Azure Cloud Computing Project: Implementation and Hands-on Experience

INTRODUCTION

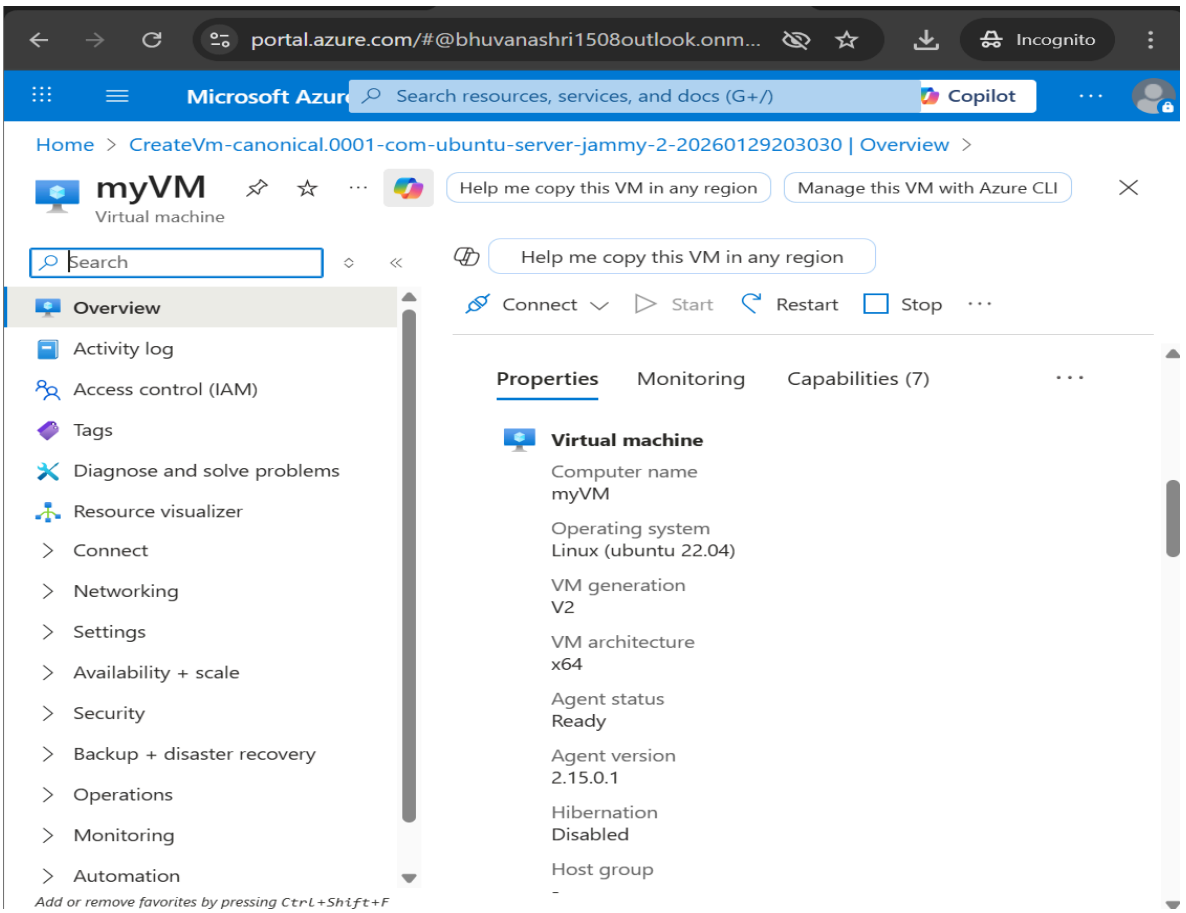
This project demonstrates an end-to-end cloud-based data analytics solution using Microsoft Azure and Power BI. The objective is to design, deploy, and analyze structured business data by integrating cloud database services with data visualization tools. The AdventureWorksLT dataset is used to represent a real-world sales and business scenario.

An Azure SQL Database is created to securely store transactional data, which is explored and validated using SQL queries. The database is then connected to Power BI Desktop to create calculated measures and interactive visualizations that analyze sales performance, product distribution, order volume, and yearly trends. The project showcases the practical application of cloud computing, SQL, and business intelligence for data-driven decision-making.

EXECUTION

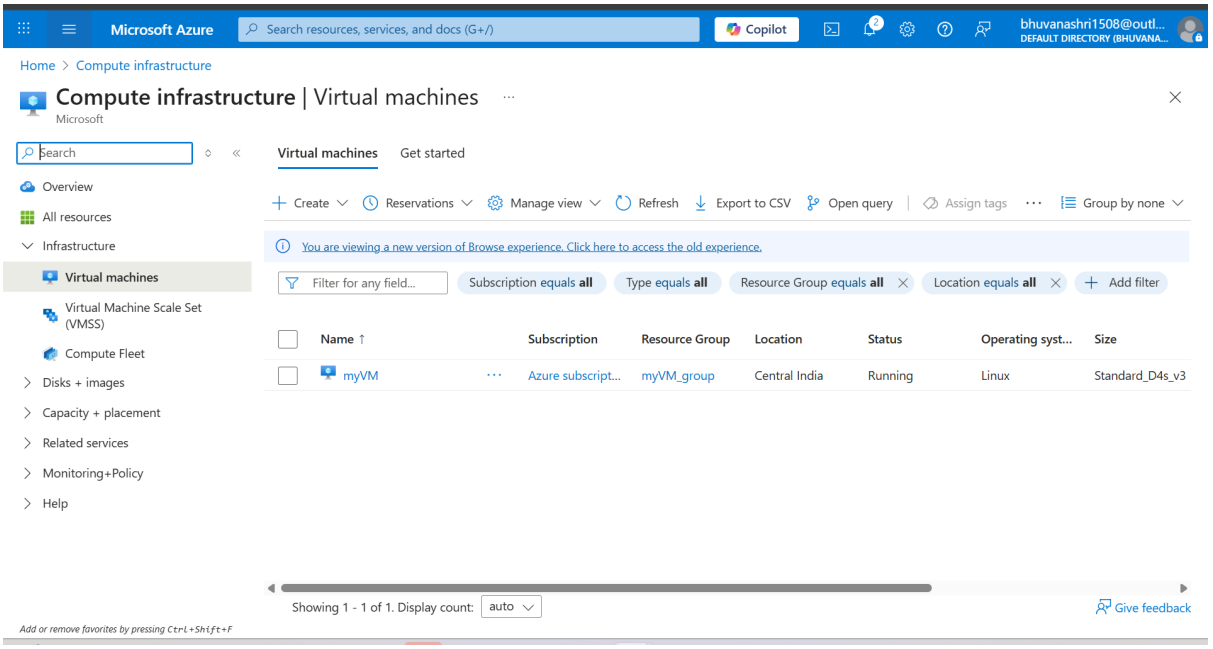
Step 1: Azure Virtual Machine – Overview Verification

- Navigated to the **Virtual Machine Overview** page in the Azure Portal.
- Verified that the Ubuntu VM **myVM** was created successfully.
- Confirmed the operating system as **Linux (Ubuntu 22.04 LTS)**.
- Checked VM status and ensured the **Azure agent status is Ready**.
- Verified that VM management options such as **Connect, Start, Restart, and Stop** are available.



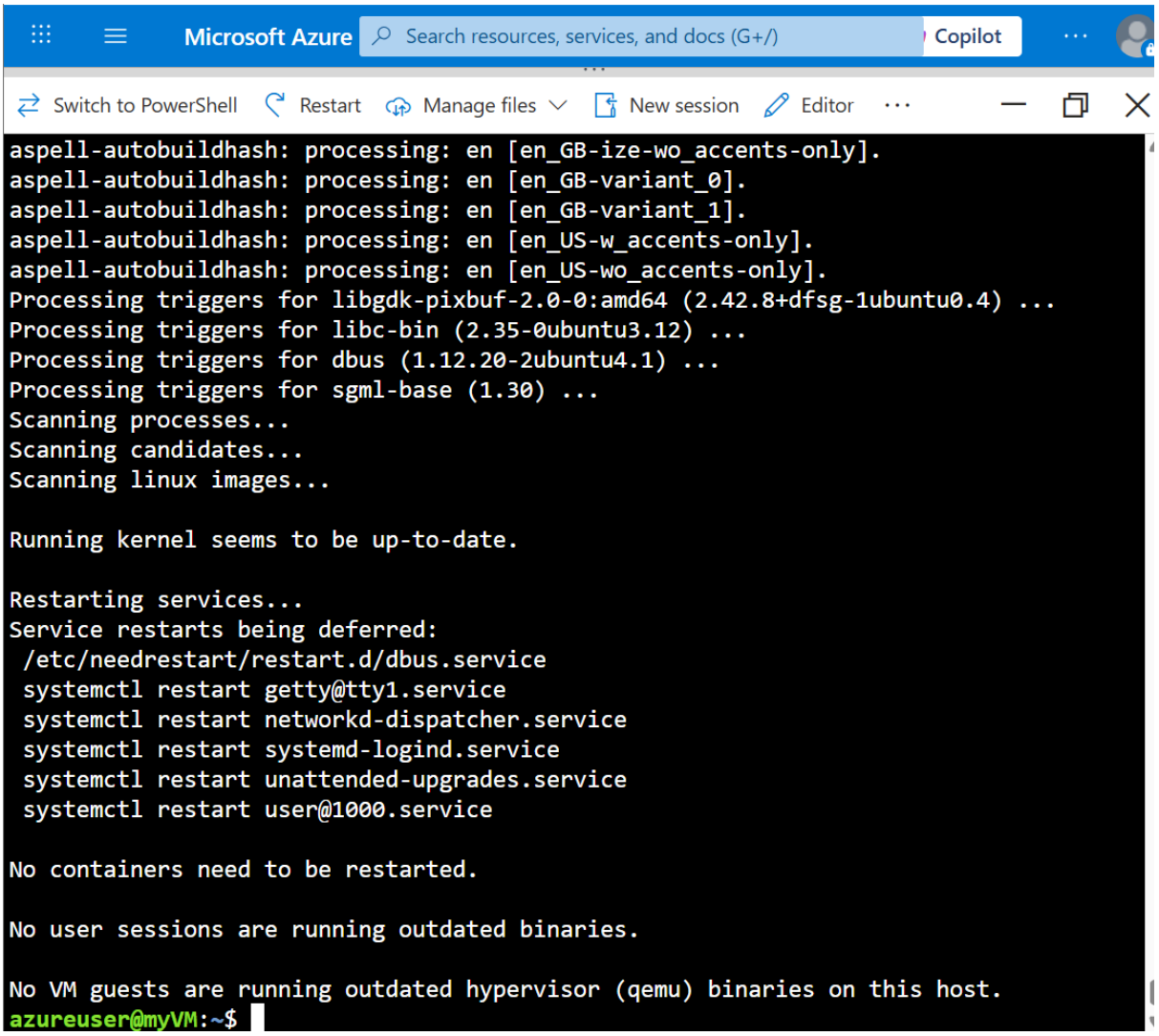
Step 2: Viewing the Virtual Machine in Azure Portal

