Python (V2 Model) Based Azure Function

Introduction:

This POC aims to do the following:

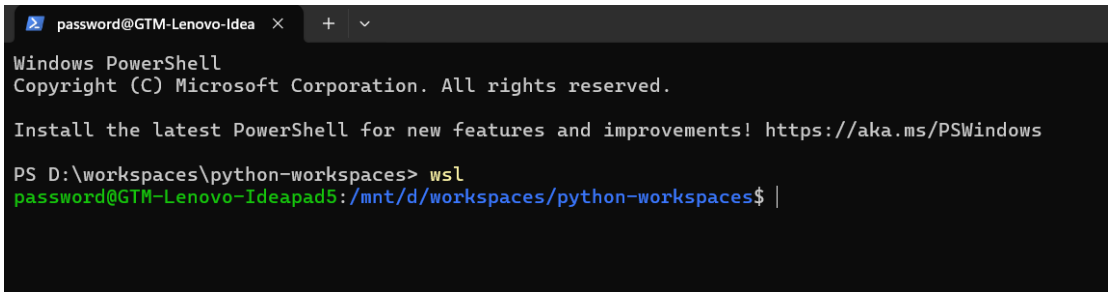
- Creation of a python (v2 model) based azure function which will work in local as well as in azure environment
- This function will be triggered through azure event-hub trigger - whenever any data is published to azure event-hub. For running in local, azure event-hub emulator has been used.
- The function will also publish the data to azure blob storage. For running in local, azurite has been used.
- Creation of a http trigger-based helper azure function as well – it will publish the data into the azure event-hub. This function will be used for testing the main function.

Pre-requisites:

- In local, everything is done in WSL environment. So, windows subsystem for Linux (WSL) installation is must.
- Docker Engine is installed in WSL
- Dotnet is installed in WSL
  sudo apt-get install -y dotnet-sdk-9.0
- Python version >=3.10 is installed
- VSCode (as IDE) is available
- Azure Functions Core Tools is installed in WSL (https://learn.microsoft.com/en-us/azure/azure-functions/functions-run-local?tabs=windows%2Cisolated-process%2Cnode-v4%2Cpython-v2%2Chttp-trigger%2Ccontainer-apps&pivots=programming-language-python)
- Azure EventHub emulator (https://github.com/Azure/azure-event-hubs-emulator-installer) and azurite (for azure storage emulator) should be available.

Steps:

- Open WSL in PowerShell



- Create a Function App

  func init MyPFunctionApp --python -m v2

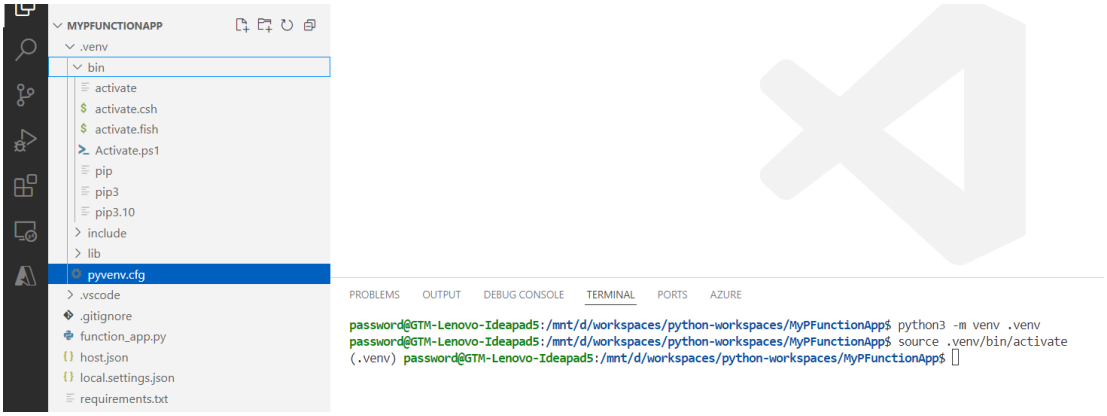
Python (V2 Model) Based Azure Function

```
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces$ func init MyPFunctionApp --python -m v2
Found Python version 3.10.12 (python3).
The new Python programming model is generally available. Learn more at https://aka.ms/pythonprogrammingmodel
Writing requirements.txt
Writing function_app.py
Writing .gitignore
Writing host.json
Writing local.settings.json
Writing /mnt/d/workspaces/python-workspaces/MyPFunctionApp/.vscode/extensions.json
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces$
```

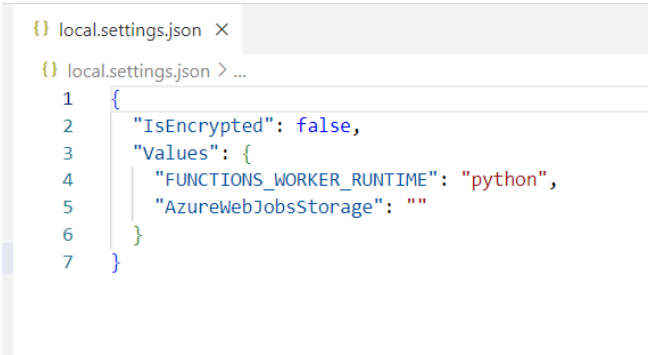
- Create a virtual environment

```
password@GTM-Lenovo-Idea
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces$ cd MyPFunctionApp/
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces/MyPFunctionApp$ ls -l
total 0
-rwxrwxrwx 1 password password  99 Apr  9 19:15 function_app.py
-rwxrwxrwx 1 password password 288 Apr  9 19:15 host.json
-rwxrwxrwx 1 password password 117 Apr  9 19:15 local.settings.json
-rwxrwxrwx 1 password password 203 Apr  9 19:15 requirements.txt
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces/MyPFunctionApp$ python3 -m venv .venv
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces/MyPFunctionApp$ ls -l
total 0
-rwxrwxrwx 1 password password  99 Apr  9 19:15 function_app.py
-rwxrwxrwx 1 password password 288 Apr  9 19:15 host.json
-rwxrwxrwx 1 password password 117 Apr  9 19:15 local.settings.json
-rwxrwxrwx 1 password password 203 Apr  9 19:15 requirements.txt
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces/MyPFunctionApp$ ls -a
.  ..  .gitignore  .venv  .vscode  function_app.py  host.json  local.settings.json  requirements.txt
password@GTM-Lenovo-Ideapad5:/mnt/d/workspaces/python-workspaces/MyPFunctionApp$
```

