Cloud storage and Retrieval - EXL Hackathon

Author(s): Sri Dhanuja R, Sri Hari KV
Date : July 10,2022
Reach us sridhanujaravi@gmail.com

Table of Contents

Problem Definition

The current solution processes backend requests and generates reports in .pdf, .csv, or .pdm file formats. These reports are then uploaded on an on-premises shared file server. Since the solution is now hosted on cloud, a file-server-based model is not the optimum solution for file storage.

Though transitioning file storage from on-premises to cloud can be achieved easily, clients have their choices of cloud provider, which requires customization for every transition. A cloud-agnostic storage solution can address this problem.

Objective

Aim of this project is to build a solution that is agnostic of the cloud provider (AWS S3 bucket, Azure Blob storage, and Google Cloud storage) for storing and retrieving files and deploying this in AWS instance.

Tools and Technologies used

- Operating system : Ubuntu, Windows
- Programming Language : Python 3.7
- Framework : Flask
- Tool Used : Microsoft Visual Code Studio
- Cloud storage : Azure, AWS and Google
- Virtual server : AWS

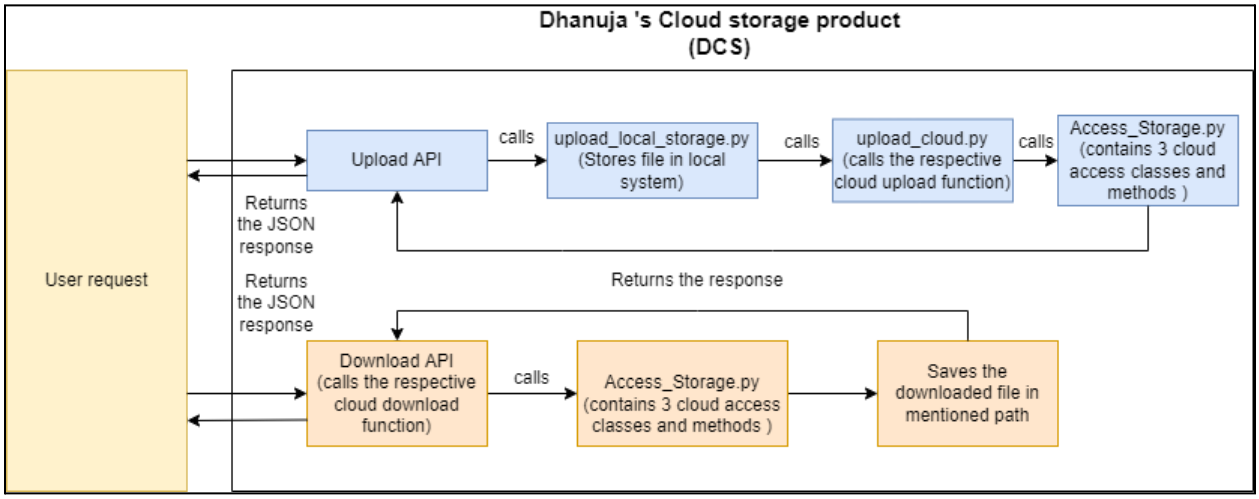
Functional description

The product is composed of the following functional components :

1. Upload_to_cloud API will upload the file to specified cloud storage. This API will get two inputs. One is file to be uploaded, one is cloud storage name to know where to upload the file
2. Download_using_name API will download the specified file/blob from the mentioned cloud storage and store it under the mentioned path. This API will get three inputs. One is file/blob name to be downloaded, another one is cloud storage name to know from where the file is to be downloaded and third one is storage path to know where to store the files. If the input value for storage path is not given, it will store it in the default location and if the cloud storage name is not mentioned, then default value will be taken. (default value for cloud storage is “gcp_storage”)
3. Azure_Storage class which comprises methods such as create_container to create a container, delete_container to delete a container, list_blobs to get the list of blobs name in the azure storage container, get_blobs_url to get the url of the blobs in the azure storage container, upload to upload the file, download_blob to download the file using the name, download_using_url to the blob by getting url of the blob as an input, download_using_listofdicturl to download multiple blobs at a stretch by giving blob name and url as an input in list of dictionary format, get_folder_name to return the blobs url by taking the virtual hierarchy name/folder name as an input. It will return only the url of the files present under the given folder name.
4. Google_Storage class which comprises methods such as download to download the file using name, download_all_files to download all files under the container, download_all_folder_files to download all files in specified folder, upload to upload the file, delete to delete the file and delete_folder to delete the folder.

5. AwsS3 class which comprises methods such as upload to upload the file, delete to delete the file and delete_folder to delete the folder, download to download the file using name, download_all_files to download all files under the bucket.

Architecture Diagram



Deployment Process

This project uses AWS, Azure and Google cloud storage accounts and AWS instances.

I Prerequisites

- **Storage account Prerequisites**

1. Create a storage account in AWS, Azure and Google cloud storage.
2. Get and Save the bucket name aws access key id and aws secret access key from the AWS account. Since it the key to access the storage account, don't share it with others
3. Get and Save the Azure connection string, container name, credential (password), account url (useful while accessing the blob using the blob url) from the Azure account.

