Universiti Kuala Lumpur
Where Knowledge is Applied

# ASSESSMENT COVERSHEET

***(STUDENTS: Fill in all sections)***

**Attach this coversheet as the cover for your submission. All sections need to be completed.**

***For online submission, attached this document as pdf or in MS Words.***

## Section A: Submission Details

| | | |
|---|---|---|
| **Programme** | : | BACHELOR IN INFORMATION TECHNOLOGY (HONOURS) (INTERNET OF THINGS) |
| **Course Code & Name** | : | IIB43203 - CLOUD COMPUTING |
| **Course Lecturer(s)** | : | MEGAT NORULAZMI MEGAT MOHAMED NOOR |
| **Type of Submission** | : | **ASSIGNMENT** |
| **Penalties** | : | • 5% will be deducted per day to a maximum of four (4) working days, after which the submission will **not** be accepted.<br>• Plagiarised work is an Academic Offence in University Rules & Regulations and will be penalised accordingly. |

## Section B: Academic Integrity

Tick (√) each box below if you agree:

| | |
|---|---|
| √ | I/You have read and understood the UniKLs' policy on Plagiarism in University Rules & Regulations. |
| √ | This assignment is own work, unless indicated with proper referencing. |
| √ | This assignment not submitted and not published previously. |
| √ | This submission follows the requirements stated in the course. |

## Section C: Submission Receipt

### Office Receipt of Submission

| Date & Time of Submission (by student) | Student Name(s) (by student) | Student ID(s) (by student) |
|---|---|---|
| 2nd JUNE 2025 | EVAN HAMO<br>MUHAMMAD UWAIS BIN MOHD IMRAN<br>MUHAMMAD IZWAN BIN HUSSIN<br>MUHAMMAD HAZMAN BIN MOHD SOFEE | 52200225007<br>52224123579<br><br>52224123217<br>52224123618 |

## Student Receipt of Submission

This is your submission receipt, the only accepted evidence that you have submitted your work. Cut along the dotted lines above & retain this for **your record**.

| Date & Time of Submission (Written) | Course Code | Submission Title | Student ID(s) & Signature(s) |
|---|---|---|---|
| | | | |

# Table of Contents

# Introduction

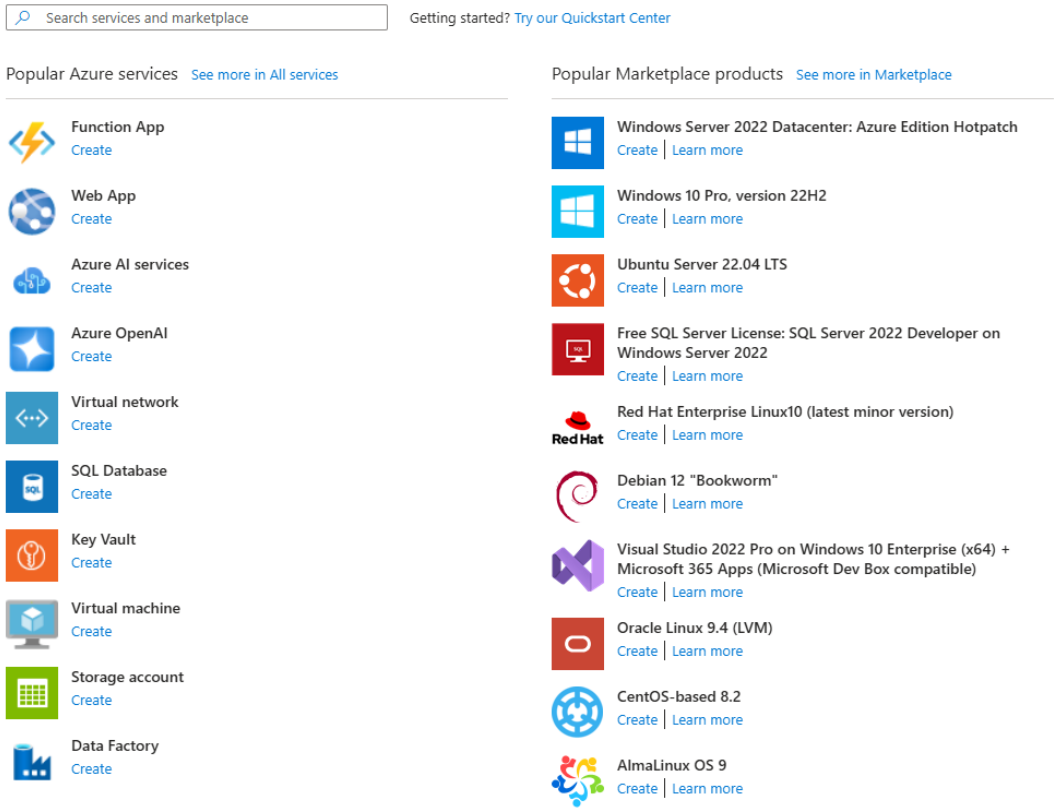
Cloud computing is a modern technology that allows users to access computing resources such as servers, storage, databases, networking, software, and more over the internet. Instead of owning and maintaining physical hardware, users can rent these resources from cloud service providers based on their needs. This approach offers greater flexibility, scalability, and cost-efficiency, making it popular among businesses, developers, and students for hosting applications, storing data, and running services.

One of the most widely used cloud platforms is Microsoft Azure, which provides a wide range of cloud services to build, deploy, and manage applications through a global network of data centers. The Azure Portal is a web-based interface that allows users to manage their cloud resources easily. Through the portal, users can create virtual machines, databases, web applications, and more with just a few clicks. It also offers monitoring tools, security features, and pricing plan options, making it a powerful and user-friendly platform for cloud computing tasks.

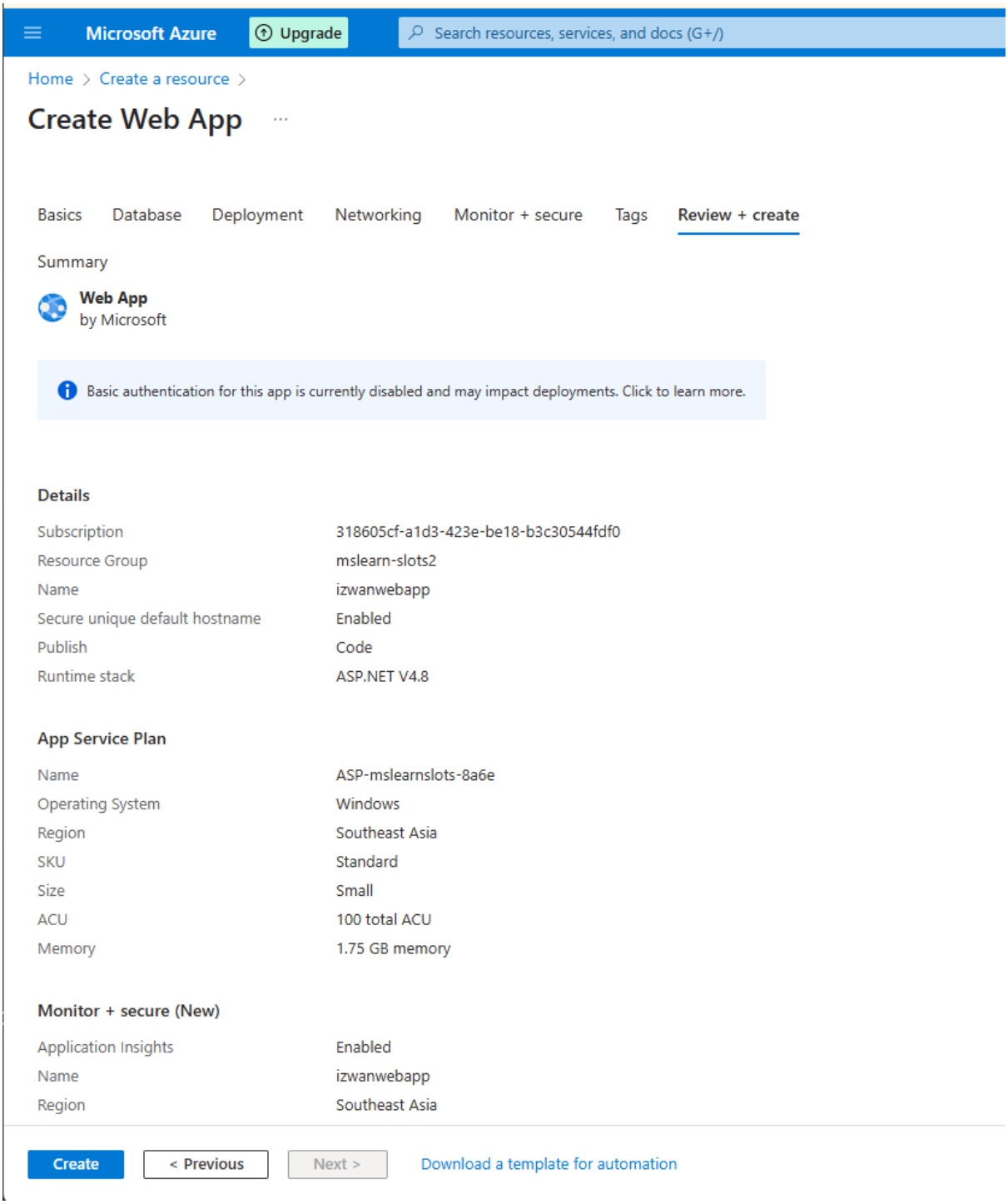
# Create deployment slots

**Azure Lab #30A**
**Name: Izwan Hussin**

➢ **Create a web app**

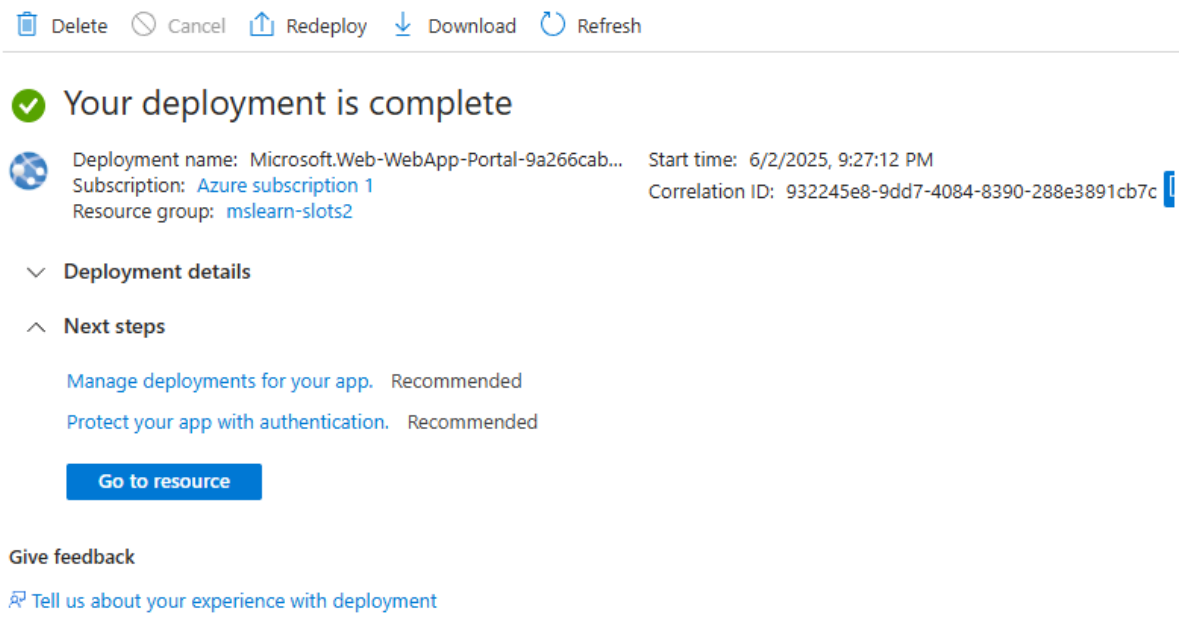



To begin, I signed in to the Azure Portal by visiting https://portal.azure.com and logging in with my Azure account. Once logged in, I started creating the web app by clicking on “Create a resource” from the Home or the left-hand menu. Then, I selected the “Web” category, searched for “Web App,” and clicked on it. After that, I proceeded by selecting the “Create” button to begin configuring the web app.