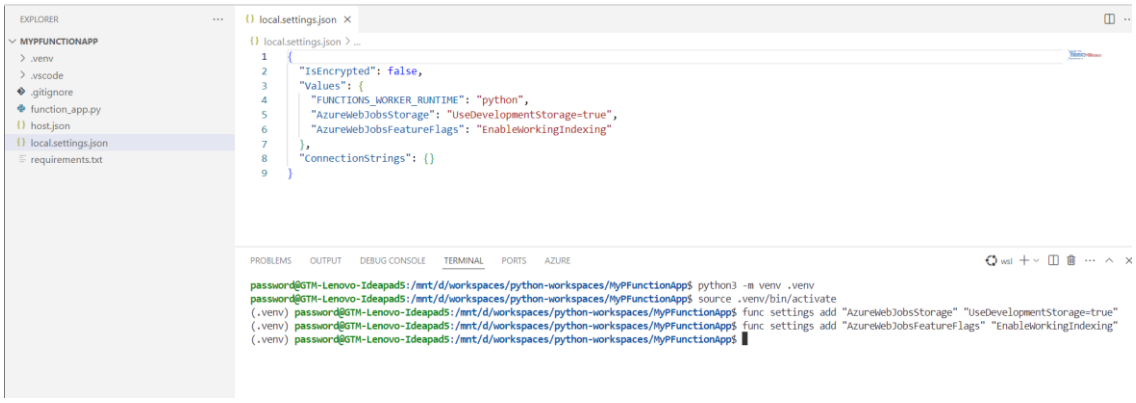
- Open the project in an IDE & switch to IDE Terminal & Activate the virtual environment



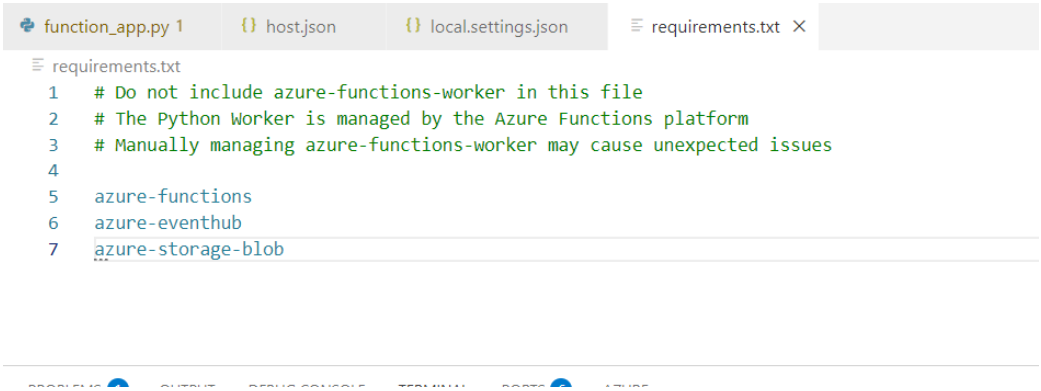
- Update local setting, using func command