- Navigated to **Compute Infrastructure → Virtual Machines** in the Azure Portal.
- Verified that the virtual machine **myVM** is listed under the active subscription.
- Confirmed the VM is associated with the resource group **myVM_group**.
- Verified the **location** of the VM as **Central India**.
- Checked the VM **status** as **Running**, confirming successful deployment.
- Confirmed the **operating system** as **Linux** and the **VM size** as **Standard_D4s_v3**.



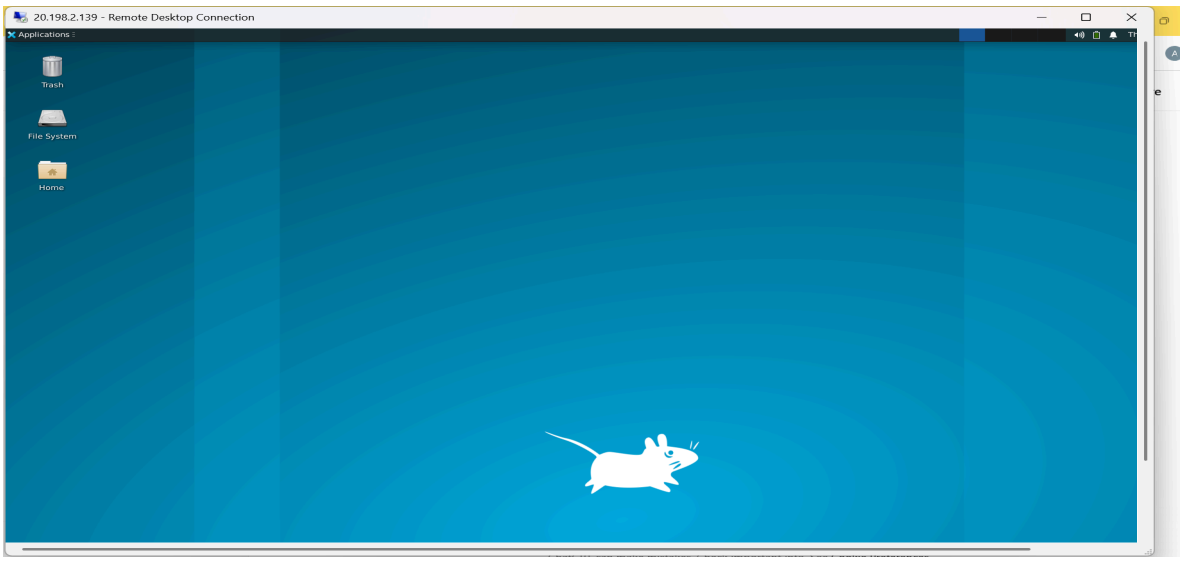
Step 3: Resetting Login Credentials for the Ubuntu Virtual Machine

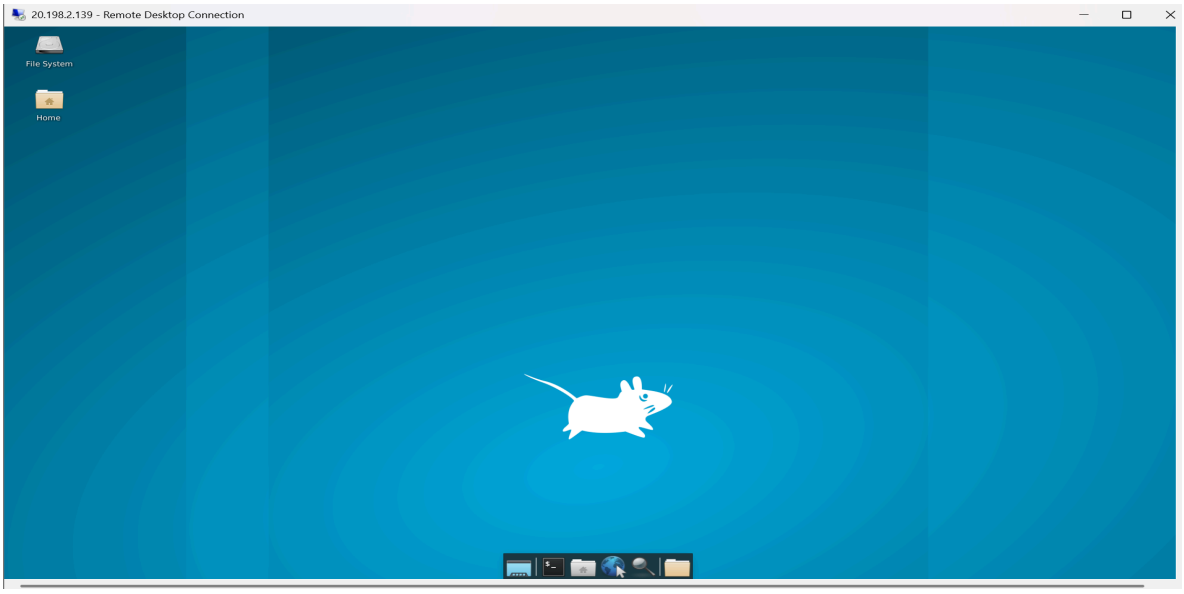
- Opened the **myVM → Reset password** option from the Azure Portal.
- Selected the **Reset password** mode using the **VMAccessForLinux** extension.
- Reset the credentials for the existing Linux user to ensure successful login access.
- Applied the changes by clicking **Update**, allowing secure authentication to the VM.
- This step ensures proper user access for connecting to the Ubuntu VM via SSH or RDP.



Step 4: Connecting to the Ubuntu Virtual Machine Using Remote Desktop (RDP)

- Established a **Remote Desktop Connection (RDP)** to the Ubuntu Virtual Machine using its public IP address.
- Successfully logged in using the configured Linux user credentials.
- Verified that the **Ubuntu GUI desktop environment** is displayed.
- Confirmed that the graphical interface is accessible, indicating successful installation of **GUI and XRDP**.
- This step validates that the Ubuntu VM can be accessed remotely using a graphical desktop interface.



