



---

## Developing Python REST APIs Using Azure Functions: A Serverless Approach

**Bhargav Bachina**

---

**Abstract** This paper delves into the versatility of Azure Functions, illustrating how developers can effortlessly deploy lightweight code snippets in the cloud, alleviating concerns about infrastructure management. With support for multiple programming languages and a consumption-based pricing model, Azure Functions offer adaptability and scalability to meet diverse application needs. Specifically focusing on creating a Python REST API using Azure Functions, the paper elucidates fundamental concepts, prerequisites, implementation steps, deployment procedures, monitoring strategies, and concludes with a succinct summary and actionable conclusions for optimizing serverless computing workflows.

**Keywords** Python, Azure, Cloud Computing, Programming, Software Development

---

### 1. Introduction

AN AZURE Function is a simple way of running small pieces of code in the cloud. You don't have to worry about the infrastructure required to host that code. You can write the Function in C#, Java, JavaScript, PowerShell, Python, or any of the languages that are listed in the Supported languages in the Azure Functions (<https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages>) article. In addition, with the consumption plan option, you only pay for the time when the code runs. Azure automatically scales your function in response to the demand from users.

In this post, we will go through how to write Python REST API using Azure Functions.

- Basics of Azure Functions
- Prerequisites
- Example Project
- Creating a Function App
- Creating a Python REST API
- Deploy to the Function App
- Demo
- Monitor the API
- Summary
- Conclusion

### 2. Basics of Azure Functions

As I stated above you can run the small pieces of code with Azure Functions without worrying about any underlying infrastructure required. Azure Functions scales based on demand. We have several templates you can choose from based on the trigger types below.

#### A. HTTP Trigger

You need to use this trigger when you want to execute the code in response to a request sent through the HTTP protocol.



### B. Timer Trigger

Use this template when you want the code to execute according to a schedule.

### C. Blob Trigger

Use this template when you want the code to execute when a new blob object is added to an Azure Storage account.

### D. CosmosDB Trigger

Use this template when you want the code to execute in response to new or updated documents in a NoSQL database.

By default, functions have a timeout of 5 minutes; you can extend up to 10 minutes. This time limit is even restricted to 2.5 minutes when the Azure functions are driven by HTTPTrigger.

## 3. Prerequisites

You need to know a lot of things as prerequisites if you want to write a serverless Python REST API. First, you need to create two accounts: a GitHub account to store the source code and Microsoft Account to deploy that code using Function App Service. Let's create these accounts by following the below links. You can start both for free.

- Github Account (<https://github.com/>)
- Microsoft Azure Account (<https://azure.microsoft.com/en-us/>)

All the API code is written in Python Azure functions. You need to be familiar with the following.

- Azure Functions (<https://azure.microsoft.com/en-us/services/functions/>)
- Azure Functions extension for Visual Studio Code (<https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions>)

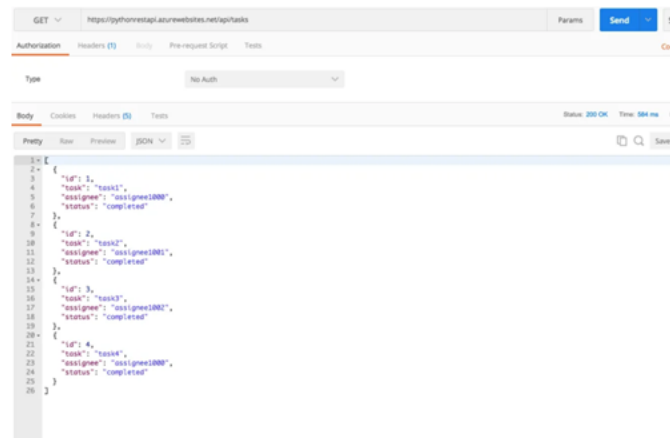
## 4. Example Project

This is a simple Python REST API where you can add tasks, delete tasks, update tasks, and get tasks. Here is an example project you can clone and run on your local machine.

