Advanced Python

Surendra Panpaliya

Week1 (Mon, Tue, Thurs)

Day 1: Python Recap + Environment & Tooling

Day 2: Functional Programming & Object-Oriented Design

Day 3: Advanced Python Concepts

Week2 (Mon, Tue, Wed, Thurs)

- Day 4: Concurrency and Async Programming
- Day 5: Web Services with FastAPI
- Day 6: Azure Functions & Cloud Deployment
- Day 7: Testing, Linting & Final Project

Day 6: Azure Functions & Cloud Deployment

- Azure Functions for Python:
- Local setup and deployment
- Comparing Dockerized App vs Azure Function
- CI/CD in Azure Pipelines / GitHub Actions

Day 6: Azure Functions & Cloud Deployment

Hands-On Lab:

Convert API to Azure Function and deploy

C# Azure Functions vs Python Azure Functions

Software Installation Requirements

Azure CLI Version: 2.50.0+

https://docs.microsoft.com/cli/azure/install-azure-cli

Azure Functions Tools:

https://docs.microsoft.com/azure/azure-functions/functions-run-local#v2

Azure Functions

What is Azure Function?

AZURE FUNCTION IS A

SERVERLESS COMPUTE SERVICE

PROVIDED BY MICROSOFT AZURE.

What is Azure Function?

ALLOWS YOU TO RUN

SMALL PIECES OF CODE

(CALLED *FUNCTIONS*)

WITHOUT HAVING TO

MANAGE INFRASTRUCTURE.

What is Azure Function?

You simply write the code,

upload it to Azure, and

Azure takes care of

provisioning, scaling, and

managing the underlying infrastructure.

Why Azure Function?

Designed for scenarios where

Run code **in response to events**

HTTP requests,

timers,

blob changes

Why Azure Function?

Designed for scenarios where

Write **event-driven,**

microservice-style code quickly

Why Azure Function?

Build **lightweight APIs**,

automated workflows, or

background tasks

without worrying about servers

Why Azure Function?

Reduce costs with

consumption-based billing

(you pay only when the function runs)

Benefits of using Azure Function

Benefit	Description
1. Serverless Architecture	No need to provision or manage servers. Perfect for microservices or lightweight modules in larger web applications.
2. Scalability	Automatically scales based on demand – from a few requests per day to thousands per second.

Benefits of using Azure Function

Benefit	Description
3. Cost-Effective	You pay only for the time your code runs (in the consumption plan), making it highly efficient for sporadic workloads.
4. Event-Driven	Can be triggered by HTTP requests, database changes, queues, etc. – enabling real-time responsiveness in your web app.

Benefits of using Azure Function

Benefit	Description
5. Quick Deployment	Easily integrate with CI/CD pipelines (e.g., GitHub Actions, Azure DevOps) to deploy updates quickly.
6. Language Support	Supports multiple languages – C#, Python, JavaScript, Java, PowerShell, etc.

Benefits of using Azure Function

Benefit	Description
7. Integration Friendly	Natively integrates with other Azure services (Blob Storage, Cosmos DB, Event Grid, Service Bus, etc.) to build full-stack solutions.
8. Secure & Reliable	Supports authentication/authorization, custom domains, TLS/SSL, and integrates with Azure Key Vault and App Insights.

Example Use Cases in Web Apps

BACKEND API FOR A WEB FORM OR CHATBOT

IMAGE PROCESSING AFTER FILE UPLOAD

EMAIL NOTIFICATIONS AFTER USER SIGNUP

Example Use Cases in Web Apps

Scheduled cleanup jobs (e.g., database pruning)

Webhooks and integrations (e.g., Slack, Stripe, GitHub)

Azure
Function

Python

Azure Functions for Python Steps

STEP-BY-STEP GUIDE FOR

LOCAL SETUP AND DEPLOYMENT OF

AZURE FUNCTIONS FOR PYTHON

Learn Azure Functions Python V2

Local Setup

Example Functions

1. Prerequisites

Python 3.8+](https://www.python.org/downloads/)

[VS Code] (https://code.visualstudio.com/)