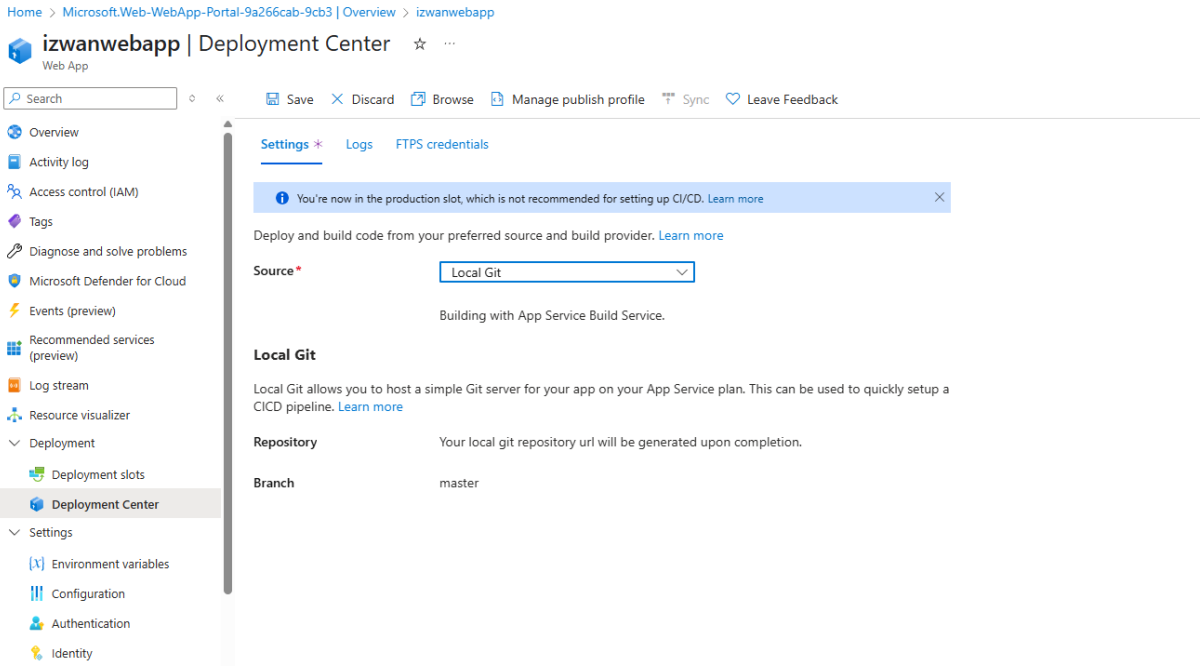



In the Basics tab of the Web App creation process, I began by selecting my Azure subscription and creating a new resource group named mslearn-slots. For the name of the web app, I entered a unique identifier, such as izwanwebapp, to ensure it wouldn't conflict with existing names. I chose "Code" for the publish method, selected "ASP.NET V4.8" as the runtime stack, and set the operating system to Windows. For the region, I picked the one closest to my location to reduce latency. I accepted the default App Service Plan (Windows Plan) and kept the default SKU and size settings.

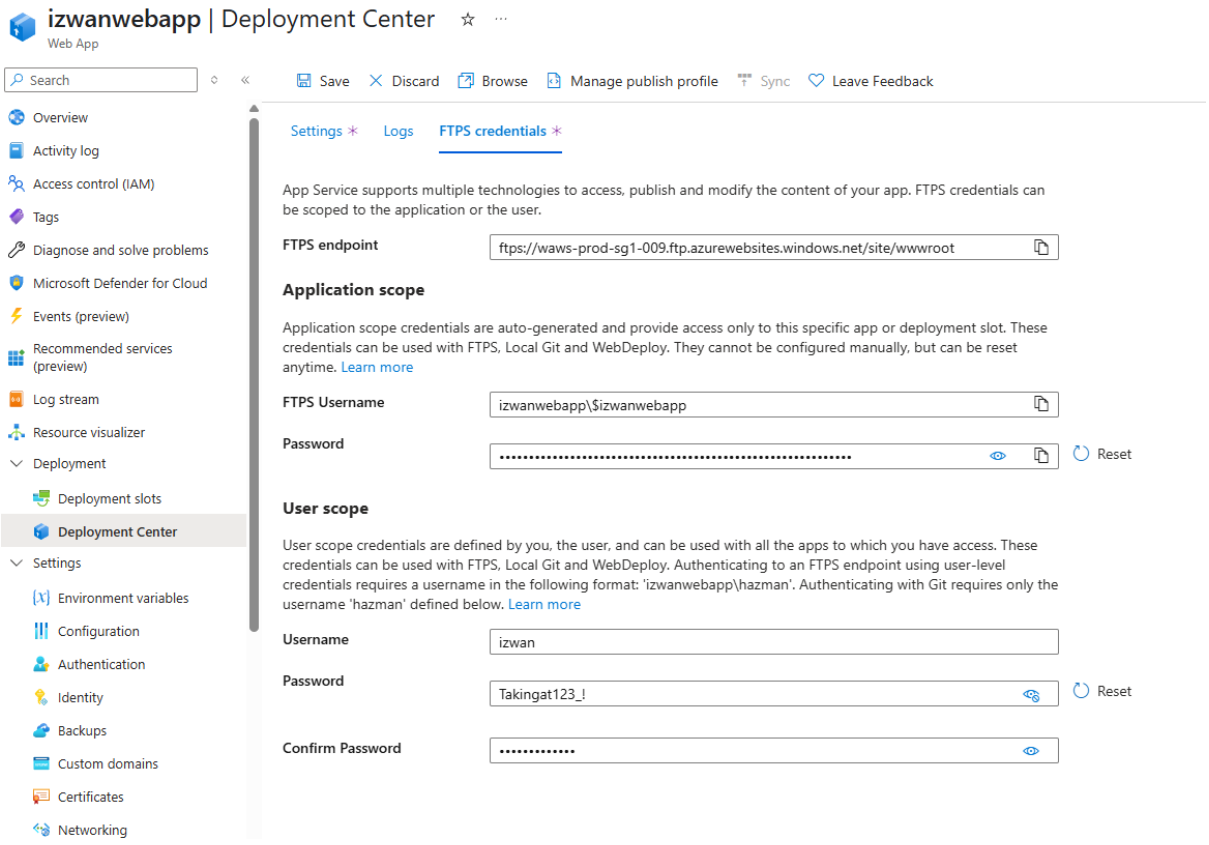
After completing the Basics tab, I moved to the Deployment tab by clicking "Next: Deployment (Preview)," where I made no changes. I then advanced to the Monitoring tab and turned off Application Insights by toggling it to "No." Finally, I proceeded to the Review + Create tab, where I reviewed all the configurations. After validation was successful, I clicked "Create" to deploy the web app.




The deployment is complete

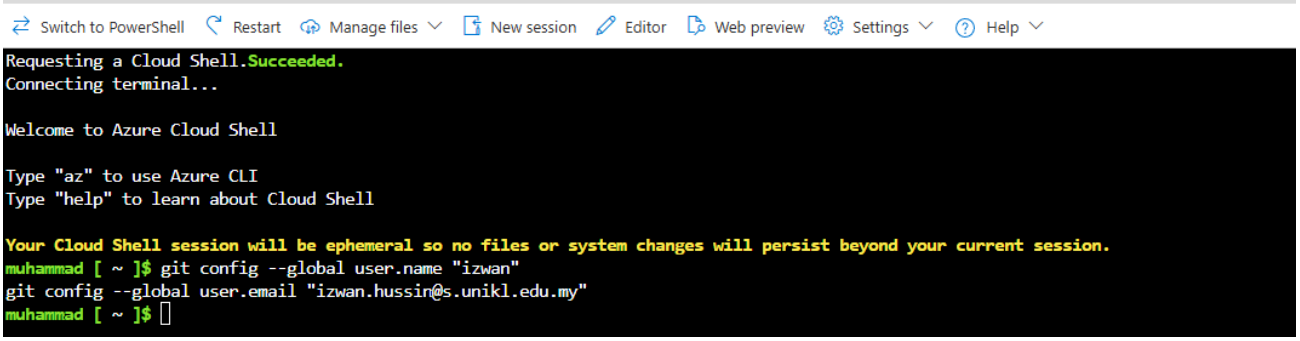


To configure the deployment method for the web app, I opened the Deployment Center by selecting it from the left-hand menu under the Deployment section. Once inside the Deployment Center, I navigated to the Settings tab where I selected "Local Git" as the deployment source. After choosing Local Git, I clicked the Save button at the top of the page to apply the changes.




Next, I set up the Git/FTP credentials required for deploying code. I navigated to the "Local Git/FTPS credentials" tab, and under the "User scope" section, I created a deployment username and set a secure password. After filling in these credentials, I clicked the Save button to store the settings and enable authentication for future deployments.

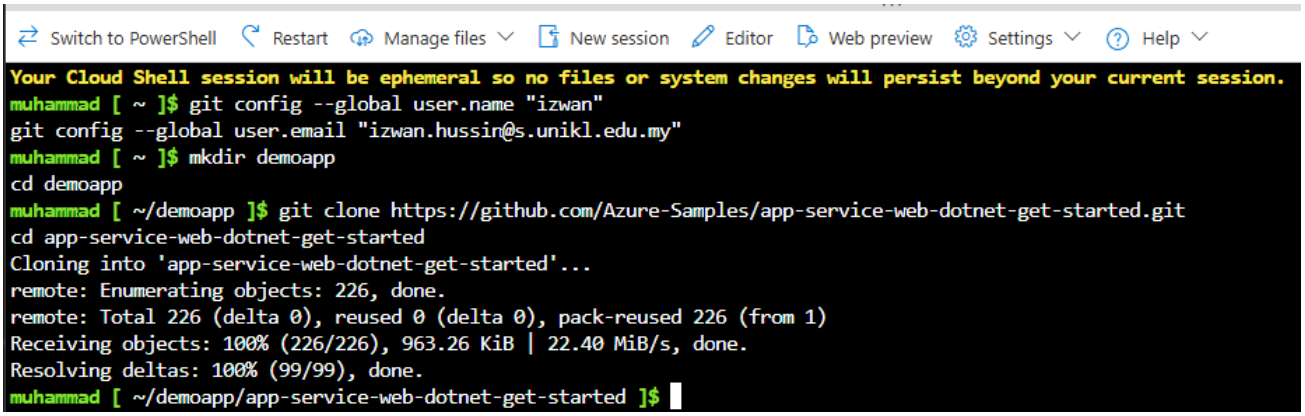
## ➢ Configure git deployment




Then , I opened the Azure Cloud Shell by clicking the terminal icon ( >_ ) located in the top navigation bar of the Azure portal. When prompted to choose a shell, I selected Bash. I configured Git to use my personal information, I put this command ,

- git config --global user.name "your-username"
- git config --global user.email "your-email@example.com"

I then replaced "your-username" and "your-email@example.com" with my actual Git username and email address. After updating, I copied the commands into the Cloud Shell and ran them by pressing Enter. This ensured that my Git commits would be properly attributed.