// clone the project `git clone https://github.com/bbachi/serverless-python-restapi-azure.git`

// With Azure Functions extension installed `func start`

### Testing



```

GET https://pythonrestapi.azurewebsites.net/api/tasks
Type: No Auth
Body:
[
  {
    "id": 1,
    "task": "task1",
    "assignee": "assignee000",
    "status": "complete"
  },
  {
    "id": 2,
    "task": "task2",
    "assignee": "assignee001",
    "status": "complete"
  },
  {
    "id": 3,
    "task": "task3",
    "assignee": "assignee002",
    "status": "complete"
  },
  {
    "id": 4,
    "task": "task4",
    "assignee": "assignee000",
    "status": "complete"
  }
]

```

Figure 1: Testing the API

## 5. Creating a Function App

Functions are hosted in an execution context called a function app. You define function apps to logically group and structure your functions and a compute resource in Azure.

You can create a function app in two ways: you can directly create that in the Azure portal and another way is to create it through VSCode. We will see how we can create a function app through VSCode.



First, you need to install the Azure Functions extension. Once you install the Azure Functions extension for Visual Studio Code you can create the entire project from VSCode itself. The first thing you should do is to authenticate with Azure Account and select the appropriate subscription by logging in to Azure.

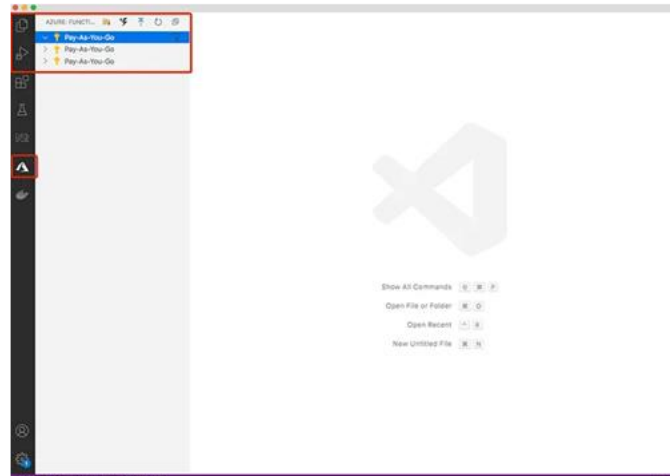


Figure 2: Azure Functions Extension

You can right-click on any available subscriptions and select the create Azure Function App option and select the tech stack. I selected Python 3.8 as a runtime stack.



Figure 3: Creating a Function App

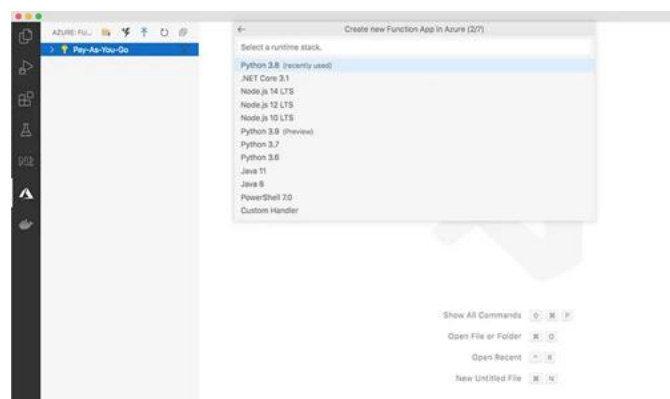


Figure 4: Selecting the running stack

You need to select the hosting plan and we usually select the consumption plan if your functions don't run longer than 10 minutes.

Function apps may use one of two types of service plans. The first service plan is the Consumption service plan. This is the plan that you choose when using the Azure serverless application platform. The Consumption service plan provides automatic scaling and bills you when your functions are running. The Consumption plan comes with a configurable timeout period for the execution of a function. By default, it is 5 minutes, but may be configured to have a timeout as long as 10 minutes.



The second plan is called the Azure App Service plan. This plan allows you to avoid timeout periods by having your function run continuously on a VM that you define. When using an App Service plan, you are responsible for managing the app resources the function runs on, so this is technically not a serverless plan. However, it may be a better choice if your functions are used continuously or if your functions require more processing power or execution time than the Consumption plan can provide.

