# Project Phases

## Project Phase - Building a Knowledge base

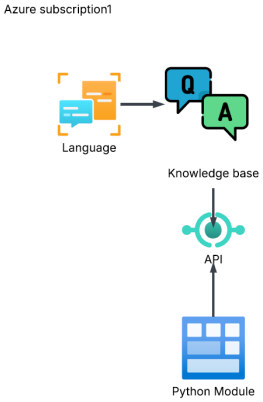
1. Build a resource based on the Azure AI Language service - Make sure to enable Custom Question and Answers.

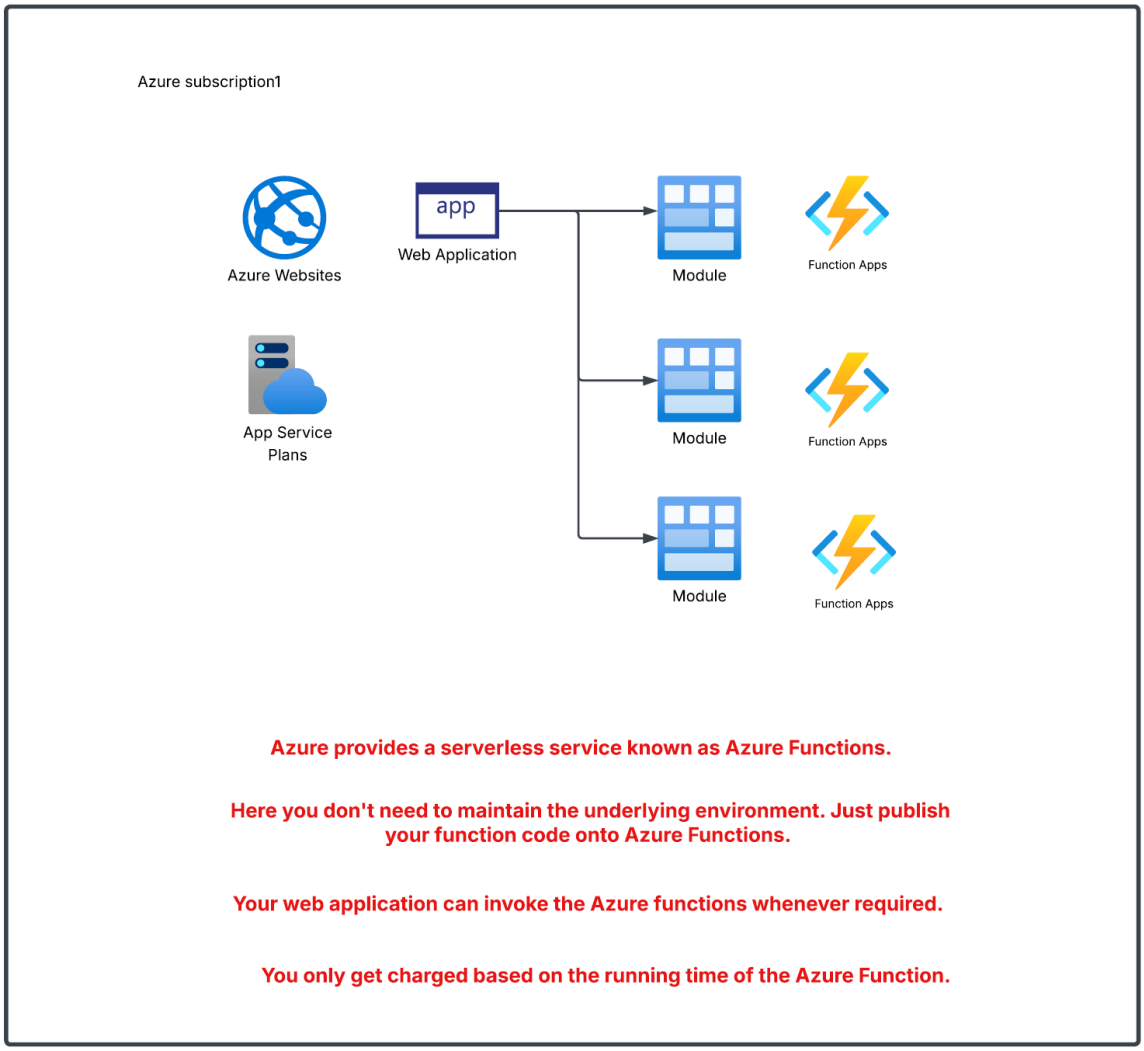
2. For the source of questions, we will import an existing set of questions - This is in the .tsv format (Tab-seperated format)