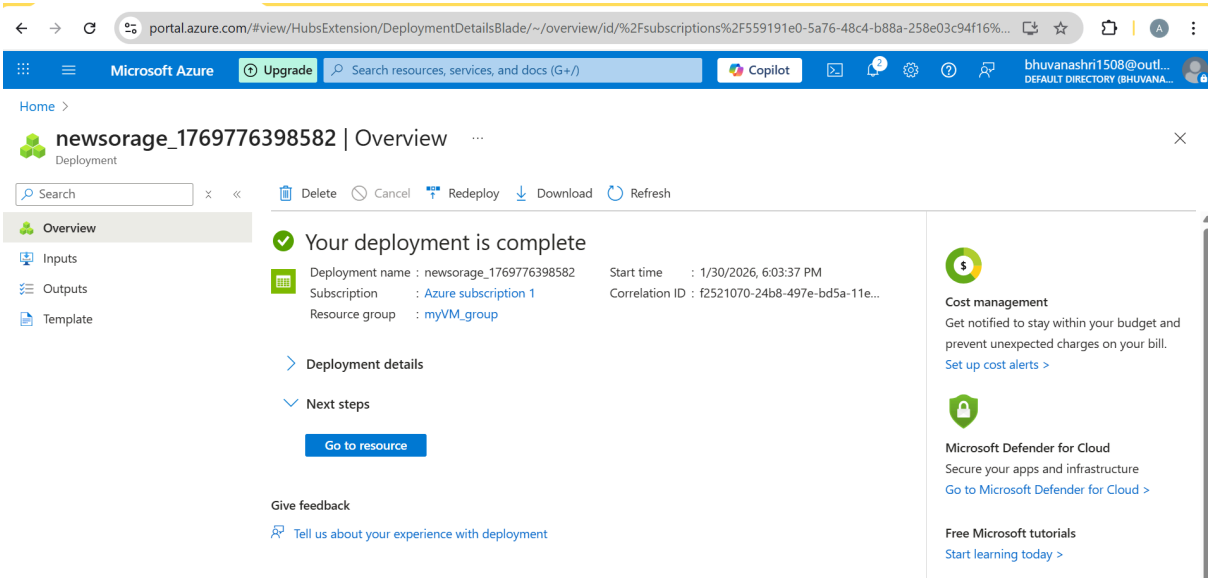


TASK 2: Azure Storage Account

Step 1: Creating an Azure Storage Account

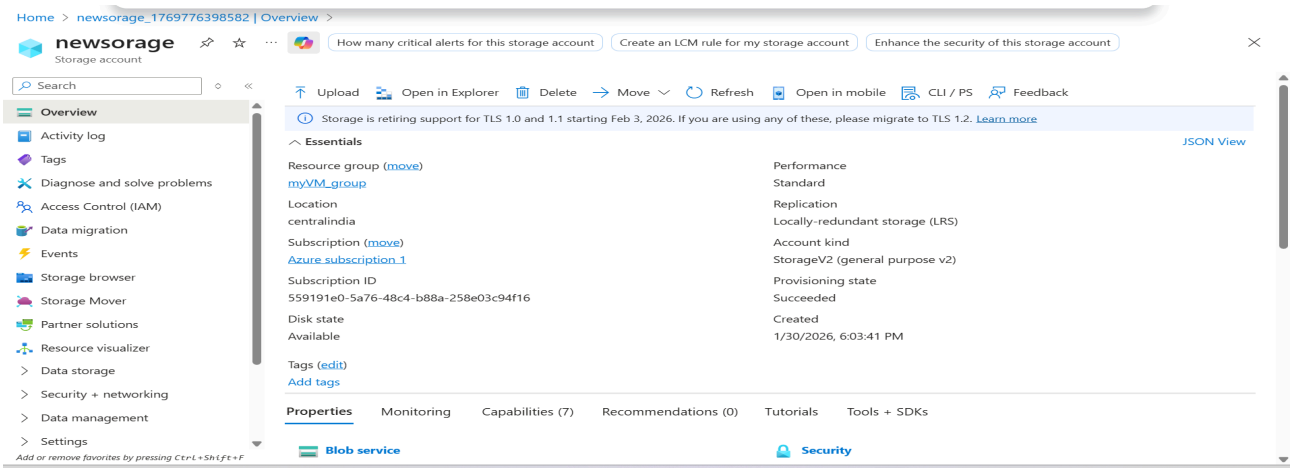
- Initiated the creation of a new **Azure Storage Account** from the Azure Portal.
- Verified that the deployment completed successfully.
- Confirmed the storage account name as **newsorage_1769776398582**.
- Associated the storage account with the resource group **myVM_group**.
- Verified the storage account location as **Central India**.

- Ensured the provisioning state shows **Succeeded**.



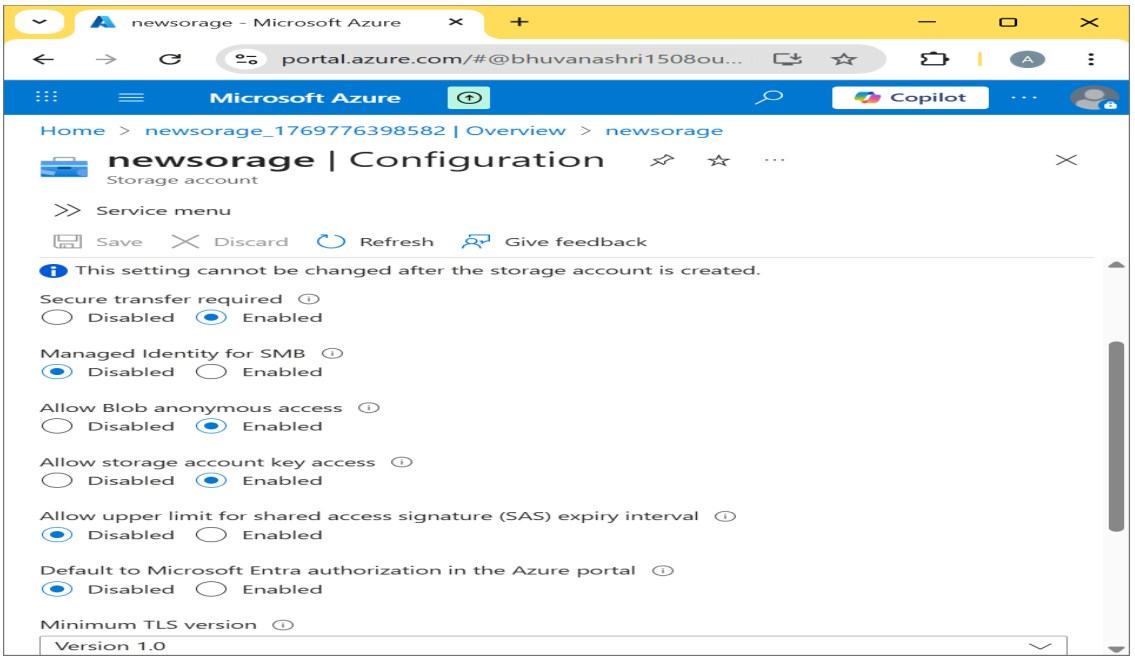
Step 2: Accessing the Storage Account Overview

- Navigated to the **Overview** section of the newly created storage account.
- Verified the **account kind** as **StorageV2 (General-purpose v2)**.
- Confirmed the **replication type** as **Locally Redundant Storage (LRS)**.
- Checked the **performance tier** as **Standard**.
- Ensured the storage account is ready for further configuration.



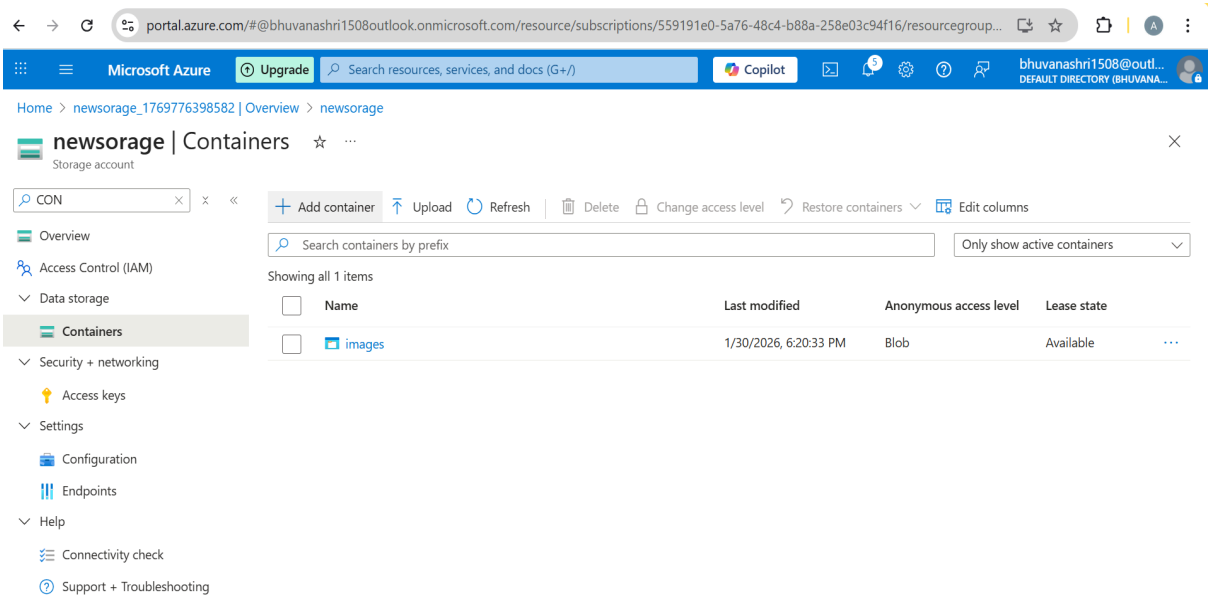
Step 3: Configuring Storage Account Settings for Public Blob Access

- Opened the **Configuration** section of the storage account **newsorage_1769776398582**.
- Verified that **Secure transfer required** is enabled.
- Enabled **Allow Blob anonymous access** to make blobs publicly accessible.
- Confirmed that **storage account key access** is enabled.
- Reviewed additional security and access settings to ensure compatibility with static website hosting.
- Saved the configuration changes.



Step 4: Creating a Blob Container

- Navigated to **Data storage → Containers** within the storage account **newsorage_1769776398582**.
- Created a new blob container named **images**.
- Set the **anonymous access level** to **Blob**, allowing public read access to blobs.
- Verified that the container status is **Available**.
- Confirmed the container creation time and successful setup.



Step 5: Uploading an Image to Blob Storage and Verifying Public Access

- Opened the **images** blob container within the storage account.
- Uploaded an image file named **Apple_image.png** to the container.
- Verified the blob type as **Block blob** with the access tier set to **Hot**.
- Confirmed that the image was successfully uploaded and is listed in the container.
- Accessed the image using the **blob URL** to verify that it is publicly accessible.
- Successfully viewed the image in a web browser, confirming public access configuration.

