Add your own data to ChatGPT

Thursday, 2 November 2023 | 16:00 - 17:45 | Startup Nights 2023 in Winterthur

This workshop guides you through setting up Azure OpenAI services, some prompt engineering basics as well as supplying your own data (in form of files). In the end, you

- can setup your own Azure OpenAI service
- understand the basics of LLMs.
- understand basics of prompt engineering
- can include your own documents for Azure OpenAI

Introduction reading time: approx. 5min

An LLM usually solves the machine learning task of predicting the next, most likely token given a series of input tokens. In simpler terms: An LLM takes input text and predicts the next word. It turns out, that even though the task itself seems rather simple, that given enough training data that LLMs can solve different tasks with astonishing results. The skill we will explore is answering questions based on given context text.

Llama 2 is an example of an LLM: It is the latest model of Meta (Facebook) and is fully open source. Hence, exact key-facts are public. LLMs are deep neural networks consisting of an enormous number of "neurons", each of them needs parameters that is used to calculate the input for the next connected neuron. Llama 2 has 70 billion of them.

Training such a neural network requires a huge amount of data *and* computational power: Llama 2 was trained on 2 trillion tokens using computational resources that currently equate to about 13.5 million USD on Azure.

Most of the process is automated: the fundamental first step is called pretrained and is self-supervised: Using a large corpus of source material, the training process can automatically select tokens that it tries then to predict, for example randomly. This way, it knows the input *and* the answer and can supervise itself: If *pizza* from "*My favorite food is pizza"* is selected, the training process can train with a sample that given "*My favorite food is*" as input, "*pizza*" should be predicted.

However, while such "foundation models" are known to excel at generating grammatically correct content, the content itself represents the structure of the source material. As a lot of it is scraped from the internet, it also has the same range of quality: From stellar factual answers to fact-free hate-speech. To avoid "rogue" chat-agents such as the infamous Microsoft Tay, LLMs such as ChatGPT or Llama are fine-tuned and aligned.

In contrast to the pretraining, fine-tuning usually requires much more manual work and is usually done in two steps:

The first one is using hand-picked / manually written chats: Llama 2 used 27.5 thousand chat interactions. The model can then learn in a similar self-supervised manner from those specific examples.

The second step is reinforcement learning from human feedback (RLHF): For Llama 2, 1.4 million interactions were manually rated and used to train a second machine learning model that can rate the answer of the LLM itself. The LLM can now continue to be trained, supervised by this RLHF model, until the answers have reached a certain level.

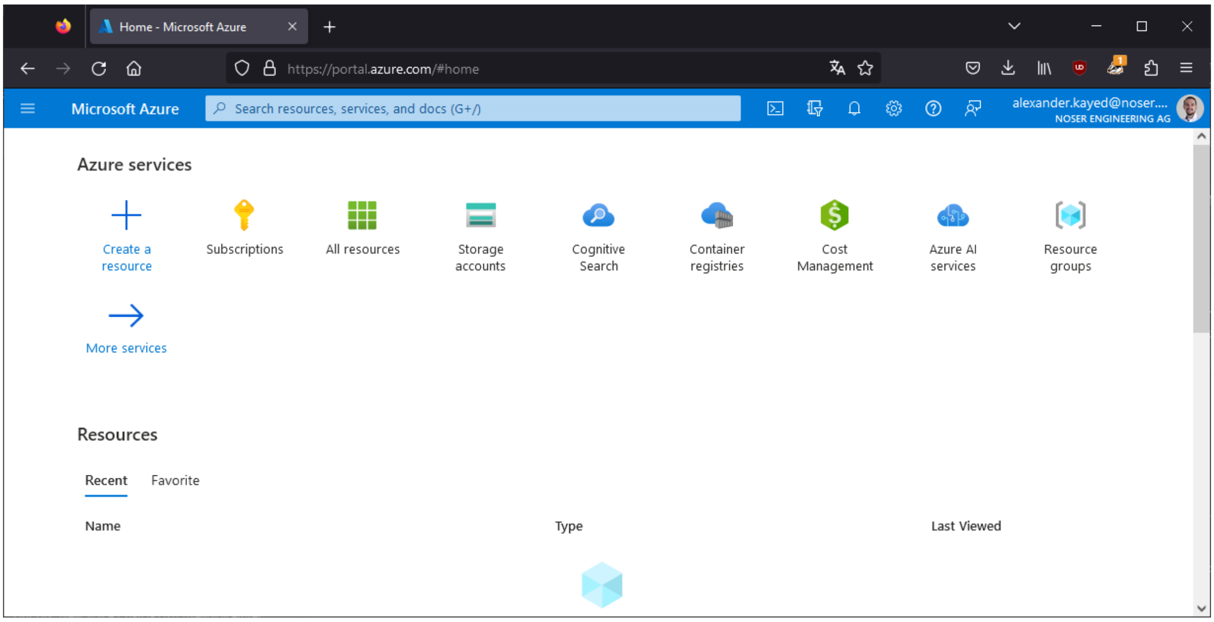
ChatGPT and GPT-4 are said to be trained in the same way, however both are much larger than Llama 2 and hence, a much bigger and costlier undertaking. ChatGPT is at [175 billion] parameters, GPT-4 is said to have [1.75 trillion] parameters. This means that training and often, even fine-tuning, is not feasible nor desirable. And hence, a new "programming discipline" was born: "Prompt Engineering".

Prompt Engineering is a relatively new discipline to create and improve inputs of LLMs to efficiently apply them to a range of different applications. A prompt is the input text you pass into an LLM. As the prompt engineering techniques are only about the prompt, it allows to adapt the LLM for the task at hand *without* (additional) training or fine-tuning.

Techniques include for example one/few-shot learning and retrieval augmented generation. Continue, to try them yourself.

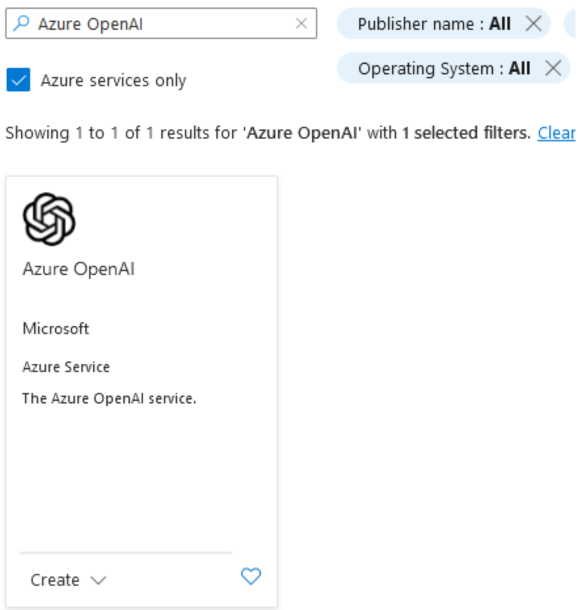
Part 1: Getting Started expected time: approx. 5 min