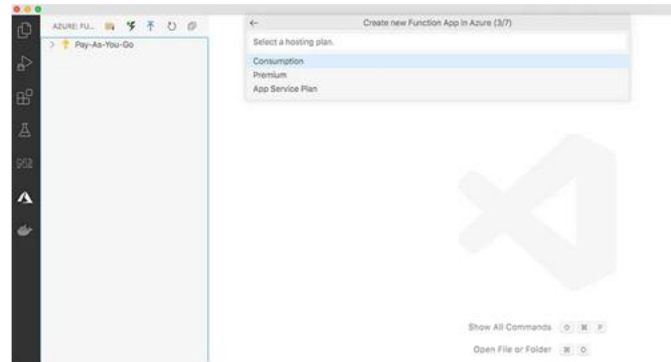


Figure 5: Selecting the plan

Select the resource group or create the new one

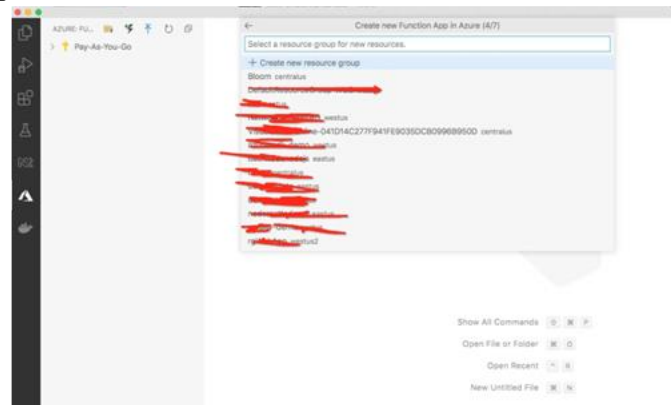


Figure 6: Creating Resource Group

Every function app should be linked with a storage account. When you create a function app, it must be linked to a storage account. You can select an existing account or create a new one. The function app uses this storage account for internal operations such as logging function executions and managing execution triggers.

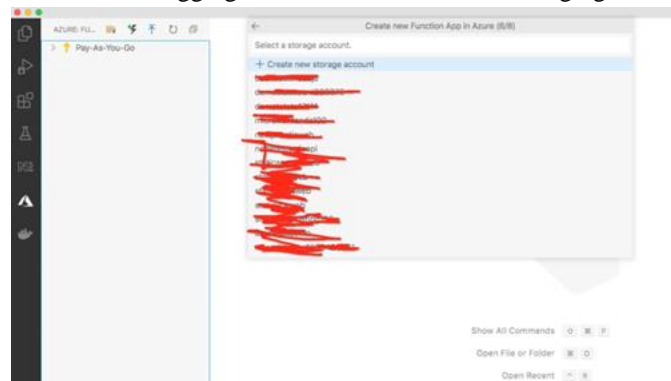


Figure 7: Creating a storage account

You can create an application insights resource to monitor your application.



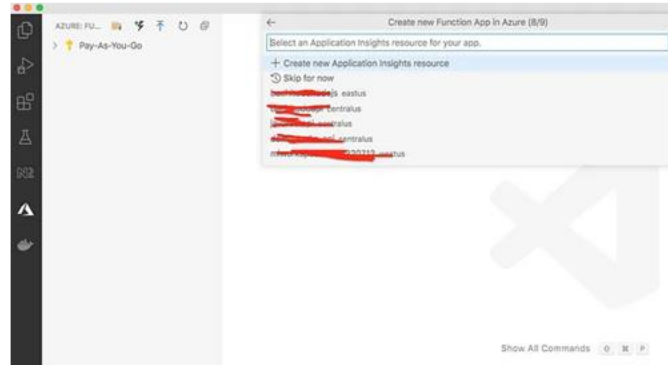


Figure 8: Creating an application insights resource

Once the creation is done you can see the function app like below.

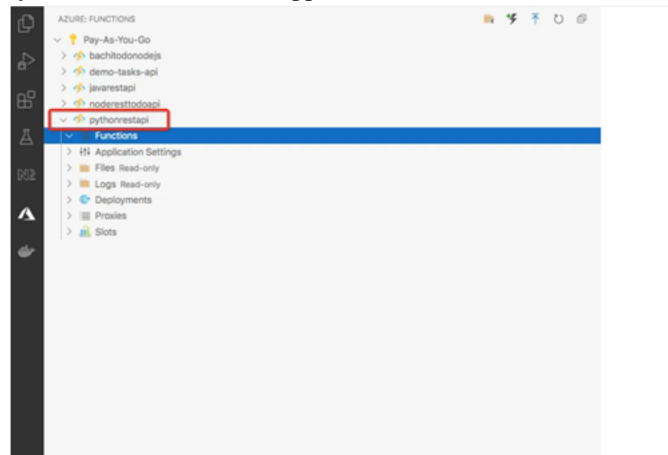


Figure 9: Function App

All the things such as resource group, storage account, function app, etc. you created in the VSCode can be seen in the Azure portal.