[Azure Functions Core

Tools] (https://docs.microsoft.com/azure/azure-functions/functions-run-local)

1. Prerequisites

[Azure CLI]

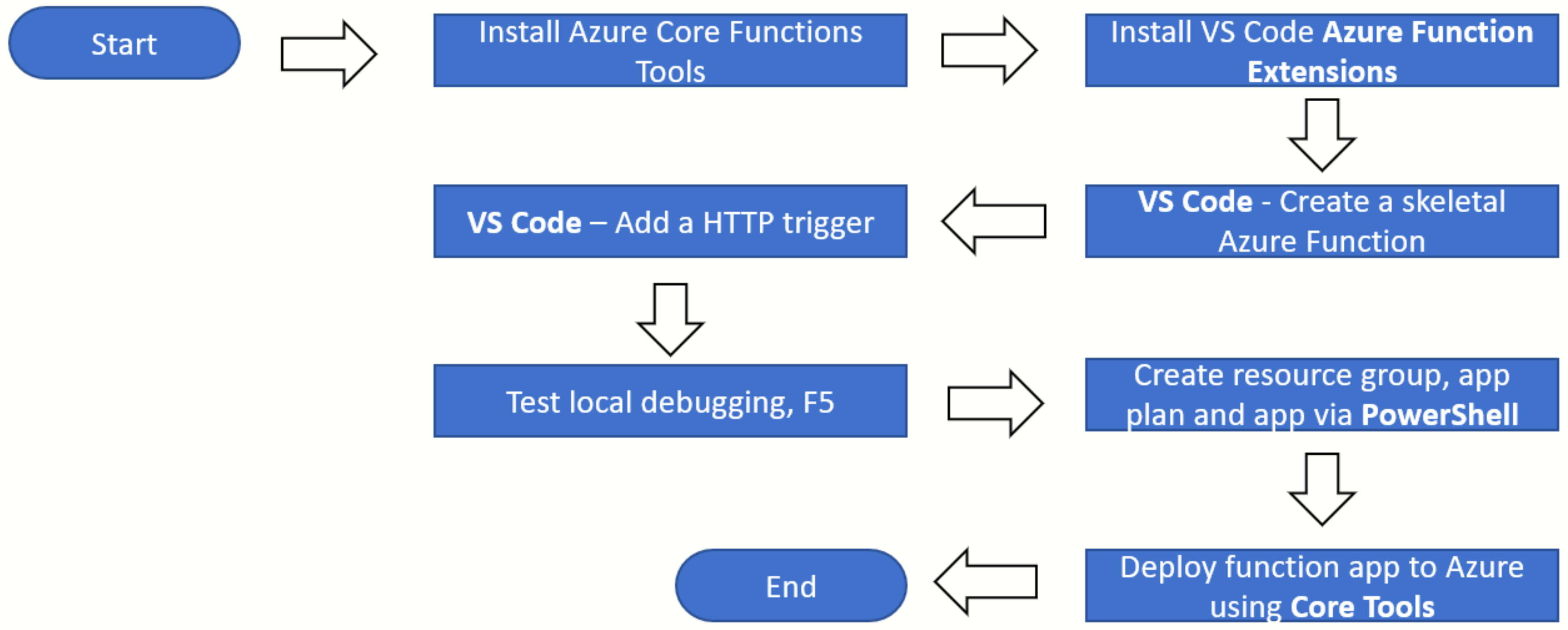
(https://docs.microsoft.com/cli/azure/install-azure-cli)

VS Code Extensions:

- Azure Functions

- Azure Account

- Python


Start
Install Azure Core Functions Tools
Install VS Code **Azure Function Extensions**
VS Code - Create a skeletal Azure Function
VS Code – Add a HTTP trigger
Test local debugging, F5
Create resource group, app plan and app via **PowerShell**
Deploy function app to Azure using **Core Tools**
End

