

Azure Cloud Exercises

Exercise 1: Create and Configure a Virtual Machine

Objective: Create and configure Ubuntu and Windows Virtual Machines on Azure Portal.

1. Create an Ubuntu VM:

- Log in to the Azure Portal.
- Navigate to Virtual Machines > Create.
- Choose Ubuntu Server 20.04 LTS.
- **Configure:**
 - **Size:** Standard_B1s (or similar)
 - **Authentication Type:** SSH (generate a key pair if not available).
 - **Inbound Port:** Allow SSH (port 22).
- Deploy and connect using SSH.

2. Create a Windows VM:

- Follow similar steps, selecting Windows Server 2022.
- **Configure:**
 - **Size:** Standard_B1s (or similar)
 - **Authentication Type:** Username and Password.
 - **Inbound Port:** Allow RDP (port 3389).
- Deploy and connect using RDP.

3. Task:

- Install Apache or IIS on the respective VMs.
- Verify by accessing the default web page from your local browser.



Ubuntu

Apache2 Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

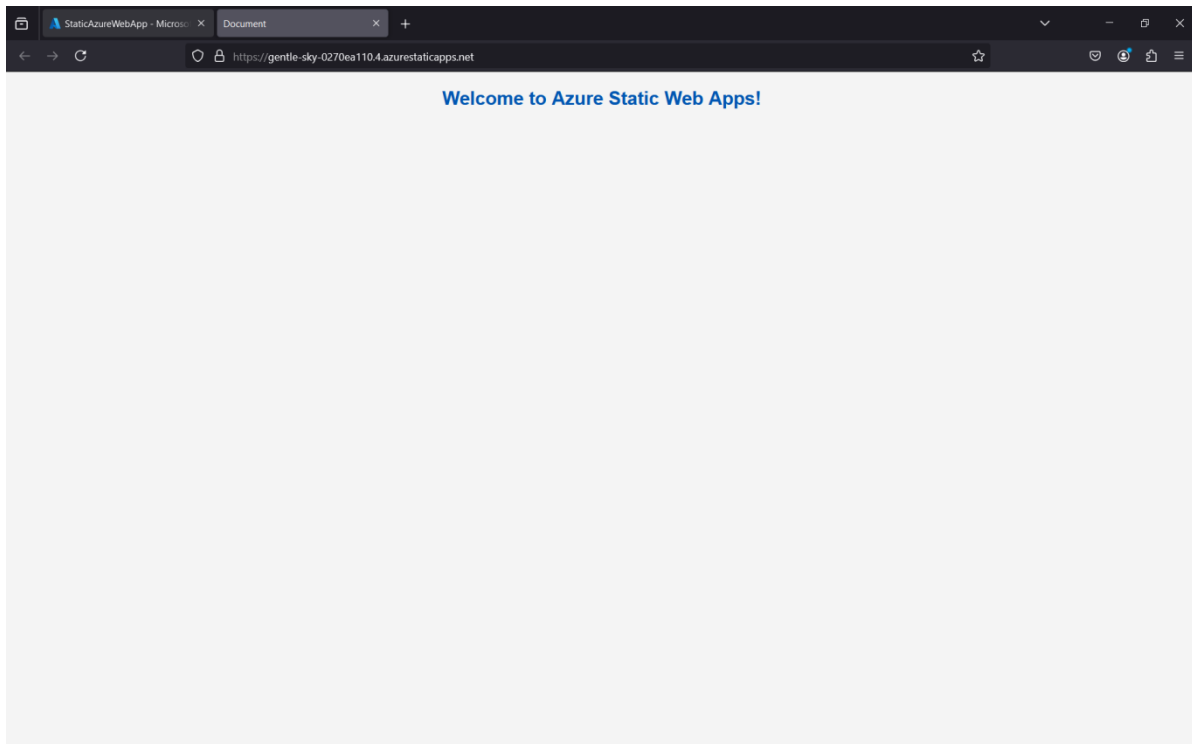
```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
```

It works!

Exercise 2: Deploy a Static Web Application

Objective: Host a static website using Azure App Service.

1. Navigate to App Services > Create.
2. **Choose:**
 - **Runtime Stack:** Python 3.10 (or latest).
 - **Operating System:** Linux.
 - **Region:** Closest to your location.
3. Deploy the application.
4. Upload a simple static website (e.g., index.html and CSS files) using FTP or the Kudu console.
5. **Task:**
 - Verify the deployment by accessing the site via its public URL.
 - **Modify the HTML to include a message like: "Welcome to Azure Static Web Apps!"**



<https://gentle-sky-0270ea110.4.azurestaticapps.net/>

Exercise 3: Deploy a Flask Application (Dynamic Web App)

Objective: Deploy a Python Flask application using Azure App Service.

1. Create a Flask app:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def home():

    return "Hello, Azure Flask App!"