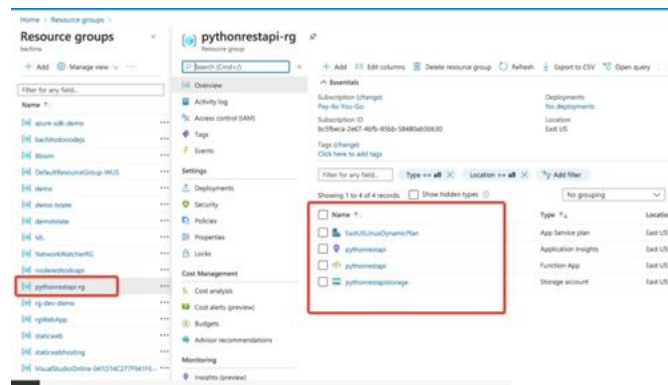


Figure 10: Azure Portal

### 6. Creating a Python REST API

We have created a function app and let's create a new local project by clicking on the folder icon in the VSCode.

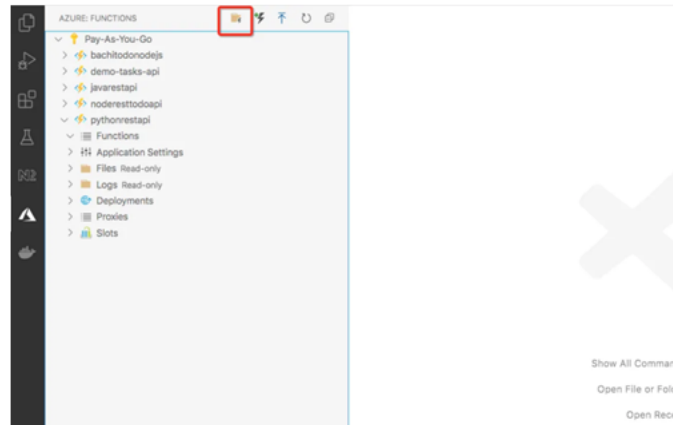


Figure 11: Creating a local project

You need to select the language and template which HTTP trigger in this case.