4. Get and Save the Google application credential json file and bucket name, from the google storage account.
5. Since the values in step 2 to 4 is used to access the account, don't share it with anyone

- **Deployment Prerequisites**

  Microsoft Visual Studio code - To easily access the script, edit and run the project. Connection to AWS VM instances can also be done using VS code

  To deploy in AWS VM instances

  - Ubuntu 18.04 server instance with a non-root user with sudo privileges configured
  - Install Conda Environment

    Steps to install Miniconda

    1. Download Miniconda3 from the internet

    `wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh`

    2. Make the file executable

    `chmod +x Miniconda3-latest-Linux-x86_64.sh`

    3. To set up miniconda

    `bash Miniconda3-latest-Linux-x86_64.sh`

  To deploy in local system:

  - Windows
  - Install Python and anaconda/miniconda

II Deployment process

1. Deploy in local system:

Create Conda Environment with python=3.7 version

To create conda environment use following command:

```
% conda create --name <conda-env name> python==3.7.7
```

Activate the conda environment

```
conda activate <conda-env name>
```

Create and Configure Flask Project

Download and unzip the project folder

Install the packages given in requirements.txt by using the following command.

```
`pip install -r requirements.txt`
```

Create environment variables to set the values (used to access the storage)

Example $Env:aws_secret_key = "sdfg"

Like this set env variables for all needed keys.

Needed key for this project.

For Azure,

Azure_connection_string, container_name, account_url, Credential

For GCP,

GOOGLE_APPLICATION_CREDENTIALS, bucket_name,location,storage_class

For AWS

aws_access_key_id,aws_secret_access_key,bucket_name_aws

Run the code

- Make sure conda env is activated or activate the conda environment
  Conda activate <conda-env name>
- Go to the root folder of the IN-D One application
  to run locally use
  python FlaskApp.py

2. Deploy in AWS instance

1. **Installing Packages from Ubuntu**

   sudo yum update

   sudo yum install python3-pip python3-dev libpq-dev

2. Do step 1 (creating new conda env) in "Deploy in local system"
3. Connect to VM

   Open vs code and in "open remote window" select "Open SSH configuration file"

   Select the file and open it

   Create 4 lines like below

```
Host <give any name as your host name>
  HostName <ip address/hostname of AWS VM instance>
  User <username>
  IdentityFile <location of downloaded pem file while creating VM
instance>
```

   Save this configuration file and again click the "open remote window" and select your host.

Create and Configure Flask Project

Upload the zip file to VM or use WINSCP or any file transfer to transfer the project zip file/ folder to the VM. If in zip, unzip it

Create Conda Environment with python=3.7 version

To create conda environment use following command:

% conda create --name <conda-env name> python==3.7.7

Activate the conda environment

conda activate <conda-env name>

Install the packages given in requirements.txt by using the following command.

`pip install -r requirements.txt`

Create environment variables to set the values (used to access the storage)

Example

$ export aws_access_key_id=" ASNFWOIJ487KNSDF"

Like this set env variables for all needed keys.

Needed key for this project.

For Azure,

Azure_connection_string, container_name, account_url, Credential

For GCP,

GOOGLE_APPLICATION_CREDENTIALS, bucket_name,location,storage_class

For AWS

aws_access_key_id,aws_secret_access_key,bucket_name_aws

Configure Flask and Gunicorn - set server_name in nginx as server's public ip address

Run the code

- Make sure conda env is activated or activate the conda environment

  `Conda activate <conda-env name>`

- Go to the root folder of the IN-D One application

  to run locally use

  `python FlaskApp.py`

  To run using Gunicorn

  gunicorn -b 0.0.0.0:5000 FlaskApp:app

API Explanation

1. **Upload_to_cloud - POST**

   Endpoint

   http://127.0.0.1:5000/api/upload

   Content-Type : multipart/form-data

   Body

{

"file":binary file,

“Storage_cloud_name”: azure_storage

}

Other values for “Storage_cloud_name” - “aws_storage”, “gcp_storage”

Response


```
{
message: "uploaded Successfully!!!",
accepted_files:[
"1200065020471sep_0Page.jpg"
],
denied_files:[]
}
```


2. **Download_using_name - POST**

Endpoint

http://127.0.0.1:5000/api/download_blob_using_name

Content-Type : multipart/form-data

Body

{

“Storage_path” : “/home/ec2-user/Dhanuja_cloud_upload/downloaded_file”,

"blob_name" : “1200065020471sep_0Page.jpg”,

“Storage_cloud_name” :”aws_storage”

}

Other values for “Storage_cloud_name” - “aws_storage”, “gcp_storage”

Response


```
{
message: "Successfuly downloaded"
```

```
}
```

Conclusion

A python script called “Access_Storage.py” has all methods to access the AWS, Azure and Google cloud storage. REST API using flask is created to upload the file to AWS, Azure and Google cloud storage and download the file from the AWS, Azure and Google cloud storage. Future enhancements can be deploying in other Cloud VM instances and creating API to utilize all the methods in Access_Storage.py