1. Go to https://portal.azure.com
2. Log in using the provided credentials. You should get to the following page:

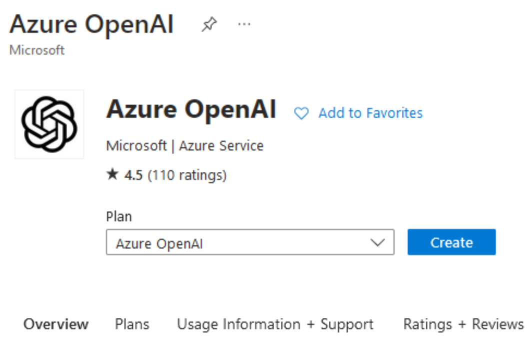


3. Click on *Create a resource.*

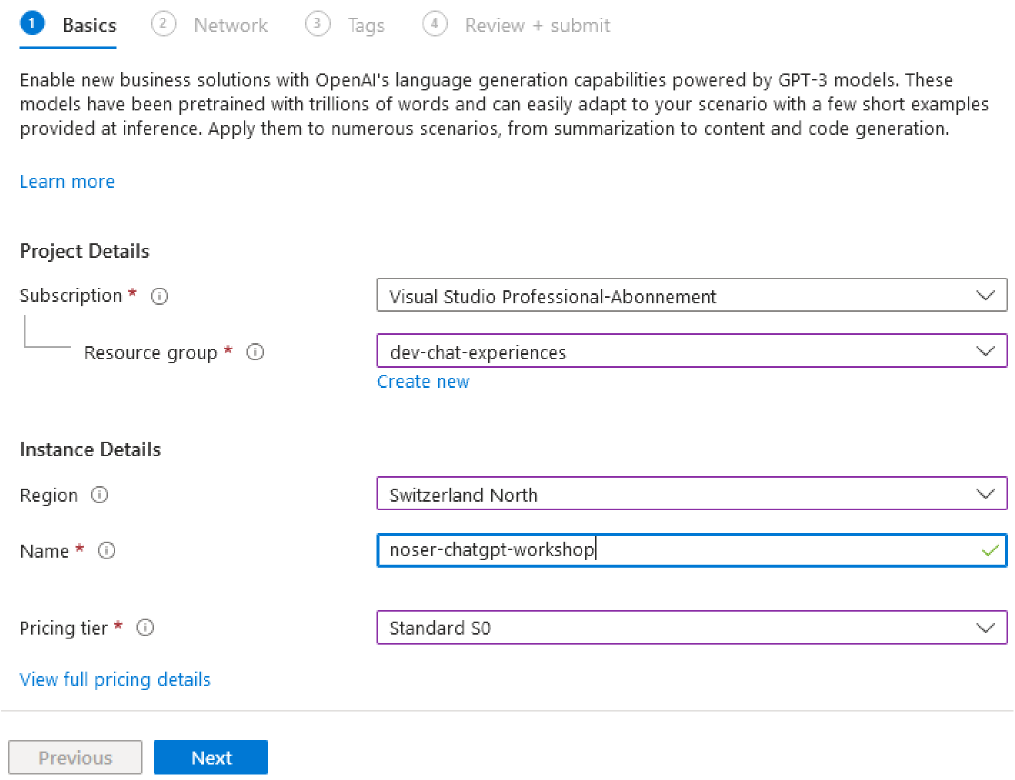
4. Search for *Azure OpenAI*. Make sure to tick *Azure services only*, you should get a single result:



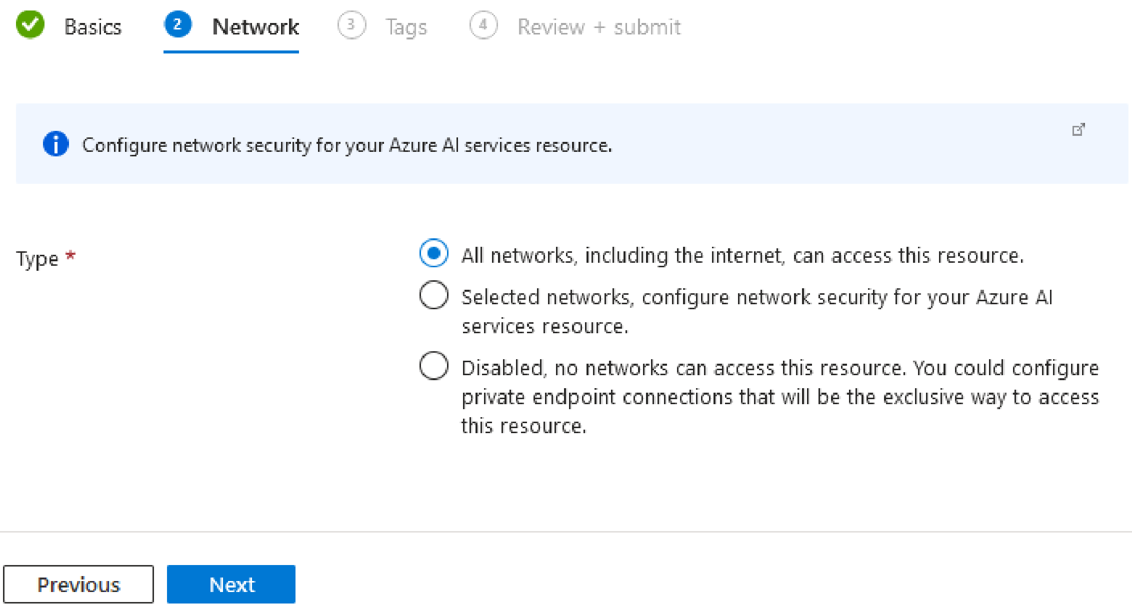
Click on it to continue, and click again on *create* on the next screen:



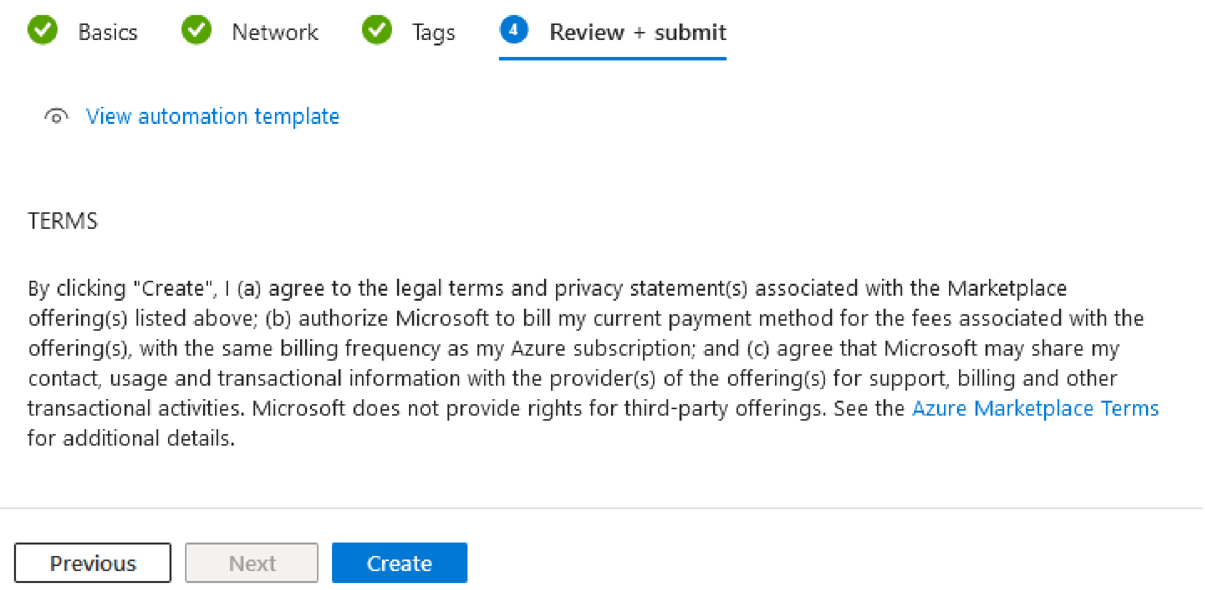
5. On the following screen, choose the following values:
    1. Subscription: Choose the only available one.
    2. Resource group: Click on *create new*. Enter a unique name, e.g., based on your username: *username-resources.*
    3. Region: *Choose Switzerland North*
    4. Choose a unique name for your own Open AI instance, e.g., based on your username: *username-openai.*
    5. Choose *Standard S0* (nothing else is available)
    6. After filing in everything, click on *next*



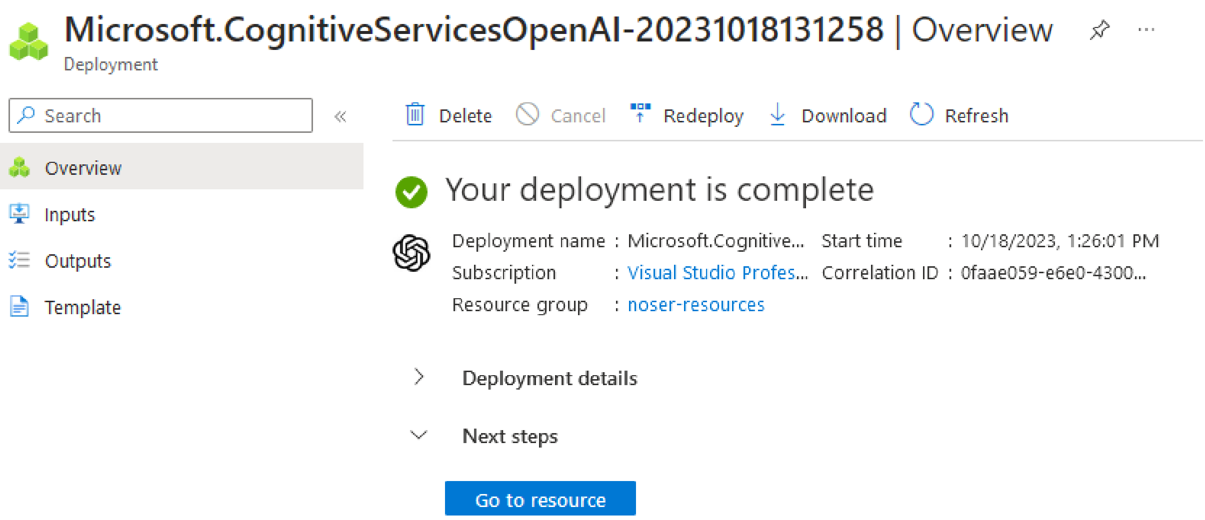
6. For simplicity for this workshop, keep the default network setting. Click *next*:



7. In the next screen, "Tags", click *next* again and click *create* on the final screen once it is ready:

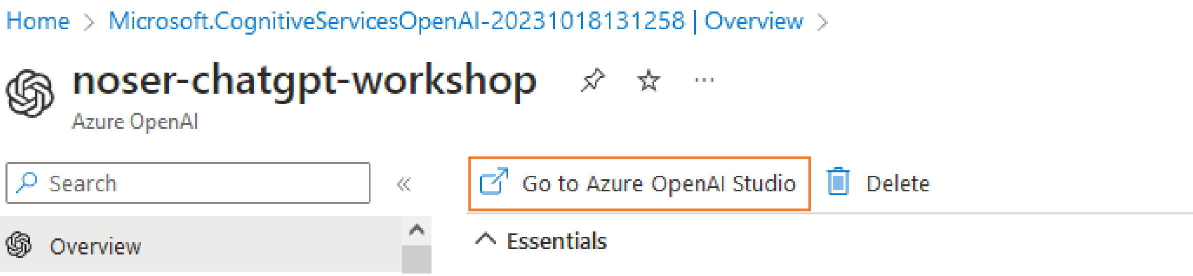


8. Voilà: Your own Azure OpenAI instance will be created. This will take a few moments and is finished as soon as you see the following. Click on *Go to resource* to continue to the next part of the workshop.

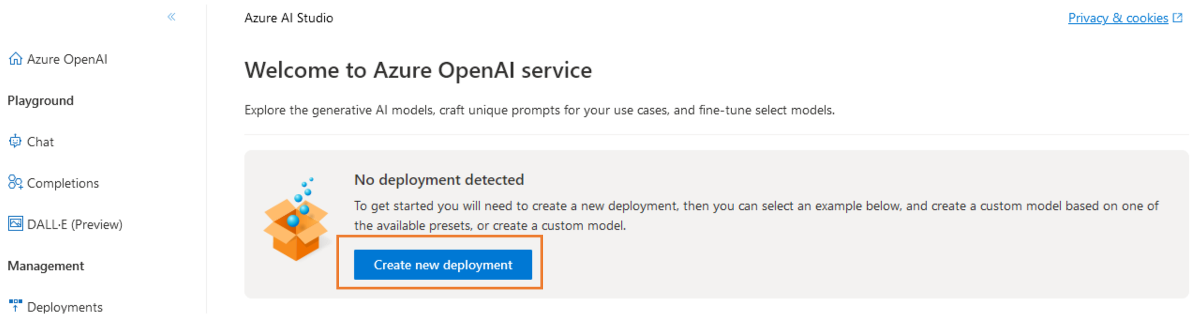


Part 2: Deploy models expected time: approx. 5 min

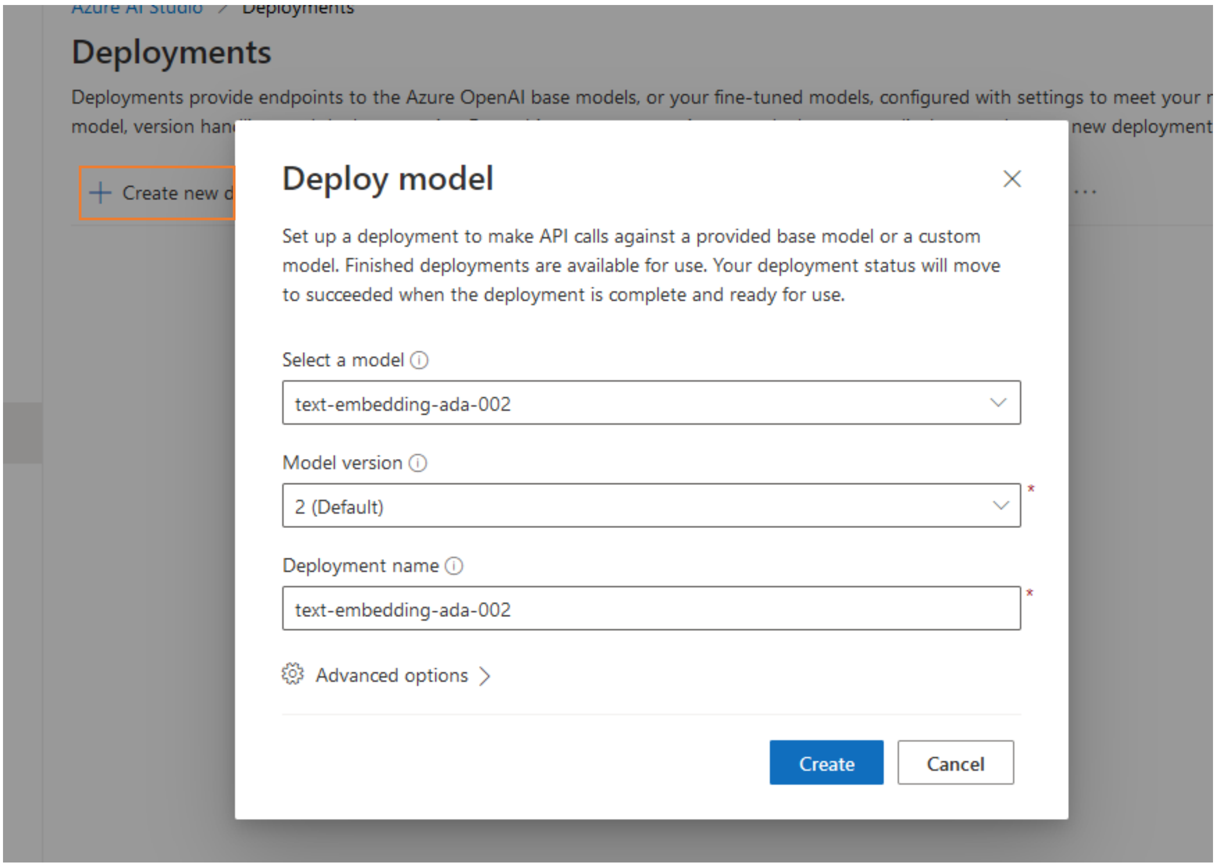
1. Click on Azure OpenAI Studio:



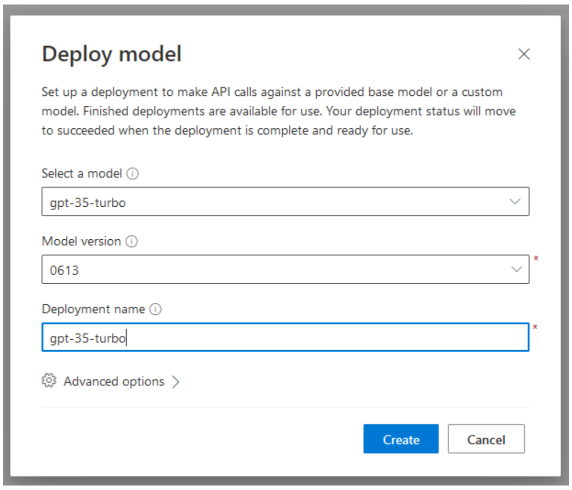
2. Click on create new deployment:



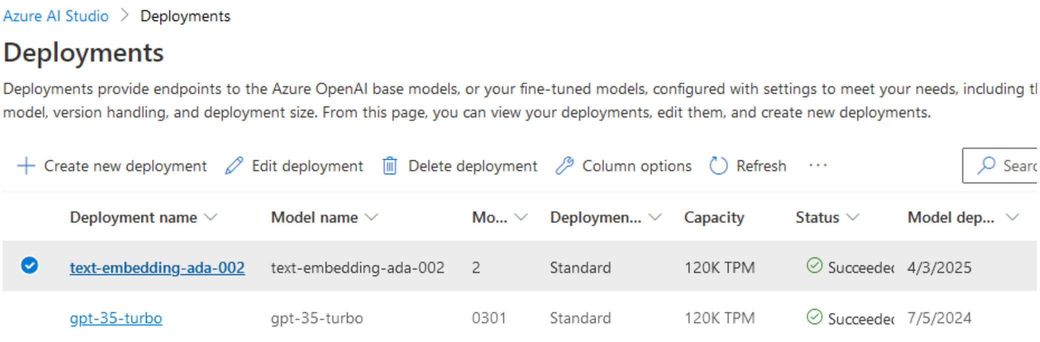
3. Click on *create new deployment*. Choose *text-embedding-ada-002* and enter a name: You can just use text-embedding-ada-002. Click on create.