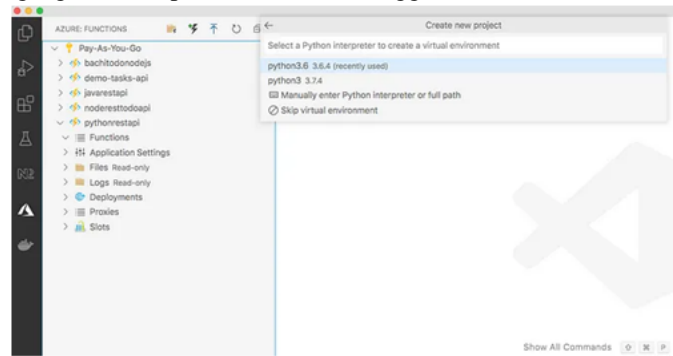


Figure 12: Selecting the language

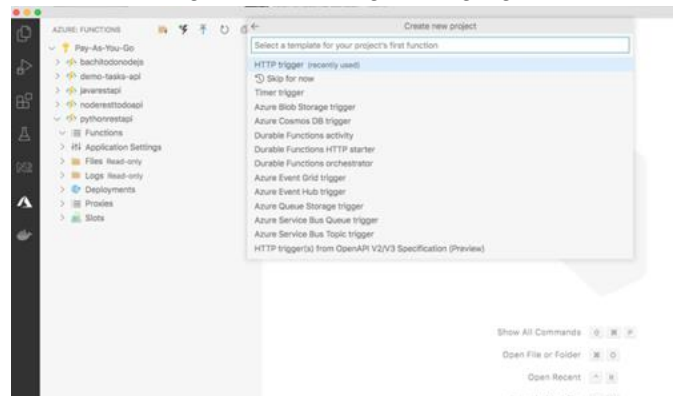


Figure 13: HTTP trigger

Once you are done with the creation you will see the below function

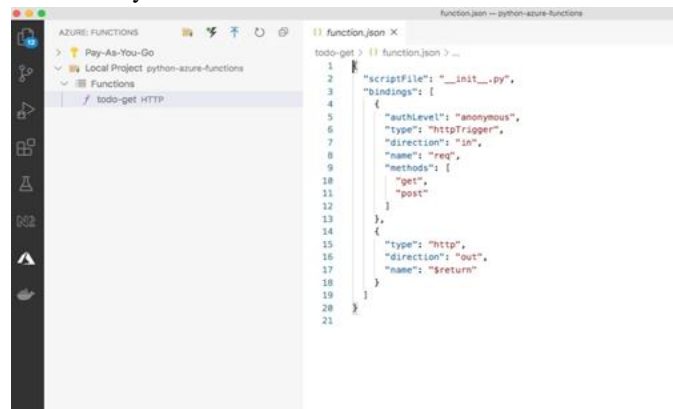


Figure 14: todo-get function





We have seen how to test the project. Let's implement the four functions for that we need some helper classes as below. I have defined these helper classes under the folder called **services**.

[https://gist.github.com/bbachi/82dd1d3b422c801457af72d57239a952#file-dummy\\_data-py](https://gist.github.com/bbachi/82dd1d3b422c801457af72d57239a952#file-dummy_data-py)

We have four functions defined for the four API routes. Let's have a look at the todos-get function. We have two files for each function: **\_\_init\_\_.py** and **function.json**.

#### A. todo-get \_\_init\_\_.py

Every function has an **\_\_init\_\_.py** file as the starting point and receives an HTTP request and sends an HTTP response back. You can access the request params and body in the HTTP request and you can add the response body in the Response Object. You can import other files into this for the processing, database access layer, etc.

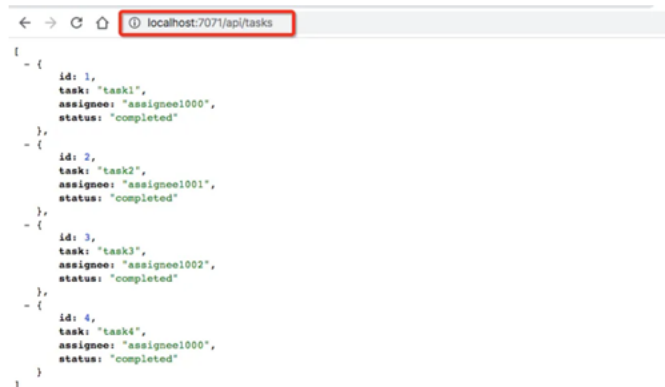
[https://gist.github.com/bbachi/c83a134b67bca20a233e6fb700b5645e#file-\\_\\_init\\_\\_.py](https://gist.github.com/bbachi/c83a134b67bca20a233e6fb700b5645e#file-__init__.py)

#### B. todo-get function.json

Every function has a **function.json** file which defines the route, type of HTTP method, directions of the request, and response objects. For example, this is the get request and the direction is in with the req object. The object is res for the direction out.

<https://gist.github.com/bbachi/d67572bfc7aa6873614fa1a51225e68e#file-function.json>

You can test the API by starting the app with the command `func start`



```

{
  - {
    id: 1,
    task: "task1",
    assignee: "assignee1000",
    status: "completed"
  },
  - {
    id: 2,
    task: "task2",
    assignee: "assignee1001",
    status: "completed"
  },
  - {
    id: 3,
    task: "task3",
    assignee: "assignee1002",
    status: "completed"
  },
  - {
    id: 4,
    task: "task4",
    assignee: "assignee1000",
    status: "completed"
  }
}

```

Figure 18: Testing the API

#### C. todo-post \_\_init\_\_.py

Since this is a post request you can see that we are reading the body of the request and passing it to helper functions.

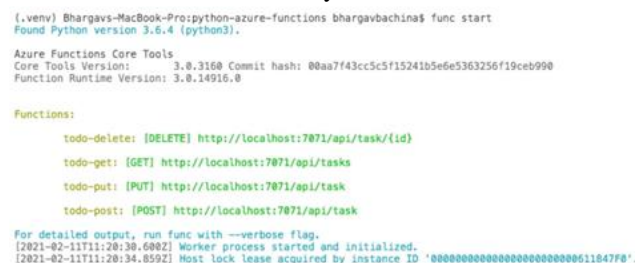
<https://gist.github.com/bbachi/c5cc88071d5572ce42289d622d2c5bfa>

#### D. todo-post function.json

You can notice that there is a change in the methods array.