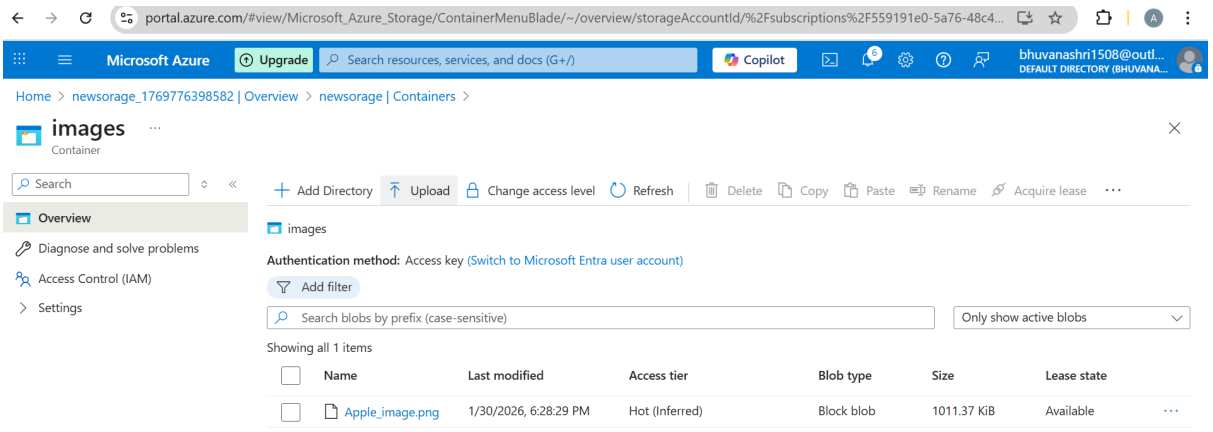
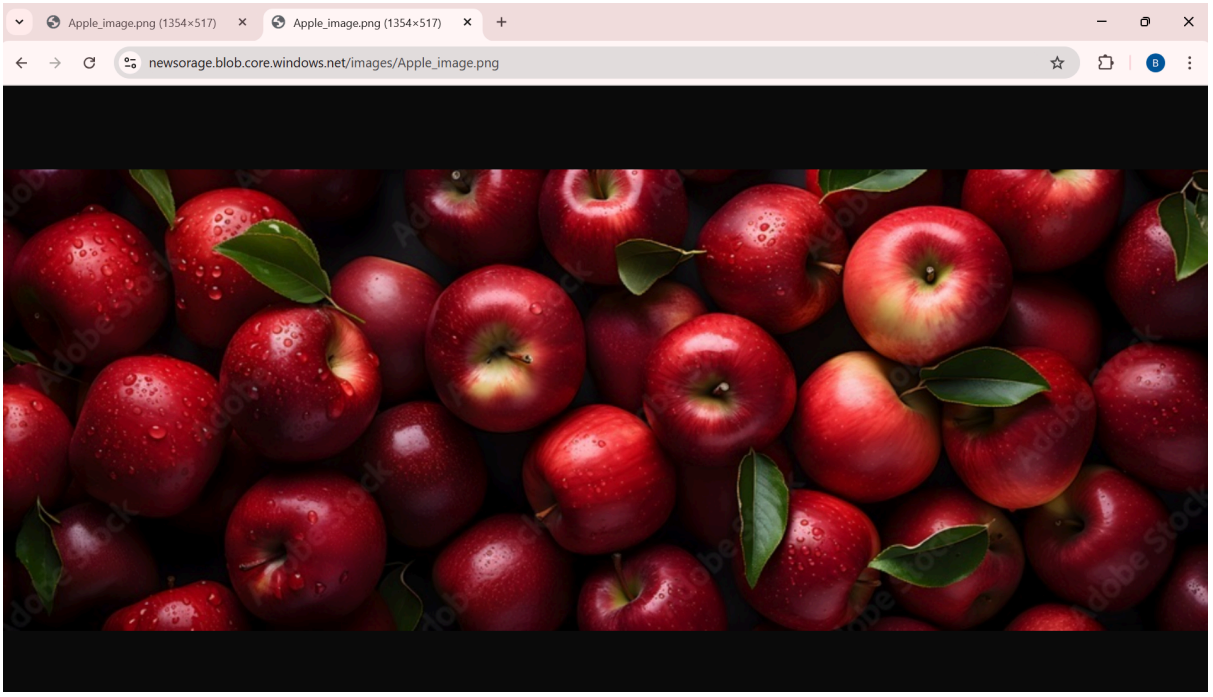


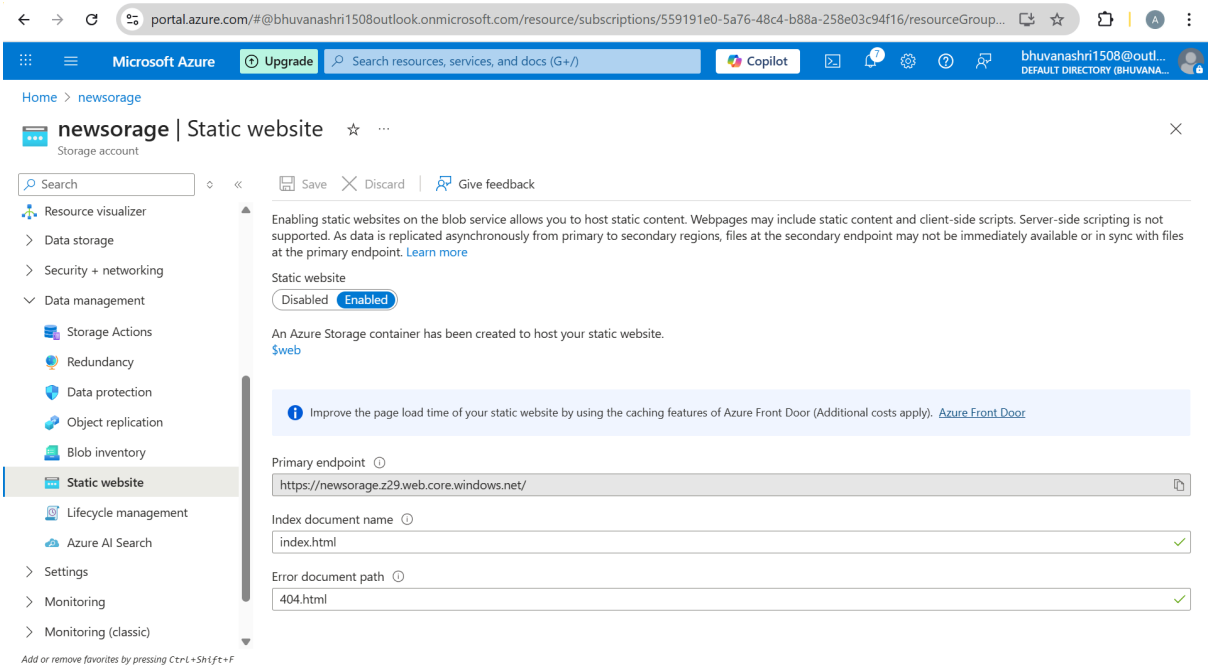
Image url: https://newsorage.blob.core.windows.net/images/Apple_image.png



Step 6: Enabling Static Website Hosting

- Navigated to **Data management → Static website** in the storage account.

- Enabled the **Static website** option.
- Set the **Index document name** as **index.html**.
- Set the **Error document path** as **404.html**.
- Noted the **primary endpoint URL** generated for the static website.
- Confirmed that the special **$web container** was automatically created.



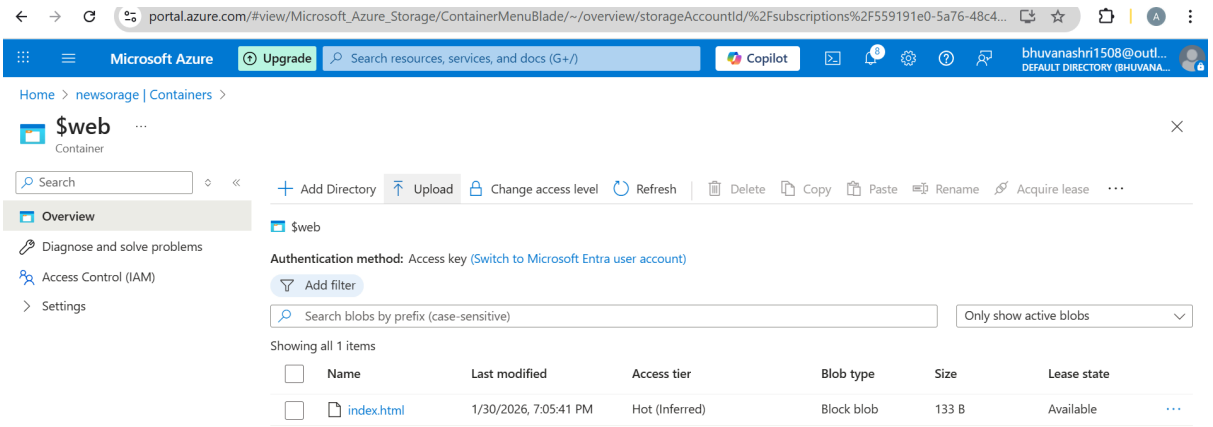
HelloWorld website URL:

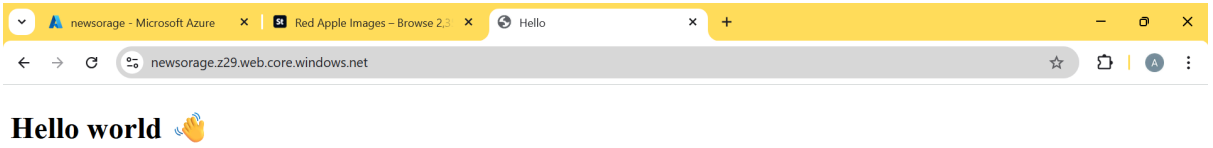
https://newsorage.z29.web.core.windows.net/

Step 7: Hosting and Verifying the Hello World Webpage

- Opened the **$web** container created for static website hosting.
- Uploaded the **index.html** file containing the Hello World content.
- Verified that the file is stored as a **block blob**.