2. Local Setup

Create a New Function Project

1. Open VS Code

2. Open Command Palette (`Ctrl+Shift+P` or `Cmd+Shift+P`)

3. Run: Azure Functions: Create New Project...

4. Choose a folder

5. Select Python as language

2. Local Setup

6. Choose a Python interpreter (3.8+)

7. Select a template (e.g., **HTTP trigger**)

8. Name your function (e.g., `HttpExample`)

9. Authorization level: **Anonymous** (for testing)

b. Explore Project Structure

__init__.py: Function code

function.json : Function configuration

requirements.txt` : Python dependencies

c. Install Dependencies

In your project folder:

```
#Bash: python -m venv .venv
source .venv/bin/activate

# On Windows: .venv\Scripts\activate

pip install -r requirements.txt
```

3. Run Function Locally

In the terminal (in your project folder):

```
# func start
```

If port 7071 is busy, use another port:

```
# func start --port 7072
```

3. Run Function Locally

Test in browser or with curl:

```
curl "http://localhost:7071/api/HttpExample?name=Azure"
```

4. Deploy to Azure

Login to Azure

```
$ az login
```

b. Deploy using VS Code

1. Click the Azure icon in the sidebar
2. Click “Deploy to Function App...”
3. Select your subscription

b. Deploy using VS Code

4. Create a new Function App

(unique name, choose Python runtime, region)

5. Wait for deployment

c. Deploy using Azure CLI

```
$ az functionapp create --resource-group <RESOURCE_GROUP> --
consumption-plan-location <LOCATION> --runtime python --
runtime-version 3.11 --functions-version 4 --name <APP_NAME> --
storage-account <STORAGE_NAME>
```

Replace `<APP_NAME>` and `<RESOURCE_GROUP>`

c. Deploy using Azure CLI

```
$func azure functionapp publish <APP_NAME>
```

5. Test Your Deployed Function

- Copy the function URL from the Azure Portal or VS Code
- Test with browser or curl:

```
$ curl "<YOUR_FUNCTION_URL>?name=Azure"
```

6. Update and Redeploy

Make code changes locally

Deploy again using

VS Code or CLI as above

7. Stopping the Function

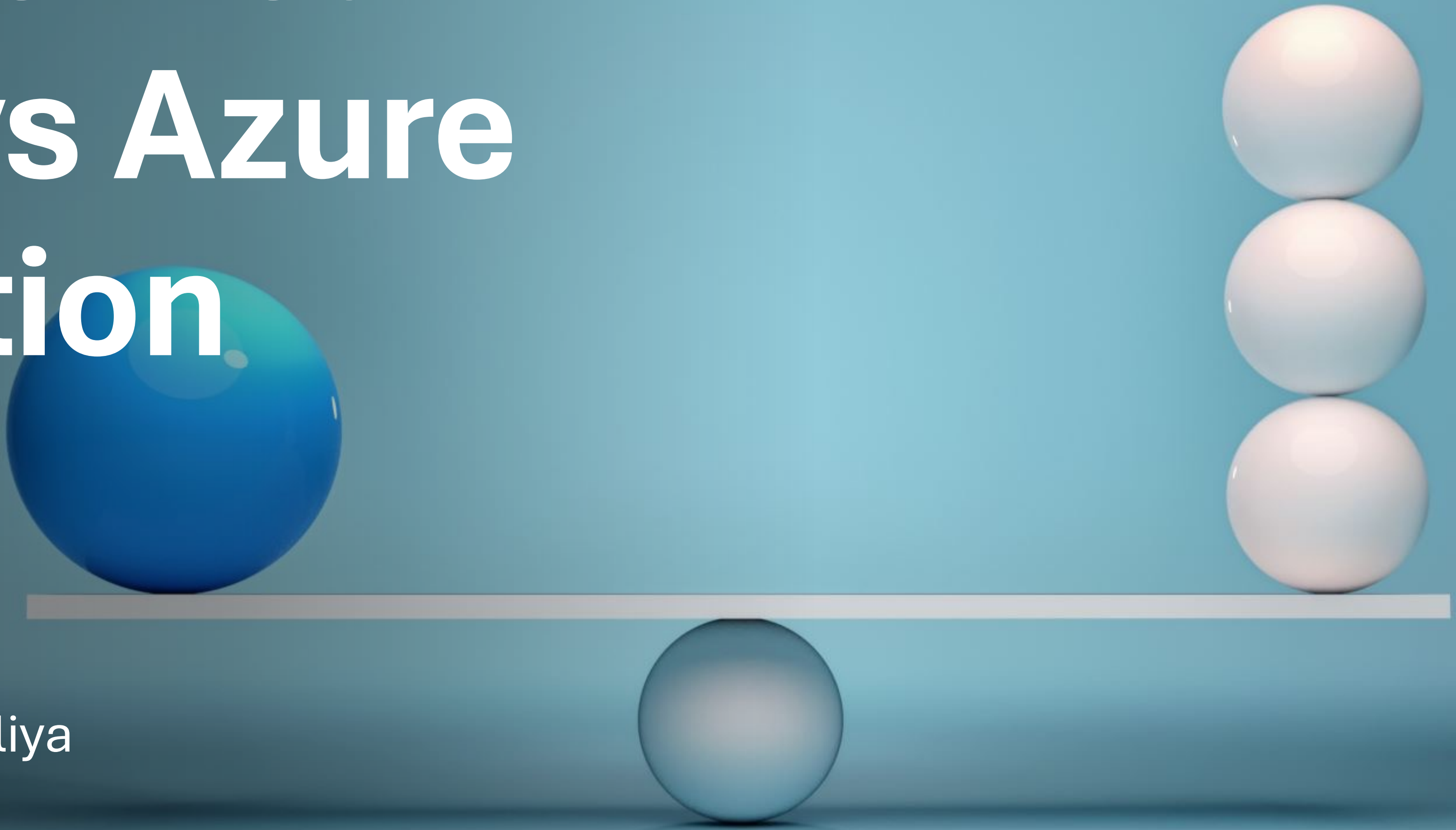
Local:

Press `Ctrl+C` in the terminal running `func start`

Azure:

In Azure Portal, click **Stop** on your Function App

Dockerized App vs Azure Function



Surendra Panpaliya

Dockerized App vs Azure Function

Step	Dockerized FastAPI App (e.g., on Azure App Service / Container Apps)	Azure Function (Serverless)
1. App Structure	Full control over FastAPI structure	Must follow Azure Function template (e.g., function.json, __init__.py) or use custom handlers

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
2. Packaging	Uses a Dockerfile to define the full environment	Azure hosts Python/Node environment — no Docker unless using custom container

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
3. Runtime	Full ASGI/WSGI compatibility, runs with uvicorn, gunicorn, etc.	Uses Azure’s runtime with function triggers (e.g., HTTP, Timer, Queue)

Dockerized App vs Azure Function

Step	Dockerized FastAPI App (	Azure Function (Serverless)
4. Hosting	App runs in a container on App Service / Azure Container Apps / AKS	Function runs in **serverless** mode, triggered only when needed

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
5. Local Testing	docker build, docker run or docker-compose	func start using Azure Functions Core Tools

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
6. Deployment	Push Docker image to Azure Container Registry or Docker Hub, then deploy to Azure	Zip deploy, GitHub Actions, or az functionapp publish (can be container-based too)

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
7. Scalability	Manual or auto-scaling at container/service level	Automatic scaling per-trigger (serverless)

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
8. Cold Start	Depends on pricing tier, but **minimal** if always-on	May experience **cold start delay** on first hit (esp. Consumption Plan)

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
9. Cost	Pay for allocated compute resources (CPU/memory even if idle)	Pay **only for actual execution time** (Consumption Plan = very cost-effective)

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
10. Control over OS/Runtime	Full control — can include OS libraries, Python version, etc.	Limited unless using custom handlers or custom containers

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
11. Best For	APIs, full-stack apps, background workers, ML models	Event-driven microservices, small APIs, cron jobs, real-time functions

Dockerized App vs Azure Function

Step	Dockerized FastAPI App	Azure Function (Serverless)
12. Vendor Lock-In	Containerized = **portable** across cloud providers	Azure Functions = tied to Azure ecosystem unless abstracted via OpenFaaS, etc.