To continue setting up the environment, I first created a new project folder by running the following commands in Azure Cloud Shell:

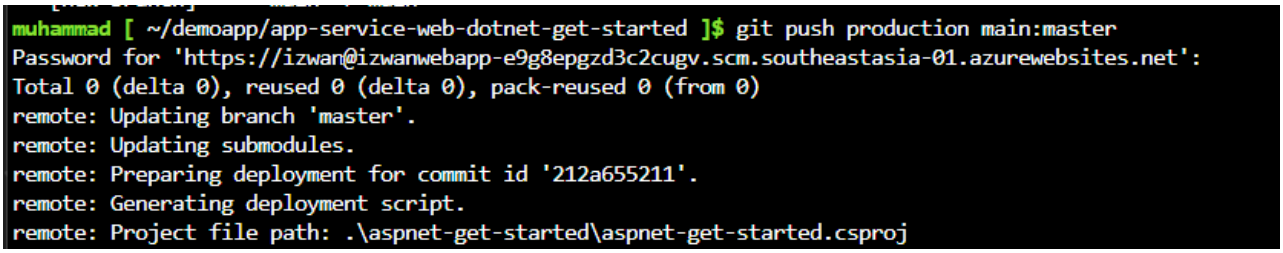
- mkdir demoapp
- cd demoapp

This created a directory named demoapp and moved me into it, providing a dedicated workspace for the web application.
Next, I cloned a sample ASP.NET web application by executing:

- git clone https://github.com/Azure-Samples/app-service-web-dotnet-get-started.git cd app-service-web-dotnet-get-started

These commands downloaded the sample project from GitHub into my demoapp directory and navigated into the cloned project folder. This gave me access to the source code needed for deployment.

- **Configure a git remote to deploy the app to production**




To deploy the web app to the production slot, I first ensured I was in the correct project directory within Azure Cloud Shell by running:

- cd ~/demoapp/app-service-web-dotnet-get-started

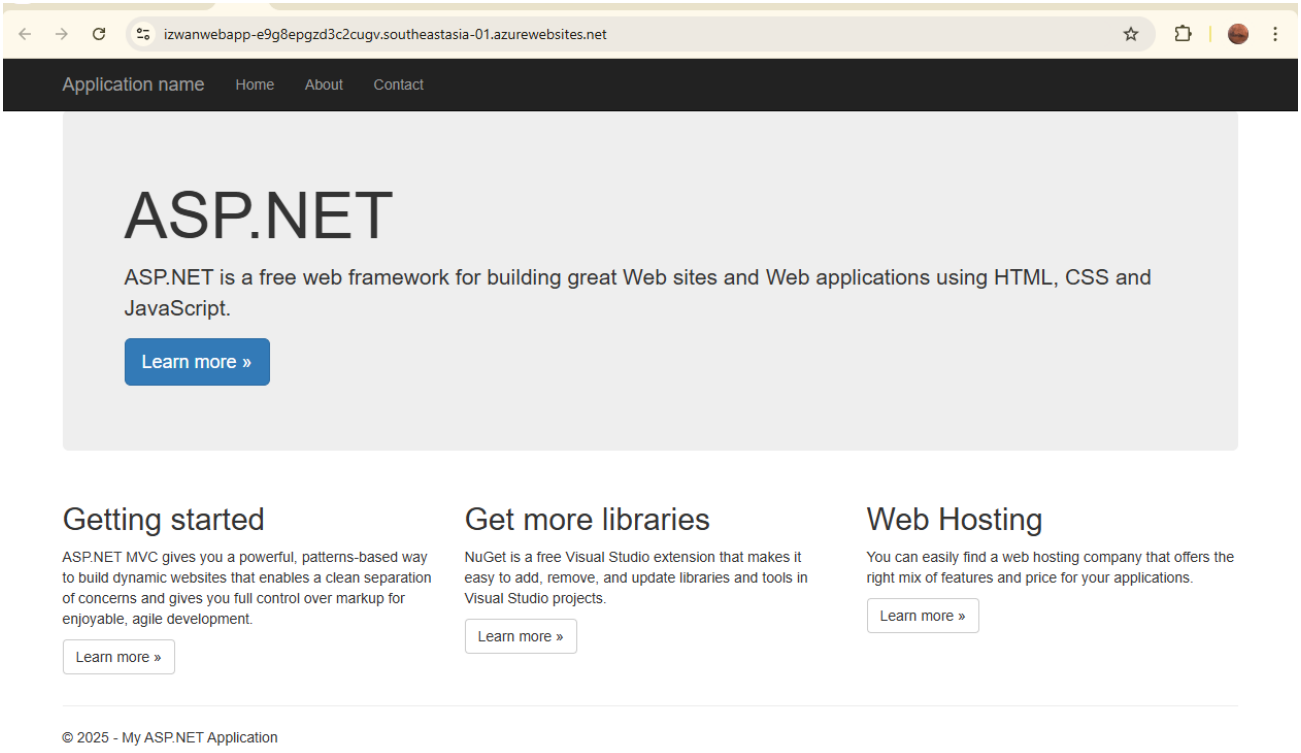
Then, I added the Git remote for the production deployment by using the Git URL from the Azure Deployment Center:

- git remote add production https://<your-deployment-username>@<your-app-name>.scm.azurewebsites.net:443/<your-app-name>.git

(Here, I replaced <your-deployment-username> and <your-app-name> with my actual deployment credentials and web app name.)
Once the remote was added, I pushed the code to the production slot using:

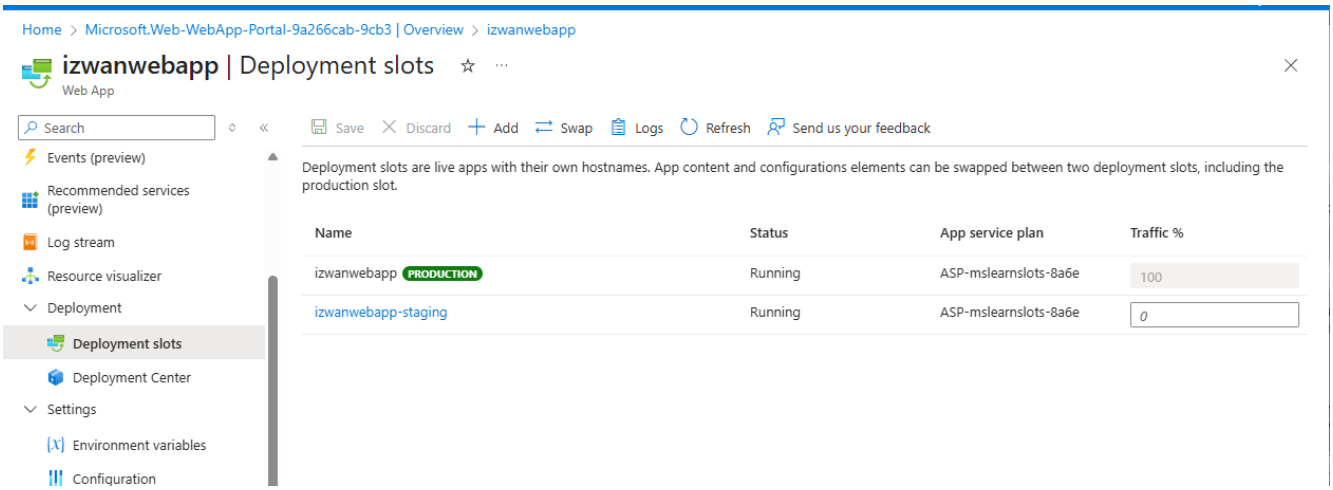
- git push production main:master

This command tells Git: "Push the local main branch to the remote master branch on the production remote." During the push, I was prompted to enter the deployment username and password I had set earlier in the Deployment Center. After the deployment completed successfully, the web application became live on the production slot.



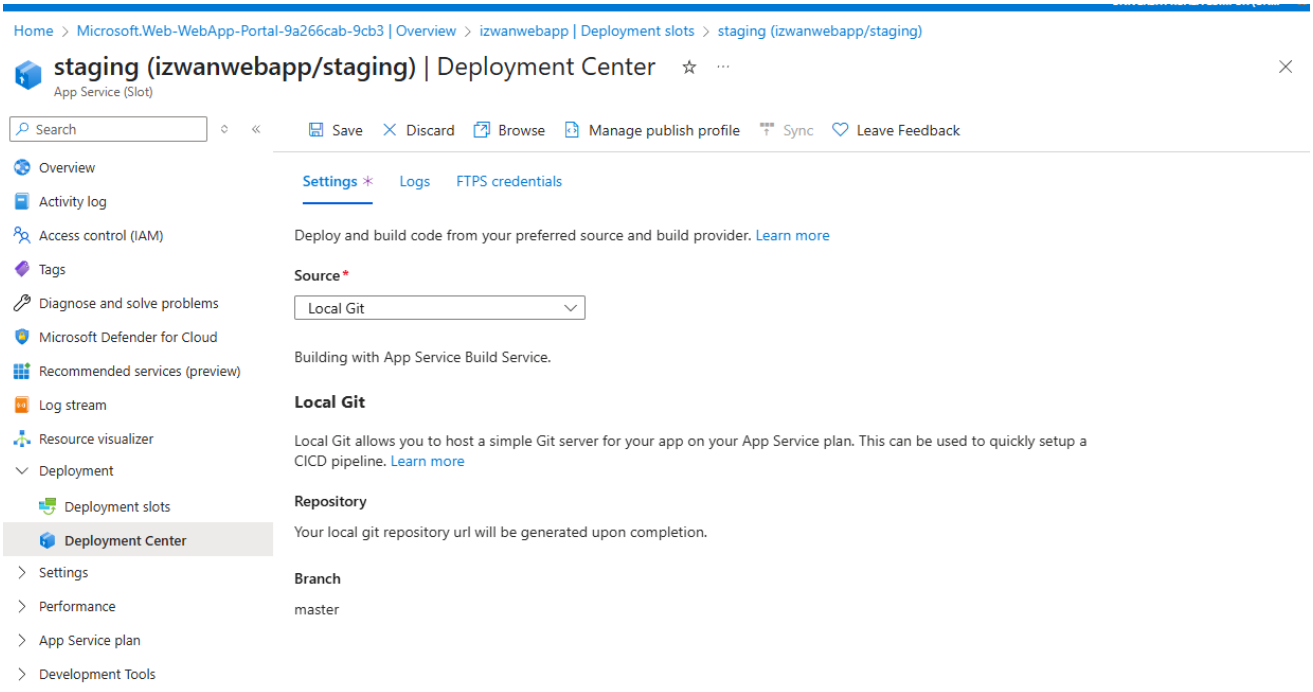

In the Azure Portal, I go back to the Overview tab, click the URL, to open the deployed web app in my browser.