3. We will add a follow-up prompt to showcase how this works.

4. We then save and deploy our knowledge base. The knowledge base is accessible via an API

5. We then develop a simple Python program that would interact with the knowledge base.



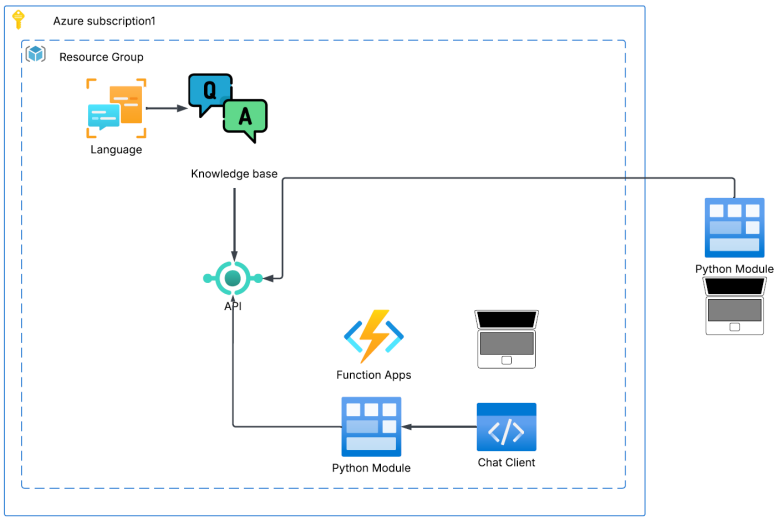



6. Now let's dpeloy a resource in Azure based on the Azure Functions service.

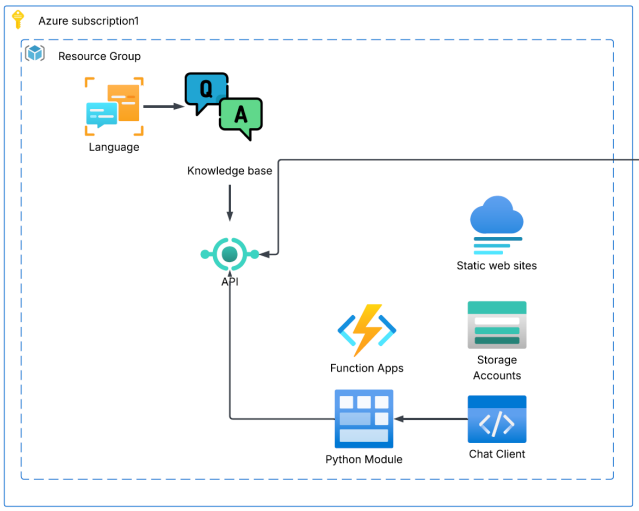
7. We need to port our module onto our Azure Function.

8. Let's test our Azure Function to check whether its working as expected.

9. Now let's build a chat client on our local machine that can interact with the Python module running in the Azure Function service.



10. Let's deploy the chat client code to an Azure storage account.




## Static Web site hosting in Azure Storage

This is a great low-cost option when you want to host a static web site that does not need a web server.

Ideal when your code is based on HTML, CSS and JavaScript.

## Project Phase - Building a Review system

1. To start off with Azure Search, we are going to upload a simple txt-based file that has reviews from a customer web site. We will upload this file to Azure Storage.

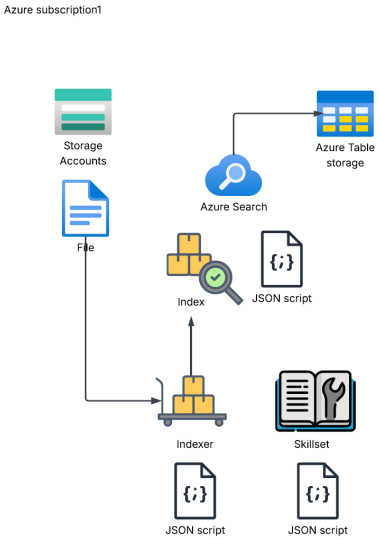
2. Next let's deploy a resource based on Azure search.