Example Use Case Comparison

- **Dockerized App**
- You have a **FastAPI Bank Application** with:
- Swagger UI
- Database connections
- JWT Auth
- Background jobs (e.g., Celery)

Example Use Case Comparison

Use Docker

Deploy to Azure App Service for Containers,

Azure Container Apps, or AKS

Example Use Case Comparison

⚡ Azure Function:

You want to:

- Send OTP to user
- Trigger balance sync every 5 minutes
- Process transactions from a queue
- Simple GET API for balance

Example Use Case Comparison

Use Azure Functions

Use HTTP trigger,

Timer trigger, or

Queue trigger

Summary Table

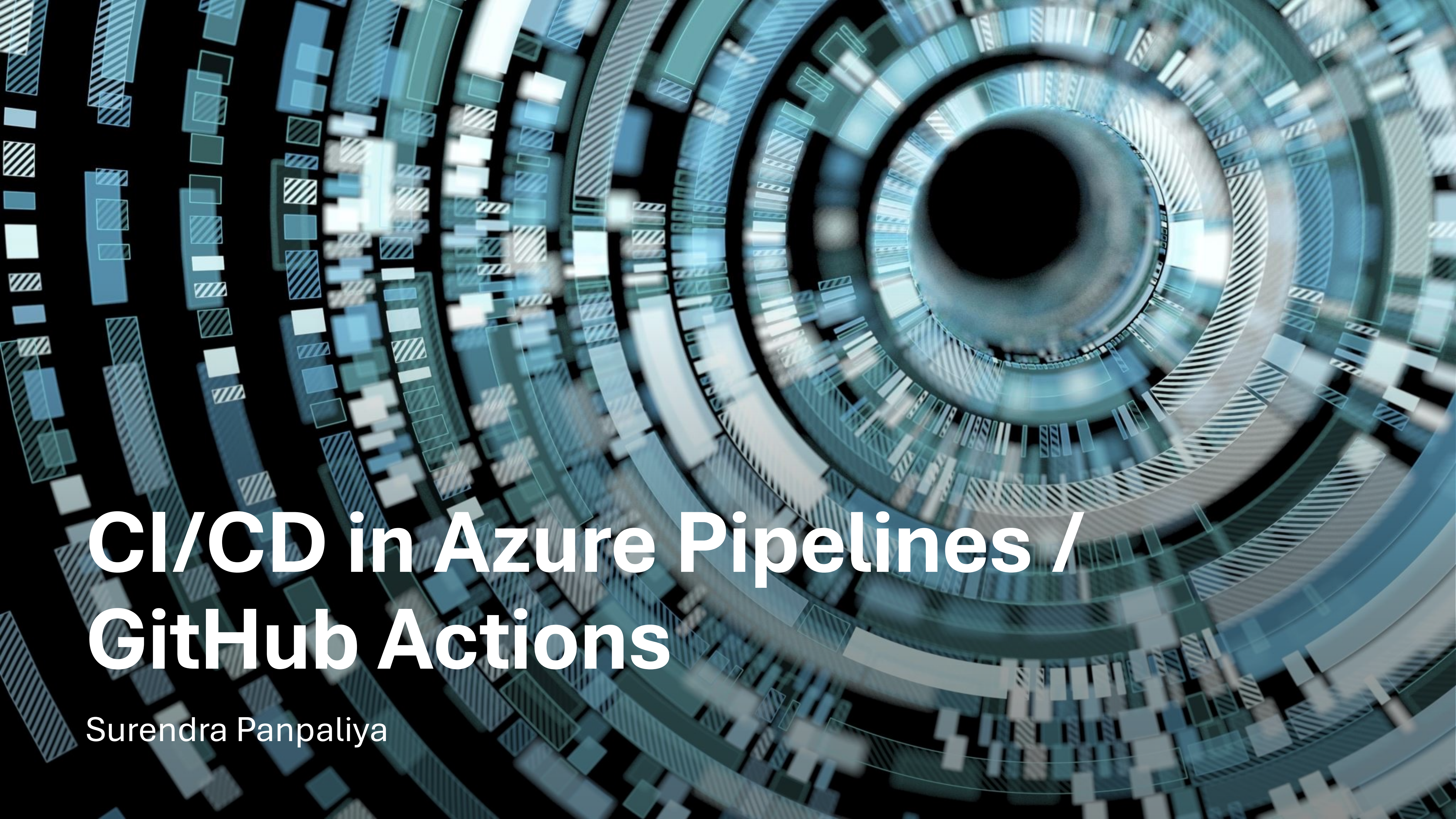
Criteria	Dockerized App	Azure Function
Full Framework Support	✅ Yes	⚠️ Partial (needs custom handler for FastAPI)
Cold Start	❌ (unless always-on)	⚠️ Yes in Consumption Plan
Billing	💸 Always-on pricing	💸 Pay-per-invocation