## ➢ Create a new staging slot

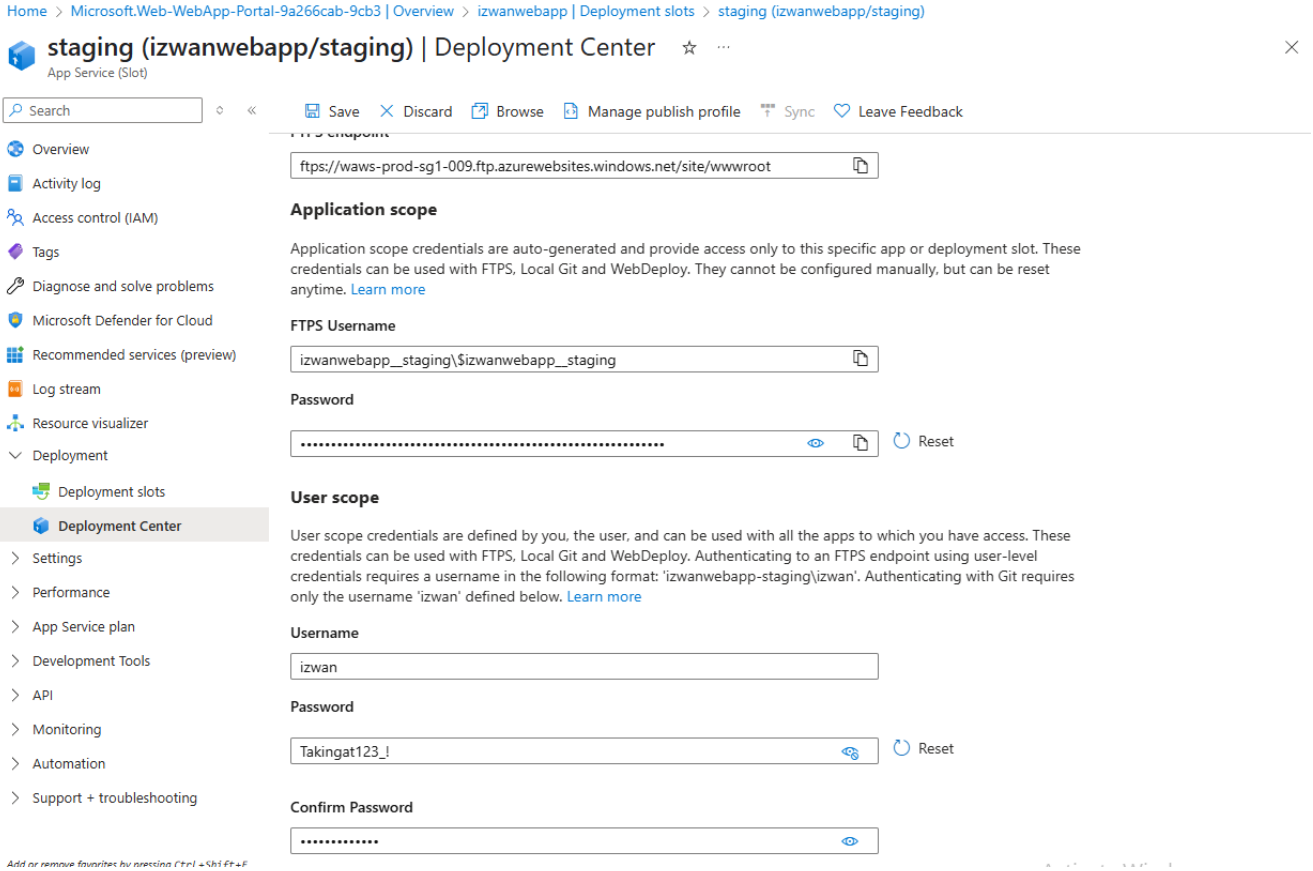



Next i create staging slot ,To create a new staging slot, start by navigating to the Azure Portal. From the Home or menu panel, select "All resources" and filter the list by setting the Type to "App Service". Choose my existing Web App from the list. In the left-hand menu under the “Deployment” section, click on “Deployment slots”. Once on the Deployment Slots page, click the "+ Add Slot" button. In the dialog that appears, enter “Staging” as the name of the slot. I Keep the default selection to clone settings from my production app to ensure configuration consistency. Finally, I click the “Add” button to create the staging slot.

## ➢ Set up git deployment for the staging slot

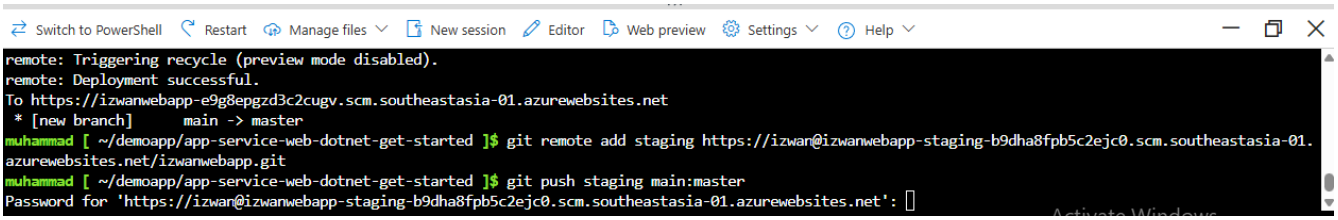


To configure Git deployment for my staging slot, first I open the Azure Portal and navigate to your Staging Web App. In the left-hand menu, under the “Deployment” section, click on “Deployment Center.” In the Deployment Center interface, go to the Settings tab. Set the deployment Source to “Local Git” to allow pushing code directly from my local environment. Once selected, click the “Save” button in the top menu bar to apply the configuration.




Next for the staging, I set up the Git/FTP credentials required for deploying code. I navigated to the "Local Git/FTPS credentials" tab, and under the "User scope" section, I created a deployment username and set a secure password. After filling in these credentials, I clicked the Save button to store the settings and enable authentication for future deployments.

## ➢ Set up git to deploy the app to the staging slot




For the next step ,I added the Git remote for my staging slot. First, I went to the Azure Portal and opened the staging slot of my web app. From the Overview page, I copied the Git clone URL found under the Essentials section. After that, I returned to Azure Cloud Shell and made sure I was inside the correct directory by running:

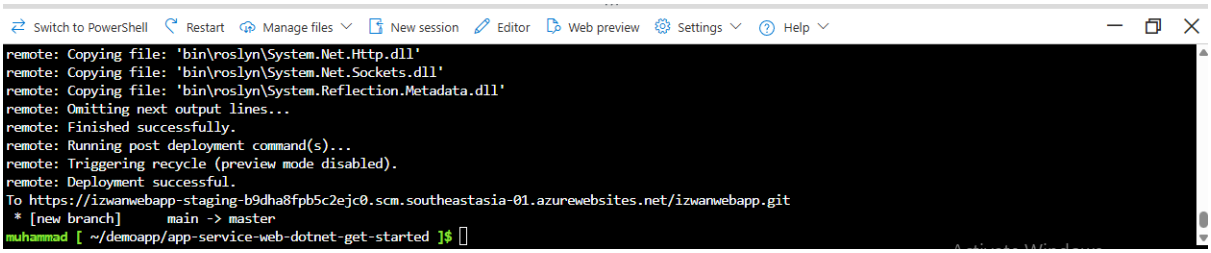
- cd ~/demoapp/app-service-web-dotnet-get-started

Once I was in the right place, I used this command:

- git remote add staging <git-clone-url>

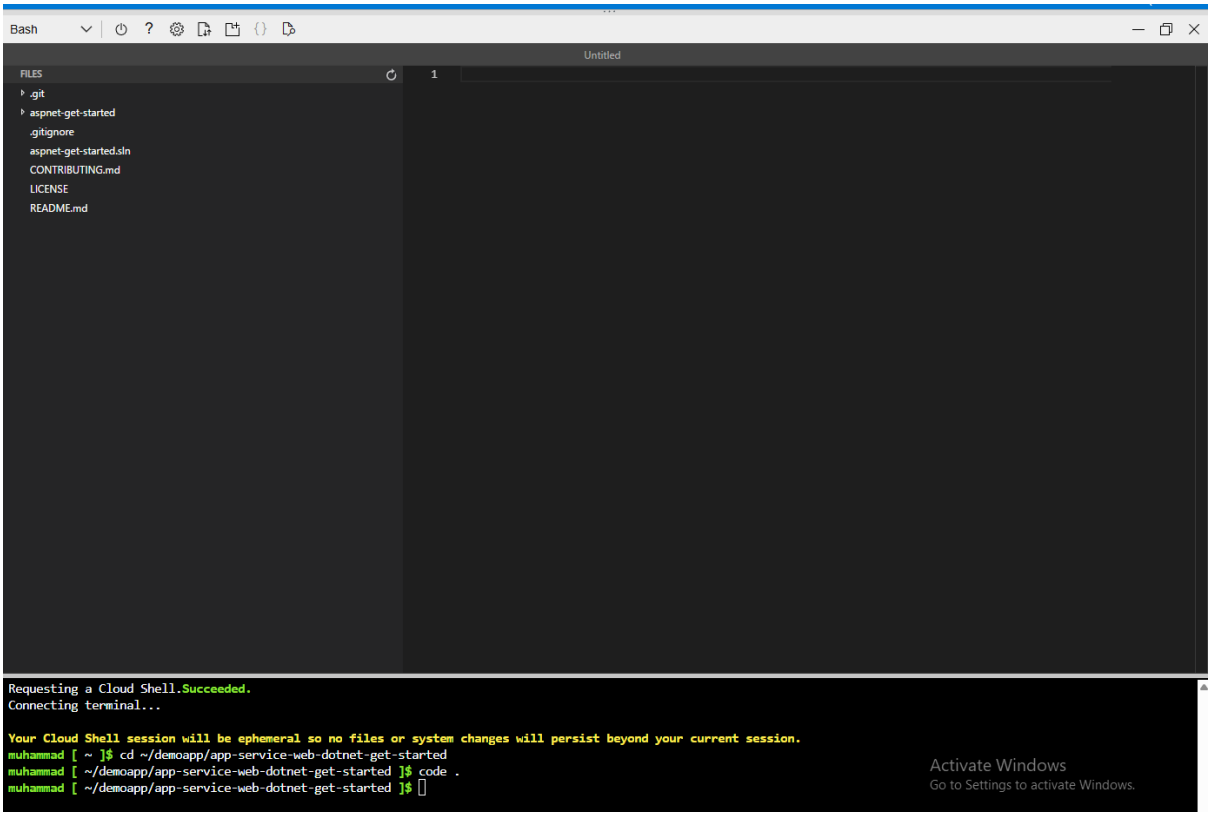
replacing the placeholder with the actual URL I copied. This step allowed me to connect my local project to the staging, after that I push the staging with this command:

- git push staging main:master

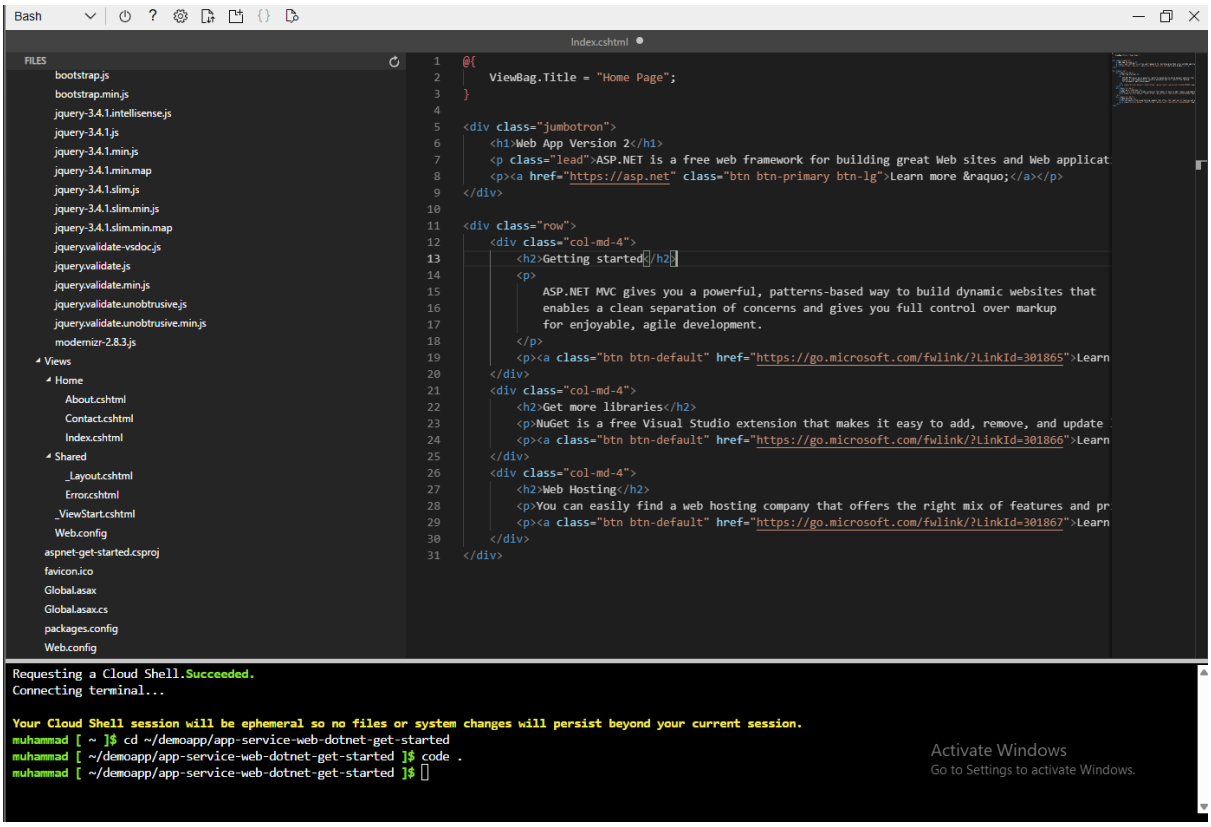



Successfully deploy

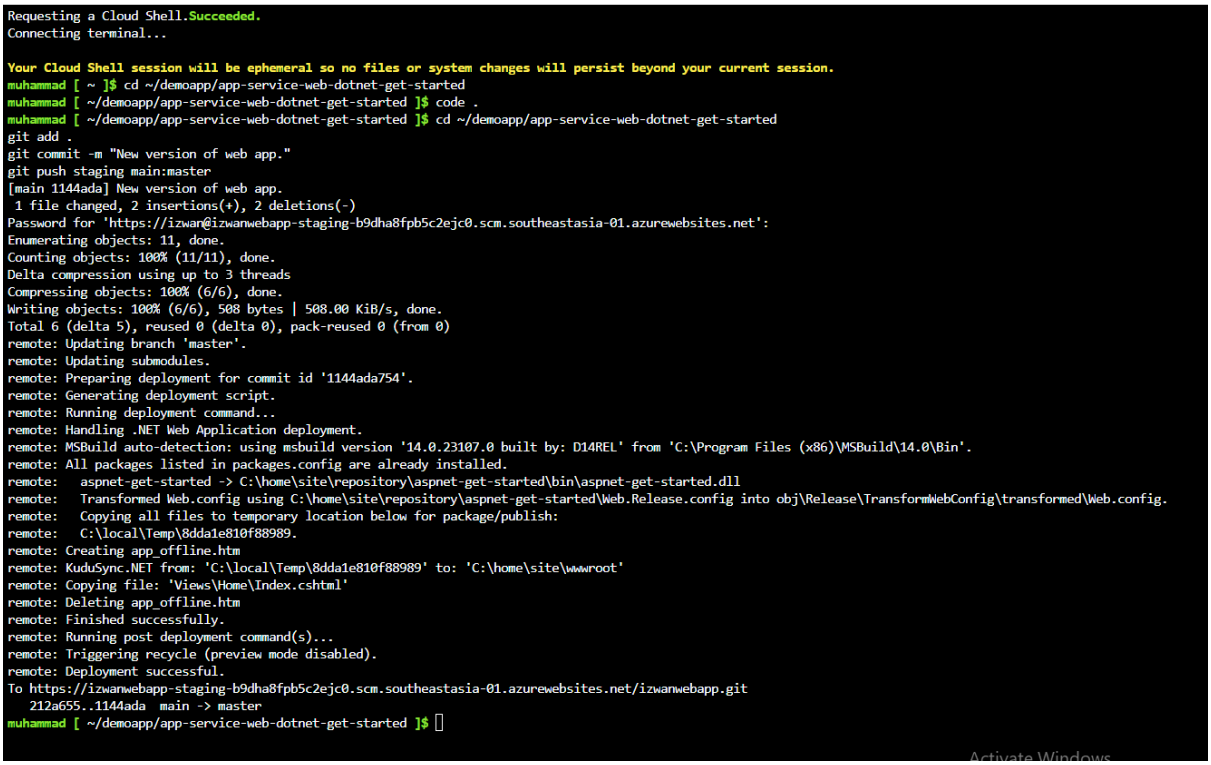
## ➢ Modify the app source code and deploy the app to the staging slot




So the next process, I opened the Cloud Shell Editor by running code . in Azure Cloud Shell. I made sure I was using the Bash environment so the editor would open correctly. Once the editor loaded, I navigated through the file tree




I navigated through the file tree by going into demoapp > app-service-web-dotnet-get-started > aspnet-get-started > Views > Home. There, I located and opened the Index.cshtml file. Inside the file, I looked for the line that had <h1>ASP.NET</h1> and I replaced it with <h1>Web App Version 2</h1>. After making that change, I saved the file using Ctrl+S and then exited the editor by pressing Ctrl+Q.




to commit the changes I made to the Index.cshtml file and push them to the staging slot. First, I made sure I was still inside the demoapp/app-service-web-dotnet-get-started directory in Cloud Shell. Then, I ran git add . to stage the changes, followed by :

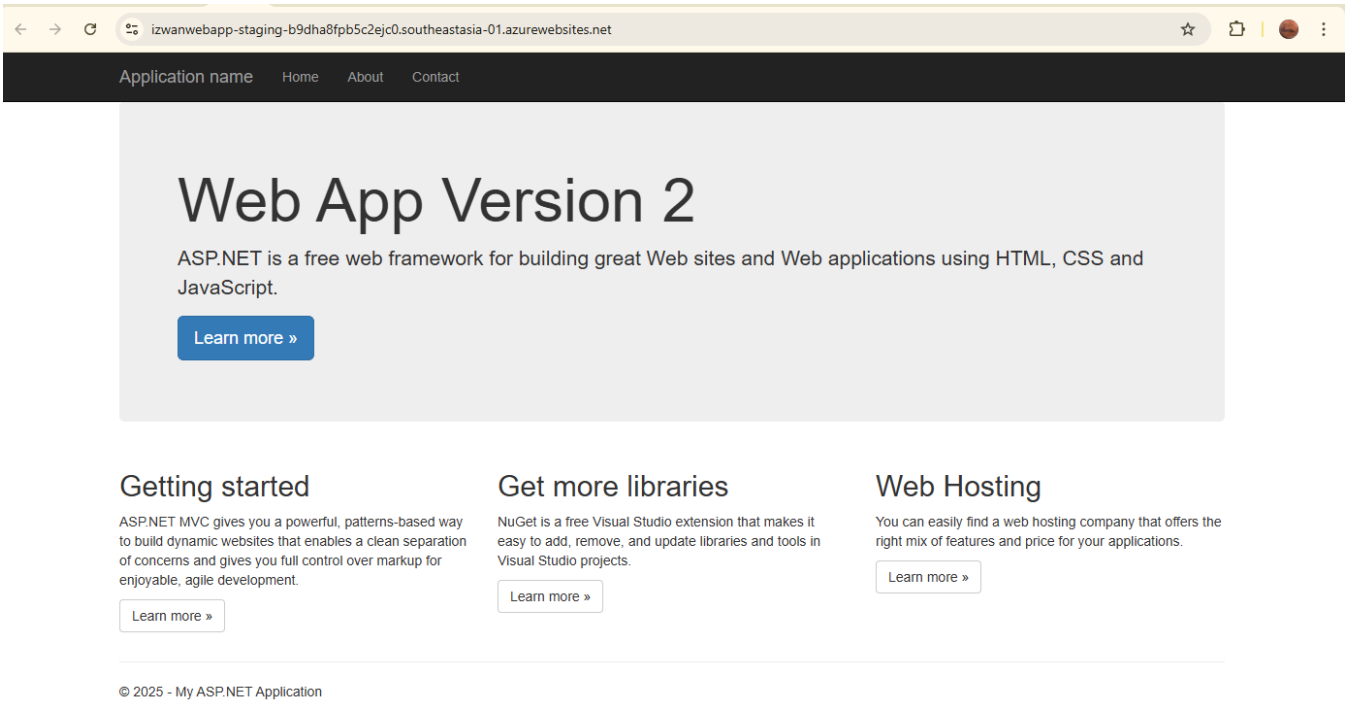
- git commit -m "Updated to Web App Version 2"

to create a commit with a message describing the update. After that, I deployed the updated version to the staging slot by running

- git push staging main:master

When prompted, I entered the deployment username and password that I had previously set in the Deployment Center. Once the push was complete, the updated web app showing “Web App Version 2” became visible on the staging URL.

- **Browse the staging slot**



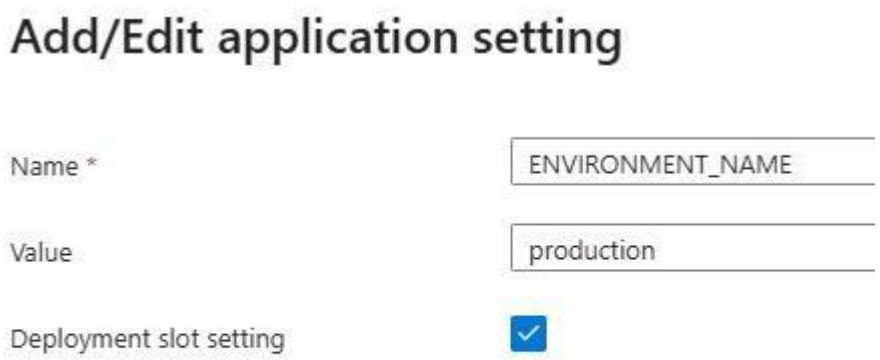

Successfully create the web app version 2 , At this point, the staging slot has the new version of the code

# Deploy a web app by using deployment slots

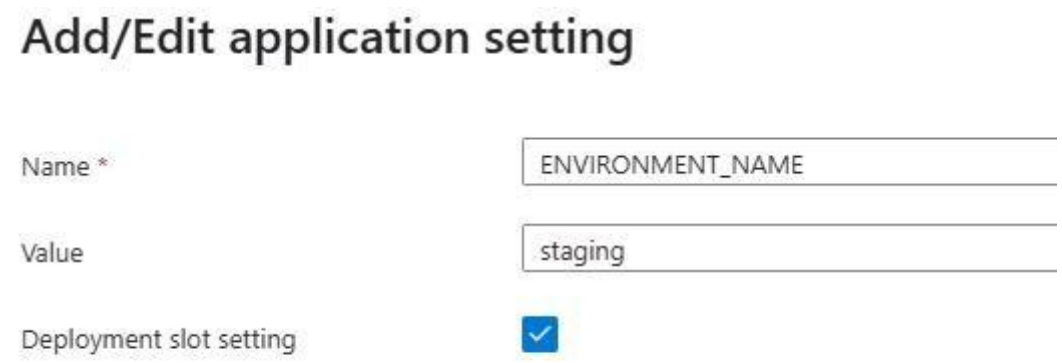
**Azure Lab #30B**

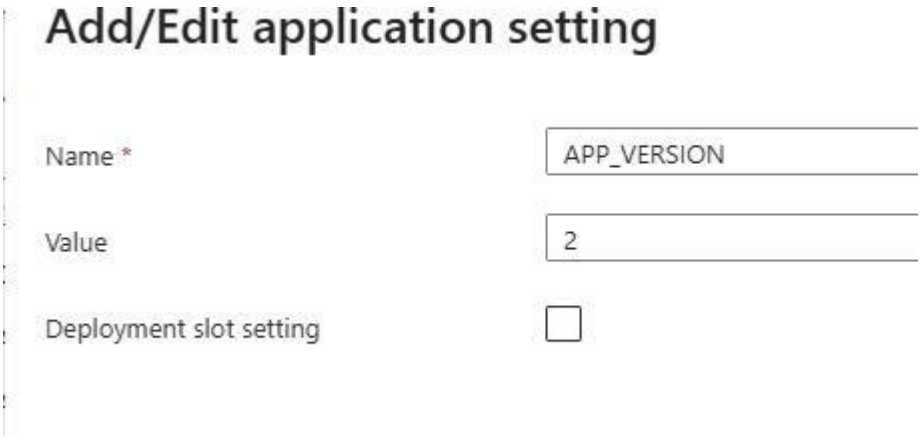
**Name: Muhammad Hazman bin Mohd Sofee**

To begin the process, I configured slot settings for both the production and staging slots of my web app. In the Azure portal, I navigated to the production slot's Configuration page and added a new application setting named ENVIRONMENT_NAME with the value production, marking it as a deployment slot setting. I also added another setting called APP_VERSION with the value 1, but did not mark this one as a slot setting. After entering these, I saved the settings.