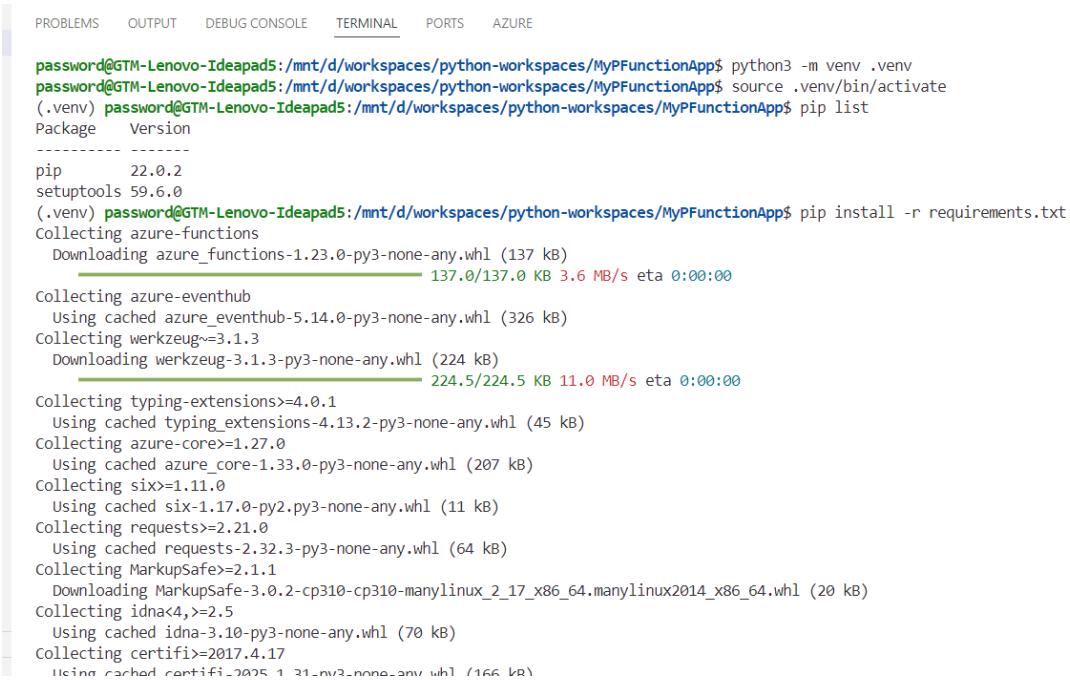


Python (V2 Model) Based Azure Function

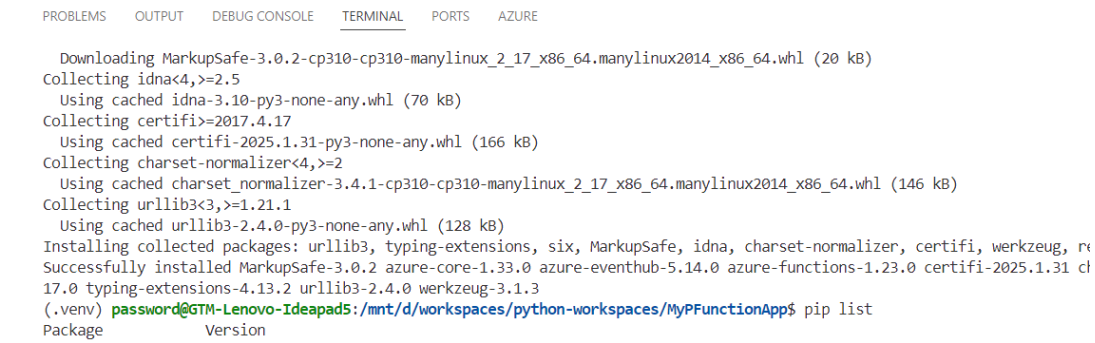


- Install packages



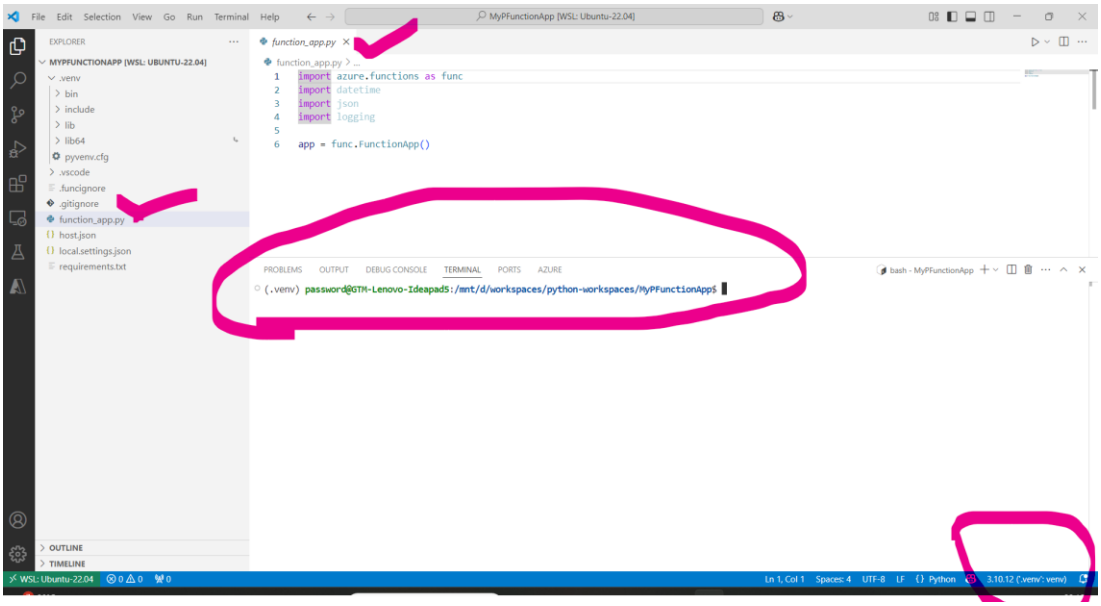


Python (V2 Model) Based Azure Function

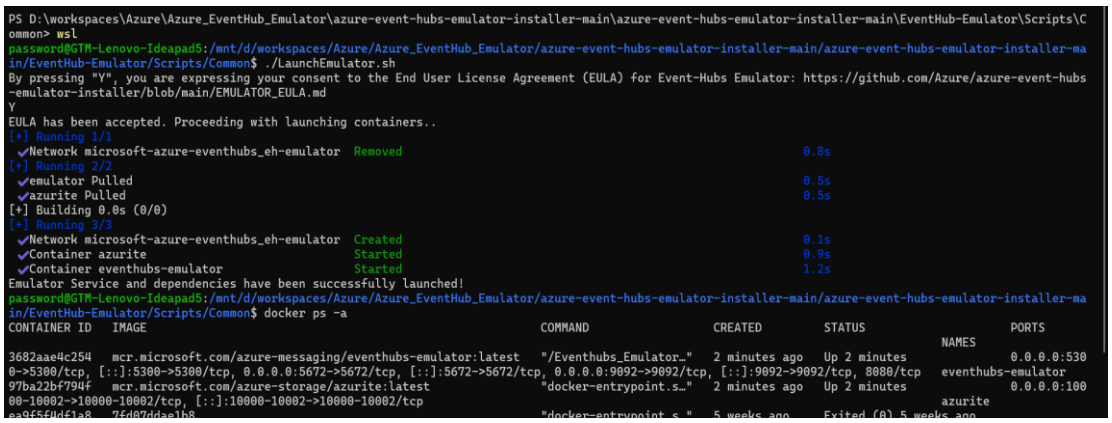


- Add python interpreter to virtual env
    - Restart VSCode and open it in WSL mode:
    - Install the WSL Extension: To work with WSL2 in VS Code, you need to install the WSL extension:
    - Open VS Code.
    - Go to the Extensions view by clicking the Extensions icon in the Activity Bar on the side of the window or by pressing Ctrl+Shift+X.
    - Search for “WSL” and click “Install.”
    - Activate WSL and open your project folder
    - Once the WSL extension is installed, you can activate WSL.
    - Open the Command Palette (Ctrl+Shift+P) in VS Code and Type “WSL: Connect to WSL”
    - VS Code will reopen with your project folder in the WSL environment.
    - Click on your python file and you will see that it is considering virtual environment’s python
    - If the project folder doesn’t open automatically, please remember to open it in wsl form (NOT like D:/… , it should be like /mnt/d/…)

Python (V2 Model) Based Azure Function



- Start Azure Event Hub Emulator in WSL:
    - Follow instructions written here - https://learn.microsoft.com/en-us/azure/event-hubs/test-locally-with-event-hub-emulator?tabs=automated-script%2Cusing-kafka
    - Download installer from https://github.com/Azure/azure-event-hubs-emulator-installer
    - Run Launchemulator.sh