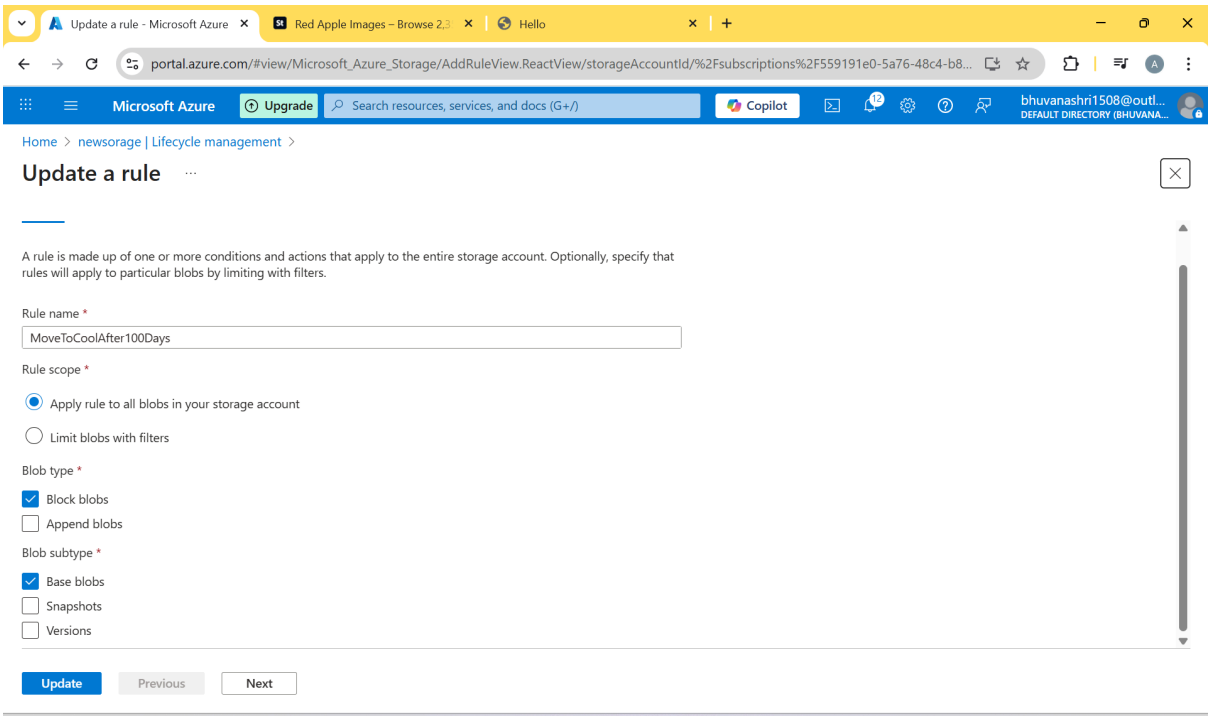
- Accessed the static website using the provided endpoint URL.
- Confirmed that the Hello World webpage loads successfully in the browser.

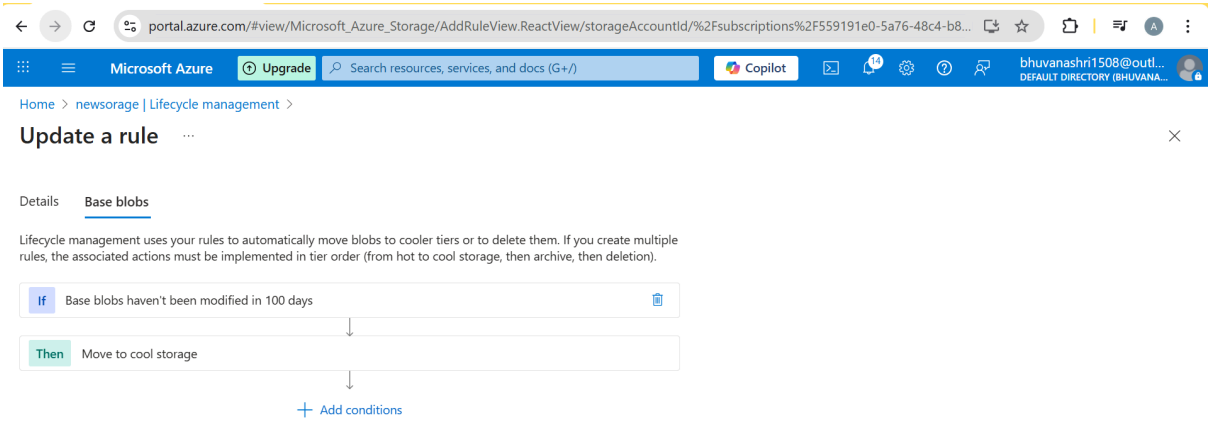




Step 8: Configuring Lifecycle Management Policy

- Navigated to **Data management → Lifecycle management** in the storage account.
- Created a new lifecycle management rule named **MoveToCoolAfter100Days**.
- Applied the rule to **all blobs** in the storage account.
- Selected the **Blob type** as **Block blobs** and **Base blobs**.
- Configured the condition to trigger when blobs **have not been modified for 100 days**.
- Set the action to **move blobs to the Cool access tier**.
- Saved and updated the lifecycle management rule successfully.

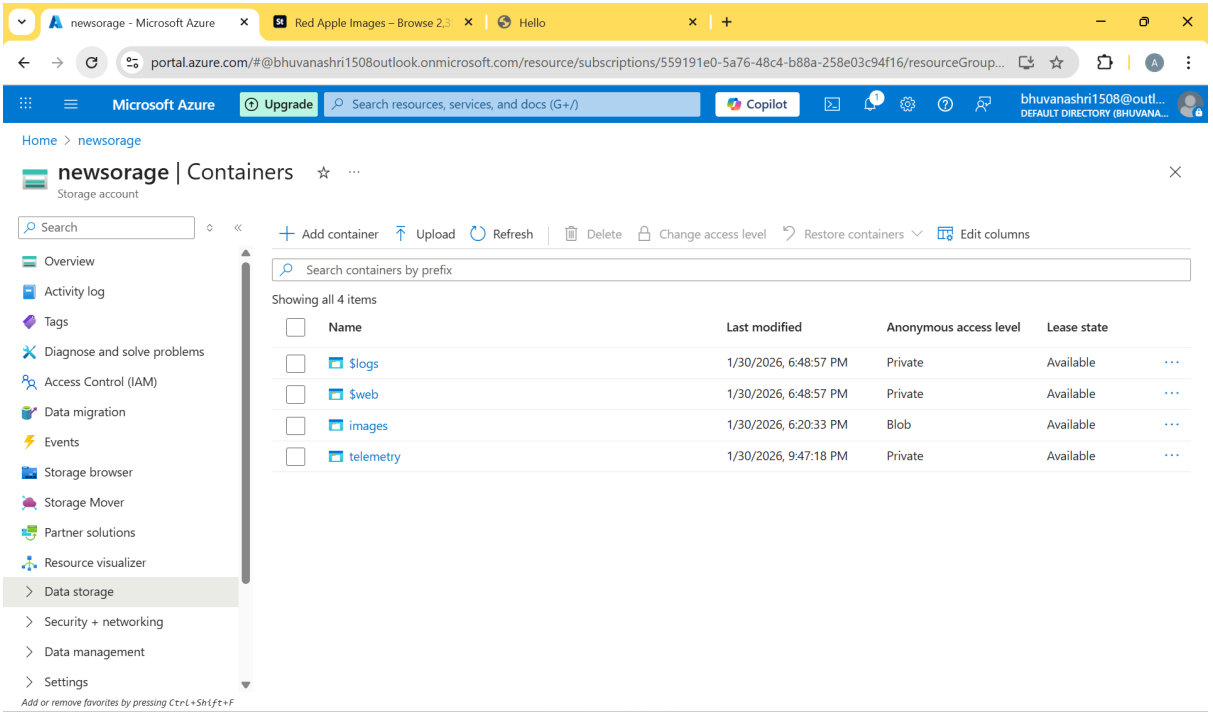




TASK 3: Azure IoT Hub – Sensor Data Ingestion and Storage

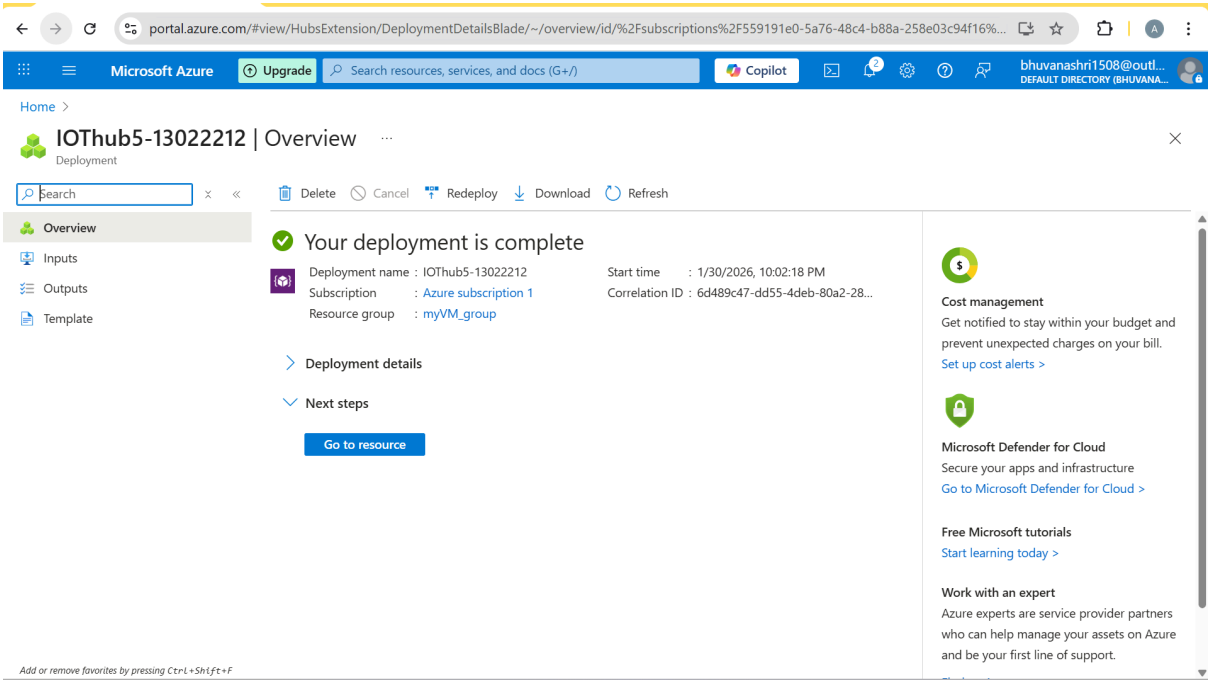
Step 1: Preparing Blob Containers for IoT Data Storage

- Opened the **Storage Account → Containers** section in the Azure Portal.
- Verified the presence of multiple containers used for different purposes.
- Confirmed the **telemetry** container is created to store IoT sensor output data.
- Observed system-generated containers such as **$logs** and **$web** for logging and static website hosting.
- Ensured the **telemetry** container is in **Available** state and ready to receive routed IoT data.
- This step prepares the storage layer required for saving temperature and humidity data from the IoT Hub.