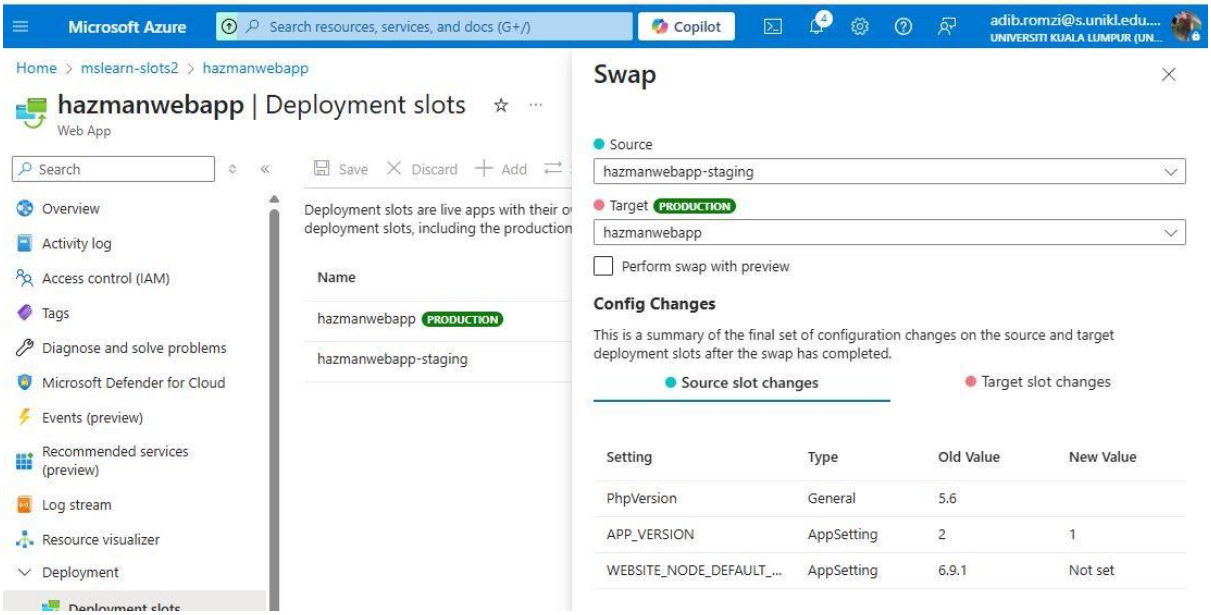



I repeated the configuration steps for the Staging slot. I added ENVIRONMENT_NAME with the value staging, marking it as a slot setting, and APP_VERSION with the value 2, leaving the deployment slot setting option unchecked. This ensured that ENVIRONMENT_NAME would not be swapped between slots, while APP_VERSION would.

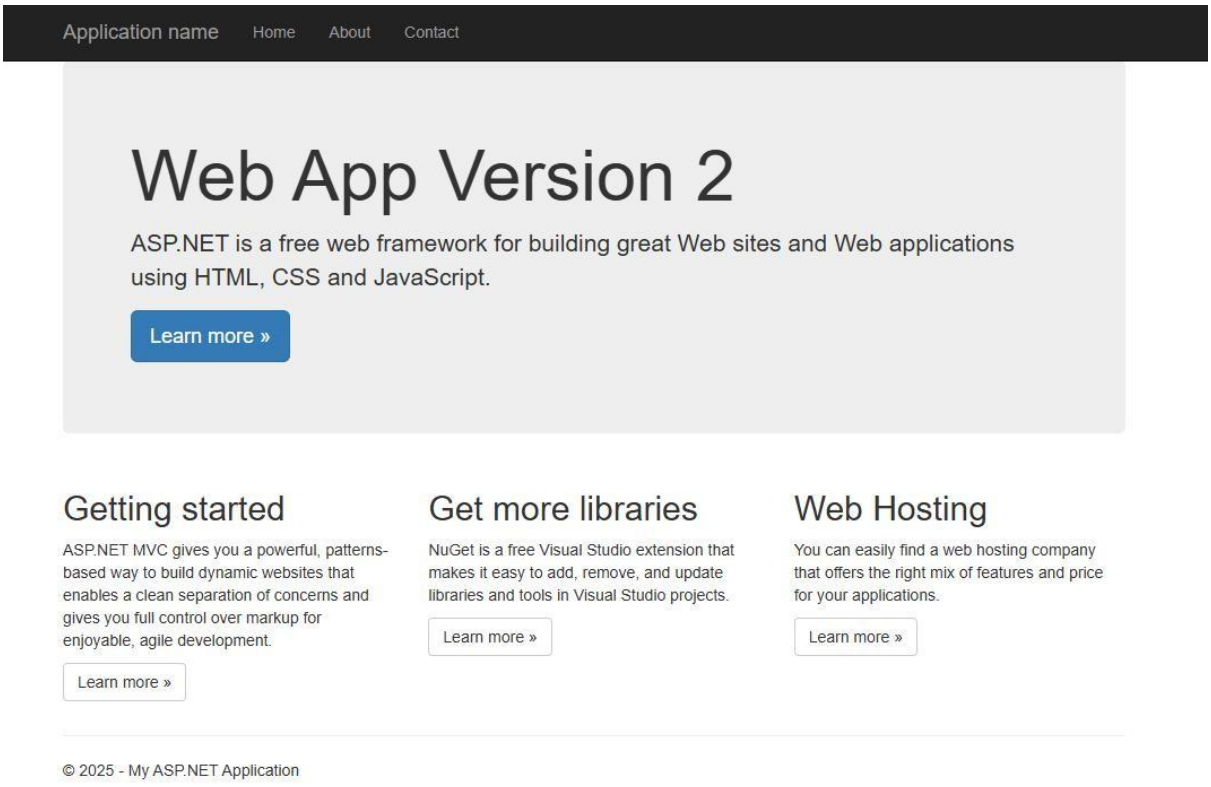


After confirming the settings, I proceeded to swap the Staging and Production slots. In the Azure portal, under the production web app's Deployment section, I chose Deployment slots Swap, verified the staging and production slots were selected correctly, and reviewed the setting behavior. I confirmed that APP_VERSION would be swapped, while ENVIRONMENT_NAME would stay in place, and then I selected Swap to execute it.

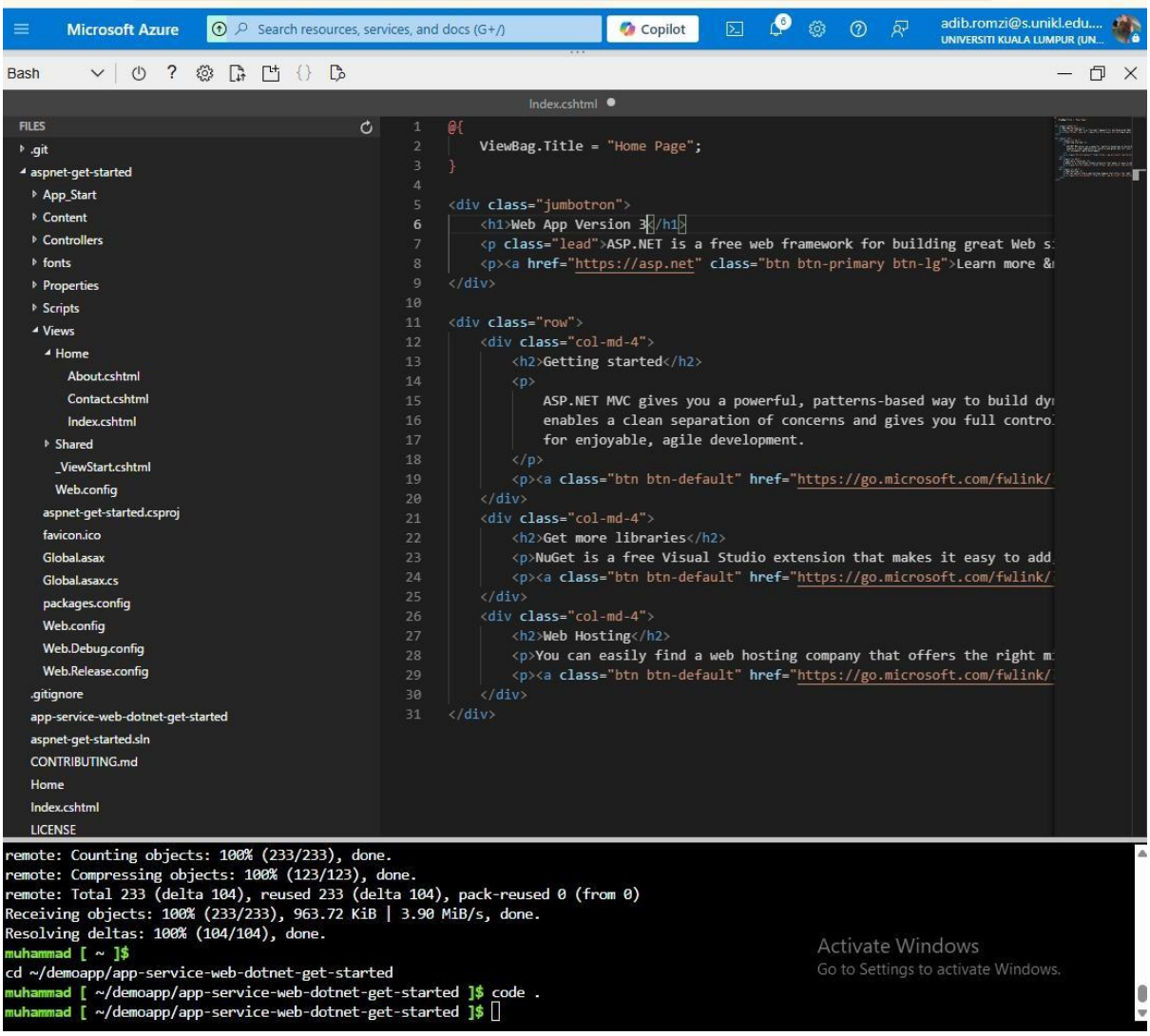


Once the swap completed, I navigated to the Overview page of the production slot and clicked Browse. The app opened in a new browser tab, now showing Version 2 of the web app as active in production. This confirmed a successful deployment.

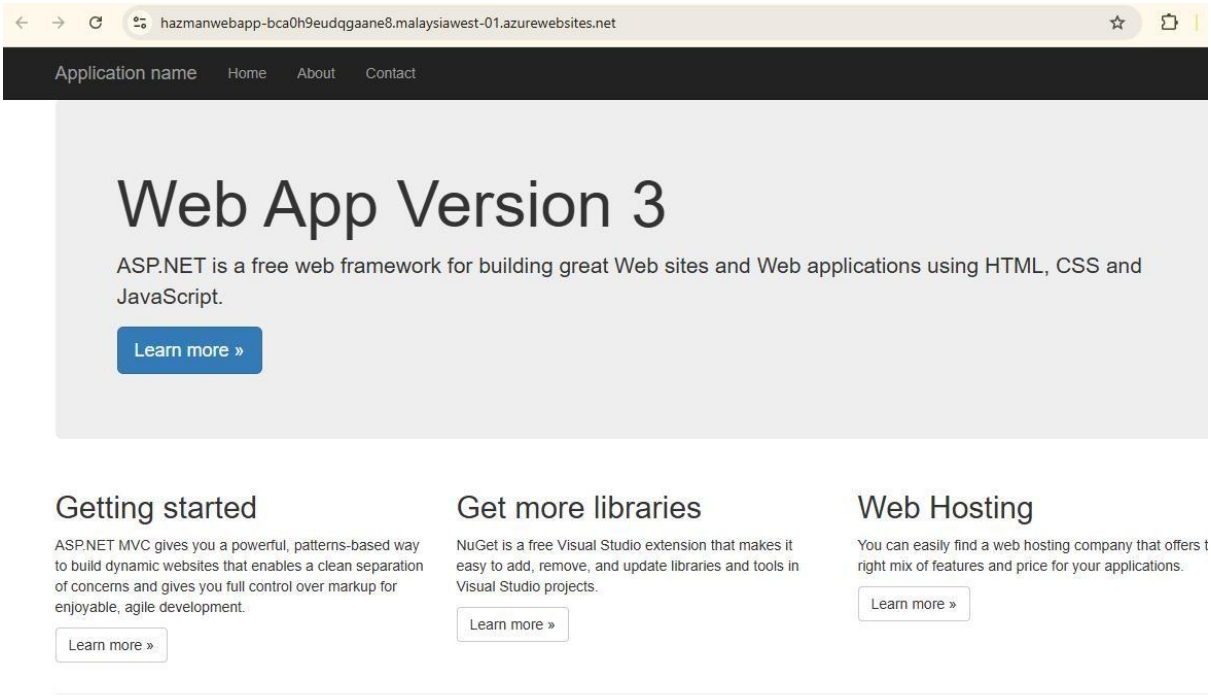



Next, I enabled Auto Swap to streamline future deployments. I went to the Configuration page of the staging slot, selected the General settings tab, and turned on the Auto swap enabled setting. I chose production as the target slot and saved the configuration. This setup ensures any deployment to staging would automatically be swapped to production once warmed up.