4. Create another deployment. Choose *gpt-35-turbo* this time. Select model version 0613:



5. You should now have two deployments and are ready for the next part:

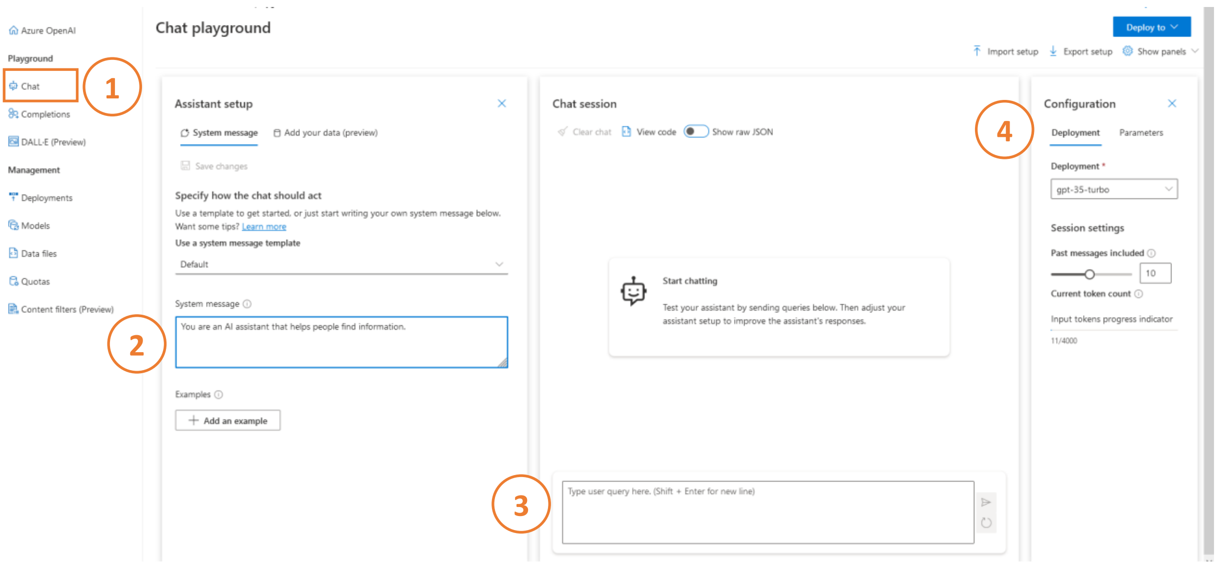


Coding Track

All required resources for the coding track have been setup now. If you want to do the coding track, you can switch to the Coding Track.

Part 3: First steps & prompt engineering expected time: approx. 30 min

Before we get started with adding data, let us play a bit around with ChatGPT to get a basic understanding on how everything works. To chat with your own instance, switch to Chat (1):



(2) is the system message, which is an instruction, in natural language, which is sent along the user input. In (3), you can chat with the AI and in (4), you can tune settings. For this workshop, we will leave the configuration at their default settings.

System Message expected time: approx. 10 min

1. Let us start with the following prompt:

   ```
   What is your name?
   ```

   What do you get back? You can try it multiple times, do not forget to reset using *Clear chat* at the top of the window.
2. Now try modifying the *system message* that the AI will respond with a name and try again. Do not forget to *Save changes* at the top of the assistant setup pane.
3. Try to solve the following exercises, *using only the system prompt*:
   - Make the AI respond in another language, even if the “What’s your name?” is in English.
   - Make the AI translate the user input (and not answer questions) into a fixed language of your choice.
   - Make the AI judge the sentiment of a user input. It should respond with a single word. Try it with inputs such as “The food was delicious” or “Not really my thing.”
   - Try to make the AI respond to the user question but with a distinctive style of writing, for example with rhymes or in old English like Shakespear.

Context Within Prompts expected time: approx. 10 min

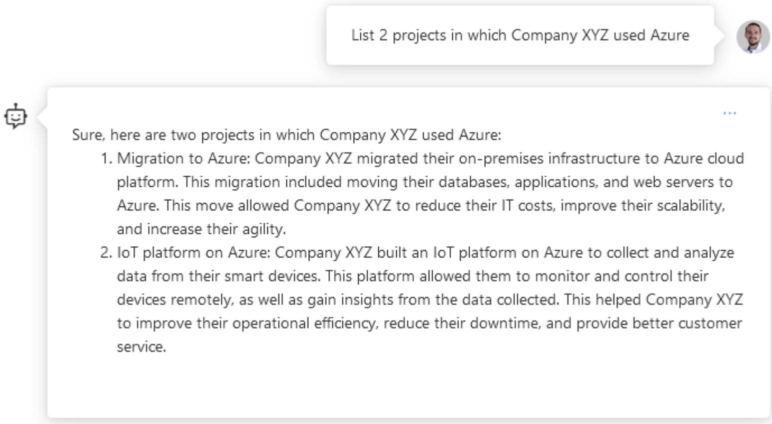
As we can see, we can already shape the answers quite well with instructions in the System Message without training for the specific task. But how do we include any more specific knowledge? For that, we will try including context within the user input. Let us create a bot that can answer questions about projects in natural language:

1. Start by defining an appropriate system message that describes what the AI should do:

   ```
   You are an AI assistant that helps people find information about projects of Company XYZ
   ```

2. Ask *"List 2 projects in which Company XYZ used Azure"* or *"List companies for which Company XYZ built projects"* and see what happens.

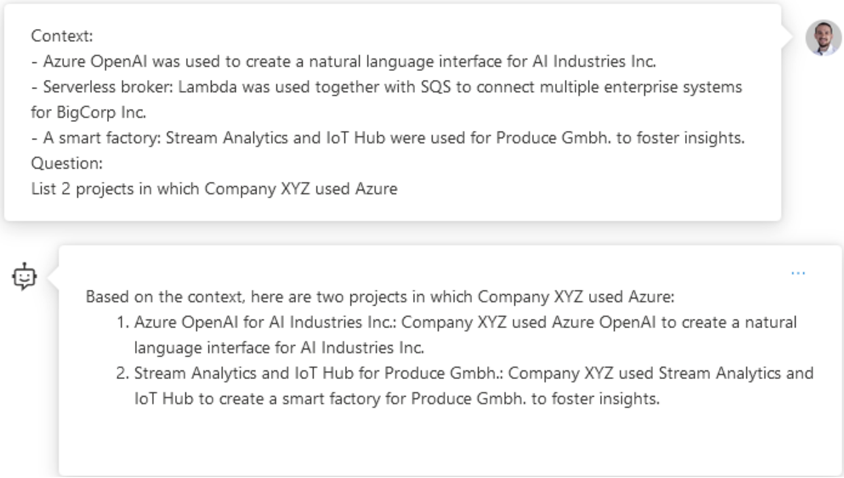
   While the answer will sound credible, it is generic and of course, completely made up. Sometimes, it will also respond by saying that it has not enough information. Example:



3. Let us provide some context and ask the question again:

   ```
   Context:
   - Azure OpenAI was used to create a natural language interface for AI Industries Inc.
   - Serverless broker: Lambda was used together with SQS to connect multiple enterprise systems for BigCorp Inc.
   - A smart factory: Stream Analytics and IoT Hub were used for Produce GmbH. to foster insights.
   Question:
   List 2 projects in which Company XYZ used Azure
   ```

This will output a result similar to this:



This demonstrates how you can provide information within a prompt to ground answers while still profiting from the “built-in understanding of ChatGPT”: The LLM “understands” that Stream Analytics and IoT Hub are Azure products while the second list entry is using AWS.

