Cloud Computing 2022/23 spring

Practice 5 - Working with Cloud databases (Azure blob storage and Cosmos DB)

In this lab, we will use Azure Cloud databases and take a look at the **Azure Blob Storage** and **Cosmos DB** database services.

Azure Blob Storage is Microsoft's object storage solution for the cloud. Blob Storage is optimized for storing massive amounts of unstructured data. Some examples of using Blob Storage used for:

- Serving images or documents directly to a browser.
- Storing files for distributed access.
- Streaming video and audio.
- Writing to log files.
- Storing data for backup and restore disaster recovery, and archiving.
- Storing data for analysis by an on-premises or Azure-hosted service.

You will use the existing account in **Azure Cloud** and set up modify the message board application to store the receive and store images in Azure Blob Storage services. You will also use the Azure **Cosmos DB** NoSQL service as the database to store the messages. In addition, You will also test the application locally and then deploy it on Azure App Services. Here is the flow of tasks:

In this practice session, we are going to re-use the message-board application code (messages stored locally in data.json) developed in Practice Session 2. We will extend the message board application by using different Azure services as mentioned below.

1. Update the home.html with an option to upload images along with text messages and further store the image into the **Azure Blob Storage** service and finally, access and display the same image on the web page along with messages.
2. Use the Azure **COSMOS DB** (No SQL database) to store the message data as a JSON document.
3. Finally, create and deploy a message board application to *'Azure App Services*.

Additional materials, tutorials, and references

- Azure Cosmos DB - https://learn.microsoft.com/en-us/azure/cosmos-db/
- Azure Blob Storage - https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction
- Azure Blob Storage Python SDK - https://learn.microsoft.com/en-us/azure/storage/common/storage-samples-python
- Azure Cosmos DB SDK - https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/sdk-python

Exercise 5.1. Introduction to application and setting up of development machine

In this task, you will update the message board application to upload images along with messages.

PS!!! You can use your machine or VM in OpenStack as a development machine.

- You can create a VM in OpenStack VM (Use ubuntu22.04).
- Install python virtual environment and Create a python virtual environment and activate as similar to Exercise 1.4 in Practice Session 1.

- You can clone your bitbucket code (message board application) used in Practice Session 3.
- Install Azure CLI `curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash` and log in to your azure account using the command `az login`.

Now, modify the application as per the below instructions:

- Update `home.html` so that users can upload an image along with their message. (Make sure your name is displayed on the web page).
  - For example, you can follow the approach explained in this guide for the home.html (However, do not include the parts about using .env, "flash" and "flash messages", they can cause conflicts when deploying to Azure later).
  - The sample code `home.html` looks like here

- Make sure you update `home.html` to display the saved image along with the text message.
- Create `./static/images` directory (inside the root Flask project directory) to store the image files.
- Update `app.py` to save the image on the disk and update data.json with a path along with the message.
  - Tip: Update the `append_message_to_file` function, so that when messages are stored on disk (data.json file), then the file path of the image is also saved in addition to the message contents (call the new property 'img_path').
  - Update the path of data.json from `/mnt/data.json` to `data.json` wherever necessary.
  - The sample code of `app.py` look like this

After these updates, the application should work something like this:

- Test the application using the `flask run --host=0.0.0.0` command.

Exercise 5.2. Working with the Azure Blob Storage

In this task, we are going to work with the Azure Blob storage. Blob storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data.

- Let us create an Azure resource group, storage account, and containers to store the images.
  - We will use Azure CLI commands as per guidelines from this manual and follow accordingly
  - PS!! Remember to replace placeholder values in angle brackets with your own values. Should ignore the angle brackets and use the values, for example *<resource-group>* should be replaced with *lab5*
  - If you have multiple Azure subscriptions then you can stick to the default Student Subscription by running the command `az account set --subscription "Azure for Students"`

Task 5.2.1: Create a Resource Group, Storage Account, and Storage Container to store the blobs.