Summary Table

Criteria	Dockerized App	Azure Function
CI/CD	GitHub Actions, ACR, DockerHub	GitHub Actions, func deploy
Ideal For	Full APIs, backend apps, ML inferencing	Microservices, event-based logic

Conclusion

Scenario	Recommended
Full FastAPI application with DB, login, and multiple routes	Dockerized App
Simple endpoint, webhook, or cron job with fast startup	Azure Function
Want lowest cost and don’t mind cold start	Azure Function (Consumption Plan)
Need custom OS, packages, or performance	Docker + Azure App Service / Container Apps



CI/CD in Azure Pipelines / GitHub Actions

Surendra Panpaliya

What is CI/CD?

Term	Description
CI	Continuous Integration – automatically test and build code on each commit
CD	Continuous Delivery/Deployment – automatically release to staging/prod

CI/CD: Azure Pipelines vs GitHub Actions

Step	Azure Pipelines	GitHub Actions
✅ 1. Platform	Built into Azure DevOps (dev.azure.com)	Built into GitHub repositories
✅ 2. Initial Setup	- Create Azure DevOps project	

CI/CD: Azure Pipelines vs GitHub Actions

- Create Pipeline via YAML or UI
- Link code repo (GitHub, Azure Repos, Bitbucket, etc.)
- Add .github/workflows/ folder
- Add workflow YAML file
- Auto-runs on push, PR, or manual

CI/CD: Azure Pipelines vs GitHub Actions

Pipeline Location

- .azure-pipelines.yml in code repo or created via UI
- .github/workflows/<file>.yml inside repo

CI/CD: Azure Pipelines vs GitHub Actions

- **UI & Experience**
- Visual editor for pipeline steps + YAML
- YAML-only but has rich UI for job monitoring

CI/CD: Azure Pipelines vs GitHub Actions

Trigger Options

- On push, PR, branch, tag, schedule, or pipeline chaining
- On push, PR, schedule (cron), release, workflow call, etc.

CI/CD: Azure Pipelines vs GitHub Actions

Secrets & Variables

Stored in Azure DevOps Library or pipeline variables

Stored in GitHub → Settings → Secrets

CI/CD: Azure Pipelines vs GitHub Actions

Permissions & RBAC

Enterprise-grade role-based access control (RBAC)

Simpler permission system (Repo → Actions access)

CI/CD: Azure Pipelines vs GitHub Actions

Built-in Agents

Hosted agents: Ubuntu, Windows, macOS

Self-hosted agents possible

GitHub-hosted runners: Ubuntu, Windows, macOS

Self-hosted runners supported

CI/CD: Azure Pipelines vs GitHub Actions

Azure Integration

Deeply integrated

(Key Vault, App Services, AKS, Functions, ARM, etc.)

Good integration using azure/login, az CLI, or

Azure/functions-action

CI/CD: Azure Pipelines vs GitHub Actions

Deployment Targets

Azure, on-prem, containers, Kubernetes, other clouds

Azure, AWS, GCP, any cloud or container platform

CI/CD: Azure Pipelines vs GitHub Actions

Pricing (Public Repos)

Free with Azure DevOps

- Free tier: 1,800 mins/month (MS-hosted agents)