- Write a basic function code (event hub trigger):

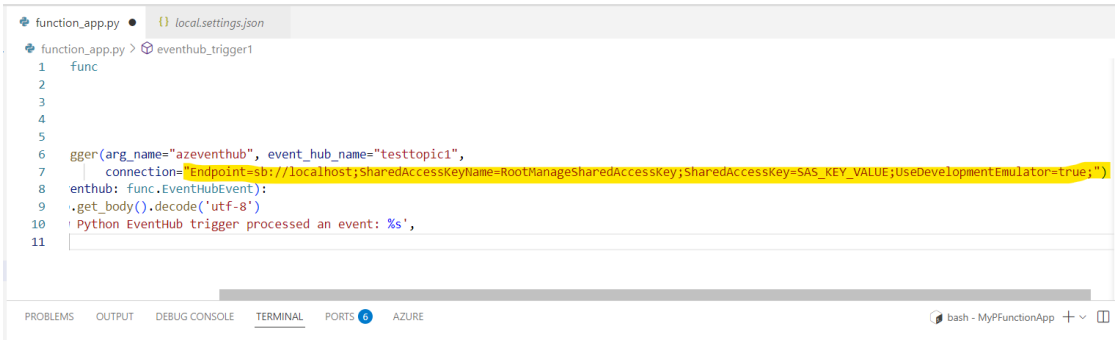
function_app.py | local.settings.json

function_app.py > eventhub_trigger1

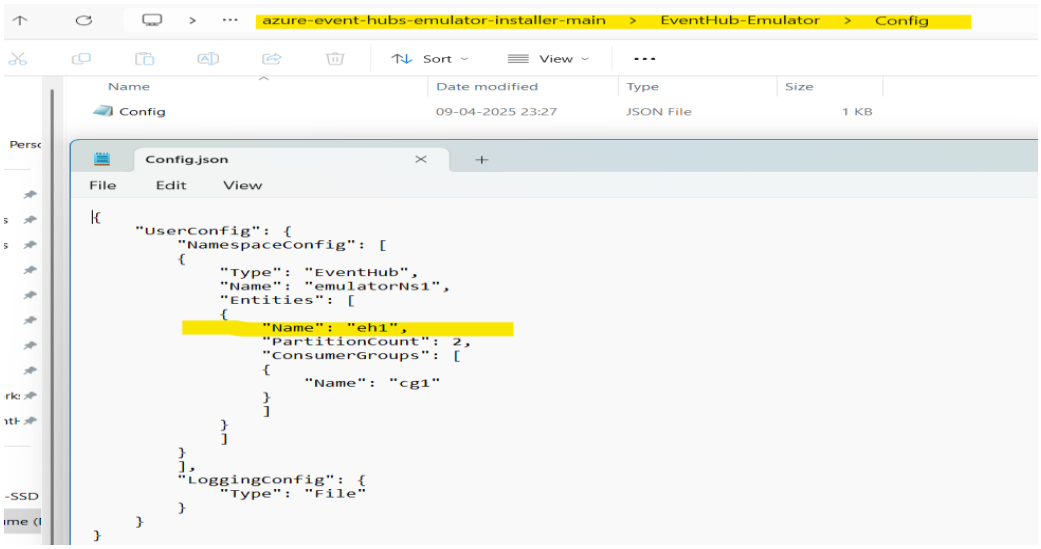
```python
import azure.functions as func
import logging

app = func.FunctionApp()

@app.event_hub_message_trigger(arg_name="azeventhub", event_hub_name="testtopic1",
                               connection="myeventhubkafkanamespace_RootManageSharedAccessKey_EVENTHUB")
def eventhub_trigger1(azeventhub: func.EventHubEvent):
    event_dec = azeventhub.get_body().decode('utf-8')
    logging.info('Gtms new Python EventHub trigger processed an event: %s',
                 event_dec)
```

- Update Connection string with Emulator connection to run it in local:

Python (V2 Model) Based Azure Function



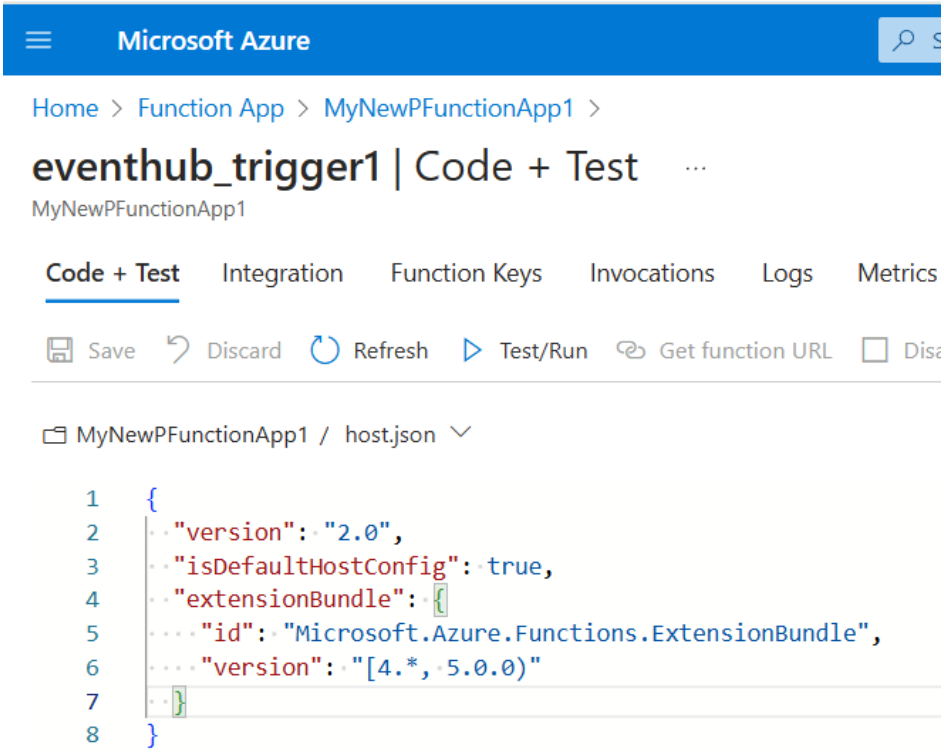
- Update event_hub_name with the name mentioned in config (Emulator):



