2 jupyter-core-5.7.2 matplotlib-inline-0.11.7 nest-asyncio-1.6.0 packaging-24.1 parso-0.8.4 platformdirs-4.2.2 prompt-toolkit-3.0.47 psutil-5.9.8 pure-eval-0.2.2 pygments-2.18.0 python-dateutil-2.9.0.post0 python-dotenv-1.0.1 pywin32-306 pyzmq-26.0.3 requests-2.32.3 six-1.16.0 stack-data-0.6.3 tornado-6.4.1 traitlets-5.14.3 typing-extensions-4.12.2 urllib3-2.2.1 wcwidth-0.2.13
```

# Create a container that supports vector search

---

In this section, we will use a Python notebook to create a container that supports vector search

## Create product container

1. Create a "Lab1" folder in the "Labs" folder
2. Open Visual Studio Code
3. In the "Lab1" folder, create a new Jupyter notebook called "create\_container.ipynb"
4. Create a new cell and add the following content

```
from azure.cosmos import CosmosClient, PartitionKey
from azure.cosmos.cosmos_client import ThroughputProperties
from dotenv import load_dotenv

load_dotenv("../.env")

COSMOS_DB_ENDPOINT = os.getenv('AZURE_COSMOSDB_NOSQL_ENDPOINT')
COSMOS_DB_KEY = os.getenv('AZURE_COSMOSDB_NOSQL_KEY')
DATABASE_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_DATABASE_NAME')
CONTAINER_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_CONTAINER_NAME')
OFFER_THROUGHPUT = 1000

throughput_properties =
ThroughputProperties(auto_scale_max_throughput=OFFER_THROUGHPUT)

indexing_policy = {
    "includedPaths": [
        {"path": "/*"},
    ],
    "excludedPaths": [
        {"path": "/\"_etag\"/?"},
        {"path": "/embedding/*"}
    ],
    "vectorIndexes": [
        {
            "path": "/embedding",
            "type": "quantizedFlat"
        }
    ]
}

embedding_policy = {
    "vectorEmbeddings": [
        {
```

```

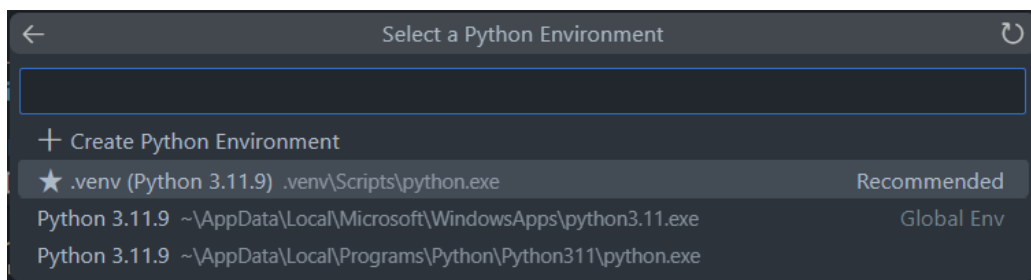
        "path": "/embedding",
        "dataType": "float32",
        "distanceFunction": "cosine",
        "dimensions": 384
    }
]
}

print("Getting database..")
client = CosmosClient(COSMOS_DB_ENDPOINT, COSMOS_DB_KEY)
database = client.get_database_client(DATABASE_NAME)

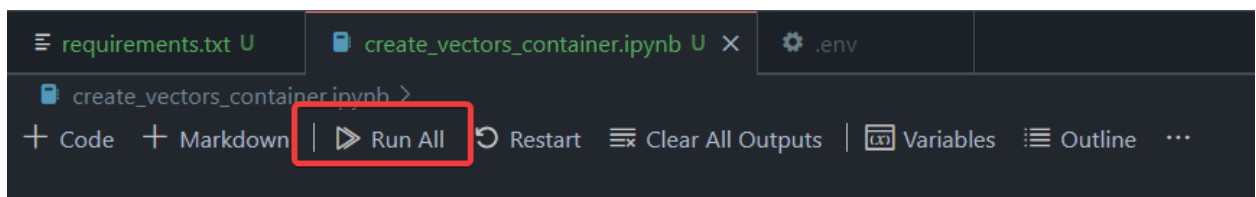
# Create "products" container
print(f"Creating '{CONTAINER_NAME}' container..")
container = database.create_container_if_not_exists(
    id=CONTAINER_NAME,
    partition_key=PartitionKey(path="/id"),
    indexing_policy=indexing_policy,
    vector_embedding_policy=embedding_policy,
    offer_throughput=throughput_properties
)
print(f"'{CONTAINER_NAME}' container created.")

