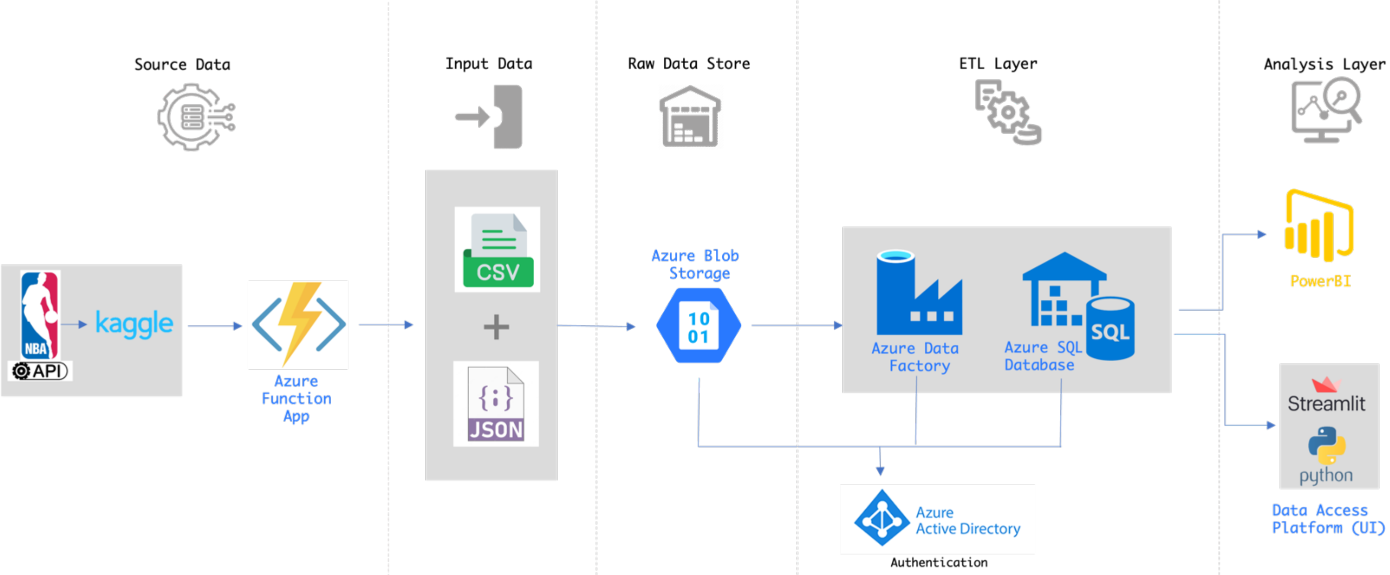
DAMG 7275 : Advanced Data Management Systems
Project P4 Submission

Topic : NBA Game Analytics
Team : Team 6 – Azure SQL Multi-model
Team Members :
Krishika Singh- 002194016
Mohammad Rafi Shaik- 001525707
Shika Shyam- 002194543
Shreya Soni – 002146758

Implementation Process :

Our Updated Project architecture accounting for Data Refresh is as shown below :



- The data we are using is available in the NBA dataset provided by Kaggle. The data is in the form of multiple csv files for different entities.
- Since these CSV files are not normalized and has a lot of redundant columns we will be needing extensive preprocessing before we can use the data for reporting.

Our Implementation Process consists of three phases :

Step 1 : Raw Data Layer

- We are using Azure Blob Storage to store the csv files from Kaggle.
- We are also using Blob storage as the staging area for JSON documents.

Step 2 : ETL Layer

- Azure Data Factory is our ETL tool. We are using Data Copy activity for some entities and full-fledged data flows for others.
- We have also enabled both schedule based and storage event-based triggers.

Step 3 : Foundation Data Layer

- The sink of our ETL pipelines point to Azure SQL tables.
- We are storing data as tables, nodes and edges for graphs as well as document.

New features added for P4 :

Ongoing data refresh :

Like mentioned in the previous steps as well in P3, our source data comes from Kaggle. But if we go more in depth, the data in Kaggle is created as a part of a Python script written by the Kaggle dataset author, which uses the publicly available NBA data feed API and creates csv files from the real NBA data based on NBA games everyday. These consolidated csv files are then uploaded to Kaggle by the data owner.

The main step in our dynamic pipeline which refreshes the data in our database as data changes in source is to access this Kaggle data everyday just as it is updated and then use that new data to refresh the data in our tables in the Foundation Data Layer and consequently in the reporting layer.

For this we make use of Azure Functions, which inherently runs a Python script, that does the following steps:

1. Use the Kaggle user's API to request data from Kaggle for our NBA dataset
2. Once the data is available, read in the csv files
3. Write these csv files to our blob storage where we store the raw data
4. This function is scheduled to run every two days as a trade-off between data availability and Azure compute costs

The Azure function scripts are below :

```python
import logging
import datetime
import os
import azure.functions as func
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient

# Set Kaggle API credentials directly
os.environ['KAGGLE_USERNAME'] = 'shshyam'
os.environ['KAGGLE_KEY'] = ''

from kaggle.api.kaggle_api_extended import KaggleApi
```

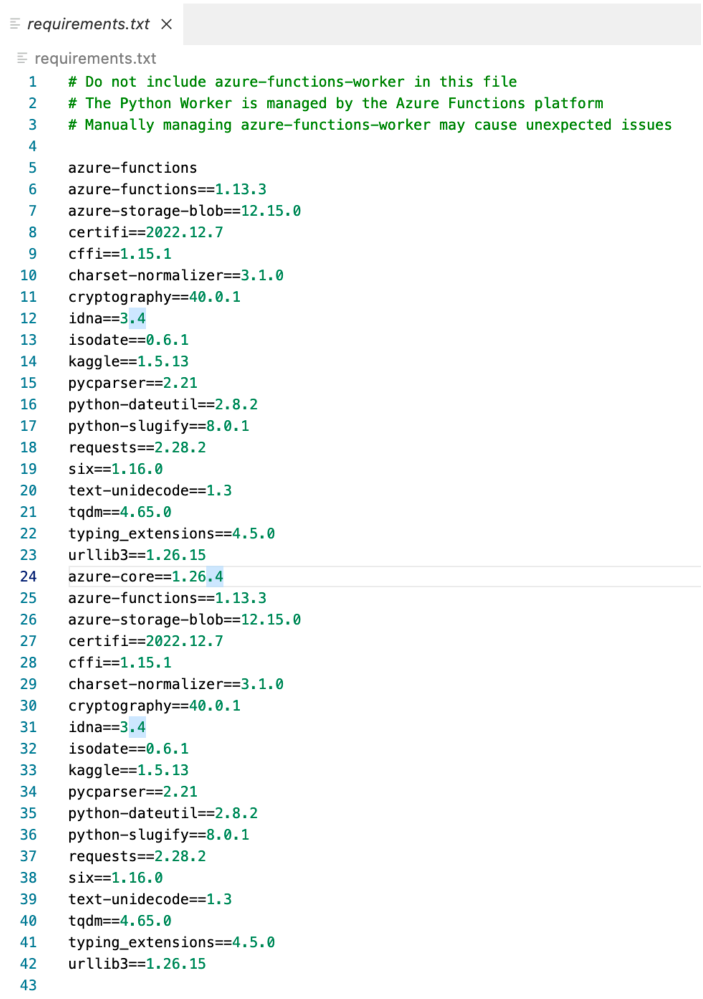
```
def main(req: func.TimerRequest, outputBlob: func.Out[func.InputStream]) -> str:
    utc_timestamp = datetime.datetime.utcnow().replace(
        tzinfo=datetime.timezone.utc).isoformat()

    logging.info('Python timer trigger function ran at %s', utc_timestamp)
    logging.info('Python timer trigger function ran at %s', req)

    api = KaggleApi()
    api.authenticate()

    # Download the dataset (replace 'dataset-id' with the actual ID of the dataset you
want to download)
    api.dataset_download_files('wyattowalsh/basketball', path="/tmp")

    # Specify the absolute path of the target directory
    target_dir = "/tmp"

    # Unzip the downloaded files
    import zipfile
    with zipfile.ZipFile('/tmp/basketball.zip', 'r') as zip_ref:
        zip_ref.extractall(target_dir)

    connect_str = os.environ['AzureWebJobsStorage']

    # Change this to nbadataset once code is finalized. Not adding it now because it
will trigger storage based event
    # trigger.
    container_name = 'nbadataset'

    directory_path = '/tmp/csv'

    blob_service_client = BlobServiceClient.from_connection_string(connect_str)

    container_client = blob_service_client.get_container_client(container_name)

    for filename in os.listdir(directory_path):
        if filename != "play_by_play.csv":
            file_name = filename
            logging.info('%s is being uploaded.....', filename)

            file_path = os.path.join(directory_path, filename)

            blob_path = file_name

            blob_client = container_client.get_blob_client(blob_path)

            with open(file_path, "rb") as data:
                blob_client.upload_blob(data, overwrite=True)
    logging.info('Python HTTP trigger function processed a request.')
```

The above python script does the requirement of going through each csv and uploading them individually to our raw data store. i.e. **nbadataset** container in Azure Storage Account.

The below JSON defines the Azure Function as a CRON job, and schedules it for every 2 days at 8PM.