Free: 2,000 mins/month

- Unlimited for public repos

CI/CD: Azure Pipelines vs GitHub Actions

Approval Gates

Built-in environment gates, manual approvals

Manual approval via workflow run with environments

CI/CD: Azure Pipelines vs GitHub Actions

Reusability

Templates, pipeline libraries

Composite actions, reusable workflows

CI/CD: Azure Pipelines vs GitHub Actions

Marketplace

Azure DevOps extensions & task templates

GitHub Marketplace with thousands of prebuilt actions

CI/CD: Azure Pipelines vs GitHub Actions

Best Use Cases

Large enterprise projects,

Azure-native CI/CD, fine-grained access control

Open-source projects,

cloud-native microservices,

lightweight automation

CI/CD for a Python App to Azure Function

Stage	Azure Pipelines	GitHub Actions
Trigger	trigger: [main]	on: push: branches: [main]
Install deps	pip install -r requirements.txt	pip install -r requirements.txt
Test	pytest	pytest
Deploy	AzureFunctionApp@1 task	azure/functions-action@v1

Sample YAML Azure Pipelines

```yaml
trigger:
  - main

pool:
  vmImage: 'ubuntu-latest'

steps:
  - task: UsePythonVersion@0
    inputs:
      versionSpec: '3.10'
```

Sample YAML Azure Pipelines

```
- script: pip install -r requirements.txt
  displayName: 'Install dependencies'

 - script: pytest
  displayName: 'Run tests'
```

Sample YAML Azure Pipelines

```yaml
- task: AzureFunctionApp@1
  inputs:
    azureSubscription: 'MyAzureServiceConnection'
    appType: 'functionAppLinux'
    appName: 'my-fastapi-func'
    package: '$(System.DefaultWorkingDirectory)'
```

GitHub Actions

```
name: Deploy FastAPI App

on:
 push:
  branches: [main]

jobs:
 build-and-deploy:
  runs-on: ubuntu-latest
```

GitHub Actions

```
steps:
  - name: Checkout
    uses: actions/checkout@v3

  - name: Set up Python
    uses: actions/setup-python@v4
    with:
      python-version: '3.10'
```

GitHub Actions

```
steps:
  - name: Install dependencies
      run: pip install -r requirements.txt

  - name: Run tests
    run: pytest
```

GitHub Actions

```
steps:
    - name: Login to Azure
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - name: Deploy to Azure Function
        uses: Azure/functions-action@v1
        with:
          app-name: 'my-fastapi-func'
          package: '.'
```

Summary

For You	Go With
You use GitHub for code, want simplicity, quick cloud deployments	✅ GitHub Actions
You use Azure DevOps, need approvals, large team management, pipelines chaining	✅ Azure Pipelines

Summary

For You	Go With
You’re a startup / solo developer	✅ GitHub Actions (easier, lighter, free)
You’re an enterprise with compliance / audit requirements	✅ Azure Pipelines (RBAC, approval gates, agent control)

C# Azure vs Python Azure Functions

Feature	C# Azure Functions	Python Azure Functions
Language Type	Statically typed	Dynamically typed
Compilation Model	Compiled (DLLs)	Interpreted
Startup Time	🔥 **Faster cold start** (especially in Premium Plan)	Slower cold start in Consumption Plan

C# Azure vs Python Azure Functions

Feature	**C# Azure Functions**	**Python Azure Functions**
Tooling Integration	Excellent with **Visual Studio / Azure DevOps**	Great with **VS Code, CLI, GitHub Actions**
Performance	✅ High performance	🟡 Medium performance (best for lightweight)

C# Azure vs Python Azure Functions

Feature	C# Azure Functions	Python Azure Functions
App Size	Larger (due to DLLs & dependencies)	Lighter and easier to debug locally
Use Case Fit	Enterprise-grade APIs, B2B, .NET-heavy apps	ML models, Data science, scripting, ETL

Summary

Area	C# Azure Function	Python Azure Function
Startup Time	✅ Faster	⚠️ Slower (cold starts)
Performance	✅ Better throughput	✅ Good for small loads
Flexibility	⚠️ More boilerplate	✅ Rapid development
Tooling	✅ Visual Studio IDE	✅ VS Code & CLI
Scripting/ML	❌ Harder	✅ Python native
Large APIs	✅ Preferred	⚠️ Limited scalability

Happy Learning !! Thanks for Your Patience ☺

Surendra Panpaliya

GKTCS Innovations