3. We will intially build the index via JSON.

4. Then we build the indexer via JSON.

5. Then we build the skillset that can be used to enrich the pipeline when the index is being built.

6. We can finally project our sentiment data onto Azure Table storage.



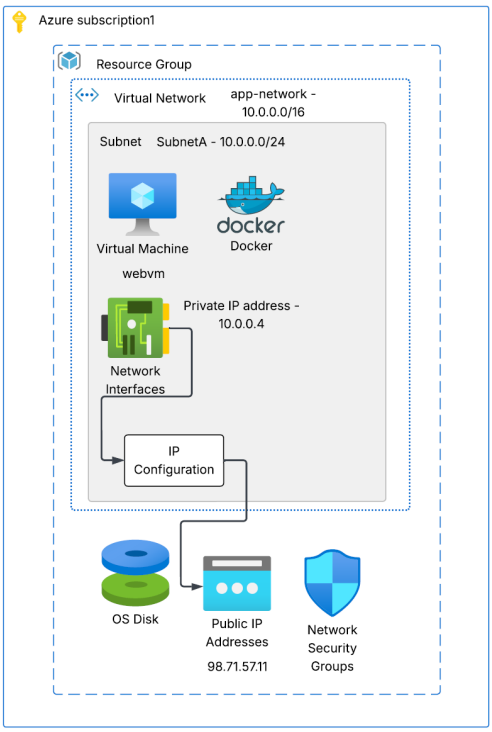
# Project Phase - Using LLM's

1. Let's first build a Python module that we can use to send our prompts to two models via deployments in Azure AI Foundry

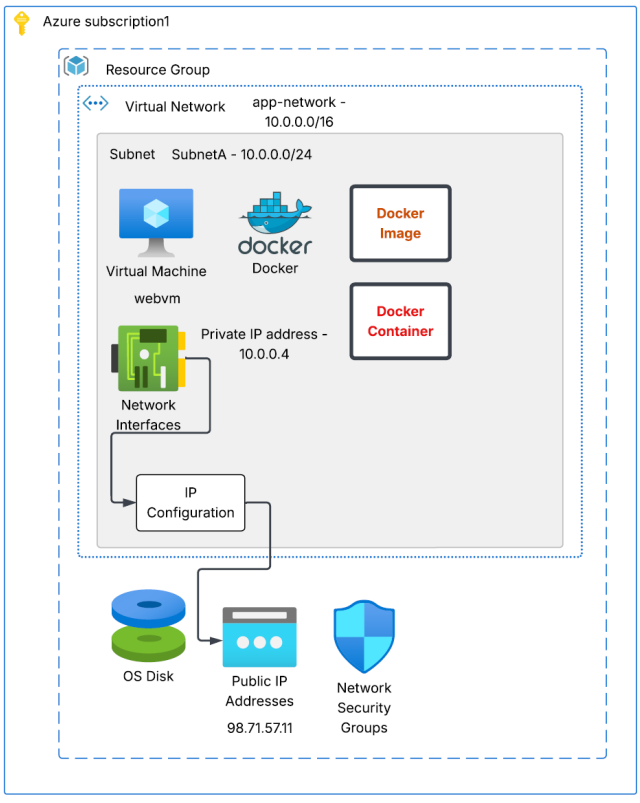
2. Next let's build an Azure Virtual Machine based on a Linux distribution.

3. Let's install Docker on the Azure virtual machine.

4. For our Python code, we will need to have an API in place that we can use to invoke the functionality of being able to send prompts to our models.





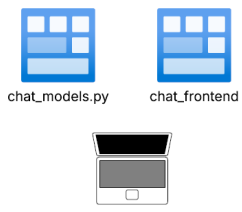



5. Now based on the code base, let's copy our files onto the Linux machine. Let's build a Docker image.

6. Once we have the Docker image in place, let's run a Docker container out of the image. We can then send a request to the application running in the container to see if we are getting the desired results back.

7. Then let's build a chat frontend in React.js. The chat frontend can then send requests to the Python application running in the Docker container.

8. The let's extend the functionality of our code to check for any harmful content via the Azure AI Content Safety service.



# Project Phase - Course Catalog Search

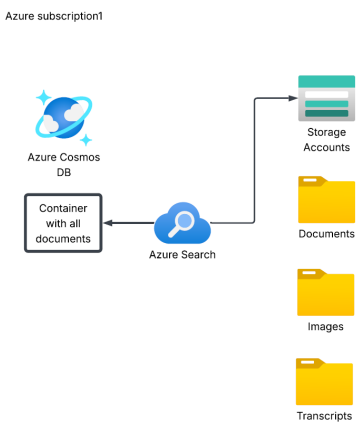
1. First we will setup Azure Cosmos DB as a data store for storing the basic information on courses. This will be based on the SQL API

2. Next we will build a search index that will provide the search capability over the data in the Azure Cosmos DB account.