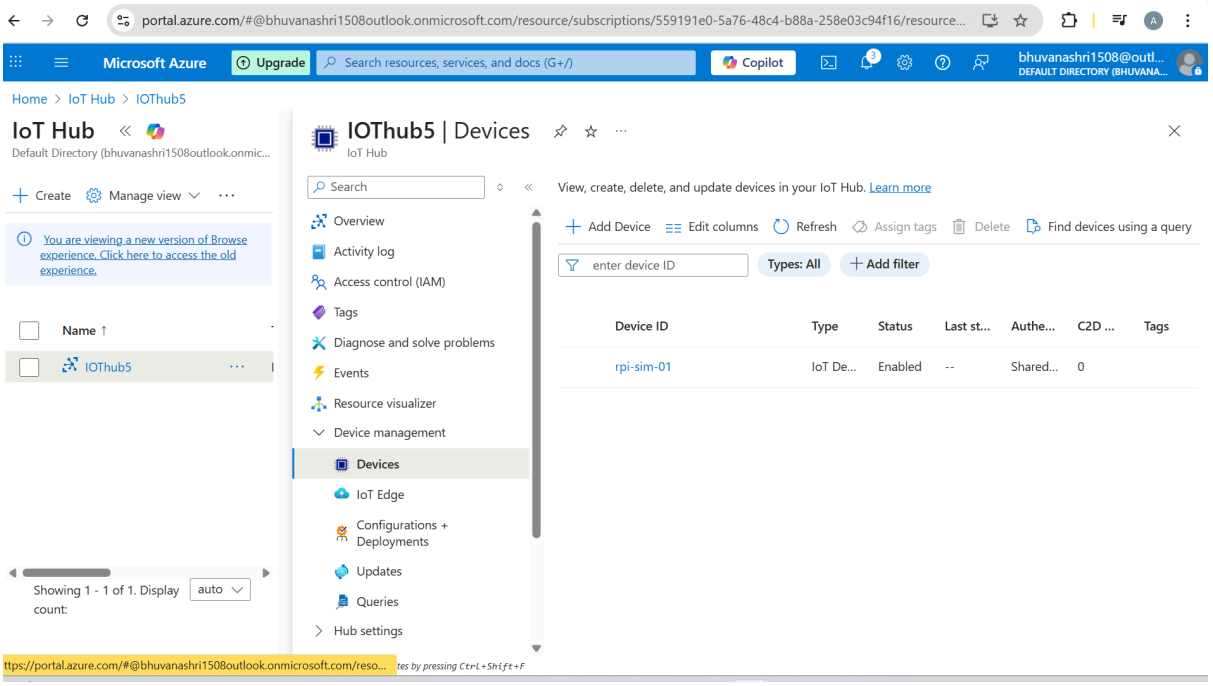
Step 2: Creating an Azure IoT Hub

- Initiated the creation of an **Azure IoT Hub** from the Azure Portal.
- Verified that the IoT Hub deployment completed successfully.
- Confirmed the IoT Hub name as **IOThub5-13022212**.
- Associated the IoT Hub with the resource group **myVM_group**.
- Verified that the provisioning state shows **Your deployment is complete**.
- Accessed the IoT Hub resource to proceed with device configuration and message routing.



Step 3: Registering a Device in Azure IoT Hub

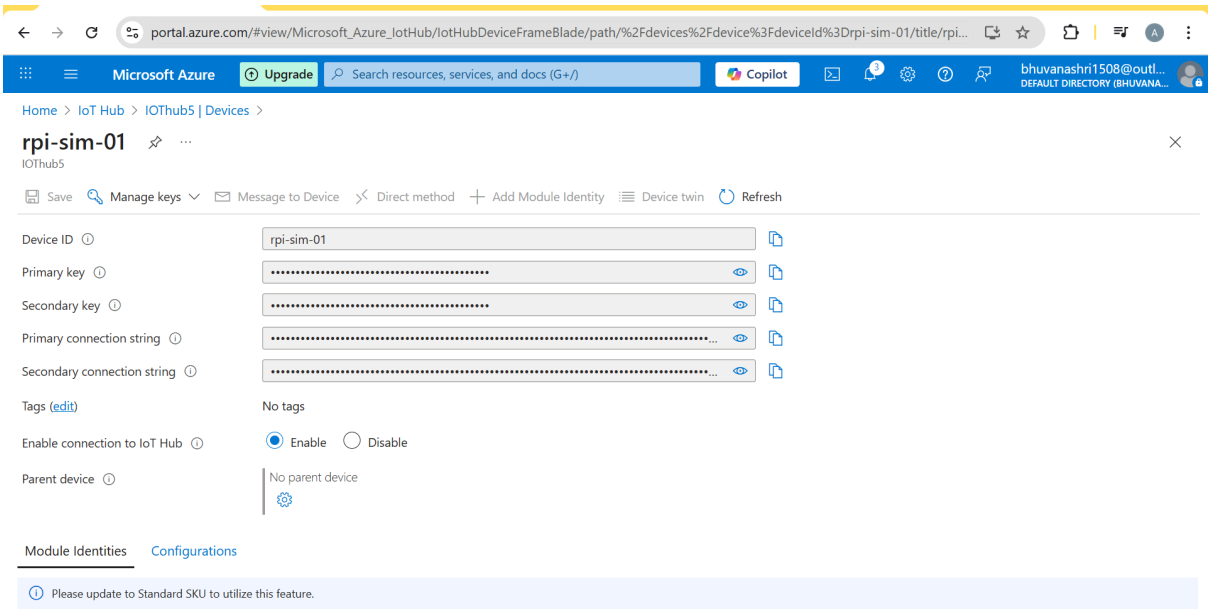
- Opened the **IoT Hub → Device management → Devices** section.
- Created and registered a new IoT device with the **Device ID: rpi-sim-01**.
- Verified the device type as **IoT Device**.
- Confirmed the device **status is Enabled**.
- Ensured the device authentication is configured successfully.
- This step registers the Raspberry Pi simulator as a device in the IoT Hub, enabling data transmission to Azure.



Step 4: Retrieving Device Connection Details

- Opened the registered IoT device **rpi-sim-01** in the IoT Hub.
- Viewed the **device authentication details**, including primary and secondary keys.
- Retrieved the **Primary Connection String** for the device.
- Confirmed that the device connection to the IoT Hub is **Enabled**.

- These credentials are used by the Raspberry Pi simulator to securely send

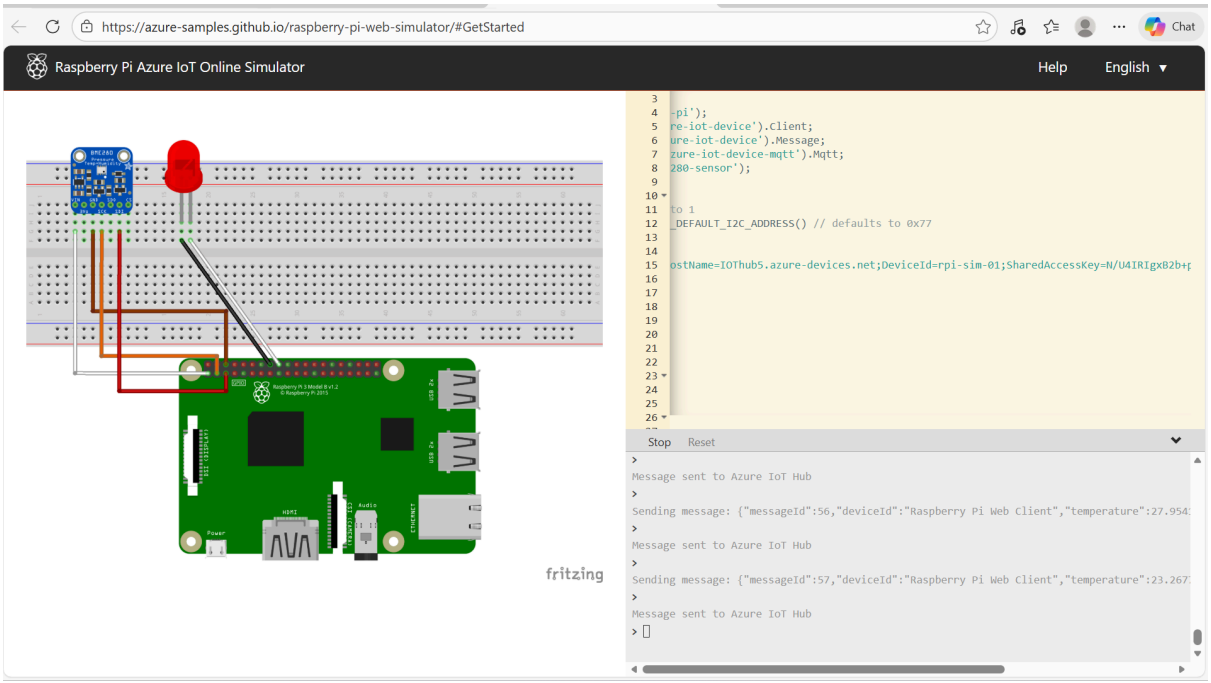


temperature and humidity data to Azure IoT Hub.