- Delete all extentionBundle from host.json (relevant only for running in local azurehub Emulator / not recommended when deploying in Azure):
    - Local:

Python (V2 Model) Based Azure Function

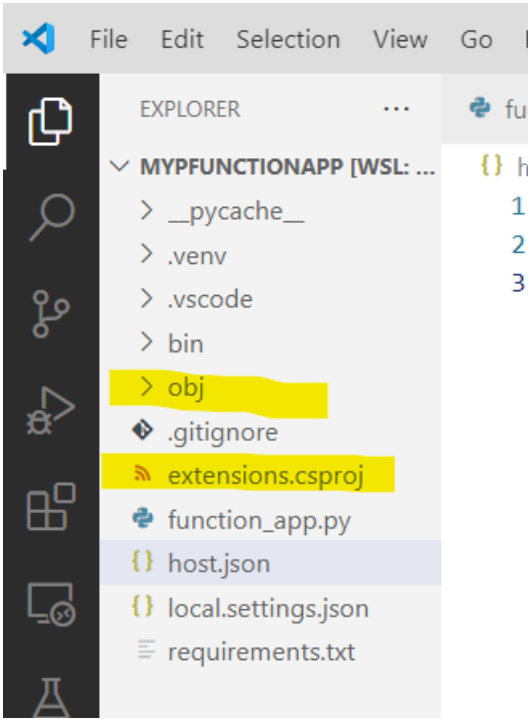
- In Azure:



- Run the following command from terminal (only for Local env):

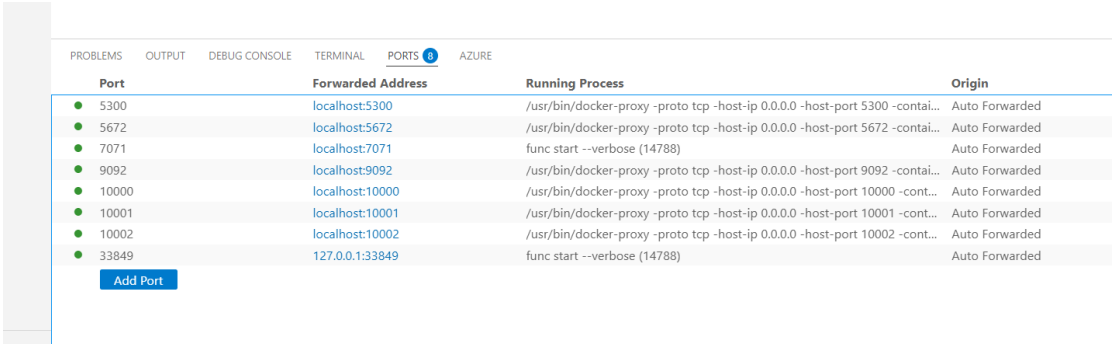
func extensions install --package Microsoft.Azure.WebJobs.Extensions.EventHubs --version 6.3.0

We can see the following files and folders in local directory (needed only for local azurehub emulator)



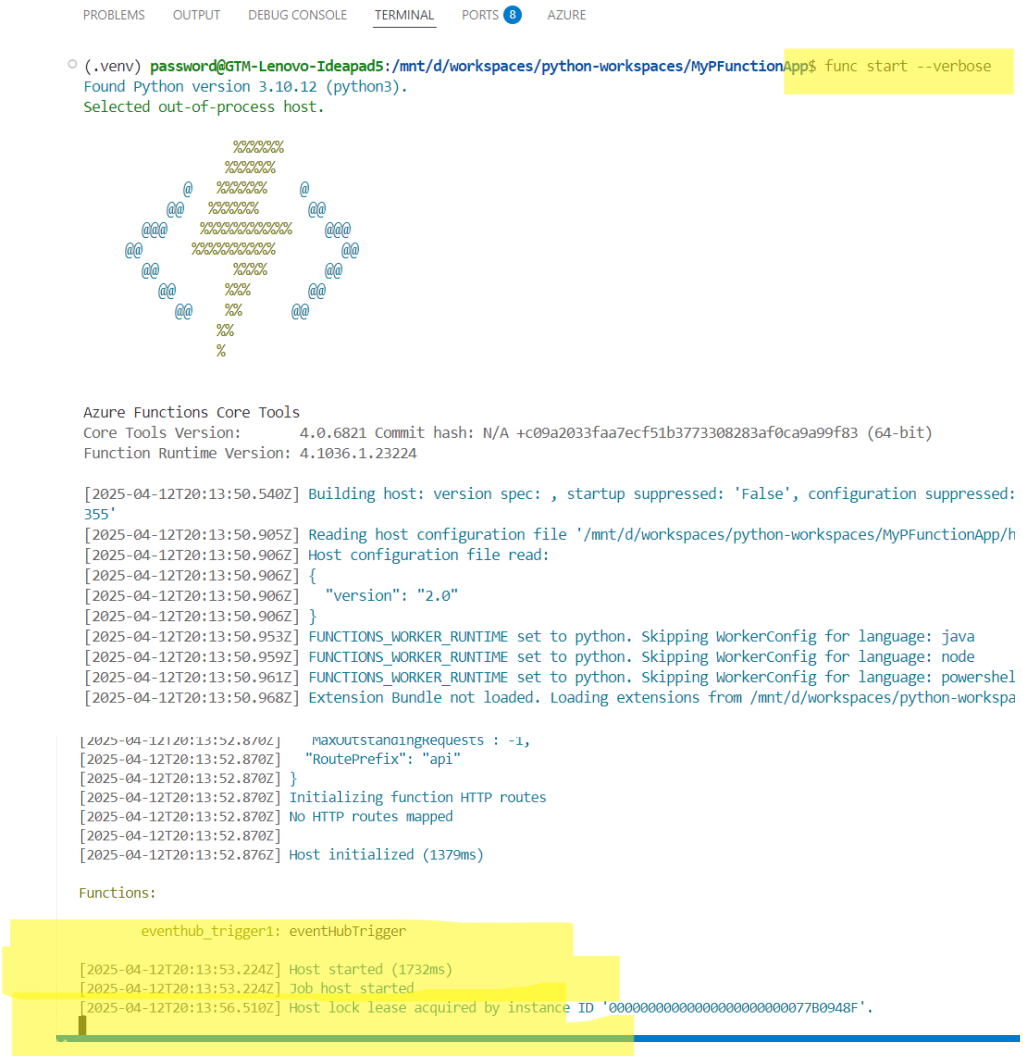
Python (V2 Model) Based Azure Function

- Check ports which are running:



- Start your function in Local:

  func start –verbose



- How to Test it?
  1. Create a http function who can publish the data to eventhub emulator

Python (V2 Model) Based Azure Function

function_app.py | host.json | local.settings.json

function_app.py > send_to_eventhub

```python
@app.event_hub_message_trigger(arg_name="azeventhub", event_hub_name="eh1",
                               connection="EventHubConnection")
def eventhub_trigger1(azeventhub: func.EventHubEvent):
    event_dec = azeventhub.get_body().decode('utf-8')
    logging.info('Gtms new Python EventHub trigger processed an event: %s',
                 event_dec)


@app.function_name(name="HttpTriggerToEventHub")
@app.route(route="send-to-eventhub", auth_level=func.AuthLevel.ANONYMOUS, methods=["POST"])
def send_to_eventhub(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('HTTP trigger function processed a request.')

    # Read data from the HTTP request
    data = req.get_json()

    # Event Hub connection string and name
    connection_str = "Endpoint=sb://localhost;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=SAS_KEY_VALUE;UseDevelopmentEmulator=tru
    eventhub_name = "eh1"

    # Create an Event Hub producer client
    producer = EventHubProducerClient.from_connection_string(
        conn_str=connection_str, eventhub_name=eventhub_name
    )

    # Send data to Event Hub
    with producer:
        event_data_batch = producer.create_batch()
        event_data_batch.add(EventData(str(data)))
        producer.send_batch(event_data_batch)

    return func.HttpResponse("Data sent to Event Hub successfully!")
```

2. Start the function and fetch the URL from the console

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 8 AZURE

```
[2025-04-12T20:39:18.549Z]   "MaxOutstandingRequests": -1,
[2025-04-12T20:39:18.549Z]   "RoutePrefix": "api"
[2025-04-12T20:39:18.550Z] }
[2025-04-12T20:39:18.550Z] Initializing function HTTP routes
[2025-04-12T20:39:18.551Z] Mapped function route 'api/send-to-eventhub' [POST] to 'HttpTriggerToEventHub'
[2025-04-12T20:39:18.551Z]
[2025-04-12T20:39:18.556Z] Host initialized (4526ms)

Functions:

        HttpTriggerToEventHub: [POST] http://localhost:7071/api/send-to-eventhub

        eventhub_trigger1: eventHubTrigger

[2025-04-12T20:39:18.905Z] Host started (4880ms)
[2025-04-12T20:39:18.906Z] Job host started
```

3. Publish data using postman or Bruno or equivalent

http://localhost:7071/api/send-to-eventhub

http://localhost:7071/api/send-to-eventhub

POST http://localhost:7071/api/send-to-eventhub

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "Category": "Eventhub Publish testing",
    "Environment" : "Local",
    "Name" : "Test using python based function"
}
```

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize Text

```
Data sent to Event Hub successfully!
```

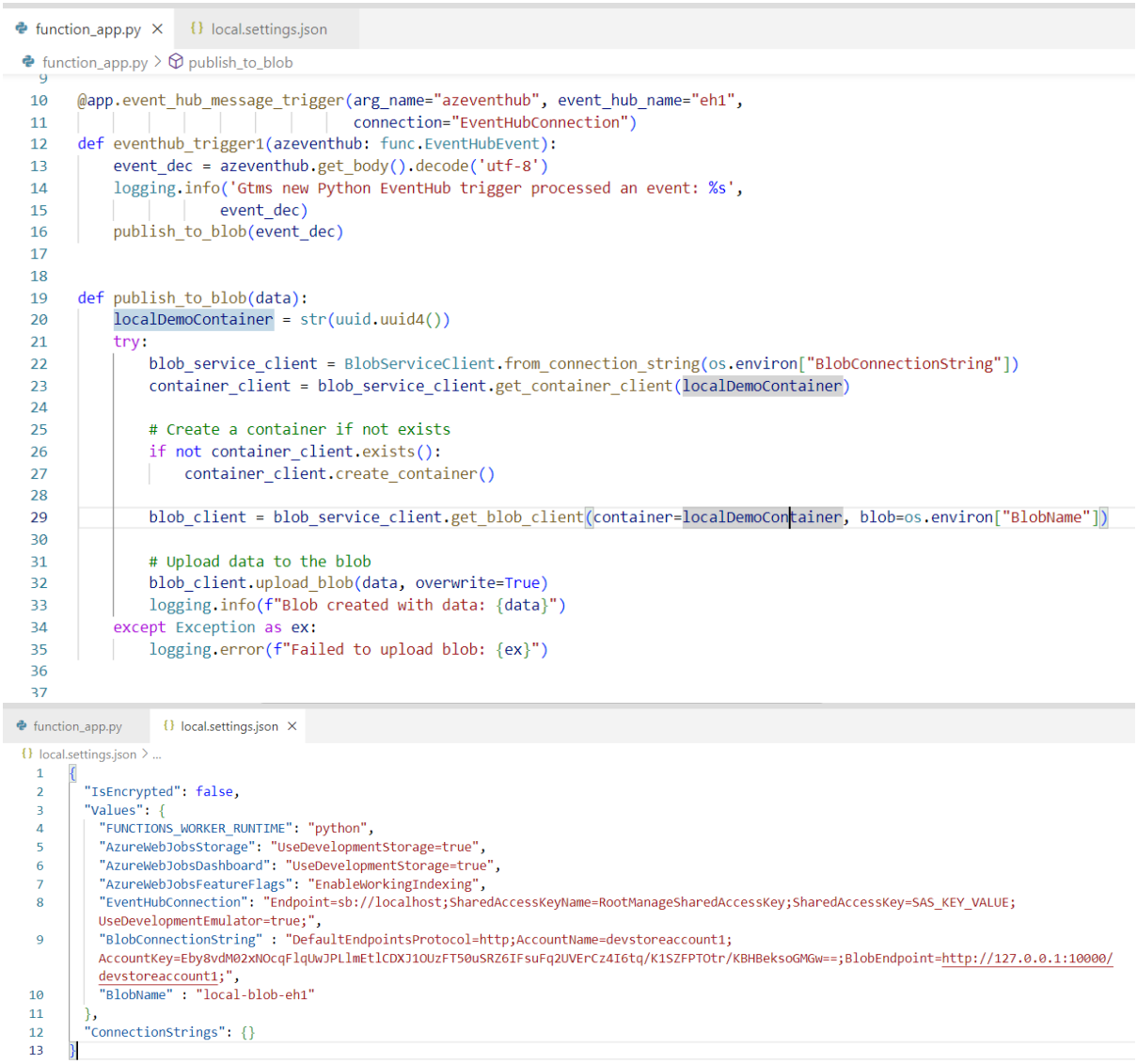
Python (V2 Model) Based Azure Function