This is exactly how “ChatGPT on your own data works”: The difference, of course, is that the included context will be a search in your documents based on the question. This workflow is called Retrieval Augmented Generation, or short: RAG-Pattern.

One-shot & Few-shot learning expected time: approx. 10 min

Another important prompt-engineering technique is few-shot learning: What we did before is called zero-shot learning as we did not provide any specific example for the task. If you provide one example, it is called one-shot learning. Accordingly, providing two or more examples is called few-shot learning.

Let us say we want the model to include a reference to the source it used, how can we achieve that. Let’s change the system prompt to

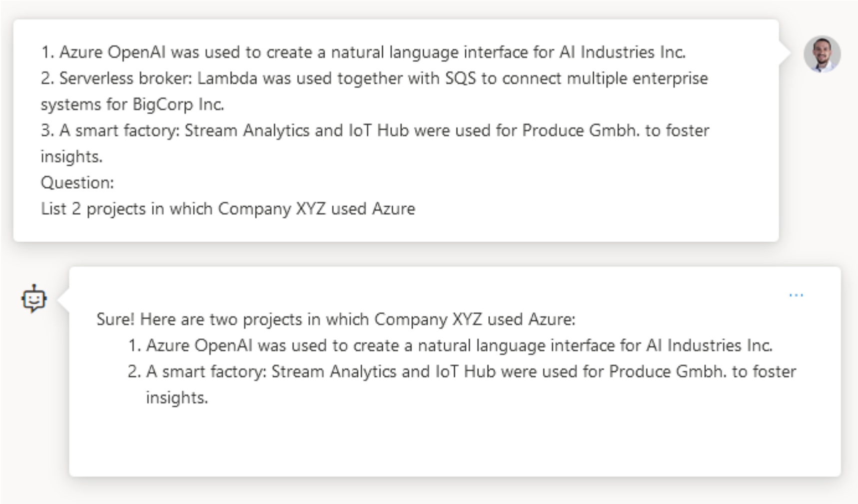
```
You are an AI assistant that helps people find information about projects of Company XYZ. Please cite the project using its number.
```

and modify the context in the prompt to include the project numbers:

```
Context:
1. Azure OpenAI was used to create a natural language interface for AI Industries Inc.
2. Serverless broker: Lambda was used together with SQS to connect multiple enterprise systems for BigCorp Inc.
3. A smart factory: Stream Analytics and IoT Hub were used for Produce GmbH. to foster insights.
Question:
List 2 projects in which Company XYZ used Azure
```

Try it out: What will you get?

You are (most likely) getting the same result:



Let us add examples by clicking on “Add an example” and you can add an example:

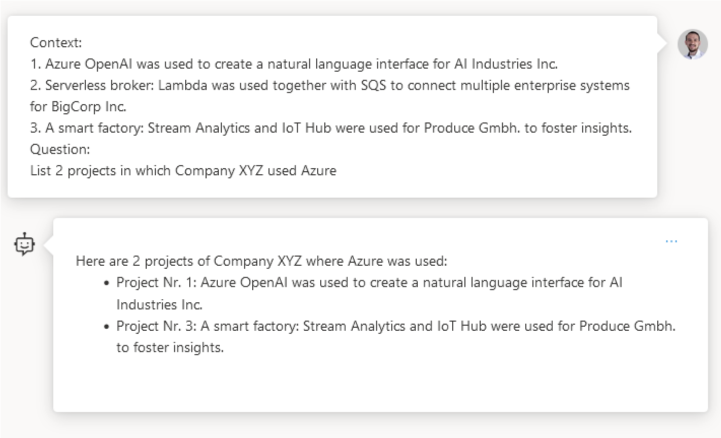
Now add the following, completely abstract example, into it:
User:

Context:
1. [...]
2. [...]
3. [...]
4. [...]
Question:
List 2 projects which ...?

Assistant:

Sure: Here you are:
- Project Nr. 2 [...]
- Project Nr. 4 [...]

After saving, try again. It should now be improved, and you will get a result similar to the following:



This few-shot-learning is the final ingredient for citations within the RAG-Pattern. Tuning the system prompt, selecting appropriate examples (and an appropriate number), post-processing the answer (and so on) is then part of a more robust LLM solution.

Here are a few exercises to try:

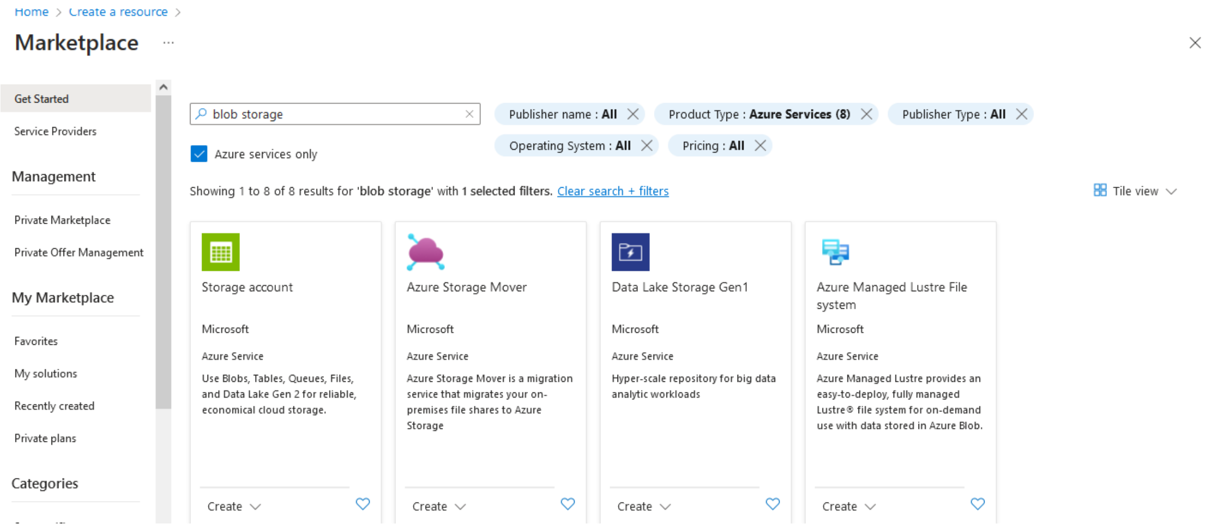
- Try to make the project number more easily recognizable for a machine: i.e., surrounded by << & >>
- If you ask for client companies instead of projects, it will most likely not include the project number. Can you improve that behavior?
- Advanced / for developers: In an integrated scenario, you might want the system to return Json to more easily post-process results: Can you change the system prompt and examples to get the following output for the original prompt *Context […] List 2 projects in which Company XYZ used Azure*:

```
[
 { "projId": 1, "company": "AI Industries Inc." },
 { "projId": 3, "company": "Produce GmbH." }
]
```

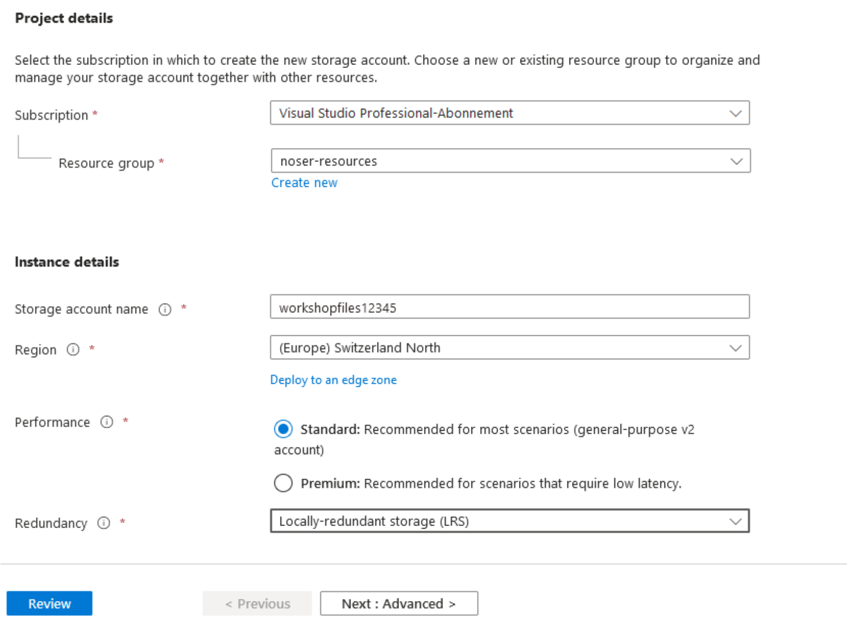
Part 4: Adding your own data to ChatGPT expected time: approx. 15 min

In this Part, we will add your data to ChatGPT by implementing the RAG-Pattern. We will use built-in Azure services even though some features are still in preview. However, in contrast to more customizable and advanced programming-based solutions, it is an approachable starting point.

