Project Selected: 2

App deployed using Azure app services: https://project-appservice.azurewebsites.net/
App deployed using Azure Kubernetes Service (AKS): http://20.166.145.157:50505/

1-Python Application:

a python FLASK app was created to connect Azure CosmosDB and perfrom CRUD operation.
Here is a breakdown of files:

-HTML files:

-index.html: home page that shows a list of all books in the database
-from.html: this page is used to the enter necessary information to add a new book to DB
-info.html: this page is used to show all information about a book, edit and delete features

-Python files:

__init__.py: connecting to Azure CosmosDB (using connection string) and initializing a Flask app
routes.py: creating app routes that are used by Flask
app.py: main file to run the python application

Here is a table that shows the main Flask routes:

URL	Method	Description
/	GET	Displaying all books in DB in index.html
/book	GET	Show from.html
/book	POST	Add new book to DB
/book/<id>	GET	Display book with id:<id> in info.html
/book/<id>	DELETE*	Delete book with id:<id>
/book/<id>	PUT*	Update book with id:<id>
/book/isbn/<id>	GET	Display book with ISBN:<id> in info.html

*Since HTML doesn't natively support PUT and DELETE methods, a javascript code was used to send these requests

2-Docker Image:

The following files were created to build the image: *Dockerfile*, *.DockerIgnore, requirements.txt*
Then an image were built and run in local port to test it before uploading it.
The Steps that I used to upload the image are as follows:
1-Create Container Registry in Azure (*named: ain3003projectregistryF*), enable Admin user and obtaining the username and password
2-use docker to login in to the Container Registry, using (*docker login ain3003projectregistryF*) command in terminal
3- build the image by running (docker build -t *ain3003projectregistryF.azurecr.io/ain3003-project:latest .) in terminal*
4-push image to Container Registry by running (docker image push *ain3003projectregistryF.azurecr.io/ain3003-project:latest*)

3-Depolyment:

For deploying the image on AKS, first I created a Kubernetes service with the following features:

-2 nodes (node size : Standard_DS2_v2) each containing up to 110 pods
-price plan: Standard
-enabling anonymous pull

Then I created 2 YAML files (*mongodb-service.yaml, mongodb-deployment.yaml*) to deploy and expose MongoDB instance, and another YAML file to connect to the previously uploaded image and I used 2 replices for scalability (*deployment.yaml*), and lastly a YAML file to use a LoadBalancer (*service.yaml*).
All fles were uploaded to Azure CLI, then applied using the command (kubectl apply -f <file_name>) in Azure CLI.
Then to access the application external IP I used the command (kubectl get service) in Azure CLI.

An Azure Virtual network was also created during this process to ensure secure communication:

Image of virtual network

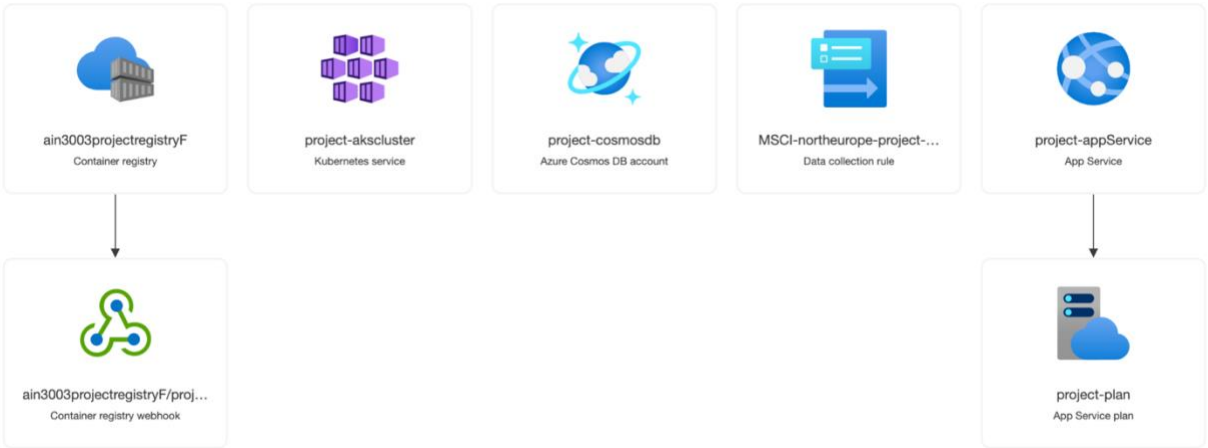

Image showing all services used in the project

A recording was also uploaded in the submission file which shows that the app is capable of performing all CRUD operations on the deployed AKS