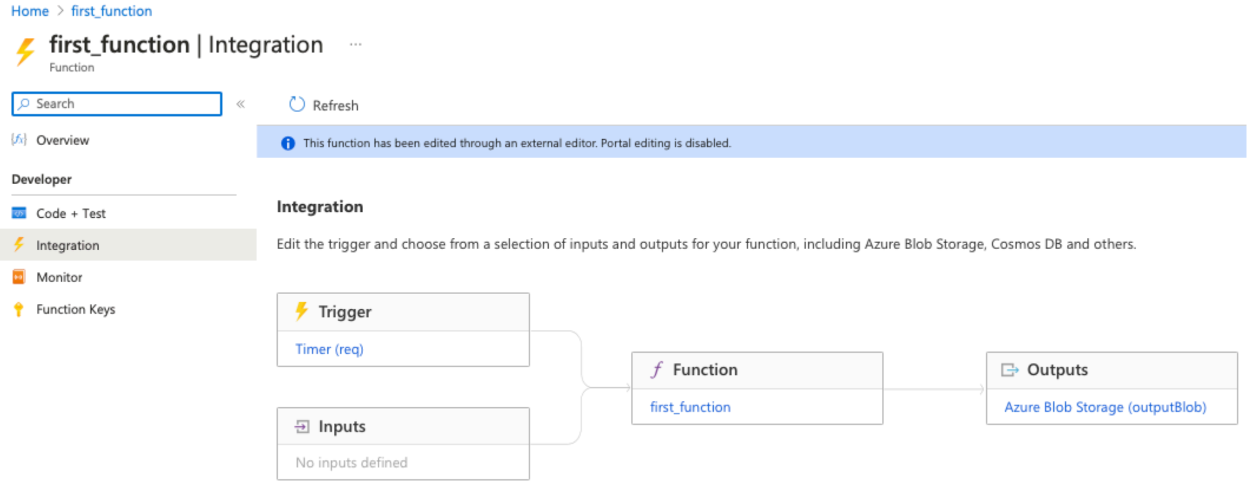
```
{
  "bindings": [
    {
      "name": "req",
      "type": "timerTrigger",
      "direction": "in",
      "schedule": "0 20 */2 * *"
    },
    {
      "name": "outputBlob",
      "type": "blob",
      "direction": "out",
      "path": "nbadataset/{name}.csv",
      "connection": "AzureWebJobsStorage"
    }
  ],
  "disabled": false
}
```

Additionally, we also include a requirements.txt file, so that the Azure Function can inherently download and install dependencies for our python script to run.



```
# Do not include azure-functions-worker in this file
# The Python Worker is managed by the Azure Functions platform
# Manually managing azure-functions-worker may cause unexpected issues

azure-functions
azure-functions==1.13.3
azure-storage-blob==12.15.0
certifi==2022.12.7
cffi==1.15.1
charset-normalizer==3.1.0
cryptography==40.0.1
idna==3.4
isodate==0.6.1
kaggle==1.5.13
pycparser==2.21
python-dateutil==2.8.2
python-slugify==8.0.1
requests==2.28.2
six==1.16.0
text-unidecode==1.3
tqdm==4.65.0
typing_extensions==4.5.0
urllib3==1.26.15
azure-core==1.26.4
azure-functions==1.13.3
azure-storage-blob==12.15.0
certifi==2022.12.7
cffi==1.15.1
charset-normalizer==3.1.0
cryptography==40.0.1
idna==3.4
isodate==0.6.1
kaggle==1.5.13
pycparser==2.21
python-dateutil==2.8.2
python-slugify==8.0.1
requests==2.28.2
six==1.16.0
text-unidecode==1.3
tqdm==4.65.0
typing_extensions==4.5.0
urllib3==1.26.15

```

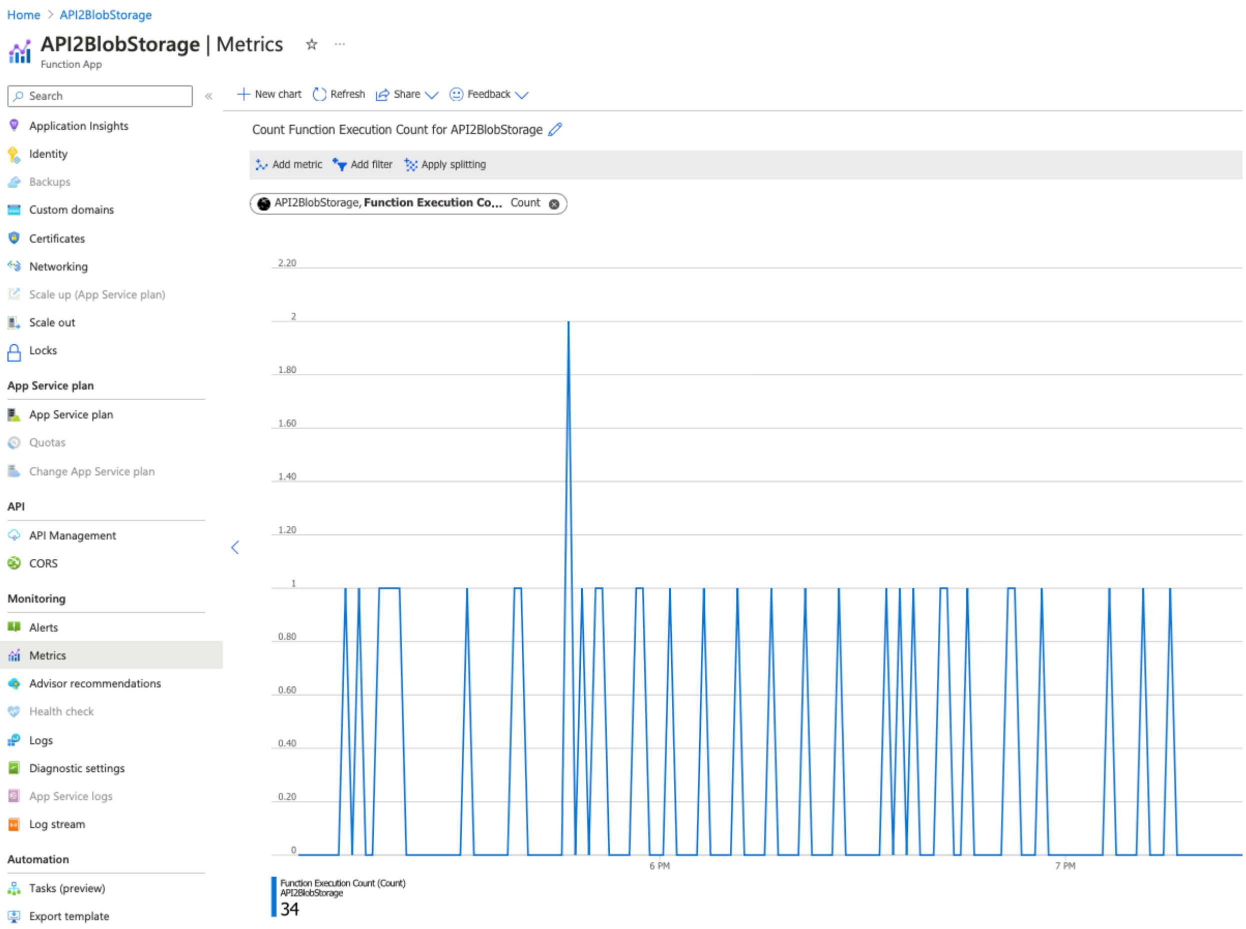
Below is a screenshot of our Azure Function set up in Azure Portal.

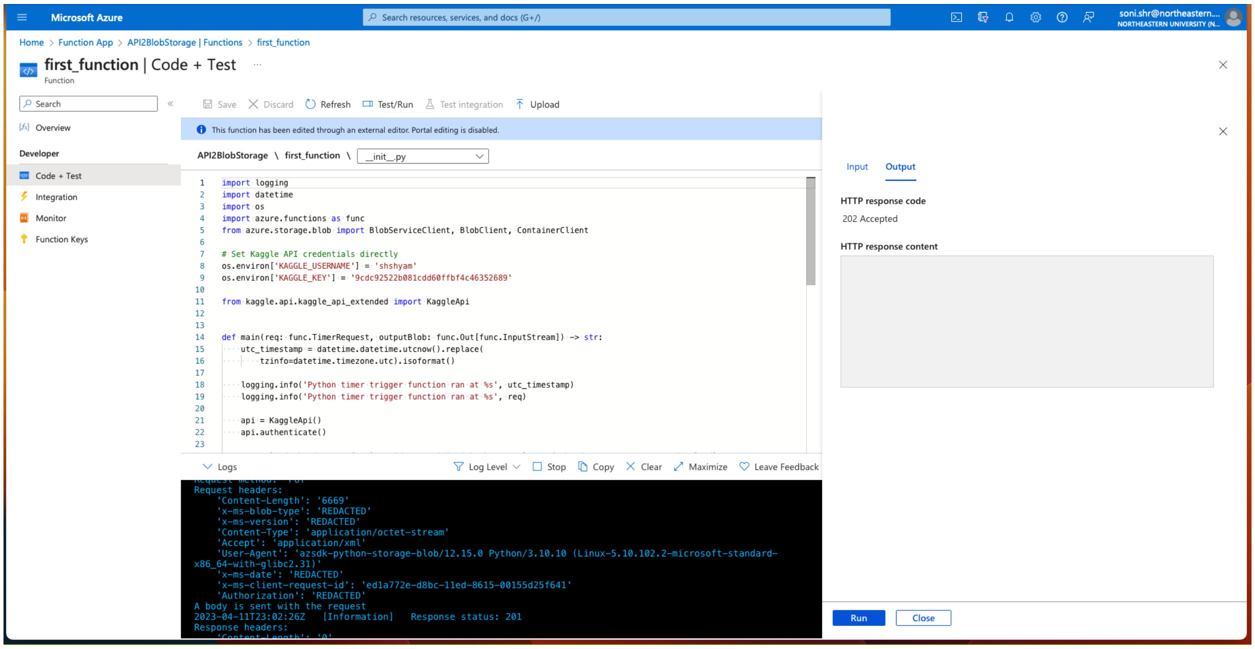


This function has been edited through an external editor. Portal editing is disabled.

Integration

Edit the trigger and choose from a selection of inputs and outputs for your function, including Azure Blob Storage, Cosmos DB and others.




Microsoft Azure
Search resources, services, and docs (G+/)
soni.shr@northeastern...
NORTHEASTERN UNIVERSITY (N...
Home > Function App > API2BlobStorage | Functions > first_function
first_function | Code + Test
Function
Search
Overview
Developer
Code + Test
Integration
Monitor
Function Keys
Save
Discard
Refresh
Test/Run
Test integration
Upload
This function has been edited through an external editor. Portal editing is disabled.
API2BlobStorage \ first_function \ __init__.py
Logs
Log Level
Stop
Copy
Clear
Maximize
Leave Feedback
Input
Output
Run
Close

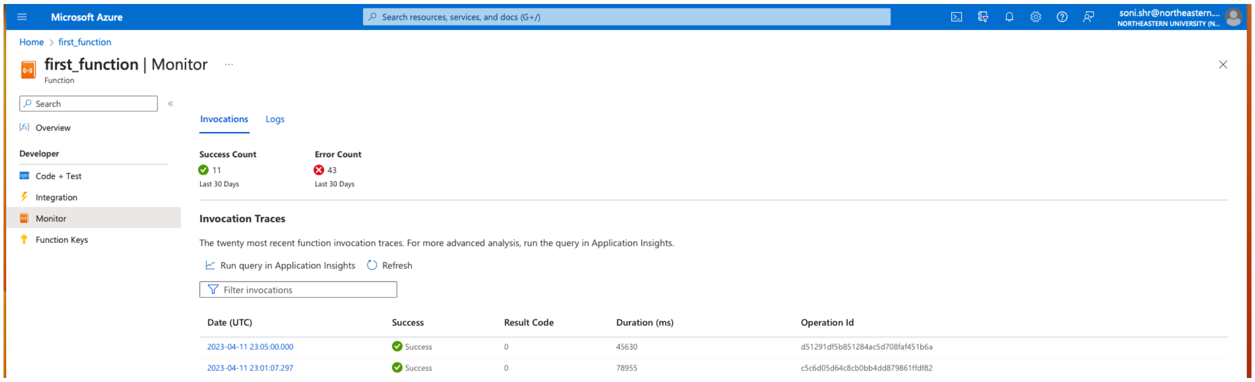
```python
import logging
import datetime
import os
import azure.functions as func
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient

# Set Kaggle API credentials directly
os.environ['KAGGLE_USERNAME'] = 'shshyam'
os.environ['KAGGLE_KEY'] = '9cdc92522b081cdd60ffbf4c46352689'

from kaggle.api.kaggle_api_extended import KaggleApi


def main(req: func.TimerRequest, outputBlob: func.Out[func.InputStream]) -> str:
    utc_timestamp = datetime.datetime.utcnow().replace(
        tzinfo=datetime.timezone.utc).isoformat()

    logging.info('Python timer trigger function ran at %s', utc_timestamp)
    logging.info('Python timer trigger function ran at %s', req)

    api = KaggleApi()
    api.authenticate()