```
@app.event_hub_message_trigger(arg_name="azeventhub", event_hub_name="eh1",
                               connection="EventHubConnection")
def eventhub_trigger1(azeventhub: func.EventHubEvent):
    event_dec = azeventhub.get_body().decode('utf-8')
    logging.info('Gtms new Python EventHub trigger processed an event: %s',
                 event_dec)


@app.function_name(name="HttpTriggerToEventHub")
@app.route(route="send-to-eventhub", auth_level=func.AuthLevel.ANONYMOUS, methods=["POST"])
def send_to_eventhub(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('HTTP trigger function processed a request.')

    # Read data from the HTTP request
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS 8  AZURE                                   func - MyPFunctionApp

[2025-04-12T20:46:46.009Z] Link state changed: <LinkState.DETACH_SENT: 4> -> <LinkState.DETACHED: 0>
[2025-04-12T20:46:46.010Z] Received FunctionInvocationRequest, request ID: 50b14f6d-1d12-413e-8bdb-811b64064a71, function ID: 7dbf07eb-c78b-5bd7-8fc8-231308e56f05, function name:
eventhub_trigger1, invocation ID: 0201d548-bb15-48b8-87ca-b8bf95649e41, function type: sync, timestamp (UTC): 2025-04-12 20:46:46.005913, sync threadpool max workers: 1000
[2025-04-12T20:46:46.011Z] Executed 'Functions.HttpTriggerToEventHub' (Succeeded, Id=99cbb852-68eb-4b4b-ba91-cfbb60b74209, Duration=557ms)
[2025-04-12T20:46:46.011Z] Gtms new Python EventHub trigger processed an event: {'Category': 'Eventhub Publish testing', 'Environment': 'Local', 'Name': 'Test using python based
function'}
[2025-04-12T20:46:46.012Z] Executed 'Functions.eventhub_trigger1' (Succeeded, Id=0201d548-bb15-48b8-87ca-b8bf95649e41, Duration=12ms)
[2025-04-12T20:46:46.013Z] Executed HTTP request: {
[2025-04-12T20:46:46.013Z]   "requestId": "71c1590a-a648-4355-868b-14e7df83b4e6",
[2025-04-12T20:46:46.013Z]   "identities": "",
[2025-04-12T20:46:46.014Z]   "status": "200",
[2025-04-12T20:46:46.014Z]   "duration": "562"
[2025-04-12T20:46:46.014Z] }
```

```
Ln 24, Col 1  Spaces: 4  UTF-8  LF  {} Python  3.10.12 ('.venv': venv)
```

4. Add code to writing the data to blob



function_app.py × | {} local.settings.json

function_app.py > publish_to_blob

```
@app.event_hub_message_trigger(arg_name="azeventhub", event_hub_name="eh1",
                               connection="EventHubConnection")
def eventhub_trigger1(azeventhub: func.EventHubEvent):
    event_dec = azeventhub.get_body().decode('utf-8')
    logging.info('Gtms new Python EventHub trigger processed an event: %s',
                 event_dec)
    publish_to_blob(event_dec)


def publish_to_blob(data):
    localDemoContainer = str(uuid.uuid4())
    try:
        blob_service_client = BlobServiceClient.from_connection_string(os.environ["BlobConnectionString"])
        container_client = blob_service_client.get_container_client(localDemoContainer)

        # Create a container if not exists
        if not container_client.exists():
            container_client.create_container()

        blob_client = blob_service_client.get_blob_client(container=localDemoContainer, blob=os.environ["BlobName"])