<https://gist.github.com/bbachi/e817cabee435fd6a4e5b4ddf8a67c1a7#file-function.json>

You can define the other two functions in the same way. You can see four functions in total.



```

(.venv) Bhargavs-MacBook-Pro:python-azure-functions bhargavbachina$ func start
Found Python version 3.6.4 (python3).

Azure Functions Core Tools
Core Tools Version:      3.0.3160 Commit hash: 00aa7f43cc5c5f15241b5e6e5363256f19ceb990
Function Runtime Version: 3.0.14916.0

Functions:

  todo-delete: [DELETE] http://localhost:7071/api/task/{id}
  todo-get: [GET] http://localhost:7071/api/tasks
  todo-put: [PUT] http://localhost:7071/api/task
  todo-post: [POST] http://localhost:7071/api/task

For detailed output, run func with --verbose flag.
[2021-02-11T11:20:30.600Z] Worker process started and initialized.
[2021-02-11T11:20:34.859Z] Host lock lease acquired by instance ID '0000000000000000000000000000000011847f8'.

```

Figure 19: Python REST API Endpoints

## 7. Deploy to the Function App

We have developed the whole REST API as the local project. Let's deploy this into the Function App. When you deploy it to the function app you can actually see it in the Azure portal.

To deploy this, you can click on the up arrow icon in the Azure Functions extension in the VSCode.





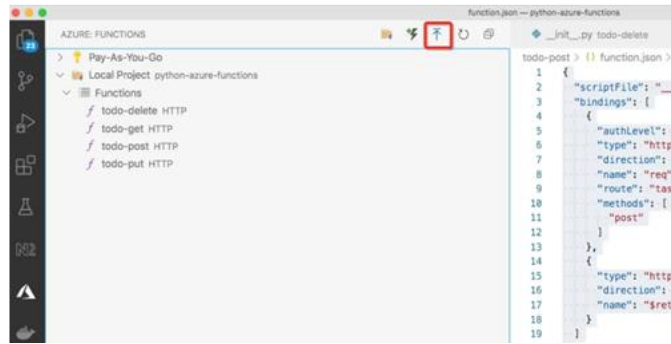


Figure 20: Deploy it to the Function App

Select the subscription and the functions app we created above

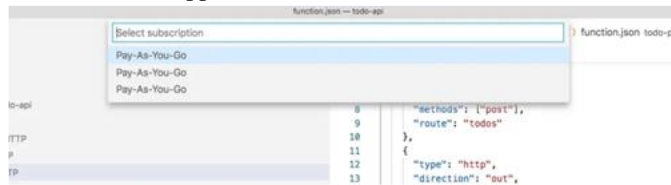


Figure 21: Selecting the subscription

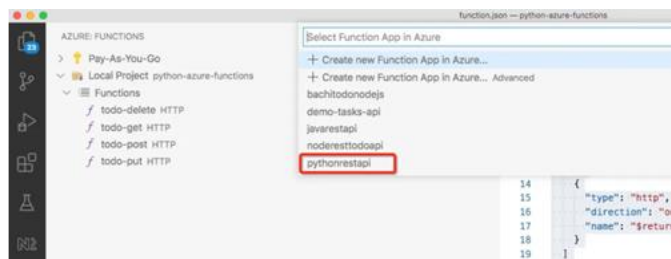


Figure 22: Selecting the Function App

Finally, Deploy

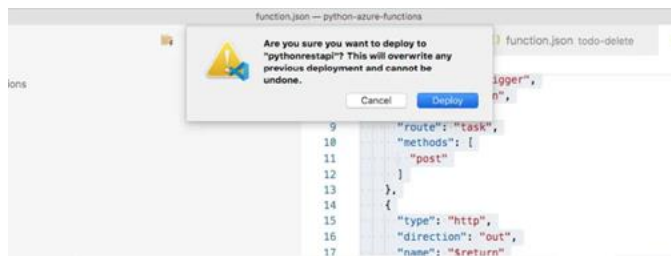


Figure 23: Deploy

Once the deployment is done, you can see all the Azure Function app functions like below.