```

```
Request headers:
    'Content-Length': '6669'
    'x-ms-blob-type': 'REDACTED'
    'x-ms-version': 'REDACTED'
    'Content-Type': 'application/octet-stream'
    'Accept': 'application/xml'
    'User-Agent': 'azsdk-python-storage-blob/12.15.0 Python/3.10.10 (Linux-5.10.102.2-microsoft-standard-
x86_64-with-glibc2.31)'
    'x-ms-date': 'REDACTED'
    'x-ms-client-request-id': 'ed1a772e-d8bc-11ed-8615-00155d25f641'
    'Authorization': 'REDACTED'
A body is sent with the request
2023-04-11T23:02:26Z   [Information]   Response status: 201
Response headers:
```

HTTP response code

202 Accepted

HTTP response content



first_function | Monitor

Function

Success Count
✓ 11
Last 30 Days

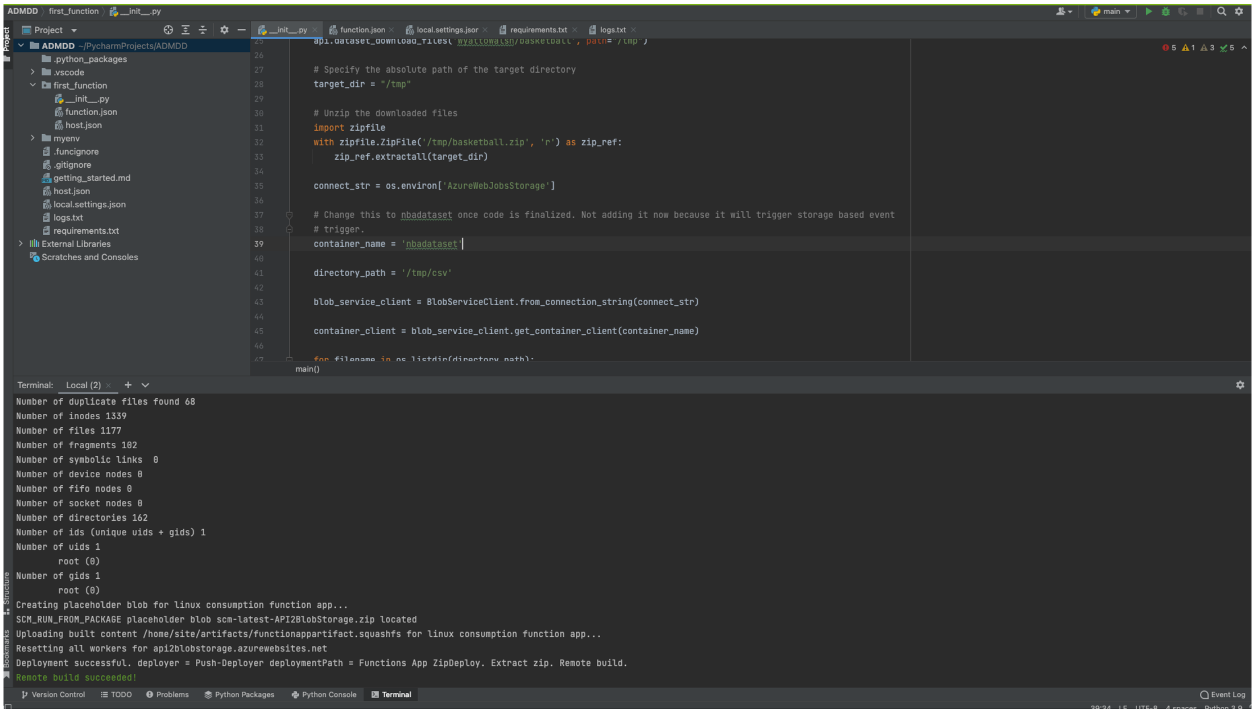
Error Count
✗ 43
Last 30 Days

Invocation Traces

The twenty most recent function invocation traces. For more advanced analysis, run the query in Application Insights.

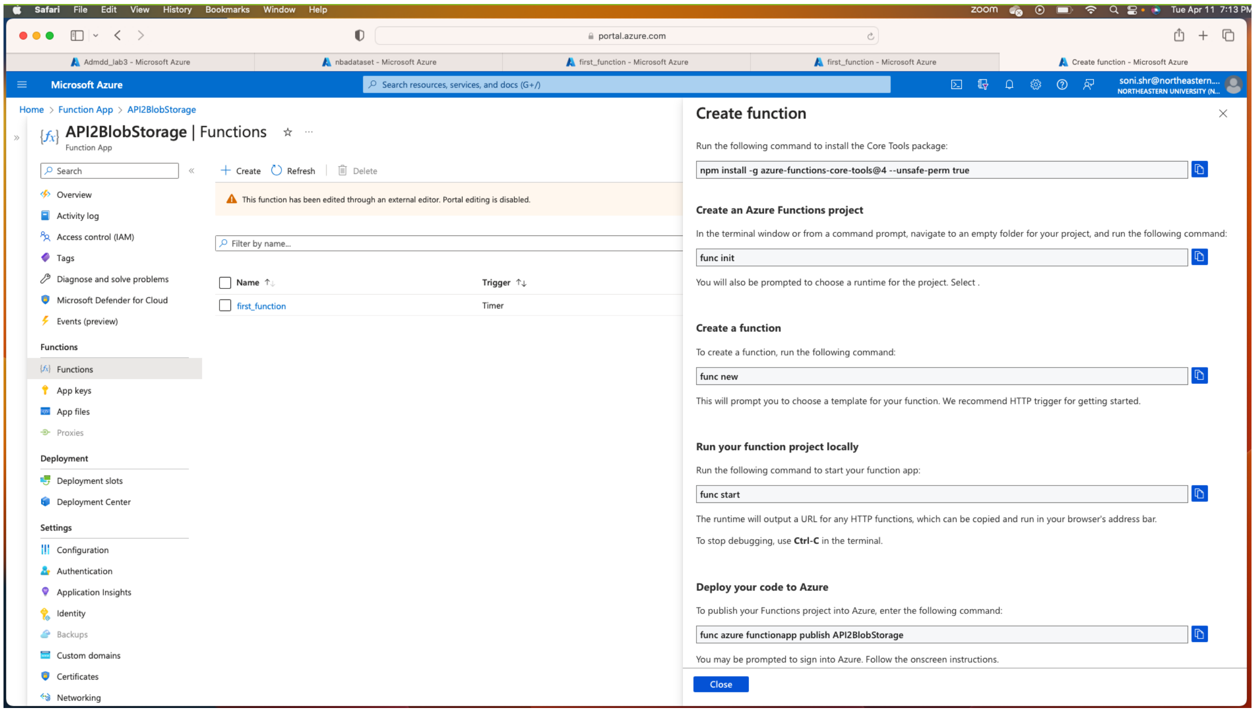
Date (UTC)	Success	Result Code	Duration (ms)	Operation Id
2023-04-11 23:05:00.000	✓ Success	0	45630	d51291df5b851284ac5d708faf451b6a
2023-04-11 23:01:07.297	✓ Success	0	78955	c5c6d05d64c8cb0bb4dd879861ffdf82

The method we followed to deploy this Azure Function App is build locally, and deploy to Azure Cloud. The local build is shown in the below screenshot and the following screenshot shows the steps taken for deployment.



```
    api.dataset_download_files('wyattowalsh/basketball', path='/tmp')

    # Specify the absolute path of the target directory
    target_dir = "/tmp"

    # Unzip the downloaded files
    import zipfile
    with zipfile.ZipFile('/tmp/basketball.zip', 'r') as zip_ref:
        zip_ref.extractall(target_dir)

    connect_str = os.environ['AzureWebJobsStorage']

    # Change this to nbadataset once code is finalized. Not adding it now because it will trigger storage based event
    # trigger.
    container_name = 'nbadataset'

    directory_path = '/tmp/csv'

    blob_service_client = BlobServiceClient.from_connection_string(connect_str)

    container_client = blob_service_client.get_container_client(container_name)

    for filename in os.listdir(directory_path):
```

```
Number of duplicate files found 68
Number of inodes 1339
Number of files 1177
Number of fragments 102
Number of symbolic links  0
Number of device nodes 0
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 162
Number of ids (unique uids + gids) 1
Number of uids 1
        root (0)
Number of gids 1
        root (0)
Creating placeholder blob for linux consumption function app...
SCM_RUN_FROM_PACKAGE placeholder blob scm-latest-API2BlobStorage.zip located
Uploading built content /home/site/artifacts/functionappartifact.squashfs for linux consumption function app...
Resetting all workers for api2blobstorage.azurewebsites.net
Deployment successful. deployer = Push-Deployer deploymentPath = Functions App ZipDeploy. Extract zip. Remote build.
Remote build succeeded!
```


Safari File Edit View History Bookmarks Window Help
portal.azure.com
Admdd_lab3 - Microsoft Azure
nbadataset - Microsoft Azure
first_function - Microsoft Azure
first_function - Microsoft Azure
Create function - Microsoft Azure
Microsoft Azure
Search resources, services, and docs (G+/)
soni.shr@northeastern....
NORTHEASTERN UNIVERSITY (N...
Tue Apr 11 7:13 PM
Home > Function App > API2BlobStorage
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Microsoft Defender for Cloud
Events (preview)
Functions
Functions
App keys
App files
Proxies
Deployment
Deployment slots
Deployment Center
Settings
Configuration
Authentication
Application Insights
Identity
Backups
Custom domains
Certificates
Networking
Create
Refresh
Delete
Filter by name...
Close

API2BlobStorage | Functions

Function App

This function has been edited through an external editor. Portal editing is disabled.