First, we need a place to store our data files. Create a new resource of type *Blob Storage*. A *blob storage* is like a disk in the cloud. *Blob storage* is available within a *Storage Account.* Search for it and create one.



On the creation page:

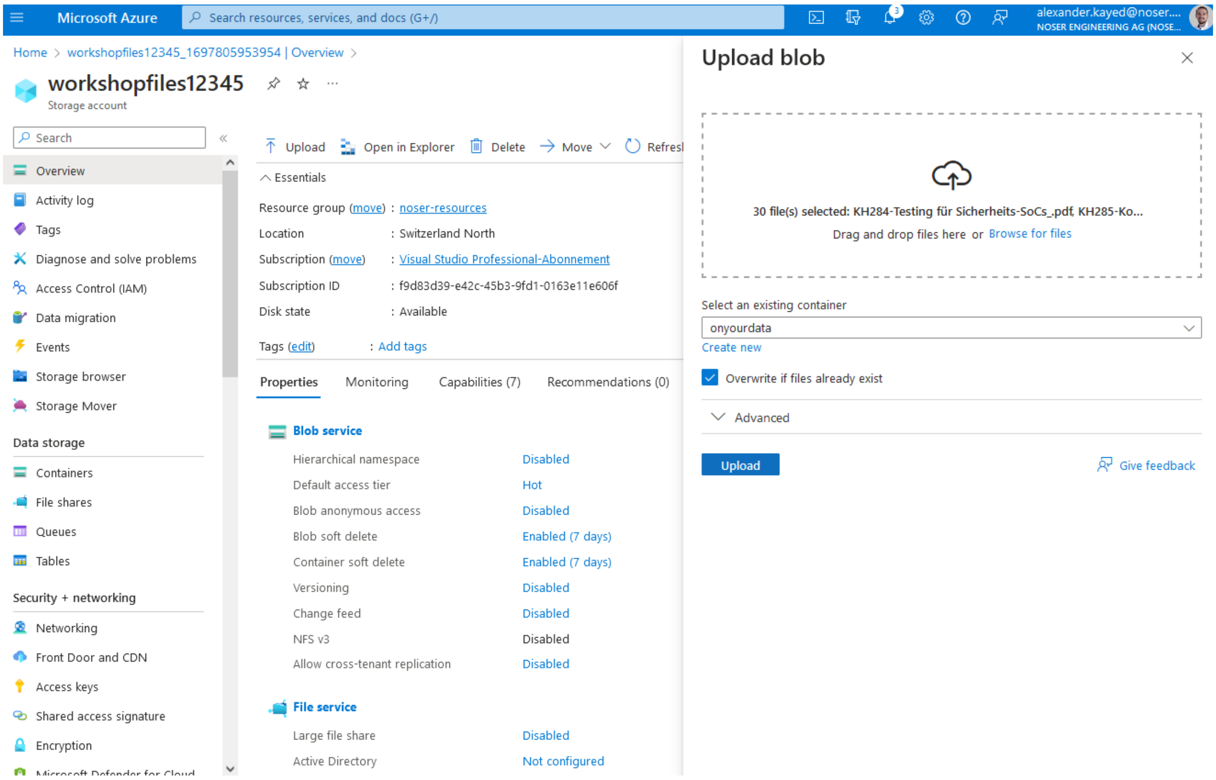


1. Set the same subscription and resource group.
2. Choose a unique storage account name.
3. Select the correct region.
4. Use *Standard Performance* and *Locally redundant storage (LRS)*.
5. Click *Review* & create the resource.

⚠ Double-check to select standard & LRS to avoid unnecessary costs!

On the following deployment screen, click on *go to resource* as soon as it is ready.

Our goal is uploading our own data. For that, we need to create a new container. A container is like a folder in a file system, but within our private Azure storage account. It will store our uploaded data. Like in a file system, you can then upload your files as shown below:



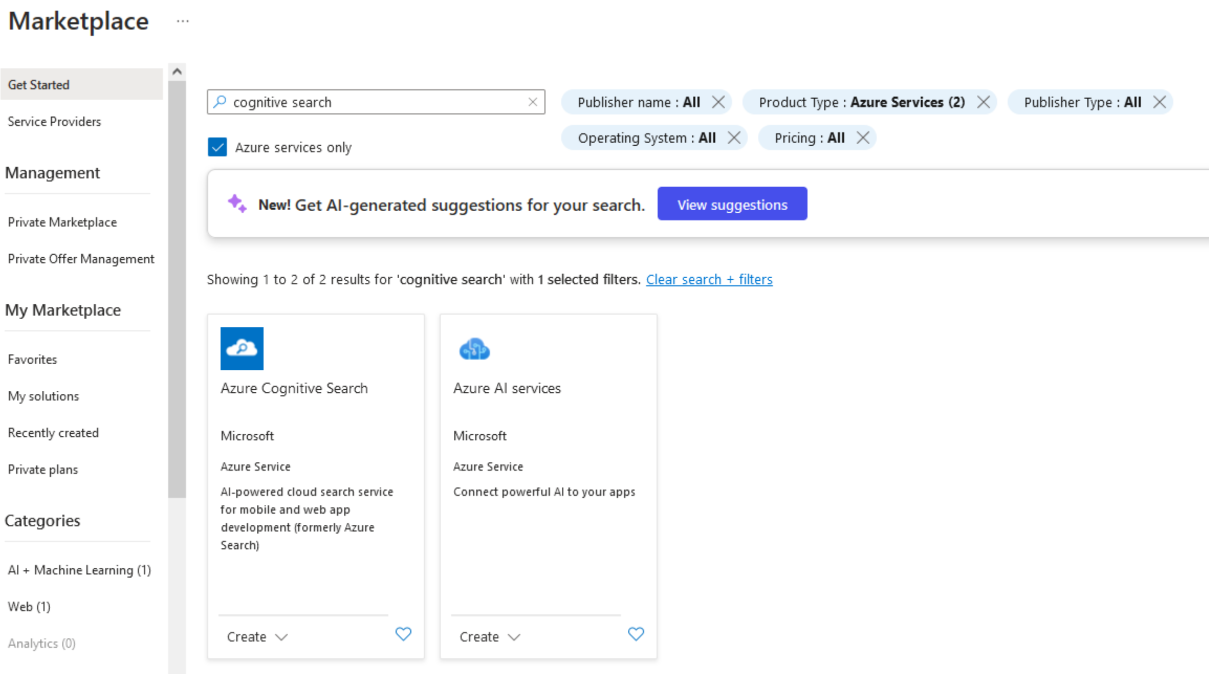
1. Click *Upload* in the upper left corner.
2. In the *Upload blob* side panel, click on *Create new* => enter something like *onyourdata*. This is your container where your files will be stored.
3. Drag & drop your files into the drop zone. You can use your own data or use some files that we have prepared for you: [View] or [Download]

> We recommend using PDFs for this workshop. The language is not important, ChatGPT will be able to handle content in a different language.