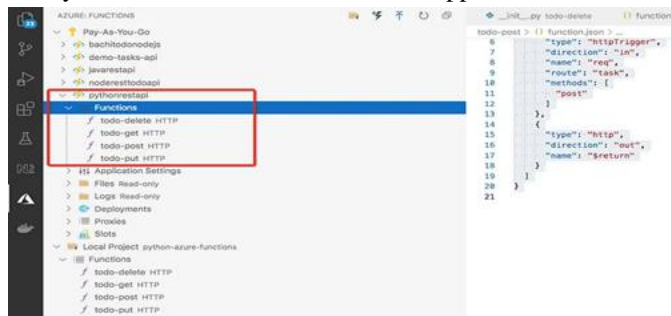


Figure 24: Deployment Done

You can actually see your functions in the Azure portal as well.



Figure 25: Azure Functions

## 8. Demo

Let's have a quick demo from the Azure portal. You can see all the functions in the Azure portal. You can even see the code and test it out by clicking on the specific function. When you go to the overview page you can see the API endpoint you can access it from the browser or any other client with it.

[https://miro.medium.com/v2/resize:fit:1400/1\\*sAq5stm6upi4-6pJnj20MQ.gif](https://miro.medium.com/v2/resize:fit:1400/1*sAq5stm6upi4-6pJnj20MQ.gif)

## 9. Monitor the API

Since we have opted in Application insights while creating the function API. You can see the logs on the Azure portal. You can check the logs when you click on any function and click on the Monitor.

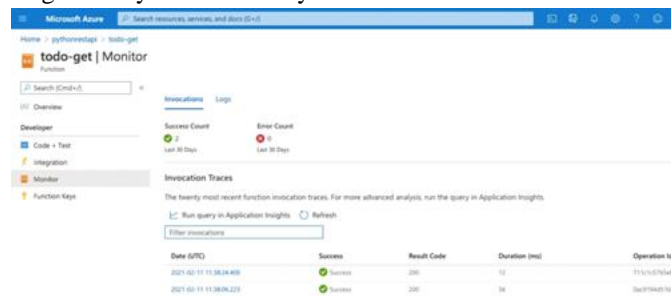


Figure 26: Monitor the API

## 10. Summary

- The container that groups functions into a logical unit for easier management, deployment and sharing of resources is called Function app.
- Azure Functions is a serverless application platform. It allows developers to host business logic that can be executed without provisioning infrastructure.
- You can write Azure functions in several languages such as C#, java, javascript, typescript, Python, etc.
- You need to use this trigger when you want to execute the code in response to a request sent through the HTTP protocol.
- You need to install the Azure Functions extension. Once you install the Azure Functions extension for Visual Studio Code you can create the entire project from VSCode itself.
- Function apps may use one of two types of service plans. The first service plan is the Consumption service plan, and another is the App Service Plan.
- Every function app should be linked with a storage account. When you create a function app, it must be linked to a storage account.
- You need an Azure Functions extension (<https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions>) for Visual Studio Code to develop the entire thing locally.
- When you deploy it to the function app you can see it in the Azure portal.



- To deploy from local, you can click on the up-arrow icon in the Azure Functions extension in the VSCode.
- You can monitor logs of the API by accessing the Monitor section of each function on the Azure portal.

## **11. Conclusion**

In summary, this paper has provided a comprehensive overview of Azure Functions, highlighting their utility in enabling developers to deploy lightweight code snippets seamlessly in the cloud. By addressing concerns surrounding infrastructure management and offering support for multiple programming languages along with a consumption-based pricing model, Azure Functions present a versatile solution adaptable to various application requirements. The focus on creating a Python REST API using Azure Functions has elucidated key concepts, implementation steps, deployment procedures, and monitoring strategies. As a result, developers are equipped with valuable insights and actionable recommendations for optimizing serverless computing workflows, paving the way for enhanced efficiency and scalability in cloud-based development endeavors.

## **References**

- [1]. Official Azure Functions Documentation <https://learn.microsoft.com/en-us/azure/azure-functions/>
- [2]. Python Documentation <https://docs.python.org/3/>
- [3]. Azure Cloud <https://azure.microsoft.com/en-us>