Name	Trigger
first_function	Timer

Create function

Run the following command to install the Core Tools package:

```
npm install -g azure-functions-core-tools@4 --unsafe-perm true
```

Create an Azure Functions project

In the terminal window or from a command prompt, navigate to an empty folder for your project, and run the following command:

```
func init
```

You will also be prompted to choose a runtime for the project. Select .

Create a function

To create a function, run the following command:

```
func new
```

This will prompt you to choose a template for your function. We recommend HTTP trigger for getting started.

Run your function project locally

Run the following command to start your function app:

```
func start
```

The runtime will output a URL for any HTTP functions, which can be copied and run in your browser's address bar.

To stop debugging, use **Ctrl-C** in the terminal.

Deploy your code to Azure

To publish your Functions project into Azure, enter the following command:

```
func azure functionapp publish API2BlobStorage
```

You may be prompted to sign into Azure. Follow the onscreen instructions.

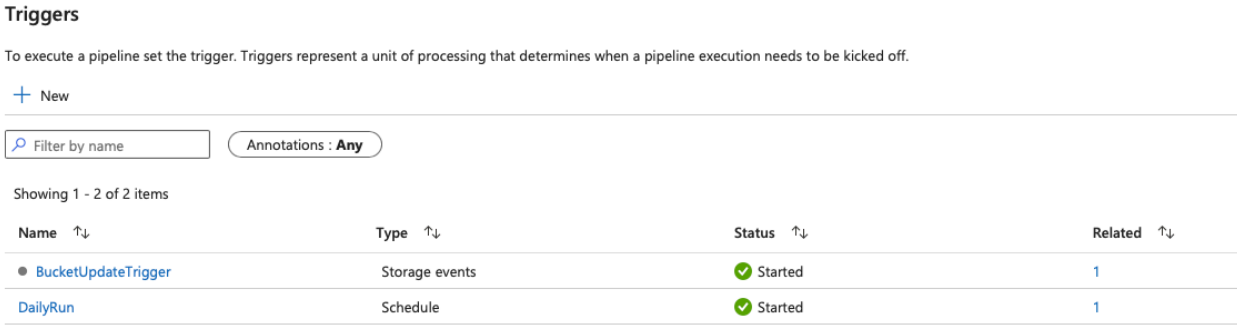
Thus, the function executes and ensures that the data available in our Raw Data store – Azure Blob Storage is always current, and as per the latest available source Data in Kaggle as well as the NBA data feeds API.

As a part of P3, we also included two other features:

Pipeline execution triggers :

We have set up two types of triggers for our pipeline –

- There is a Scheduled trigger called Daily Run – which is scheduled to run everyday at 12AM UTC
- There is a Storage Events Trigger called BucketUpdateTrigger- which will run every time there is a new file added or updated in our Raw Data Storage Container

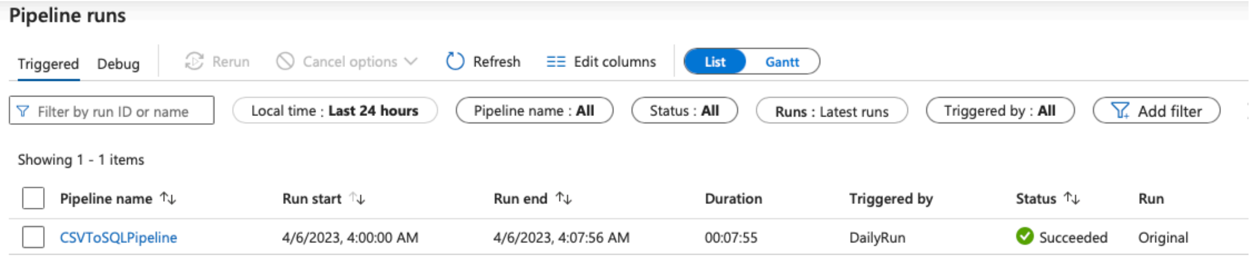


Triggers

To execute a pipeline set the trigger. Triggers represent a unit of processing that determines when a pipeline execution needs to be kicked off.

Showing 1 - 2 of 2 items

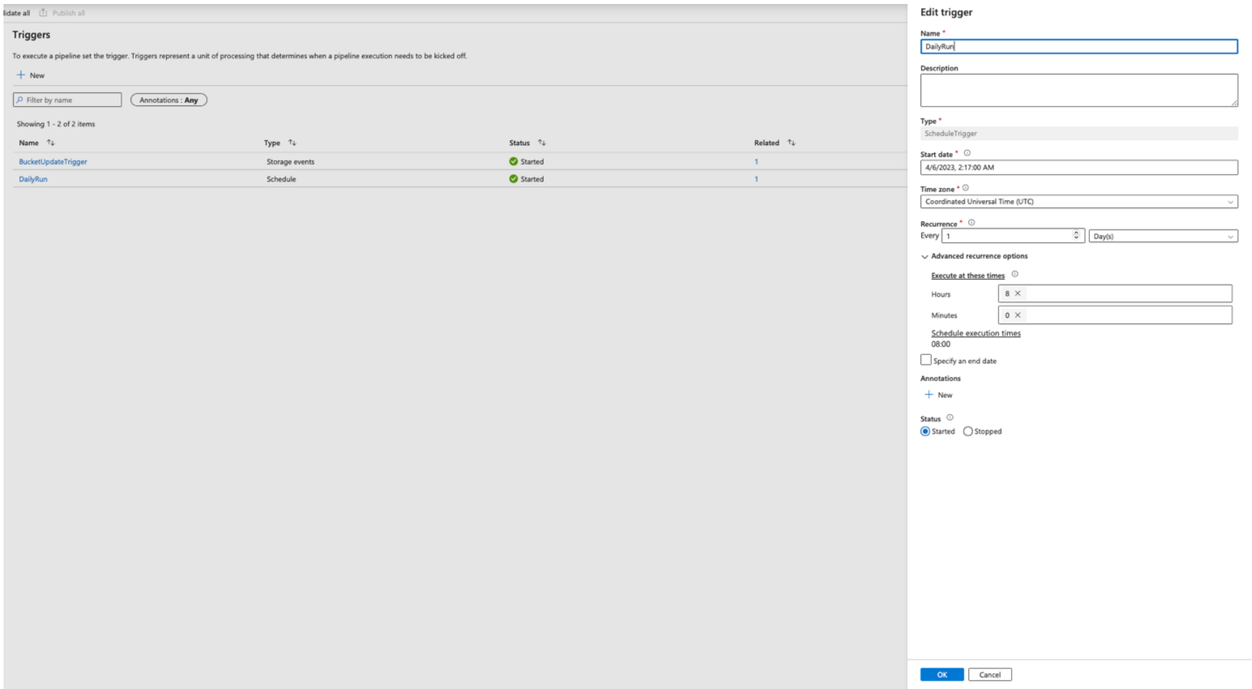
Name	Type	Status	Related
● BucketUpdateTrigger	Storage events	✓ Started	1
DailyRun	Schedule	✓ Started	1



Pipeline runs

Showing 1 - 1 items

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Run
CSVToSQLPipeline	4/6/2023, 4:00:00 AM	4/6/2023, 4:07:56 AM	00:07:55	DailyRun	✓ Succeeded	Original



Triggers

To execute a pipeline set the trigger. Triggers represent a unit of processing that determines when a pipeline execution needs to be kicked off.

Showing 1 - 2 of 2 items

Name	Type	Status	Related
BucketUpdateTrigger	Storage events	✓ Started	1
DailyRun	Schedule	✓ Started	1

Edit trigger

Name * DailyRun

Description

Type * ScheduleTrigger

Start date * 4/6/2023, 2:17:00 AM

Time zone * Coordinated Universal Time (UTC)

Recurrence * Every 1 Day(s)