Step 5: Simulating Raspberry Pi Sensor Data

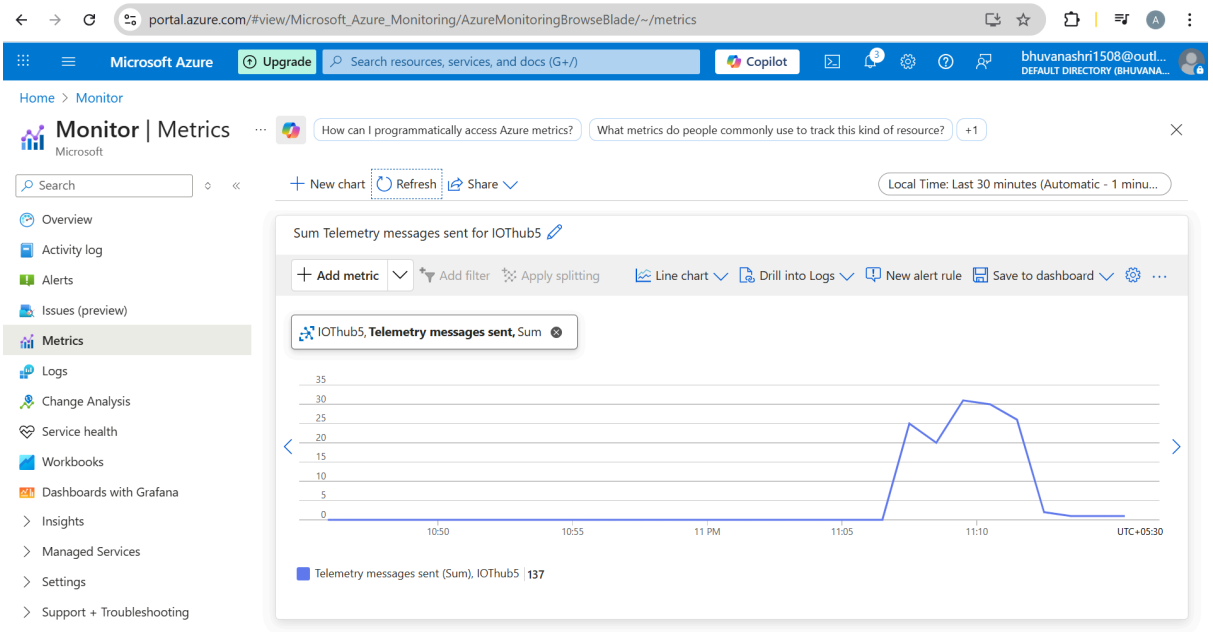
- Opened the **Raspberry Pi Azure IoT Online Simulator**.
- Configured the simulator using the **IoT Hub device connection string**.
- Simulated temperature and humidity sensor readings.
- Verified that telemetry messages are generated and sent to Azure IoT Hub.

- Confirmed successful message transmission in the simulator console.



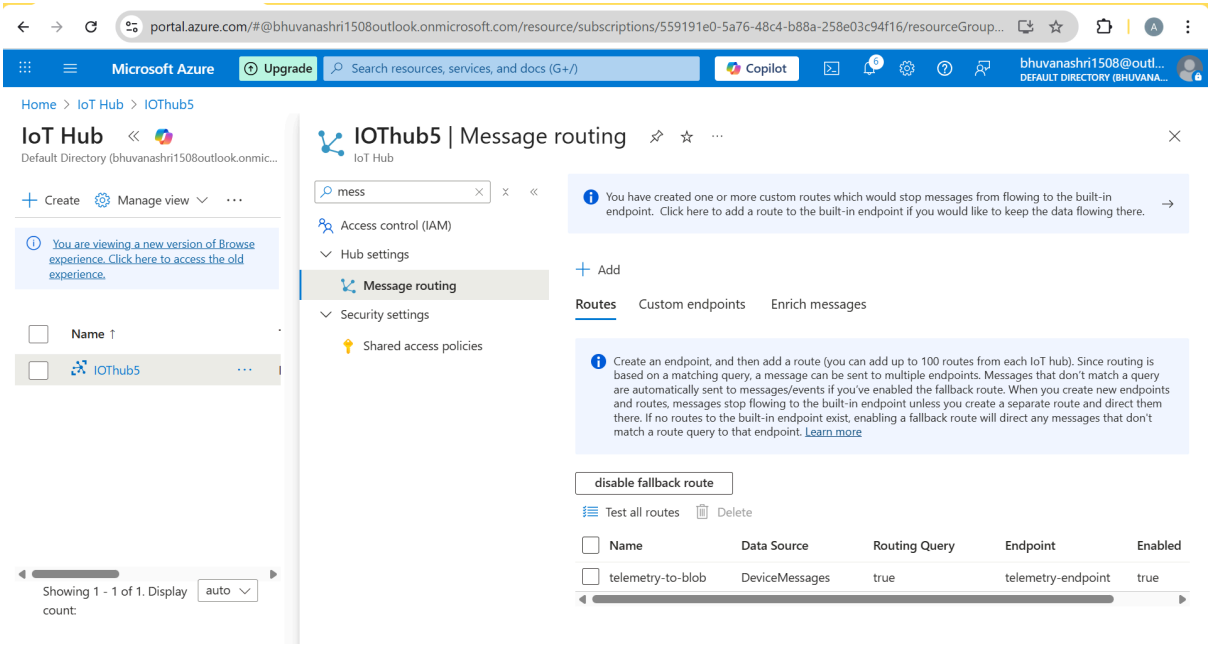
Step 6: Monitoring IoT Telemetry in Azure

- Navigated to **Azure Monitor → Metrics** for the IoT Hub.
- Selected the **Telemetry messages sent** metric.
- Verified that telemetry data from the simulated device is being received.
- Observed message trends in the metrics graph, confirming active data flow.



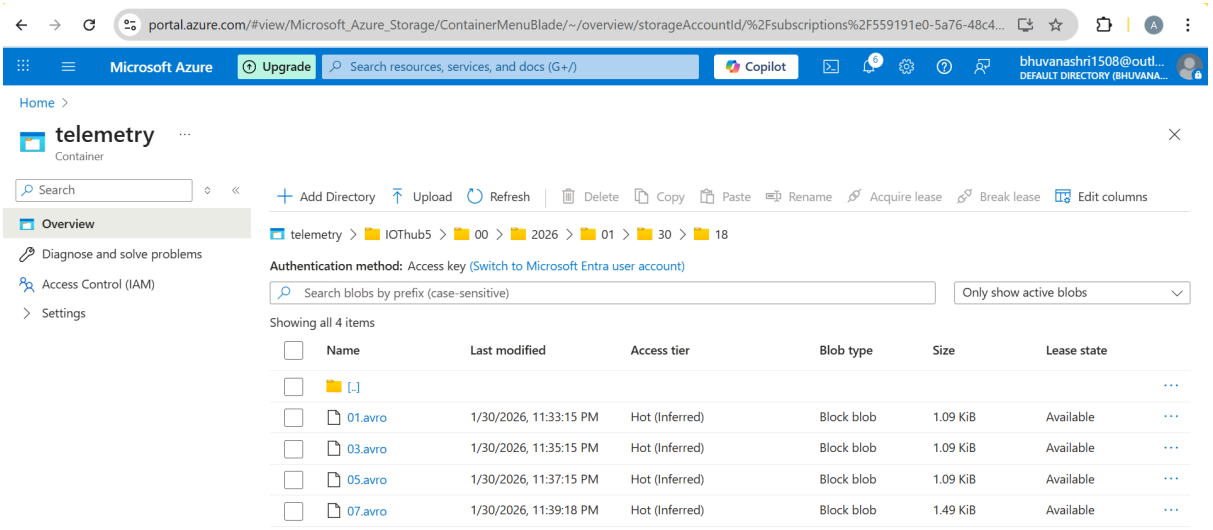
Step 7: Configuring Message Routing from IoT Hub to Blob Storage

- Opened the **IoT Hub → Message routing** section.
- Created a custom route named **telemetry-to-blob**.
- Selected the **data source** as **DeviceMessages**.
- Configured the **routing query** as `true` to route all incoming messages.
- Selected the **custom endpoint** pointing to **Azure Blob Storage**.
- Verified that the route is **Enabled**.
- This step ensures that all telemetry data from the IoT device is automatically routed to Blob Storage.



Step 8: Verifying IoT Telemetry Data Stored in Blob Storage

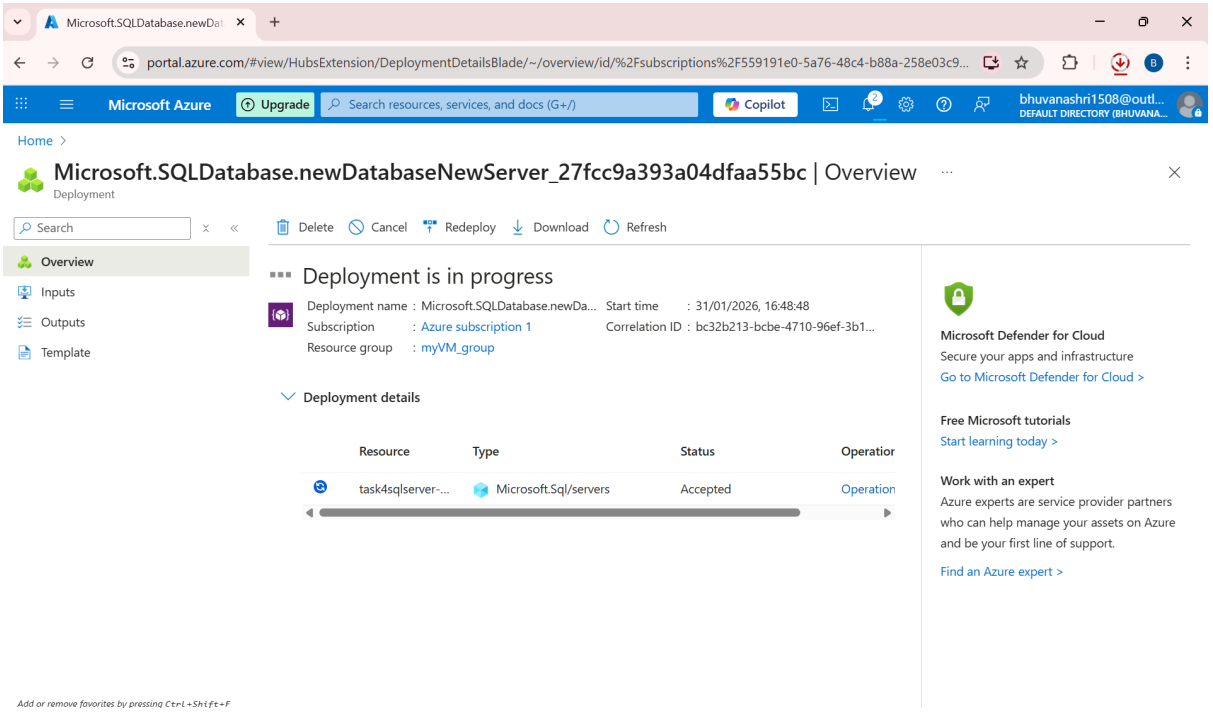
- Opened the **telemetry** container in the Azure Storage Account.
- Navigated through the auto-generated folder hierarchy created by IoT Hub routing.
- Verified multiple **.avro** files generated from incoming telemetry data.
- Confirmed the blob type as **Block blob** with access tier **Hot (Inferred)**.
- Observed timestamps matching the telemetry transmission time.
- This confirms that temperature and humidity data from the simulated device is successfully stored in **Azure Blob Storage**.