if __name__ == '__main__':

    app.run(debug=True)
```

2. Push the code to a GitHub repository.

3. In the Azure Portal, navigate to App Services > Create.

4. Configure:

- **Runtime Stack:** Python 3.10 (or latest).
- **Deployment Source:** Connect your GitHub repository.

5. Deploy the Flask app and verify it by accessing the public URL.

The screenshot shows a GitHub Actions workflow run for the repository 'harishezhi / FlaskPythonApp'. The workflow is titled 'Build and deploy Python app to Azure Web App - DynamicWebApp'. The current job is 'Add or update the Azure App Service build and deployment workflow config #1', which has a status of 'Success' and a total duration of '9m 14s'. The workflow consists of two jobs: 'build' (15s) and 'deploy' (8m 38s). The 'deploy' job is linked to the 'build' job. The 'deploy' job is configured with the deployment source 'https://dynamicwebapp-cbamcgd4gc3a2...'.

github.com/harishezhi/FlaskPythonApp/actions/runs/12890121790

harishezhi / FlaskPythonApp

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Build and deploy Python app to Azure Web App - DynamicWebApp

✓ Add or update the Azure App Service build and deployment workflow config #1

Summary

Jobs

- ✓ build
- ✓ deploy

Run details

- Usage
- Workflow file

Triggered via push 14 minutes ago

harishezhi pushed → 3f80c0f master

Status: Success

Total duration: 9m 14s

Artifacts: 1

master_dynamicwebapp.yml

on: push

build 15s

deploy 8m 38s

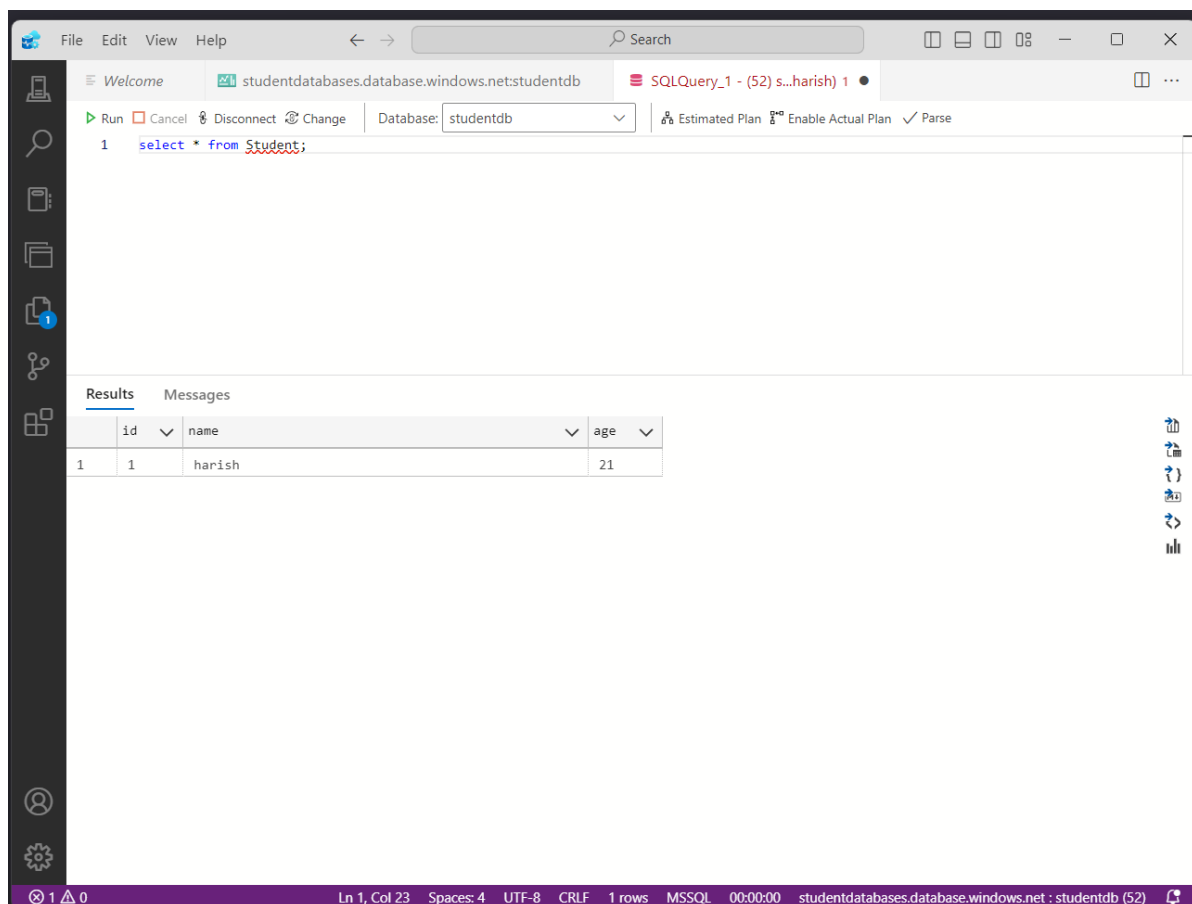
https://dynamicwebapp-cbamcgd4gc3a2...



Exercise 4: Set Up and Use an Azure SQL Database

Objective: Create an Azure SQL Database and connect to it from your local machine.

1. Navigate to SQL Databases > Create.
2. **Configure:**
 - Database Name: StudentDB.
 - **Server:** Create a new server with username and password.
 - Compute + Storage: Use the free tier.
3. Deploy the database.
4. Connect using Azure Data Studio or SQL Server Management Studio (SSMS).
5. **Task:**
 - Create a table Students with columns ID, Name, and Age.
 - Insert sample data and query it.



Exercise 5: Integrate Flask App with Azure SQL Database

Objective: Connect a Flask app to Azure SQL Database and perform CRUD operations.

1. Use the Flask app from Exercise 3.

2. Install required libraries:

```
pip install flask pyodbc
```

3. Modify the app to connect to the SQL Database:

```
import pyodbc

conn = pyodbc.connect(
    'DRIVER={ODBC Driver 17 for SQL Server};'
    'SERVER=<your_server>.database.windows.net;'
    'DATABASE=StudentDB;'
    'UID=<your_username>;'
    'PWD=<your_password>'
)

cursor = conn.cursor()
```

4. Add a route to fetch and display data from the Students table.

5. Deploy the updated app to Azure App Service.

6. Task:

- Verify CRUD functionality by interacting with the app via its public URL.