Advanced recurrence options

Execute at these times

Hours 8

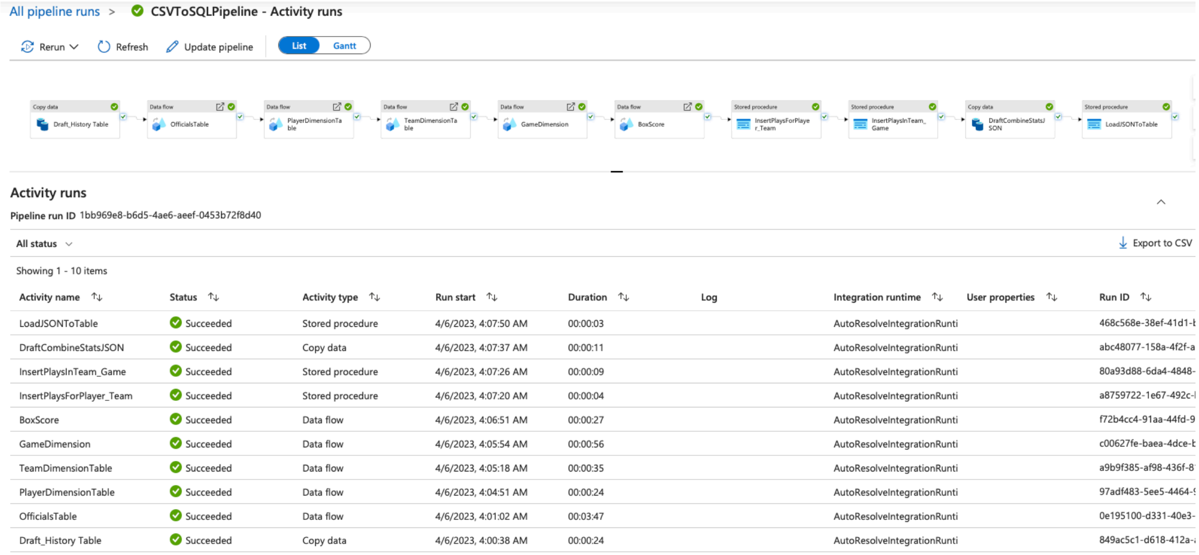
Minutes 0

Schedule execution times
08:00

□ Specify an end date

Annotations

Status ● Started ○ Stopped



All pipeline runs > CSVToSQLPipeline - Activity runs

Activity runs

Pipeline run ID 1bb969e8-b6d5-4ae6-aeef-0453b72f8d40

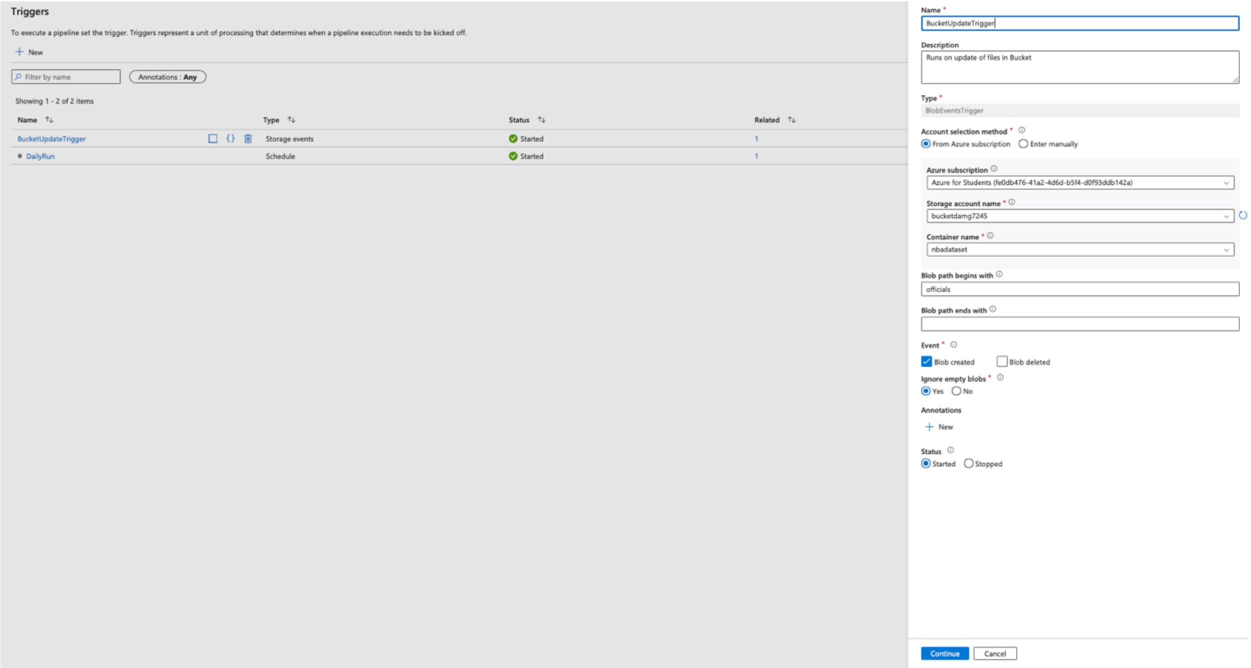
Showing 1 - 10 items

Activity name	Status	Activity type	Run start	Duration	Log	Integration runtime	User properties	Run ID
LoadJSONToTable	✓ Succeeded	Stored procedure	4/6/2023, 4:07:50 AM	00:00:03		AutoResolveIntegrationRunti		468c568e-38ef-41d1-t
DraftCombineStatsJSON	✓ Succeeded	Copy data	4/6/2023, 4:07:37 AM	00:00:11		AutoResolveIntegrationRunti		abc48077-158a-4f2f-a
InsertPlaysInTeam_Game	✓ Succeeded	Stored procedure	4/6/2023, 4:07:26 AM	00:00:09		AutoResolveIntegrationRunti		80a93d88-6da4-4848-
InsertPlaysForPlayer_Team	✓ Succeeded	Stored procedure	4/6/2023, 4:07:20 AM	00:00:04		AutoResolveIntegrationRunti		a8759722-1e67-492c-l
BoxScore	✓ Succeeded	Data flow	4/6/2023, 4:06:51 AM	00:00:27		AutoResolveIntegrationRunti		f72b4cc4-91aa-44fd-9
GameDimension	✓ Succeeded	Data flow	4/6/2023, 4:05:54 AM	00:00:56		AutoResolveIntegrationRunti		c00627fe-baea-4dce-t
TeamDimensionTable	✓ Succeeded	Data flow	4/6/2023, 4:05:18 AM	00:00:35		AutoResolveIntegrationRunti		a9b9f385-af98-436f-8
PlayerDimensionTable	✓ Succeeded	Data flow	4/6/2023, 4:04:51 AM	00:00:24		AutoResolveIntegrationRunti		97adf483-5ee5-4464-(
OfficialsTable	✓ Succeeded	Data flow	4/6/2023, 4:01:02 AM	00:03:47		AutoResolveIntegrationRunti		0e195100-d331-40e3-
Draft_History Table	✓ Succeeded	Copy data	4/6/2023, 4:00:38 AM	00:00:24		AutoResolveIntegrationRunti		849ac5c1-d618-412a-;

The above image shows all steps of our ADF pipeline has run and completed successfully.

As shown above our pipeline is scheduled to run everyday at 4AM EST. This ensures that our Scheduled trigger runs after the data refresh done by the Azure Function.

Our Storage Events trigger is implemented to track for changes in any of the files in the nbadataset storage location.



Triggers

To execute a pipeline set the trigger. Triggers represent a unit of processing that determines when a pipeline execution needs to be kicked off.

Showing 1 - 2 of 2 items

Name	Type	Status	Related
BucketUpdateTrigger	Storage events	Started	1
DailyRun	Schedule	Started	1

Name *
BucketUpdateTrigger

Description
Runs on update of files in Bucket

Type *
BlobEventsTrigger

Account selection method *
● From Azure subscription ○ Enter manually

Azure subscription
Azure for Students (fe0db476-41a2-4d6d-b5f4-d0f93ddb142a)

Storage account name *
bucketdamg7245

Container name *
nbadataset

Blob path begins with
officials

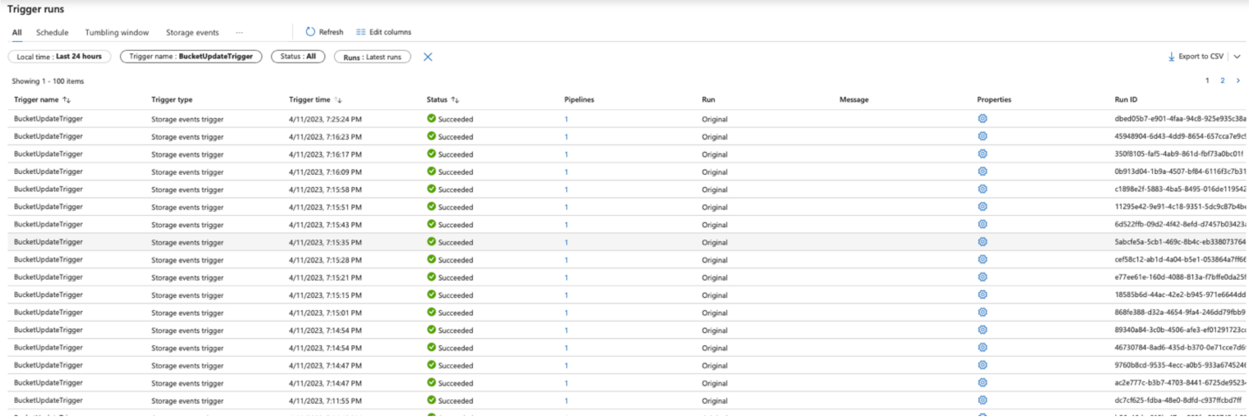
Blob path ends with

Event *
☑ Blob created ☐ Blob deleted

Ignore empty blobs *
● Yes ○ No

Annotations

Status
● Started ○ Stopped



Trigger runs

Local time : Last 24 hours Trigger name : BucketUpdateTrigger Status : All Runs : Latest runs

Showing 1 - 100 items

Trigger name	Trigger type	Trigger time	Status	Pipelines	Run	Message	Properties	Run ID
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:25:24 PM	✓ Succeeded	1	Original		⚙	dbed05b7-e901-4faa-94c8-925e935c38a
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:16:23 PM	✓ Succeeded	1	Original		⚙	45948904-6d43-4dd9-8654-657cca7e9c
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:16:17 PM	✓ Succeeded	1	Original		⚙	350f8105-faf5-4ab9-861d-fbf73a0bc01f
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:16:09 PM	✓ Succeeded	1	Original		⚙	0b913d04-1b9a-4507-bf84-6116f3c7b31
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:58 PM	✓ Succeeded	1	Original		⚙	c1898e2f-5883-4ba5-8495-016de119542
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:51 PM	✓ Succeeded	1	Original		⚙	11295e42-9e91-4c18-9351-5dc9c87b4b
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:43 PM	✓ Succeeded	1	Original		⚙	6d522ffb-09d2-4f42-8efd-d7457b03423
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:35 PM	✓ Succeeded	1	Original		⚙	5abcfe5a-5cb1-469c-8b4c-eb338073764
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:28 PM	✓ Succeeded	1	Original		⚙	cef58c12-ab1d-4a04-b5e1-053864a7ff6
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:21 PM	✓ Succeeded	1	Original		⚙	e77ee61e-160d-4088-813a-f7bffe0da25
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:15 PM	✓ Succeeded	1	Original		⚙	18585b6d-44ac-42e2-b945-971e6644dd
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:15:01 PM	✓ Succeeded	1	Original		⚙	868fe388-d32a-4654-9fa4-246dd79fbb9
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:14:54 PM	✓ Succeeded	1	Original		⚙	89340a84-3c0b-4506-afe3-ef01291723c
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:14:54 PM	✓ Succeeded	1	Original		⚙	46730784-8ad6-435d-b370-0e71cce7d6
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:14:47 PM	✓ Succeeded	1	Original		⚙	9760b8cd-9535-4ecc-a0b5-933a674524
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:14:47 PM	✓ Succeeded	1	Original		⚙	ac2e777c-b3b7-4703-8441-6725de9523
BucketUpdateTrigger	Storage events trigger	4/11/2023, 7:11:55 PM	✓ Succeeded	1	Original		⚙	dc7cf625-fdba-48e0-8dfd-c937ffcbd7ff

In conclusion, our Azure Function ensures that the data in raw layer – Azure blob storage is always the current NBA data. The function in combination with these two triggers activated on our ETL pipeline ensures a near-real time updation of our Foundation Data Layer implemented in Azure SQL Database.

Additional Implementation for P4:

An additional implementation that we did for P4 is the creation of a User Interface. The purpose behind this is two-fold.