3. We will perform some simple searches over the Azure Cosmos DB-based index.

4. Next we will setup an Azure storage account that stores documents, images and transcripts.

5. We will create another index that would enable search over the files in Azure Storage.



# Introduction to Azure Cosmos DB

## What is NoSQL


SQL Databases

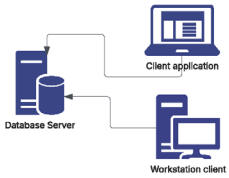
Familiar with SQL-based database engines - Oracle , MySQL, Microsoft SQL Server

| ID | Name | Description |
|---|---|---|
| C01 | AZ-104 Azure Adminsitrator | Azure Administration |
| C02 | AZ-204 Azure Developer | Azure Development |

Course

| ID | Name | CourseID |
|---|---|---|
| S01 | Mark | C01 |
| S02 | James | C02 |

Student




Applications, users connect to the database hosted on the database server.

There were issues when it came to traditional SQL database engines.

Ability to store large amounts of data - These were meant to be transactional systems and not able to manage large amounts of data.

Tables in the databases needed to have a predefined schema. But in today's world , data comes in all sizes, shapes and form.

Needed to have a more flexible way of storing data.

There were many NoSQL-based data stores developed for this purpose - MongoDB , Cassandra.

The NoSQL data stores were designed to be flexible in nature when it came to data storage.

Initially the main focus was not on building relationships across entities, but across efficient storage and retrieval of data.

Azure Cosmos DB

Azure Cosmos DB is a fully managed NoSQL, relational and vector database.

You get fast access to your data.

## Different API's

NoSQL

Data is stored in document format.

You can query for items using Structured Query Language (SQL)

MongoDB

Here documents are stored in BSON

PostgreSQL

Managed open source relational database with better performance.

Apache Cassandra

Here data is stored in a column-oriented schema.

Gremlin

This allows you to store graph-based databases.

Table

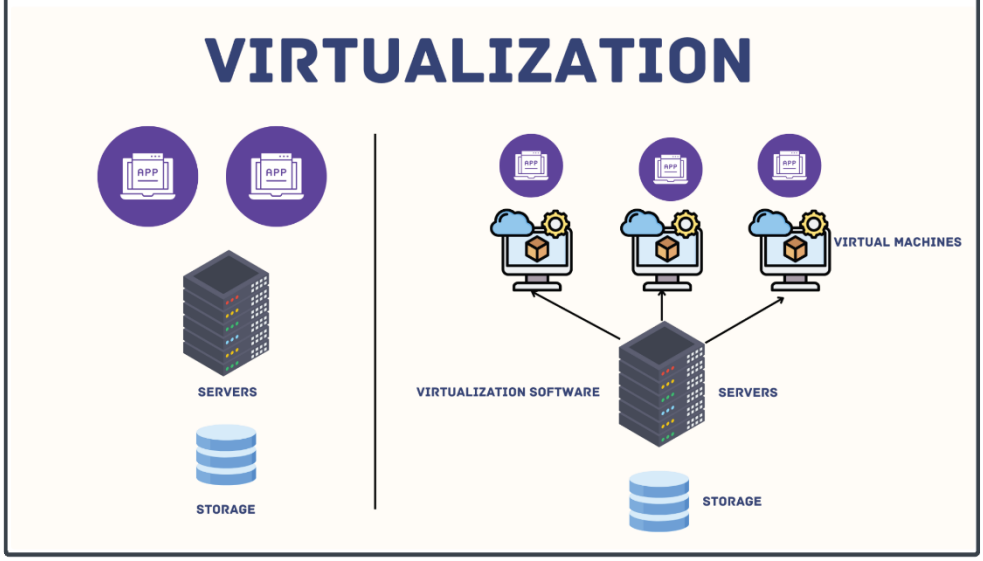
Store data in the form of key/value pairs.