- Create a **resource group** `az group create --name <resource-group> --location <location>`
  - name `lab5`
  - location `northeurope`
- Create a **storage account**

```
az storage account create \
    --name <storage-account> \
    --resource-group lab5 \
    --location northeurope \
    --sku Standard_ZRS \
    --encryption-services blob
```

  - Replace <storage-account> with *lab5<last_name>*. Ex: lab5poojara
- Create a **storage container** `az storage container create --account-name <storage-account> --name images`

Task 5.2.2: Create, download, and list the blobs

- Let us download the sample image file or you can use any other `wget https://estonianworld.com/wp-content/uploads/2022/03/University-of-Tartu.-Photo-by-Andres-Tennus.-1536x864.jpg -O UT.jpg`
- Upload to the blob

```
az storage blob upload --account-name <storage-account> \
--account-key <key2> \
--container-name images \
--file <path to file> \
--name <blob name>
```

  - You can get the value of key2 by using the command `az storage account keys list --account-name <storage-account>` PS!! Use the key2 value as the account-key.
  - `--file` could be `./UT.jpg` (Pointing to the local file)and `--name` could be `my_blob_UT.jpg` (To be blob file name)

- List the blob using the command

```
az storage blob list --account-name <storage-account> \
--account-key <key2> \
--container-name images \
--output table
```

- Download the blob using the command

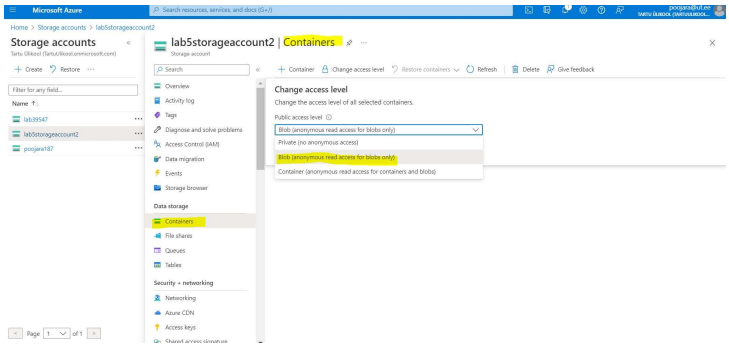
```
az storage blob download --account-name <storage-account> \
--account-key <key2> \
--container-name images \
--name <blob name> \
--file <path-to-file>
```

    - Here, <path-to-file> is a path in your local directory, where you want to store the blob. Ex: ./my_blob_image.jpg

Task 5.2.3: Working with blob permissions

Let's try to access the blobs and set the public access permissions.

- Get the URL of the blob.
    - * You can log in to the Azure portal and under Storage Accounts--><storage-account>-->Containers-->images, Click on the my_blob_UT.jpg.
    - Copy the URL and open in it browser.
    - You will see the error with ResourceNotFound. This indicates that blob doesn't have public access.
- Let us change the access level with **anonymous-read-access** as shown in the figure or the guide here. This should be done for container **images**



- Again open the URL in the browser and now should see the contents of the blob.

- You can also use the Azure CLI to set blob and container permissions. Please refer to the az command here

Exercise 5.3 Using Azure blobs in flask application

Here, we are going to modify the message board application to store the images uploaded by the end-user in azure blob storage using python SDK.

- Make sure you are under the application directory and activated the python virtual environment.
- Add python SDK for azure blob storage `azure-storage-blob==12.3.1` in `requirements.txt`.
- Set environment variables `STORAGE_ACCOUNT=<storage-account>`, `CONN_KEY=<key1>` as an environment variables.
    - In Linux can be set using *export*, for example, `export STORAGE_ACCOUNT=<storage-account>`
    - These variables are temporarily stored and need to be set again if you exit and open the terminal, So would recommend to notedown also in notepad with varibale_name and value.
    - Get the account key (key1) of storage account using `az storage account keys list --account-name <storage-account>`. Copy the `key1` part.
- Now, let us update `app.py` to use azure blobs
    - Import the Azure blob storage library `from azure.storage.blob import BlobServiceClient`
    - Read environment variables
        - `CONN_KEY= os.getenv('CONN_KEY')`
        - storage account name `storage_account = os.getenv('STORAGE_ACCOUNT')`
        - Container name `images_container = "images"`
    - Create connection client `blob_service_client = BlobServiceClient(account_url="https://"+storage_account+".blob.core.windows.net/",credential=CONN_KEY)`
    - Now, write a function `insert_blob` in `app.py` to upload the image into Azure blob.
        - The function receives input name as filename `insert_blob(img_path)`
        - Get the filename of the image `filename = (img_path).split('/')[-1]`
        - You can refer to this document for an example to upload an item to blob using the Azure blob python library.
            - Create a blob client using the local file name as the name for the blob `blob_client = blob_service_client.get_blob_client(container=images_container, blob=filename)`
            - Upload the image file to blob

```
with open(file=img_path, mode="rb") as data:
    blob_client.upload_blob(data,overwrite=True)
```

- Update `home():` method
    - Now call the function `insert_blob(img_path)` after `image.save(img_path)` with the saved image path as an argument, received from end user from the web page.