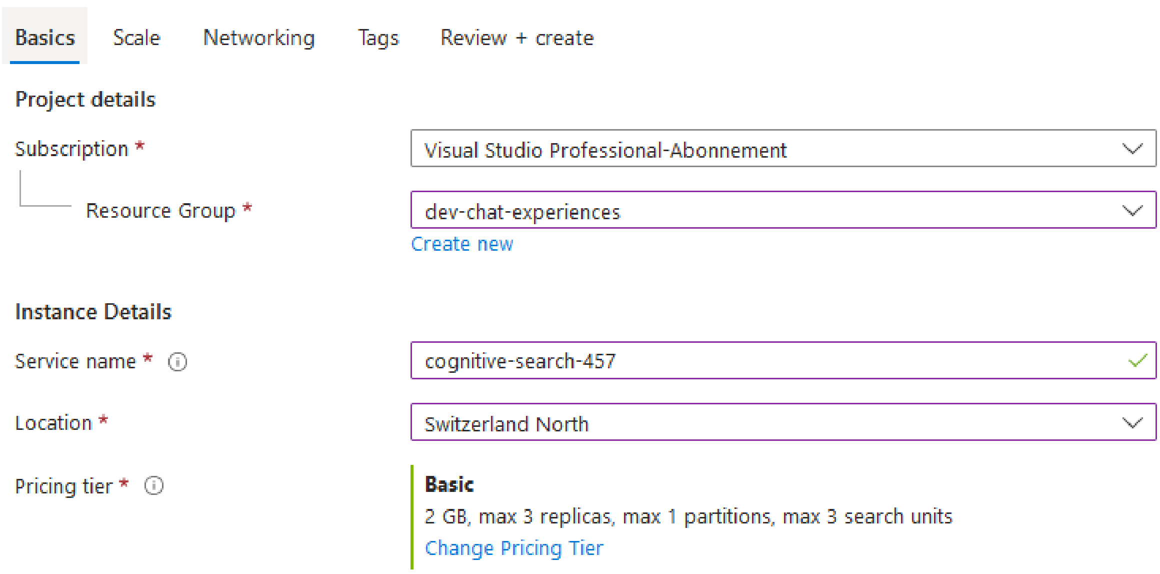
4. Click *Upload* in the side panel to start uploading the documents.

Once completed, go back, and create another resource:

Remember, for the RAG-Pattern we will need to make our content searchable. Azure Cognitive Search will index and search the documents for us. Search for it, and create a resource:



On the next screen:

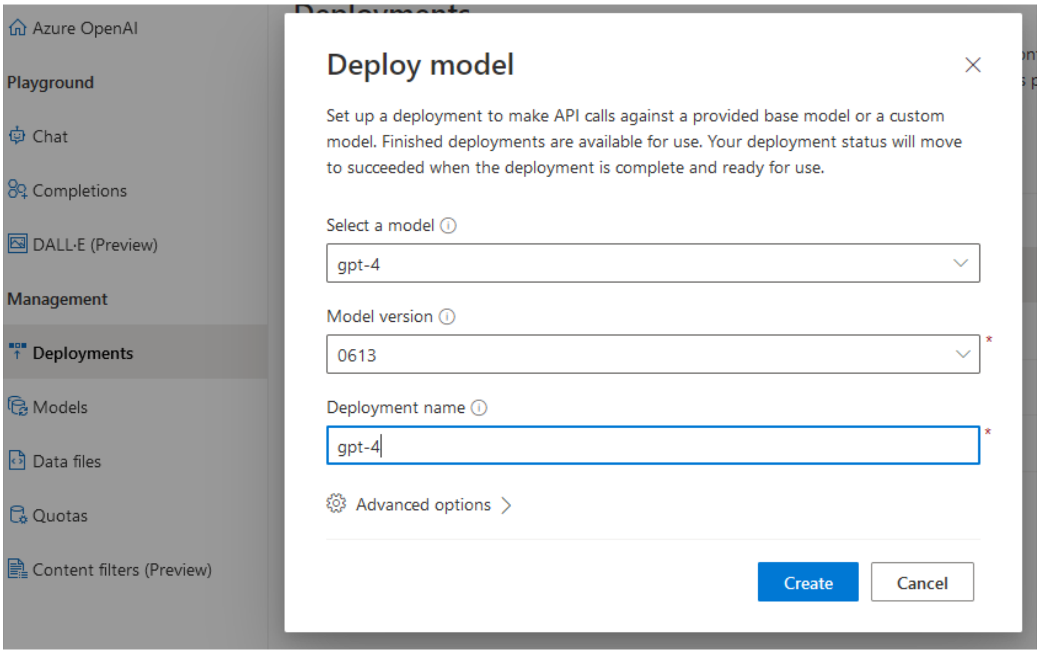


1. Choose the same subscription and resource group as with the other resources before
2. Give the service a name.
3. Choose Switzerland North as a region.
4. Select the pricing tier by clicking on *Change Pricing Tier* to *Basic*
5. Click on *Review & create* and *then again* on *Create.*

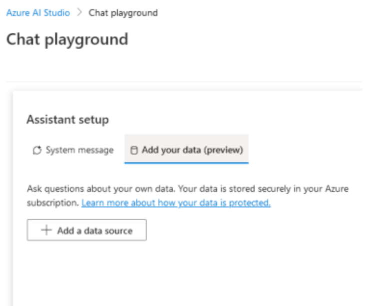
⚠ Double-check to select *Basic* as a pricing tier to avoid unnecessary costs!

Make sure that the deployment finished. Once completed, return to the Azure AI Studio and head to the model section.

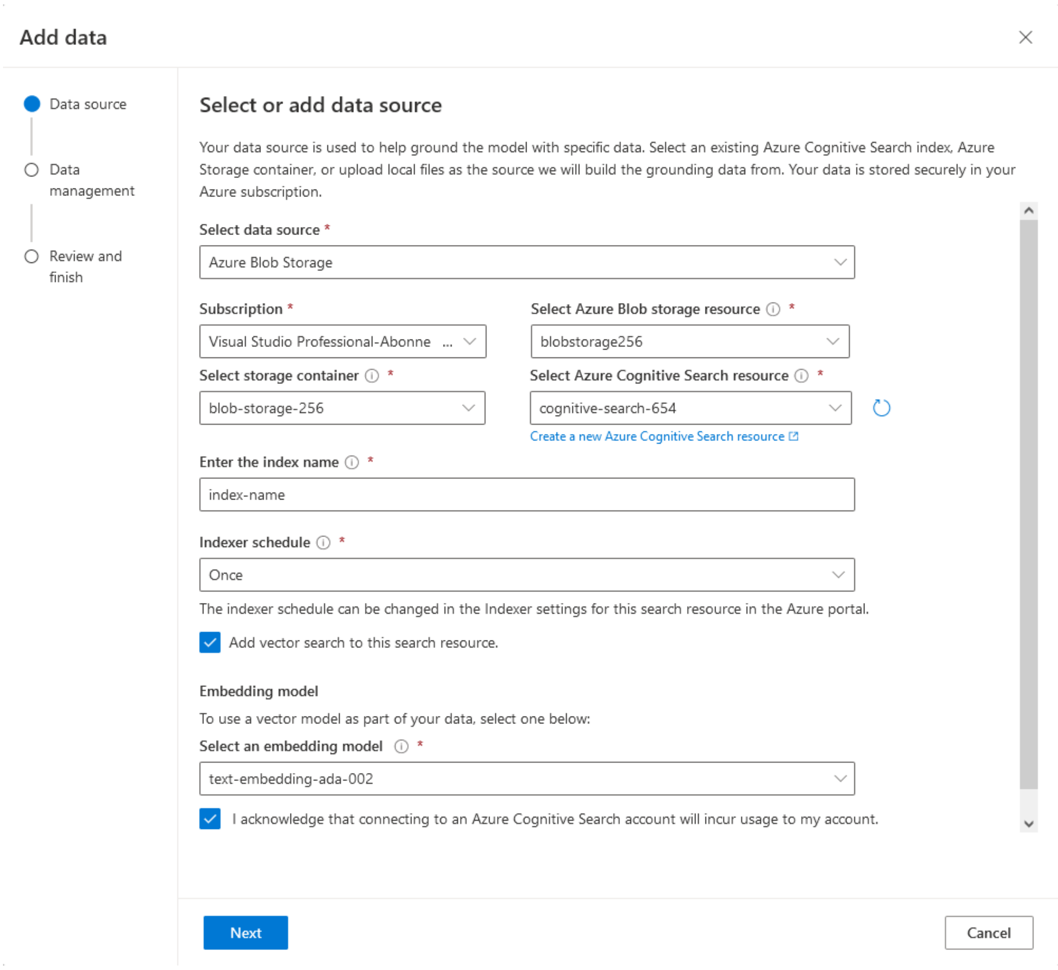
Currently (as of 01.11), you cannot use the available model version of the faster GPT 3.5 Turbo for querying your documents. Hence, let us deploy a GPT-4 model version 0613:



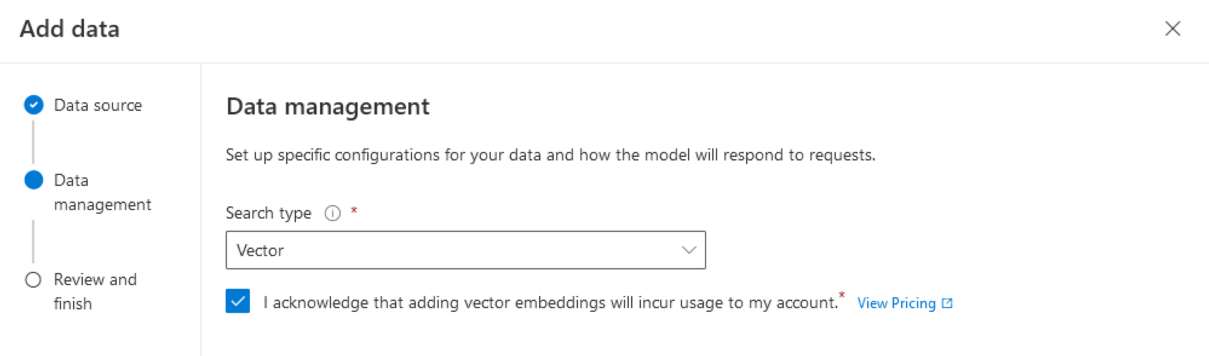
When this has completed, head back to the Chat-Playground and switch to *Add your data (preview)*.