To test the auto swap feature, I updated the code to create Version 3 of the app. I navigated to the file Index.cshtml under Views Home, replaced the existing <h1>Web App Version 2</h1> line with <h1>Web App Version 3</h1>, and saved the changes. Then, using the Cloud Shell, I ran git add ., git commit -m "Third version of web app.", and git push staging to deploy the new version to the staging slot.




After the push completed, Azure automatically swapped the staging slot into production. I went back to the Azure portal, accessed the production slot's Overview, and clicked Browse. The web app now displayed Version 3, confirming that auto swap worked as expected. If the old version appeared at first, refreshing the browser after a short wait resolved it.

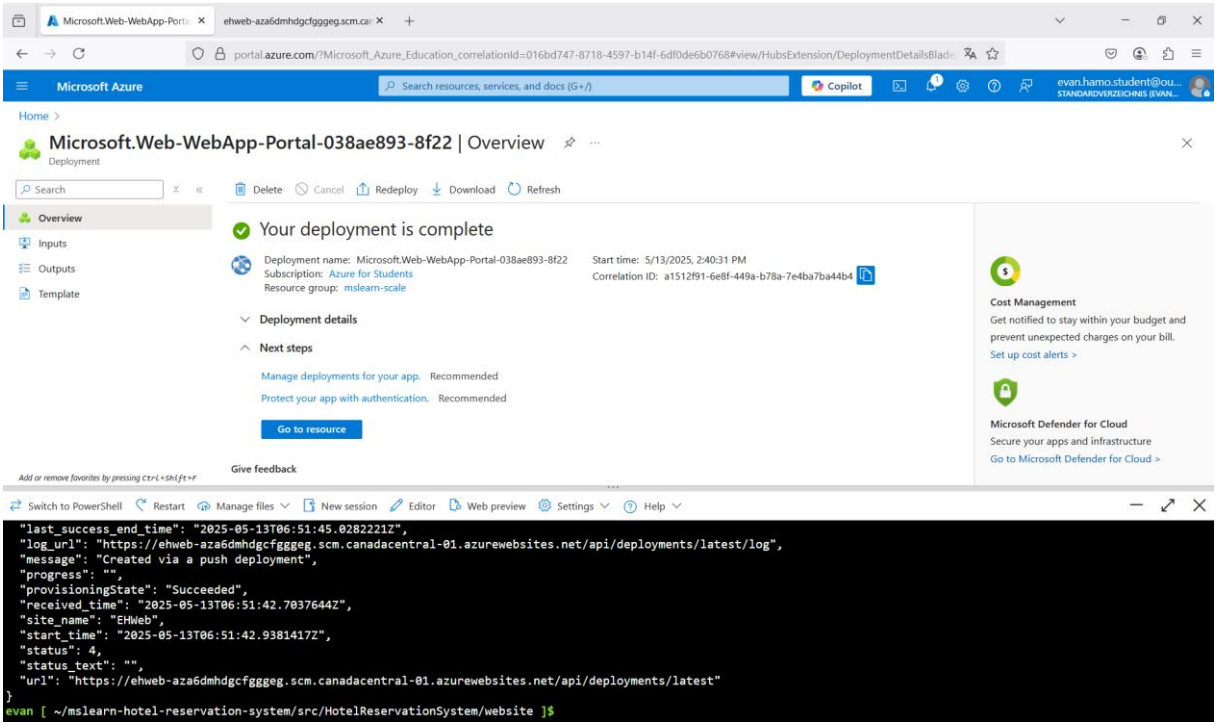



Using Azure Deployment Slots made deploying the web app safe, efficient, and reliable. I was able to test updates fully in staging before pushing to production, automate deployments using Auto Swap, and quickly roll back to a previous version if necessary. This approach reduces downtime and minimizes risk during updates, improving the overall deployment process.

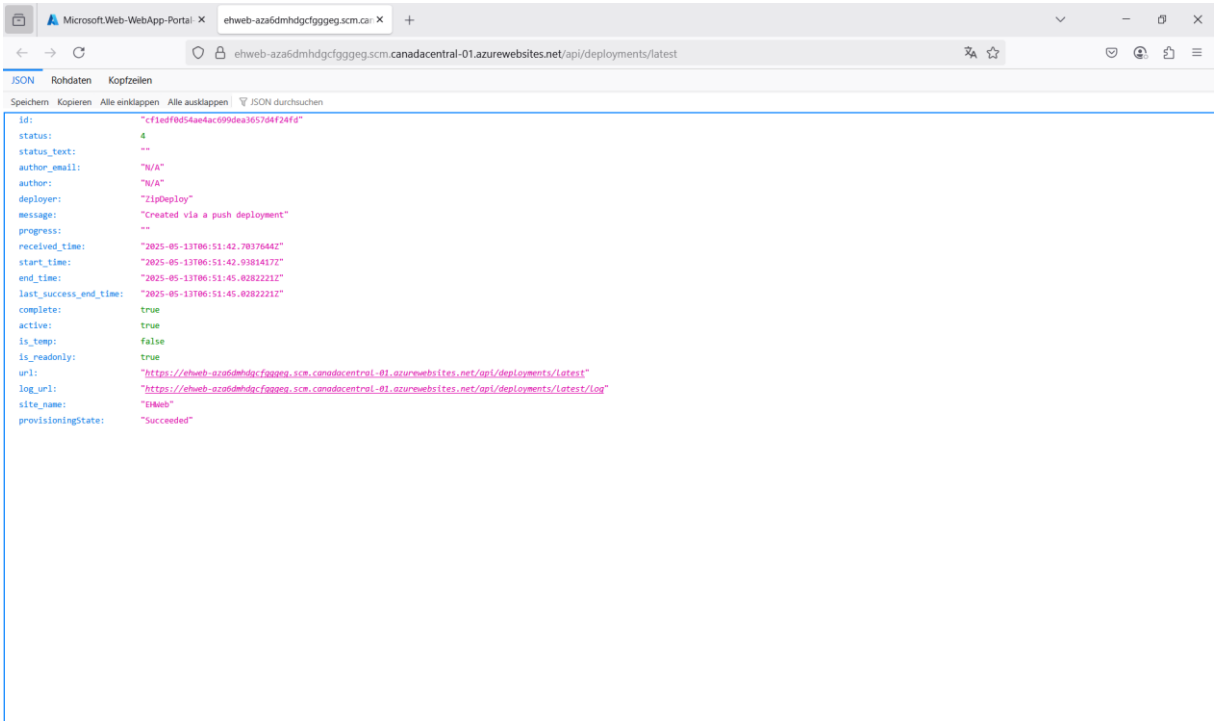
# Scale a web app manually

**Azure Lab #31**
**Name:** Evan Hamo

**Deployment Setup with Cloud Shell and Azure Portal**
In this step, I used Azure Cloud Shell to set up the project. I cloned the repository using git clone, built the app, and then packaged it with zip website.zip *. After that, I deployed it to Azure using a ZIP deployment. The Azure Portal confirms that the deployment was completed successfully, and the app is now running in the cloud.

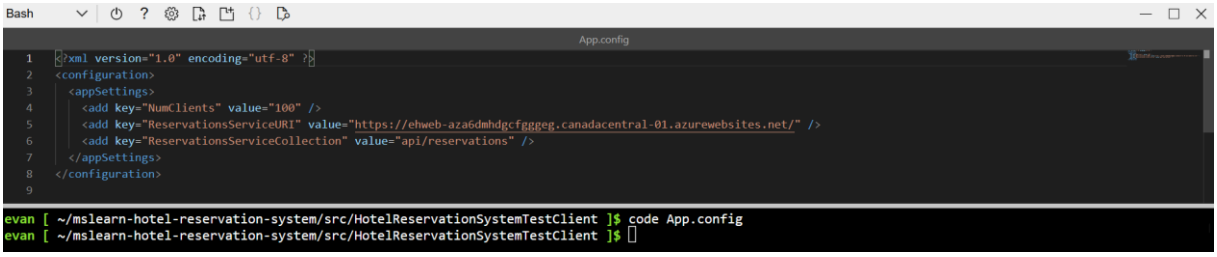


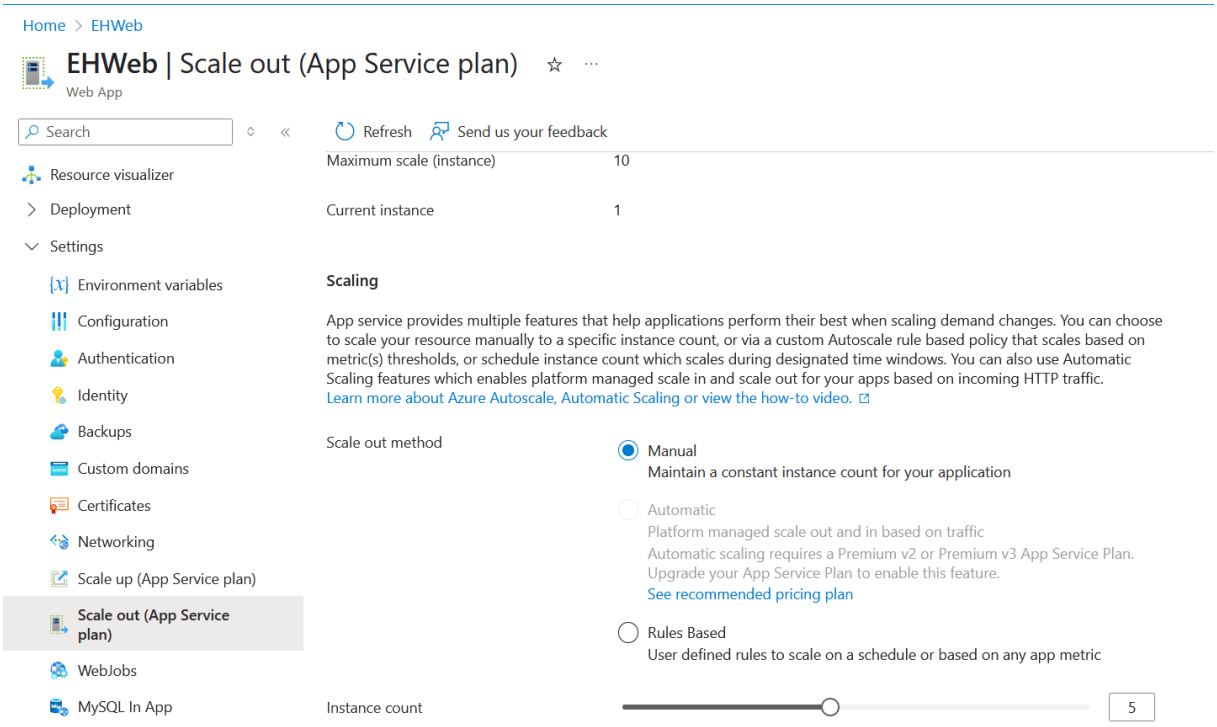

This screenshot shows that the deployment of the HotelReservationSystem app to Azure was successful. In the Azure portal, the message “Your deployment is complete” confirms that everything worked. In the terminal below, you can see the zip deployment command and the success message. I also tested the web app by opening the API link in the browser. It returned a JSON response with reservation data, which means the web app is running and correctly configured.