- Further, let's change how the `append_message_to_file` gets called in `home()`. We will update the `img_path` value in `data.json` to store the image Blob Storage URL instead of the local disk path. For this, you can change the input parameters of `append_message_to_file` calling method to: `append_message_to_file(blob_path,new_message)`

```
blob_path = 'https://'+storage_account+'.blob.core.windows.net/'+images_container+'/'+image.filename
append_message_to_file(blob_path,new_message)
```

- Install the requirements.txt packages
- Test the application by adding few message and images
- (Not mandatory) You can also add the code to handle exceptions, when no images uploaded by the user.
- Deliverable: Take a screenshot of the terminal output by running the command to list the blobs (az storage blob list).

Exercise 5.4 Using COSMOS Database in flask application

Azure Cosmos DB is a fully managed NoSQL database service for modern app development. In this task, we are going to use COSMOS DB to store the messages in the database as similarly used in serverless functions. But we use COSMOS python SDK to interact with azure cosmos DB.

- Create a COSMOS DB No SQL database as similar to Exercise 4.2.
  - Resource group should be `lab5` as created in previous Excercise.
  - You can choose
    - Account name: lab5cosmoslast_name (Ex: lab5cosmospoojara)
    - Database id: lab5messagesdb
    - Container id: lab5messages

We are modifying the message-board flask application to store the messages in the database instead of locally at `data.json`. Here we will modify `app.py` to interact with cosmos db.

- Make sure that your under the project directory in the development machine
- Add entry of `azure-cosmos` in `requirements.txt`
  - Don't forget to do `pip install`
- Now let us get the endpoint COSMOS_URL and Master key to access the COSMOS DB using python SDK, You can use Azure CLI for this and set the values as environment variables (**COSMOS_URL** and **MasterKey**)
  - Replace <resource-group> and <account-name> (COSMOS DB account name) in the command
    - Endpoint **COSMOS_URL** `az cosmosdb show --resource-group <resource-group> --name <account-name> --query documentEndpoint --output tsv`
    - **MasterKey** `az cosmosdb keys list --resource-group <resource-group> --name <account-name> --query primaryMasterKey --output tsv`

- Recommend to notedown COSMOS_URL, DATABASE_ID,CONTAINER_ID,MasterKey in notepad.

- Now let us modify the `app.py`
  - Import the cosmos lib `import azure.cosmos.cosmos_client as cosmos_client`
  - Declare COSMOS_URL, MasterKey
    - Values can be gathered for example `COSMOS_URL= os.getenv('COSMOS_URL')`
  - Declare the variables
    - DATABASE_ID='lab5messagesdb', CONTAINER_ID='lab5messages'
  - Create cosmos client `cosmos_db_client = cosmos_client.CosmosClient(COSMOS_URL, {'masterKey': MasterKey} )`
  - Database client for connection `cosmos_db = cosmos_db_client.get_database_client(DATABASE_ID)`
  - Container connection string `container = cosmos_db.get_container_client(CONTAINER_ID)`
  - Write a function `insert_cosmos`, as similar to `append_message_to_file`
    - Which takes `content` and `img_path` as input
    - Add one more element `id` in **new_message** json and value can be generated using uuid.uuid4()
    - Insert a item in to the data base and also need to import `import azure.cosmos.exceptions as exceptions`

```
    try:
        container.create_item(body=new_message)
    except exceptions.CosmosResourceExistsError:
        print("Resource already exists, didn't insert message.")
```

- Write a function to read the messages from the cosmos `read_cosmos` as similar to `read_messages_from_file`
  - Declare list to store the messages `messages = []`
  - Read the items from cosmos `messages = list(container.read_all_items(max_item_count=10))`
  - return the data `messages`

- Now, update the home function of `app.py`
  - Replace `read_messages_from_file()` with `read_cosmos`
  - Replace `append_message_to_file` with `insert_cosmos`
  - Update `render_template`-s 2nd argument to "*data*" instead of *data["messages"]*
- Test the application
- Test the application by inserting the messages and images.
  - Deliverable: Take a screenshot of the development machine terminal with the command output of "flask run". The screenshot should contain a few GET and POST operations handled by the flask application.