1. In most organizations, the data layer needs to be abstracted from the business users due to many reasons, these can be security reasons or simply the need to keep the data engineers and data users entities separate.
2. Each user might need different types of data – Although our project does create a reporting layer in PowerBI, there could be different kinds of users for our data. For

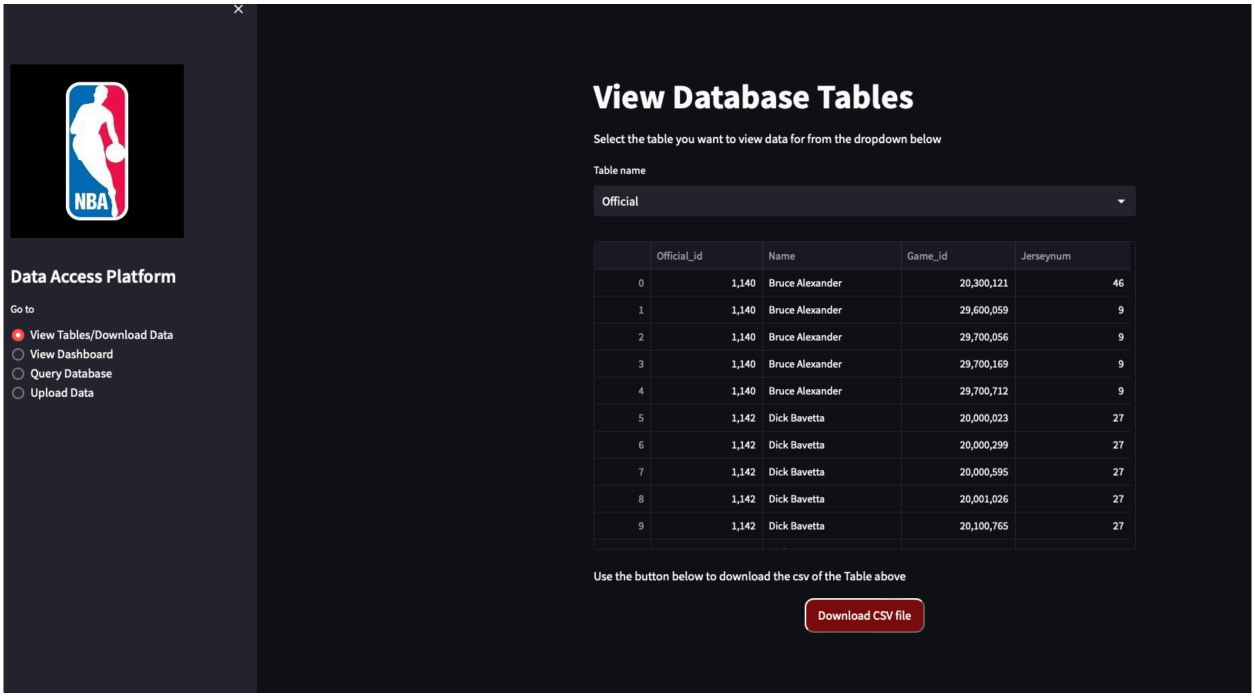
example, one team might use the powerBI visualizations in their progress reports, another team might want to use the csv of the Table to use within their own internal excel models or another team might need the functionality to just look at one row of data based on a condition.

Hence we have implemented a Data Access Platform for our users which gives them this functionality while also protecting the actual data from updations or deletions.

Our Data Access platform is available at :
https://shikashyam-streamlitazure-streamlit-9b7mzs.streamlit.app

The UI screenshots are below:

Function 1 : **View Tables**



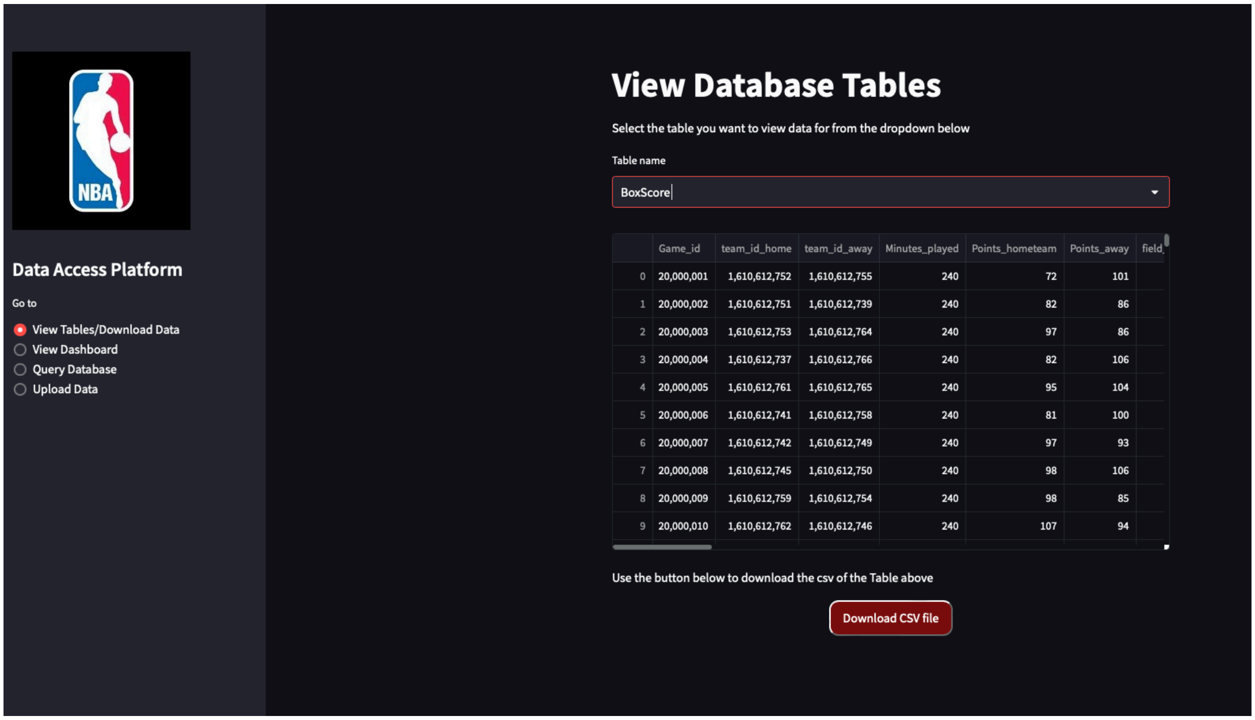
View Database Tables

Select the table you want to view data for from the dropdown below

Table name: Official

	Official_id	Name	Game_id	Jerseynum
0	1,140	Bruce Alexander	20,300,121	46
1	1,140	Bruce Alexander	29,600,059	9
2	1,140	Bruce Alexander	29,700,056	9
3	1,140	Bruce Alexander	29,700,169	9
4	1,140	Bruce Alexander	29,700,712	9
5	1,142	Dick Bavetta	20,000,023	27
6	1,142	Dick Bavetta	20,000,299	27
7	1,142	Dick Bavetta	20,000,595	27
8	1,142	Dick Bavetta	20,001,026	27
9	1,142	Dick Bavetta	20,100,765	27

Use the button below to download the csv of the Table above



Data Access Platform

Go to

● View Tables/Download Data
○ View Dashboard
○ Query Database
○ Upload Data

View Database Tables

Select the table you want to view data for from the dropdown below

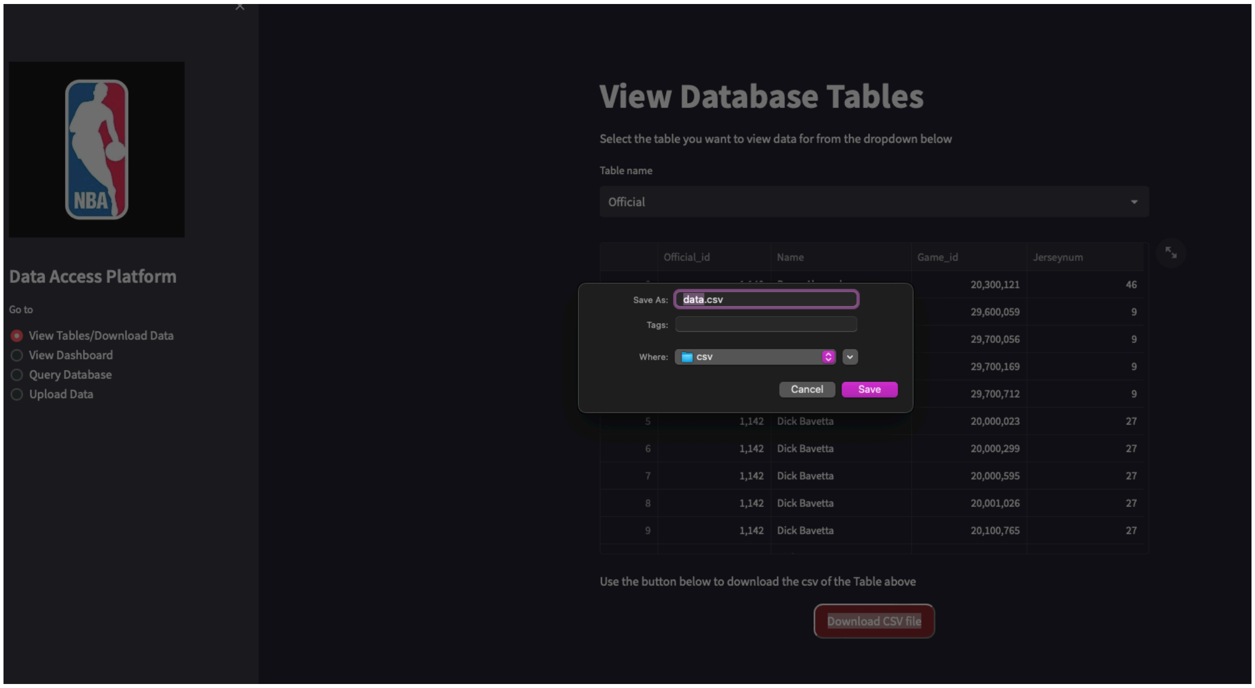
Table name

BoxScore

	Game_id	team_id_home	team_id_away	Minutes_played	Points_hometeam	Points_away	field_
0	20,000,001	1,610,612,752	1,610,612,755	240	72	101	
1	20,000,002	1,610,612,751	1,610,612,739	240	82	86	
2	20,000,003	1,610,612,753	1,610,612,764	240	97	86	
3	20,000,004	1,610,612,737	1,610,612,766	240	82	106	
4	20,000,005	1,610,612,761	1,610,612,765	240	95	104	
5	20,000,006	1,610,612,741	1,610,612,758	240	81	100	
6	20,000,007	1,610,612,742	1,610,612,749	240	97	93	
7	20,000,008	1,610,612,745	1,610,612,750	240	98	106	
8	20,000,009	1,610,612,759	1,610,612,754	240	98	85	
9	20,000,010	1,610,612,762	1,610,612,746	240	107	94	

Use the button below to download the csv of the Table above

Function 2: **Download Data**



Data Access Platform

Go to

● View Tables/Download Data
○ View Dashboard
○ Query Database
○ Upload Data

View Database Tables

Select the table you want to view data for from the dropdown below

Table name

Official

	Official_id	Name	Game_id	Jerseynum
[illegible]	[illegible]	[illegible]	20,300,121	46
[illegible]	[illegible]	[illegible]	29,600,059	9
[illegible]	[illegible]	[illegible]	29,700,056	9
[illegible]	[illegible]	[illegible]	29,700,169	9
[illegible]	[illegible]	[illegible]	29,700,712	9
5	1,142	Dick Bavetta	20,000,023	27
6	1,142	Dick Bavetta	20,000,299	27
7	1,142	Dick Bavetta	20,000,595	27
8	1,142	Dick Bavetta	20,001,026	27
9	1,142	Dick Bavetta	20,100,765	27