        # Upload data to the blob
        blob_client.upload_blob(data, overwrite=True)
        logging.info(f"Blob created with data: {data}")
    except Exception as ex:
        logging.error(f"Failed to upload blob: {ex}")
```

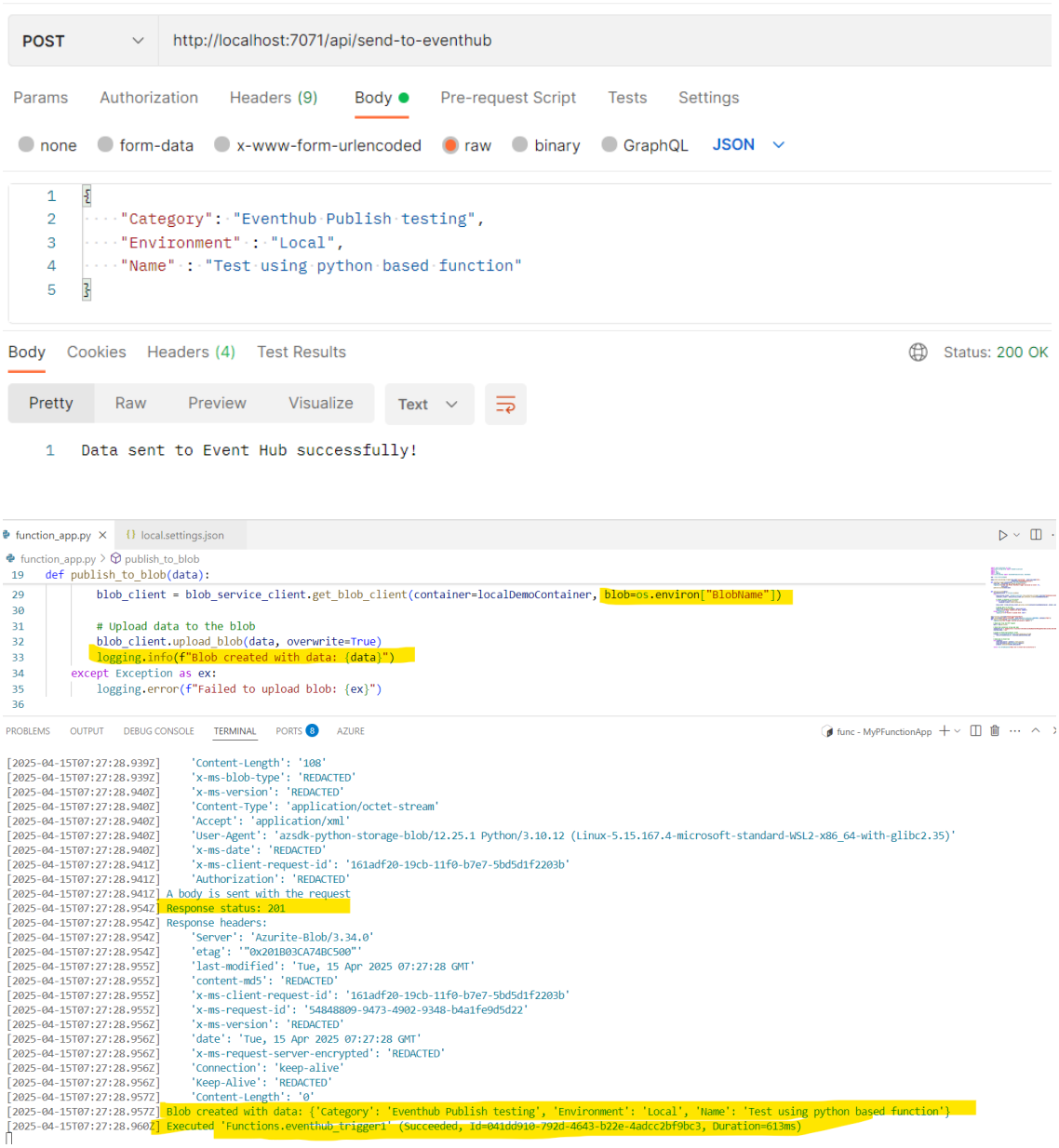
function_app.py | {} local.settings.json ×

{} local.settings.json > ...

```
{
  "IsEncrypted": false,
  "Values": {
    "FUNCTIONS_WORKER_RUNTIME": "python",
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "AzureWebJobsDashboard": "UseDevelopmentStorage=true",
    "AzureWebJobsFeatureFlags": "EnableWorkingIndexing",
    "EventHubConnection": "Endpoint=sb://localhost;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=SAS_KEY_VALUE;
    UseDevelopmentEmulator=true;",
    "BlobConnectionString" : "DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;
    AccountKey=Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==;BlobEndpoint=http://127.0.0.1:10000/
    devstoreaccount1;",
    "BlobName" : "local-blob-eh1"
  },
  "ConnectionStrings": {}
}
```

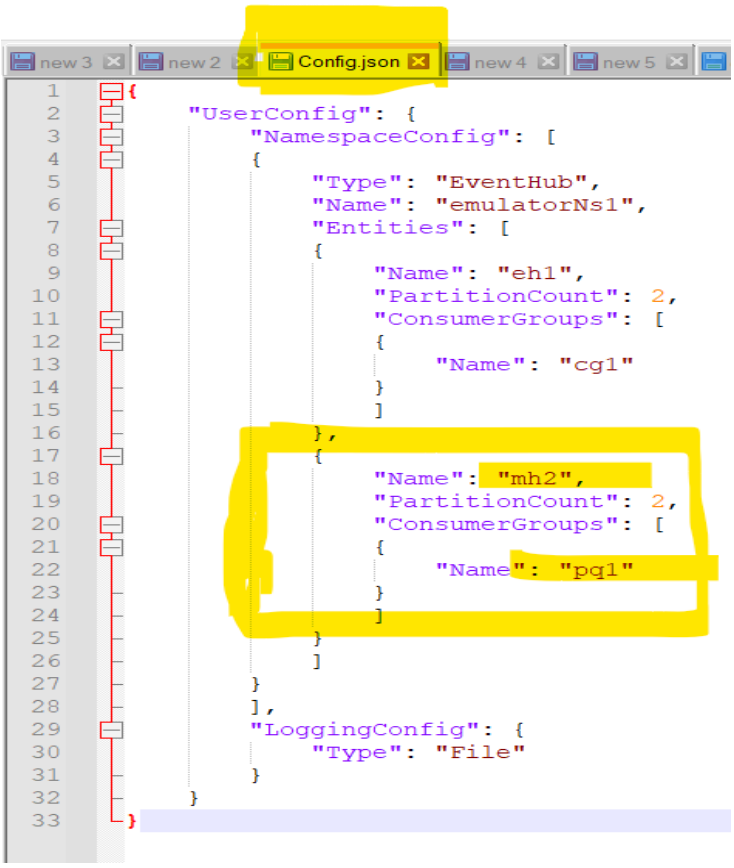
5. Run the function again and test

Python (V2 Model) Based Azure Function



6. Consuming from two topics
   - Add a new topic in azurehub emulator config file

Python (V2 Model) Based Azure Function



- Add another function in the same file (beauty of V2 model)

- Restart Azurite
- Restart Functions
- Test

Python (V2 Model) Based Azure Function