Exercise 5.5 Deploying application in Azure App Service

We are going to use Azure git-based deployment of message-boad application to Azure App Service. We will use Azure CLI to perform the tasks.

- Initially, need to create a Azure Service Plan `az appservice plan create --resource-group lab5 --name lab5plan --is-linux --sku F1`
- Now, create a Azure APP Service which is git-enabled deployment `az webapp create --resource-group lab5 --plan lab5plan --name <app-name> --runtime "PYTHON:3.9" --deployment-local-git`
    - Replace <app-name> with lab5<last_name>app (Ex: lab5poojaraapp)
    - The output of the above command contains a URL like: https://None@<app-name>.scm.azurewebsites.net/<app-name>.git with some other json data.
- Set the remote repository
    - Get the credetinals (Username and Password) used to push the code to Azure git

`az webapp deployment list-publishing-credentials --name <app-name> --resource-group lab5 --query "{Username:publishingUserName, Password:publishingPassword}" --output table`

- This command output the username and password used to push in the code in next step.
- Add remote URL `git remote add azure https://<deployment-username>@<app-name>.scm.azurewebsites.net/<app-name>.git`, here URL should for example `https://$lab5xx@lab5xx.scm.azurewebsites.net/lab5xx.git`
- Push the code to master branch `git push azure master:master`. It will ask for a username and passord, use the credetinals obtained in the previous step.
    - You will see the set of application deployment logs.
- After successful deployment, open the application using URL https://<app-name>.azurewebsites.net/. However, you will see the application with Error Message. This is because you need to create the Application Settings (Environment Variables).
- Now let us create the application settings
    - Blob-related variables: STORAGE_ACCOUNT, CONN_KEY
    - Cosmos DB related varibales: COSMOS_URL,MASTER_KEY

`az webapp config appsettings set --name <app-name> --resource-group lab5 --settings STORAGE_ACCOUNT=$STORAGE_ACCOUNT CONN_KEY=$CONN_KEY COSMOS_URL=$COSMOS_URL MASTER_KEY=$MASTER_KEY`

- You need to modify `app.py` to access these variables. You should prefix with APPSETTING_ for application setting variables. For example `STORAGE_ACCOUNT=os.getenv('APPSETTING_STORAGE_ACCOUNT')`.
- Commit the git code `git add . && git commit -m "Modifed app.py for app settings"`
- Deploy the code again `git push azure master:master`
- Get the logs using in your terminal `az webapp log tail --name <app-name> --resource-group lab5`
- Open your application in the browser.
- Deliverable: Take screenshot of the web page of the application containing message and image (Your web address should be visible).

PS! You can go to the development web interface of your App service to get a nice overview of logs, parameters and files:

- https://<APP_SERVICE_NAME>.scm.azurewebsites.net/

Deliverables:

1. Screenshots from Exercise 5.3, 5.4 and 5.5
2. Application code
3. Link to your messageboard deployed to App Services

Task	Lab 5 - Databases
Current submission	ZIP (01:44 14.03.2023)
	If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
Report	Choose File No file chosen
File	Choose File No file chosen
Comment	https://lab5kadalipp.azurewebsites.net/
	Submit

In case of issues, check these potential solutions to common issues:

- If you experience an error message "Sorry, we are currently experiencing high demand in this region ..." on service creation, than choose the different **Region** for example Sweden Central
- If you get an error about `./static/images` folder or `./static/images/somefile.jpg` not existing
  - Make sure you created the `static/images` folder inside the Flask project **root** folder.
- If the application does not load.
  - If you have not used the application for a while, you are likely experiencing the **Cold-Start** issue, and it may take several minutes for Azure to provision your code again in the cloud.
- If you get an Error in Azure logs:
  - ```
    2023-03-15T15:30:24.126415205Z if not 0 <= time_low < 1<<32L:
    2023-03-15T15:30:24.126427005Z ^
    2023-03-15T15:30:24.126432005Z SyntaxError: invalid syntax
    ```
  - You should not use `uuid` in requirements.txt file.
  - rebuilding the Python virtual environment (.env folder) from scratch without custom uuid, removing uuid from requirements.txt and redeploying should solve i

The courses of the Institute of Computer Science are supported by following programs:


    ```