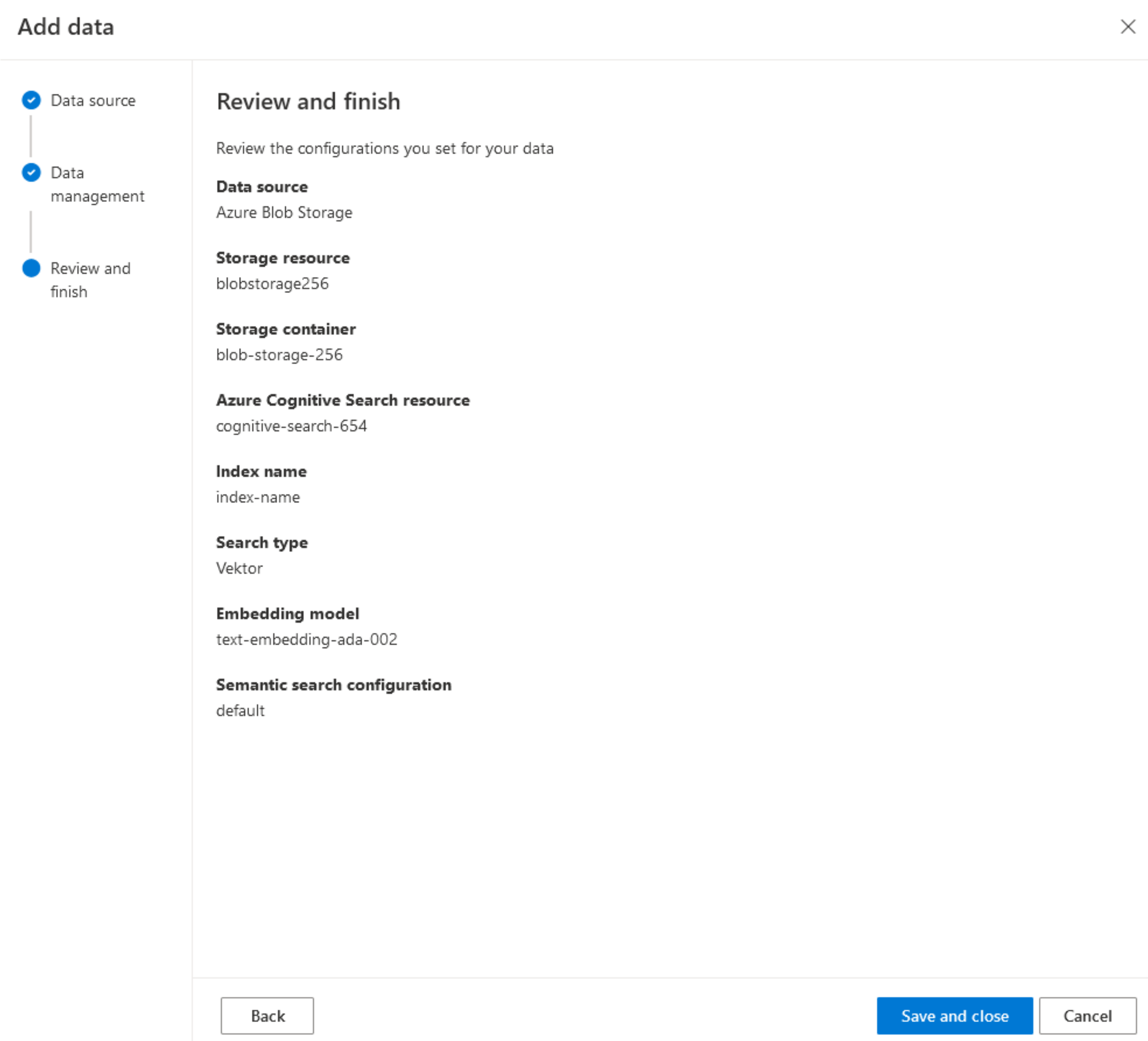
Click on *Add a data source*.



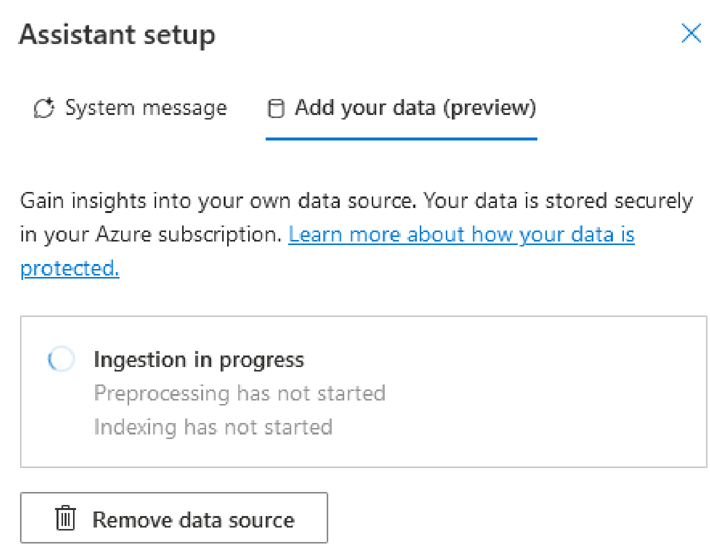
1. Select *Azure Blob Storage* as *data source.*
2. Select your *subscription* and the previously created *blob storage.*
3. Select the *storage container* and the *Azure Cognitive Search* resources.
4. Enter a name for index name.
5. Check the *Add vector search checkbox.*
6. Select your embedding model that you have created in the beginning.
7. Click on *next.*



1. Select Vector as Search type.
2. Click on *next.*



Click *Save and Close* to start ingesting your documents:

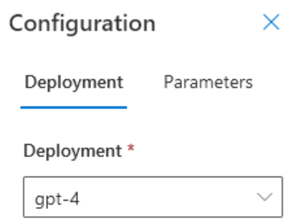


This might take a few moments, depending on the number of documents. While this process is running, let us quickly recap how this vector search works. Azure Cognitive Search will create embeddings using the selected embedding model. Embeddings are high-dimensional vectors that represent the semantic meaning of text.

A simple mental model is a flat 2D map of words with two-dimensional embedding vectors. The vectors in this model are normal Euclidian x-y coordinates. Each word gets x-y coordinates assigned through the embedding model that place it on a map based on their semantic meaning: Tiger and Lion are words that would be close to each other on that semantic map. This cluster would then be far away from words such as Car, Plane or Helicopter, which in turn would be close to each other.

Searching for “Cat” is done in the same fashion: Get the embedding of “Cat” aka the location on the map and then, looking for everything close to that location: Tiger and Lion in our case. This is usually called semantic search or vector search is exactly what will now allow you to “talk” to your own data.

You can now chat to your data. Do not forget to select the GPT-4 model on the right side under configuration if you get any error message:



Try different questions. If you have used the PDFs supplied by us, here are some examples:

- Tell me about a single project using ChatGPT.
- Does Noser realize projects that use C#?
- List some IoT projects.
- How many projects has Noser realized using Azure?

Some questions may work better than others. Try to find limitations and think about why this might happen.

Congratulations!

You have completed the workshop and should have obtained a basic understanding of LLMs and Prompt Engineering as well as how to setup ChatGPT with your own data on Azure.