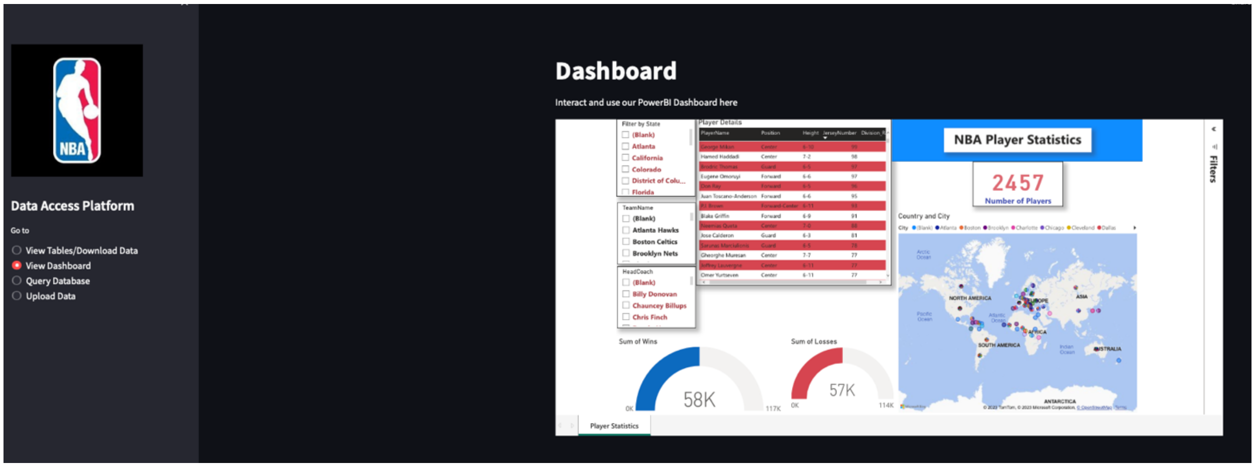
Save As: data.csv

Tags:

Where: csv

Use the button below to download the csv of the Table above

Function 3 : **View and Interact with Embedded Dashboard**



Dashboard

Interact and use our PowerBI Dashboard here

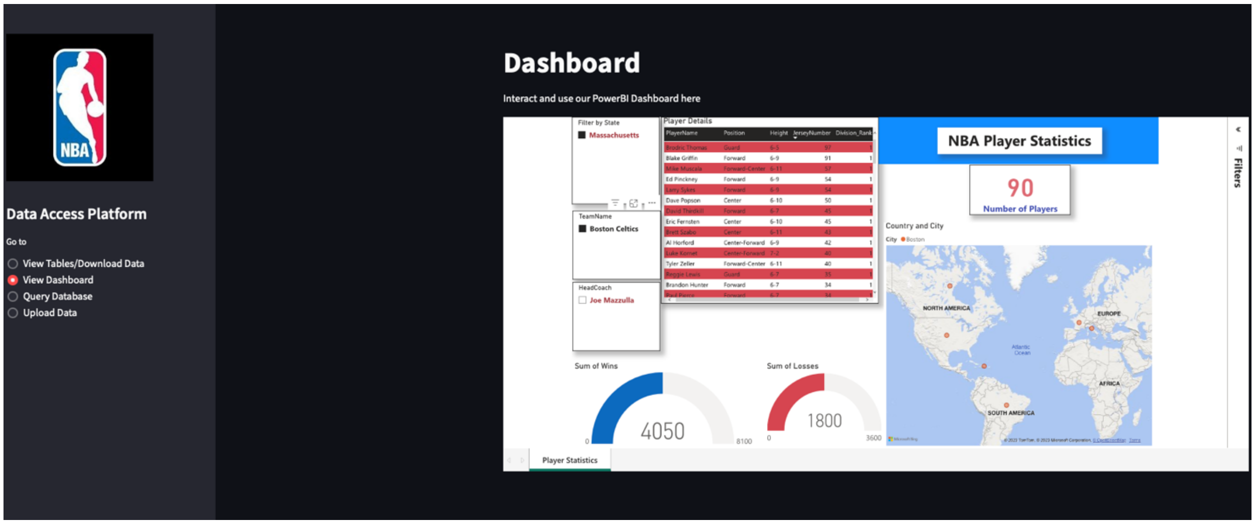
Filter by State
☐ (Blank)
☐ Atlanta
☐ California
☐ Colorado
☐ District of Colu...
☐ Florida

TeamName
☐ (Blank)
☐ Atlanta Hawks
☐ Boston Celtics
☐ Brooklyn Nets

HeadCoach
☐ (Blank)
☐ Billy Donovan
☐ Chauncey Billups
☐ Chris Finch

Player Details

PlayerName	Position	Height	JerseyNumber	Division_R
George Mikan	Center	6-10	99	
Hamed Haddadi	Center	7-2	98	
Brodric Thomas	Guard	6-5	97	
Eugene Omoruyi	Forward	6-6	97	
Don Ray	Forward	6-5	96	
Juan Toscano-Anderson	Forward	6-6	95	
P.J. Brown	Forward-Center	6-11	93	
Blake Griffin	Forward	6-9	91	
Neemias Queta	Center	7-0	88	
Jose Calderon	Guard	6-3	81	
Sarunas Marciulionis	Guard	6-5	78	
Gheorghe Muresan	Center	7-7	77	
Jeffrey Lauvergne	Center	6-11	77	
Omer Yurtseven	Center	6-11	77	



Dashboard

Interact and use our PowerBI Dashboard here

Filter by State
■ Massachusetts

TeamName
■ Boston Celtics

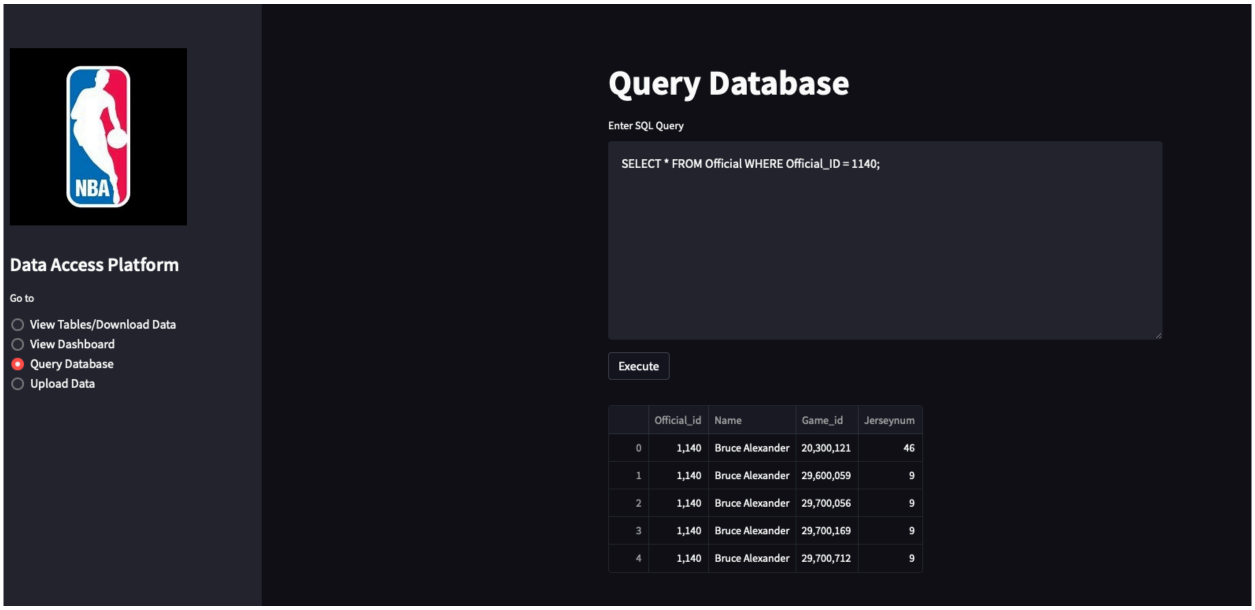
HeadCoach
☐ Joe Mazzulla

Player Details

PlayerName	Position	Height	JerseyNumber	Division_Rank
Brodric Thomas	Guard	6-5	97	1
Blake Griffin	Forward	6-9	91	1
Mike Muscala	Forward-Center	6-11	57	1
Ed Pinckney	Forward	6-9	54	1
Larry Sykes	Forward	6-9	54	1
Dave Popson	Center	6-10	50	1
David Thirdkill	Forward	6-7	45	1
Eric Fernsten	Center	6-10	45	1
Brett Szabo	Center	6-11	43	1
Al Horford	Center-Forward	6-9	42	1
Luke Kornet	Center-Forward	7-2	40	1
Tyler Zeller	Forward-Center	6-11	40	1
Reggie Lewis	Guard	6-7	35	1
Brandon Hunter	Forward	6-7	34	1
Paul Pierce	Forward	6-7	34	1

Function 4 : **Query Database**

This function allows the user to query the database for specific requirements that cannot be satisfied in Function 1. This ensures that the query the user tries to execute is a SELECT, and never a DELETE, TRUNCATE or UPDATE or any similar destructive queries.