Microsoft.Web-WebApp-Portal
ehweb-aza6dmhdgcfgggeg.scm.ca
ehweb-aza6dmhdgcfgggeg.scm.canadacentral-01.azurewebsites.net/api/deployments/latest
JSON
Rohdaten
Kopfzeilen
Speichern
Kopieren
Alle einklappen
Alle ausklappen
JSON durchsuchen
id: "cf1edf0d54ae4ac699dea3657d4f24fd"
status: 4
status_text: ""
author_email: "N/A"
author: "N/A"
deployer: "ZipDeploy"
message: "Created via a push deployment"
progress: ""
received_time: "2025-05-13T06:51:42.7037644Z"
start_time: "2025-05-13T06:51:42.9381417Z"
end_time: "2025-05-13T06:51:45.0282221Z"
last_success_end_time: "2025-05-13T06:51:45.0282221Z"
complete: true
active: true
is_temp: false
is_readonly: true
url: "https://ehweb-aza6dmhdgcfgggeg.scm.canadacentral-01.azurewebsites.net/api/deployments/latest"
log_url: "https://ehweb-aza6dmhdgcfgggeg.scm.canadacentral-01.azurewebsites.net/api/deployments/latest/log"
site_name: "EHWeb"
provisioningState: "Succeeded"


ehweb-aza6dmhdgcfgggeg.canadacentral-01.azurewebsites.net/api/reservations/1
JSON
Rohdaten
Kopfzeilen
Speichern
Kopieren
Alle einklappen
Alle ausklappen
JSON durchsuchen
reservationID: 1
customerID: "208876526"
hotelID: "Hotel 785982216"
checkin: "2025-06-05T12:44:12.570231+00:00"
checkout: "2025-08-12T13:07:11.4412964+00:00"
numberOfGuests: 1
reservationComments: "W9Pb5QU4/sbRzYVBZlpyszlxlMP/GJkprKwuLP2E3cSNtNkWYnA0q4Z5tyMhqi/gNbne0hMekj3zOYkrnd30/d/tk02QI22jw30CGjTT+Nl5R+PZPMJtKWzlE0Iql+43hlsdo1ZcEWTQ2fPbwWUSZqbDxGPZ/
zLsFwlpdJFFdTokLDNXk4qmMz1C9WeOE26CW2iuntvc4BKJ7GaWzwXP16VsKSE+orDOqgaSAMTaC/5QG/nDrHntWJLryBKzmSuh+rQyzAa9q+kDTla3SoAyt2K4/FNVApoPW6Z//z/do0Yjj4/
+mEauSBzV1aBeK2bNZhLRZc1qwTZ456K+S899hfaH1OZ/2AOHHHMG/5vYDJR3QQSnnDFqN2GdJUvAhnM+ZdHKkQMaO5"

In this step, I edited the App.config file to connect the test client to my deployed Azure web app. I set the NumClients to 100 and inserted the correct URL for the ReservationsServiceURI. This setup makes sure that the client app will send requests to the right web service for the performance test.



# Scale-Out 1 vs 5 Instance

This chart shows the performance of the web app before and after scaling out. On the left side, with only 1 instance, the response time and error rate were higher. After increasing to 5 instances (right side of the red line), the CPU time increased slightly, but the average response time and HTTP errors decreased, showing that the app handled the load better.

# Scale up the pricing plan

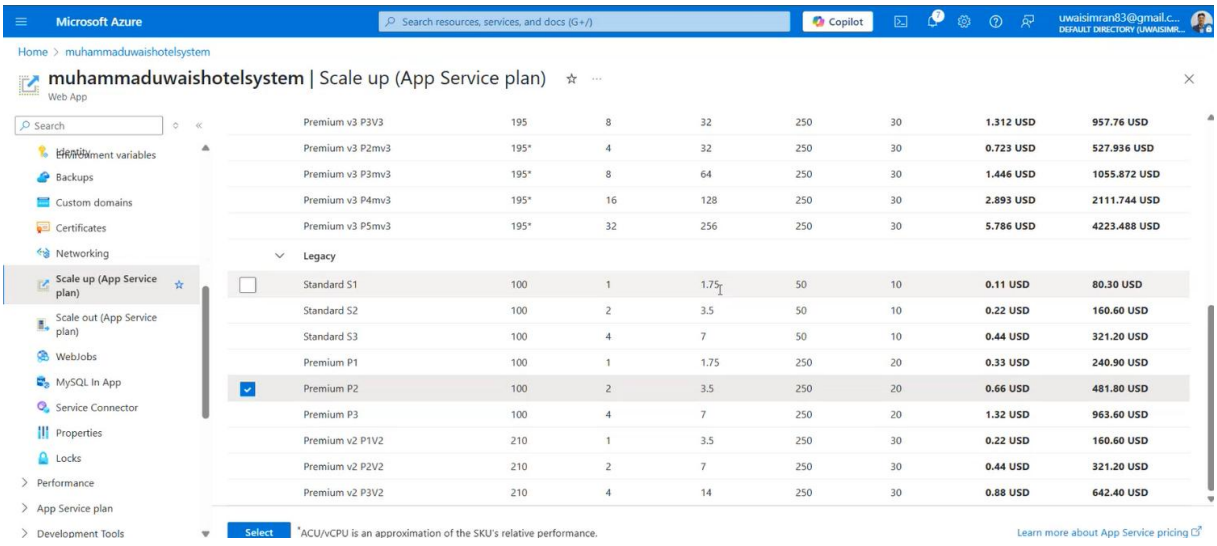
**Azure lab #32**

In Azure cloud computing, scaling up refers to increasing the capacity or performance of an existing resource by upgrading it to a higher-tier or more powerful configuration. This is also known as vertical scaling. For example, in the Azure Portal, scaling up can involve moving an App Service Plan to a higher pricing tier that offers more CPU power, memory, and additional features. Similarly, it could mean upgrading a virtual machine (VM) to a larger size that can handle heavier workloads. In the context of a hotel reservation system, scaling up is important when the application needs to support more advanced features or handle more data processing. It provides the necessary resources to maintain performance as the app becomes more complex. Scaling up can also support scaling out, which means adding more instances of a service to handle increased traffic. Therefore, scaling up is an essential step to ensure the system remains reliable and efficient as demand grows.

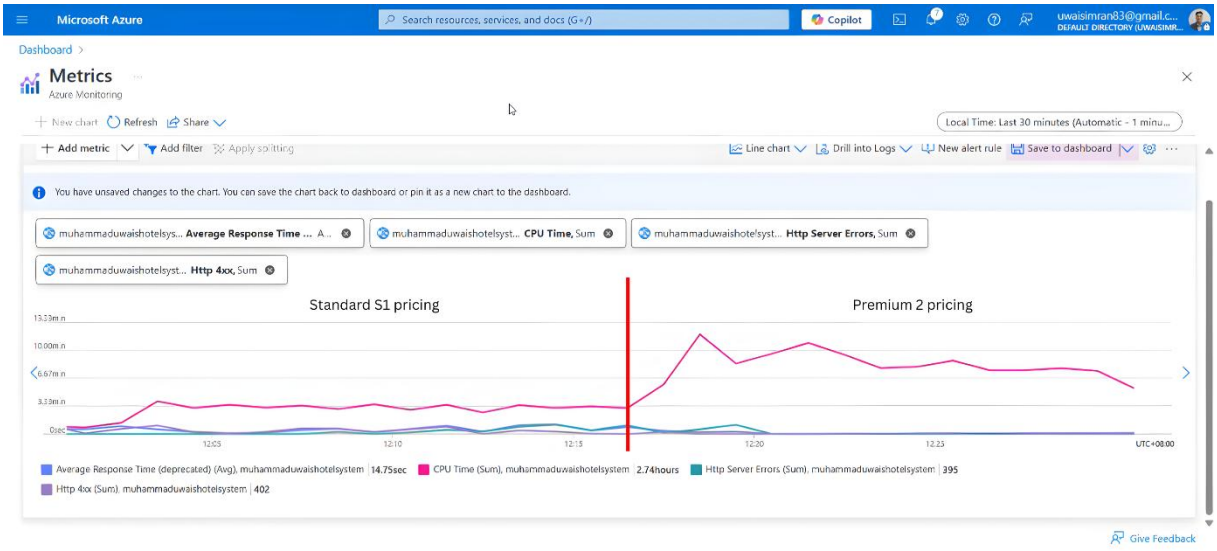
For this web application project, the default pricing tier is set to Standard S1, which offers limited memory and processing power. Specifically, the Standard S1 plan includes only 1 vCPU, 1.75 GB of RAM, and 50 GB of storage, which is not sufficient for handling larger volumes of client data or delivering faster response times. As a result, the performance of the web application is weak, especially when it comes to processing and managing data from registered clients efficiently.

To improve performance, the pricing plan needs to be upgraded to a more powerful tier. Initially, Premium v2 P2v2 was considered because it offers better performance, but due to quota limitations in the selected Azure region, it could not be used. Therefore, the plan was upgraded to Premium P2, which provides 2 vCPU, 3.5 GB of RAM, and 250 GB of storage. This Premium P2 tier offers better computing power and storage capacity, allowing the web application to perform faster and manage client data more efficiently, even though it comes at a higher cost.

To upgrade the pricing plan, users need to access the web application that has been created in the Azure Portal and navigate to Settings > App Service (Pricing Plan). In this section, Azure provides various pricing tiers that users can choose from, depending on their system requirements. Each tier offers different levels of performance in terms of CPU, memory, and storage capacity. However, it is important to note that the more powerful the plan, the higher the cost will be. Because of this, users must make careful and informed decisions when selecting a pricing plan. It is essential to balance performance and budget, and to choose a plan that can meet the system's current and future needs without overspending unnecessarily.




After upgrading the pricing plan, the graph statistics data displayed in the Azure Portal shows a clear increase in CPU performance and several other system processes. The diagram below highlights a significant difference in system performance before and after the upgrade. It is evident that the Standard S1 plan could only handle client data at a slower speed due to its limited CPU and memory capacity. In contrast, the Premium P2 pricing plan, with its higher CPU power and larger memory storage, is able to process data more efficiently and handle higher workloads. The average response time has also improved noticeably, resulting in faster interactions for users. Additionally, the number of server errors has decreased, indicating better overall system stability. This demonstrates that selecting a higher pricing plan, when necessary, can significantly improve the efficiency and reliability of a web application especially if the plan is chosen based on the actual needs of the system.



# Conclusion

This project has provided valuable hands-on experience in managing web applications using Microsoft Azure, with a strong focus on deployment strategies, performance scaling, and configuration best practices. By exploring deployment slots, we were able to implement safe and structured app updates with minimal downtime, leveraging staging environments to test changes before pushing to production. The integration of Git and Azure's Deployment Center simplified version control and enabled efficient team collaboration.

Furthermore, scaling out and scaling up the web app allowed us to observe the impact of resource allocation on application performance, especially under load. Through detailed performance metrics, we saw firsthand how additional instances and higher-tier pricing plans can enhance responsiveness and reduce errors.
Overall, this assignment not only reinforced theoretical concepts from the Cloud Computing syllabus but also sharpened our practical skills in real-world cloud operations. It highlights the importance of planning, testing, and resource management in building robust, scalable, and efficient cloud-based systems. These are essential skills for any future IT professional working in cloud environments.

# Link video of our project

IIB43203 - CLOUD COMPUTING (MARCH 2025) :
https://youtu.be/ZhzSjRXQP4c