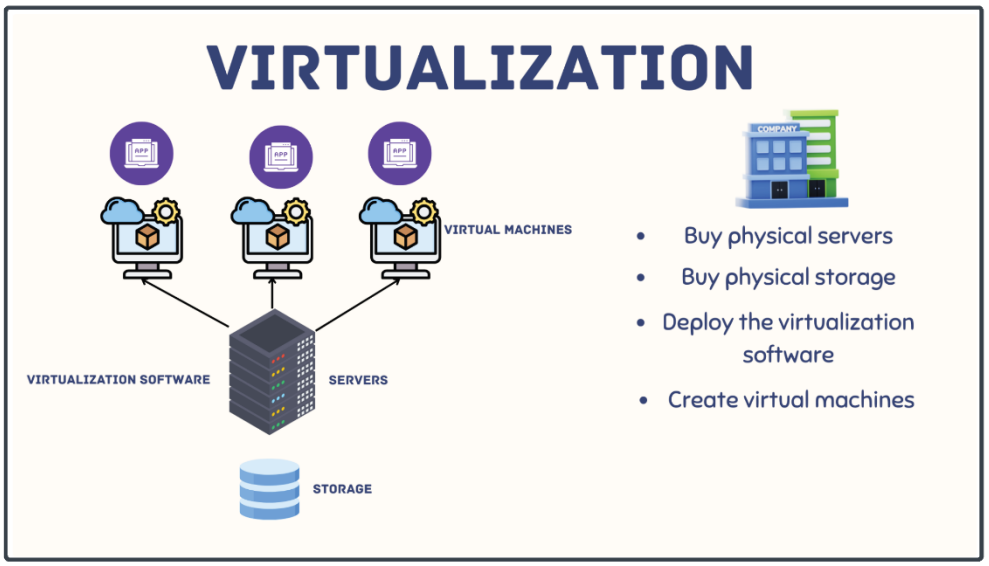
# Quick overview of Azure Virtual Machines and Docker

## The virtual machine service

This service allows you to host virtual machines without the need of buying and maintaining the underlying physical hardware.

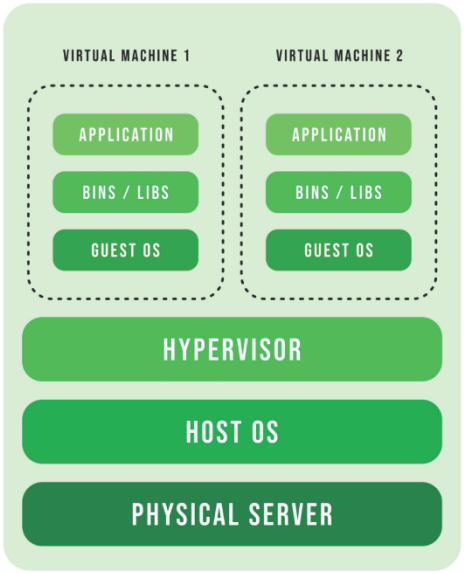
Here you only pay based on the usage of the virtual machine service.




When it comes to the Azure platform, you don't need to buy or maintain the physical infrastructure.

This is managed by Azure. You can just use the service to start deploying virtual machines.




**VIRTUAL MACHINE**
ARCHITECTURE

Virtual Machines were designed to make the most of the hardware specifications of the underlying physical machine.

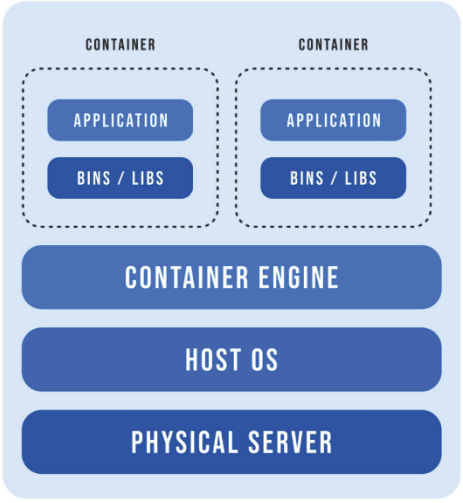
You could have multiple virtual machines hosted on the same physical server.

But then there were still some problems with it came to application deployments.

Companies would deploy multiple applications on the same virtual machine. They want to make the most of each virtual machine.

But let's say that one application requires an update at the virtual machine operating system level. This update could break the other applications running on the same virtual machine.

The other most common issue was when applications were copied from one environment to another - From development to production. The application would be perfectly working on development but would not work on production due to configuration differences.




**CONTAINER**
ARCHITECTURE

Then came the advent of container-based tools such as Docker. These tools allowed you to package your code and all dependencies required by the application within a container.

That container could the run on any machine that had the underlying supported container runtime installed such as Docker.

Each container can run independently.

You can install updates for each seperate container as required by the application and it would not affect other containers running on the same host system.

Docker is a popular container platform that provides tools to
1) Package applications into Docker images
2) Runs the images as Docker containers