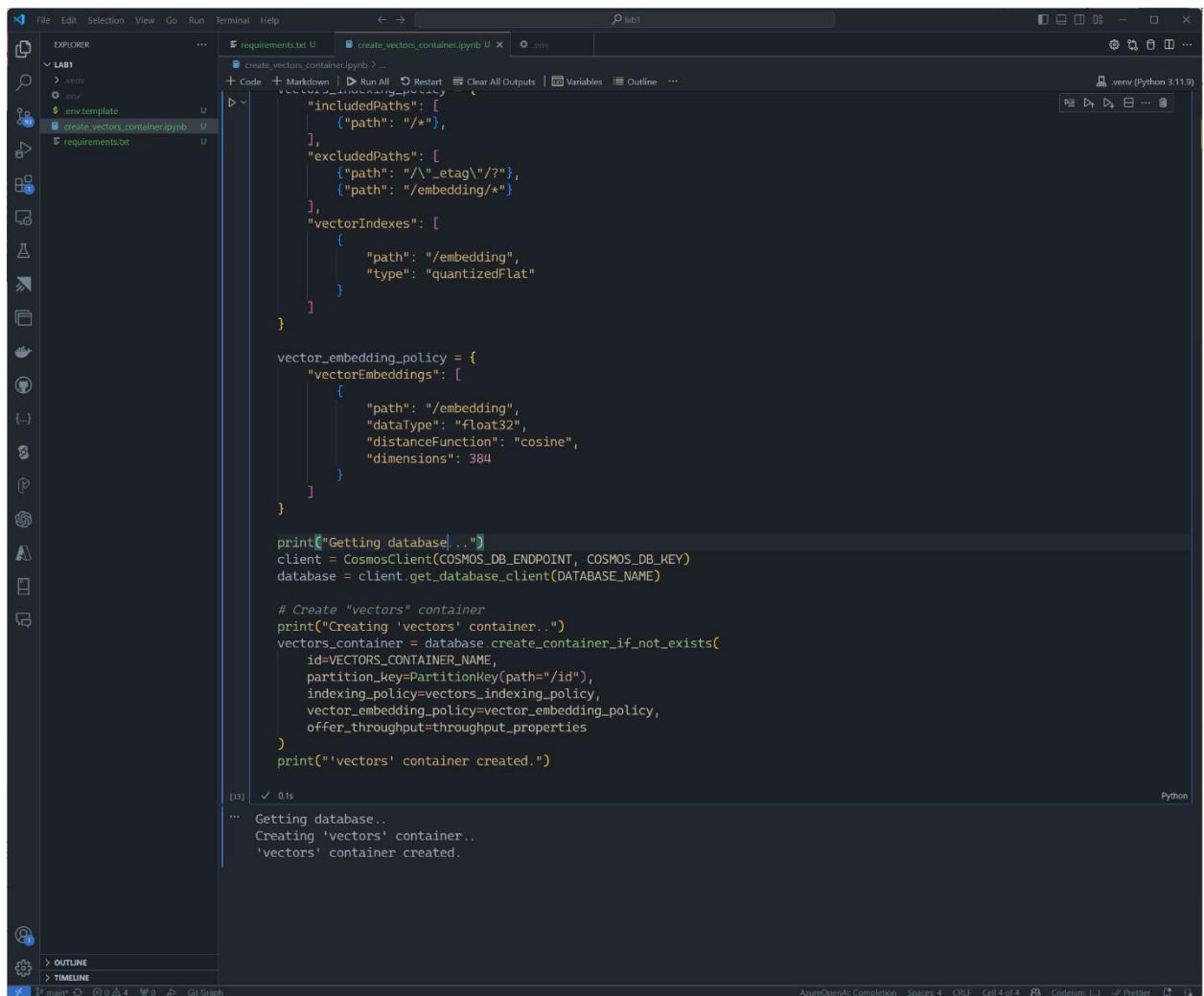
```

5. In the top-right corner, click on "Select kernel"
6. Select "Python environment" and select the recommended environment (that should point to the .venv environment that you created in the preview section)



7. In the menu bar, click on "Run all"





```
File Edit Selection View Go Run Terminal Help
create_vectors_container.py:nb U x
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ...
venv (Python 3.11.9)

"includedPaths": [
  {"path": "/*"},
],
"excludedPaths": [
  {"path": "/\\\"_etag\\\"/?"},
  {"path": "/embedding/*"}
],
"vectorIndexes": [
  {
    "path": "/embedding",
    "type": "quantizedFlat"
  }
]

vector_embedding_policy = {
  "vectorEmbeddings": [
    {
      "path": "/embedding",
      "dataType": "Float32",
      "distanceFunction": "cosine",
      "dimensions": 384
    }
  ]
}

print("Getting database...")
client = CosmosClient(COSMOS_DB_ENDPOINT, COSMOS_DB_KEY)
database = client.get_database_client(DATABASE_NAME)

# Create "vectors" container
print("Creating 'vectors' container..")
vectors_container = database.create_container_if_not_exists(
  id=VECTORS_CONTAINER_NAME,
  partition_key=PartitionKey(path="/id"),
  indexing_policy=vectors_indexing_policy,
  vector_embedding_policy=vector_embedding_policy,
  offer_throughput=throughput_properties
)
print("'vectors' container created.")

[13] ✓ 0.1s Python

... Getting database..
Creating 'vectors' container..
'vectors' container created.
```

8. Verify in the Azure portal that the “products” container has been successfully created.