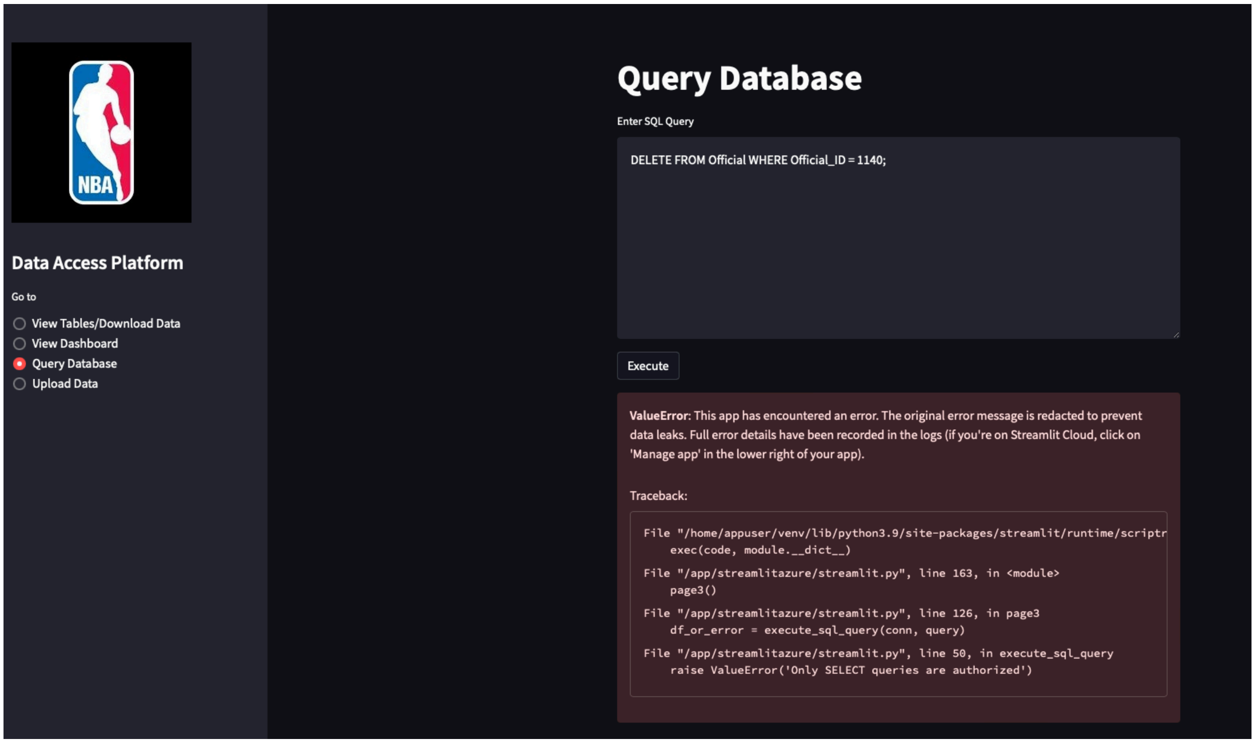


Query Database

Enter SQL Query

```
SELECT * FROM Official WHERE Official_ID = 1140;
```

	Official_id	Name	Game_id	Jerseynum
0	1,140	Bruce Alexander	20,300,121	46
1	1,140	Bruce Alexander	29,600,059	9
2	1,140	Bruce Alexander	29,700,056	9
3	1,140	Bruce Alexander	29,700,169	9
4	1,140	Bruce Alexander	29,700,712	9



Query Database

Enter SQL Query

```
DELETE FROM Official WHERE Official_ID = 1140;
```

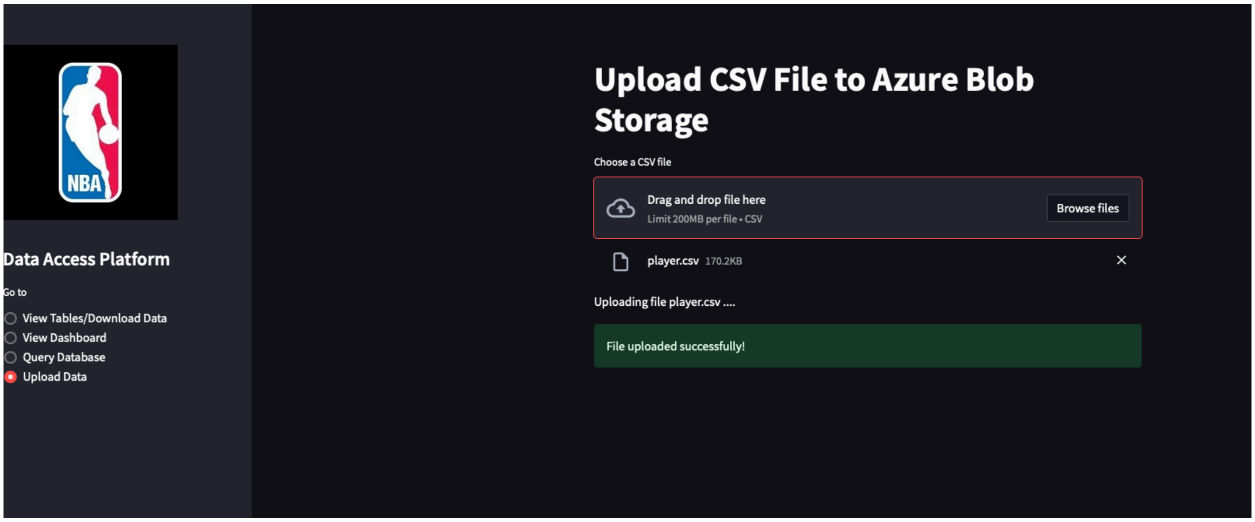
ValueError: This app has encountered an error. The original error message is redacted to prevent data leaks. Full error details have been recorded in the logs (if you're on Streamlit Cloud, click on 'Manage app' in the lower right of your app).

Traceback:

```
File "/home/appuser/venv/lib/python3.9/site-packages/streamlit/runtime/scriptr
    exec(code, module.__dict__)
File "/app/streamlitazure/streamlit.py", line 163, in <module>
    page3()
File "/app/streamlitazure/streamlit.py", line 126, in page3
    df_or_error = execute_sql_query(conn, query)
File "/app/streamlitazure/streamlit.py", line 50, in execute_sql_query
    raise ValueError('Only SELECT queries are authorized')
```

Function 5 : **Upload data csv directly**

This functionality lets the user directly upload a csv file to blob storage through the app. This checks that the file is a csv, and that it is a filename that we expect for our ETL, and then uploads it to blob in the backend. Once the blob is updated, Our Storage event trigger will kick in and will update the SQL Database within minutes.



Upload CSV File to Azure Blob Storage

Choose a CSV file

player.csv 170.2KB

Uploading file player.csv ....

File uploaded successfully!

Appendix – Python code for the UI

```
import streamlit as st
import pymssql
import pandas as pd
import base64
import re
from PIL import Image
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient

server_name = ‘<server_name>’
database_name = '<databasename>'
username = '<username>'
password = '<password>'

# Define function to download the table as CSV
def download_csv(data):
    csv = data.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()
    href = f'<a href="data:file/csv;base64,{b64}" download="data.csv" style = "text-decoration: none;"><button style="background-color: #7c0d0e; color: #ffffff; border-radius: 12px; padding: 8px 16px; display: block; margin: 0 auto;text-decoration: none;">Download CSV file</button></a>'
    return href

# Define a connection function to Azure SQL Database

def connect_to_database(server, database, username, password):
    conn = pymssql.connect(server=server, database=database, user=username, password=password)
    return conn
```

```
# Define a function to retrieve table names from the database

def get_table_names(conn):
    cursor = conn.cursor()
    cursor.execute("SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE
TABLE_TYPE='BASE TABLE'")
    table_names = [row[0] for row in cursor]
    return table_names

# Define a function to retrieve data from the selected table

def get_table_data(conn, table_name):
    cursor = conn.cursor()
    cursor.execute(f"SELECT TOP(1000) * FROM {table_name}")
    data = cursor.fetchall()
    column_names = [column[0] for column in cursor.description]
    print(table_name)
    print(column_names)
    df = pd.DataFrame(data, columns=column_names)
    return df

def execute_sql_query(conn, query):

    if not re.match(r'^\s*SELECT\s+.*\s+FROM\s+', query, re.IGNORECASE):
        raise ValueError('Only SELECT queries are authorized')
    else:
        cursor = conn.cursor()
        try:
            cursor.execute(query)
            if cursor.description:
                column_names = [column[0] for column in cursor.description]
                data = cursor.fetchall()
                df = pd.DataFrame(data, columns=column_names)
                return df
            else:
                return None
        except Exception as e:
            return str(e)

def upload_to_azure(filecontent,file_name, container_name):
    connect_str = '<blob_connection_string>
    blob_service_client = BlobServiceClient.from_connection_string(connect_str)
    container_client = blob_service_client.get_container_client(container_name)
    blob_client = container_client.get_blob_client(blob=file_name)
    blob_client.upload_blob(filecontent,overwrite=True)


# Define page 1
```

```python
def page1():
    st.title('View Database Tables')

    # Create connection to database
    conn = connect_to_database(server_name, database_name, username, password)

    # Get the table names from the database
    table_names = get_table_names(conn)
    st.write('Select the table you want to view data for from the dropdown below')
    # Create a dropdown of table names
    selected_table = st.selectbox('Table name',table_names)

    # Retrieve the data from the selected table
    table_data = get_table_data(conn, selected_table)
    hide_dataframe_row_index = """
            <style>
            .row_heading.level0 {display:none}
            .blank {display:none}
            </style>
            """

    st.markdown(hide_dataframe_row_index, unsafe_allow_html=True)

    # Display the table
    st.dataframe(table_data, width=1000)

    st.write('Use the button below to download the csv of the Table above')
    # Provide an option to download the table as a CSV file
    st.markdown(download_csv(table_data), unsafe_allow_html=True)

# Define page 2
def page2():
    st.title('Dashboard')
    st.write('Interact and use our PowerBI Dashboard here')
    # Embed PowerBI report
    st.markdown("""<iframe title="Player Statistics - Player Statistics" width="1140"
height="541.25" src="https://app.powerbi.com/reportEmbed?reportId=3eb21850-9017-41b1-
8fb8-646fb8b85004&autoAuth=true&ctid=a8eec281-aaa3-4dae-ac9b-9a398b9215e7"
frameborder="0" allowFullScreen="true"></iframe>""", unsafe_allow_html=True)

def page3():
    hide_dataframe_row_index = """
            <style>
            .row_heading.level0 {display:none}
            .blank {display:none}
            </style>
            """
    st.title('Query Database')
    # Create connection to database
    conn = connect_to_database(server_name, database_name, username, password)
    query = st.text_area('Enter SQL Query', height=250)
    if st.button('Execute'):
        if not query:
            st.warning('Please enter a query')
        else:
            df_or_error = execute_sql_query(conn, query)
            if isinstance(df_or_error, pd.DataFrame):
                st.markdown(hide_dataframe_row_index, unsafe_allow_html=True)
                st.dataframe(df_or_error)
            else:
                st.error(df_or_error)

def page4():
    st.title('Upload CSV File to Azure Blob Storage')

    # Create a file uploader
    uploaded_file = st.file_uploader('Choose a CSV file')

    if uploaded_file is not None:
        # Save the file to a temporary directory
        print('filename:',uploaded_file.name)
        filename = uploaded_file.name
        filecontents = uploaded_file.getvalue()
        # Upload the file to Azure Blob storage
        container_name = 'nbadataset'
        upload_to_azure(filecontents, filename, container_name)

        st.success('File uploaded successfully!')


# Create a sidebar with navigation
st.sidebar.image(Image.open("./nba.jpg"), width=225)
st.sidebar.title('Data Access Platform')
options = ['View Tables/Download Data', 'View Dashboard', 'Query Database', 'Upload
Data']
selection = st.sidebar.radio('Go to', options)

# Show the appropriate page based on the user's selection
if selection == 'View Tables/Download Data':
    page1()
elif selection == 'View Dashboard':
    page2()
elif selection == 'Query Database':
    page3()
else:
    page4()
```

Next Steps:

As a part of next steps, specifically for P5, we are excited to showcase our entire dashboard as well as presentation. We are also looking forward to demoing our user interface.

****