TASK 4: Azure SQL Database Creation and Power BI Integration

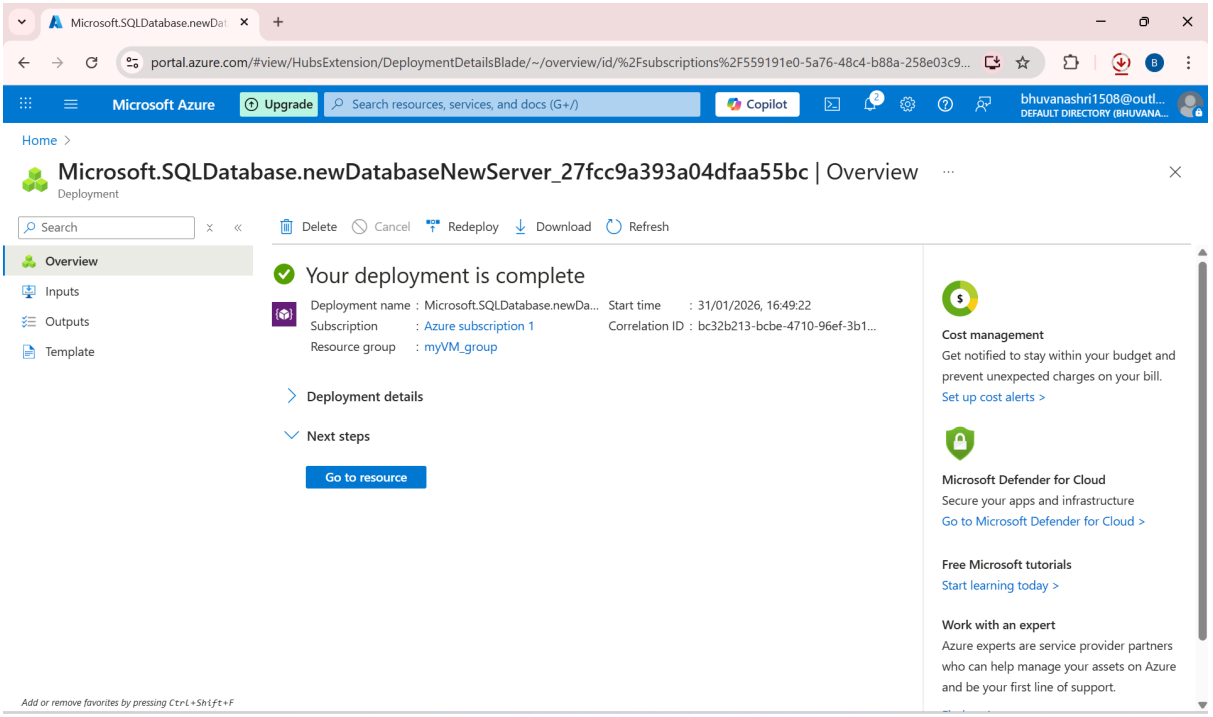
Step 1: Initiating Azure SQL Database Deployment

- Started the deployment of a new **Azure SQL Database** from the Azure Portal.
- Selected the **Azure subscription** and assigned the database to the resource group **myVM_group**.
- Triggered the creation of a **new SQL Server** along with the database.
- Verified that the deployment status shows **“Deployment is in progress”**, indicating successful initiation.
- This step sets up the backend database infrastructure required for further data analysis and reporting.



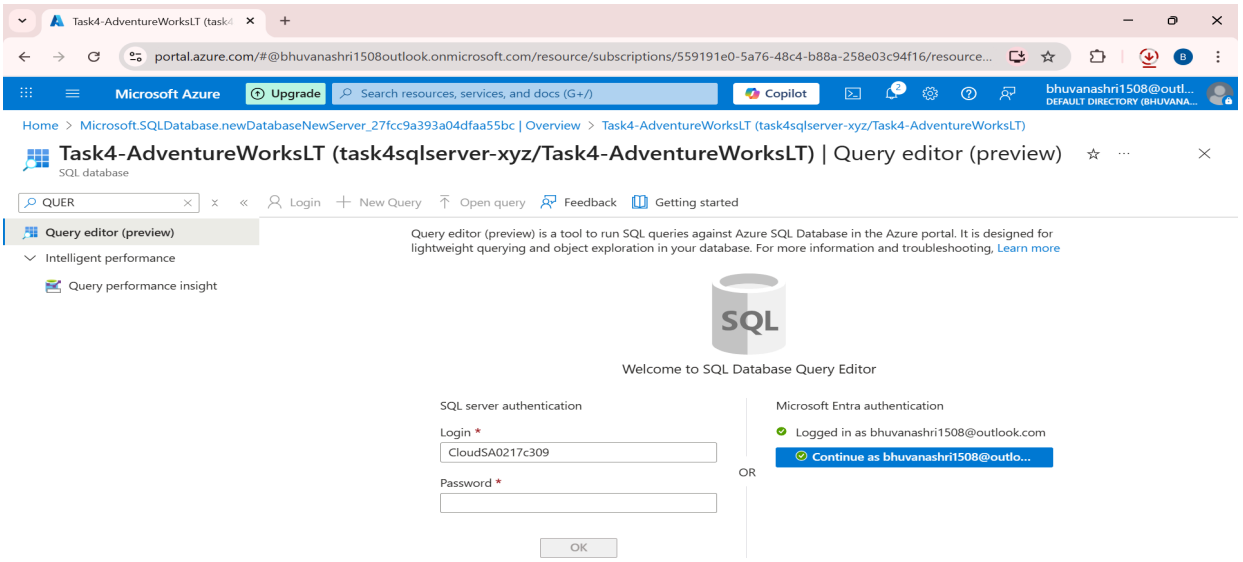
Step 2: Verifying Successful Azure SQL Database Deployment

- Monitored the deployment status of the Azure SQL Database in the Azure Portal.
- Confirmed that the deployment status shows **“Your deployment is complete”**.
- Verified the database was created under **Azure subscription 1** and resource group **myVM_group**.
- Noted the successful completion timestamp and correlation ID for audit/reference.



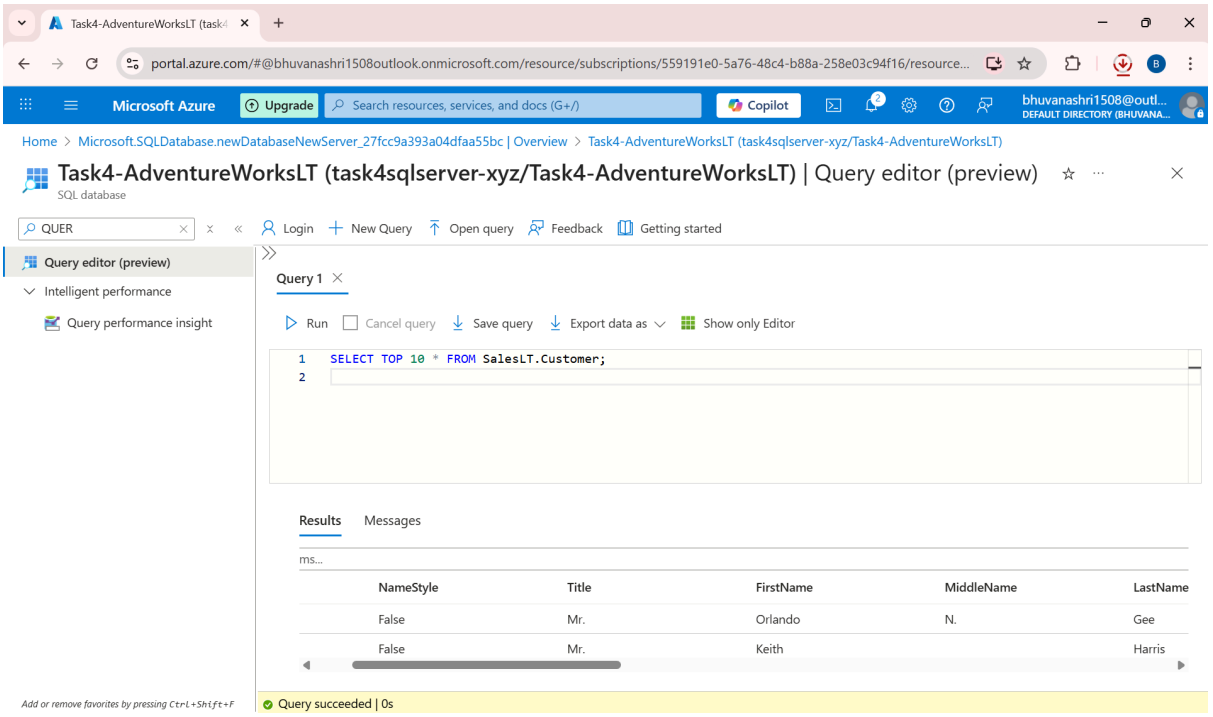
Step 3: Login to Azure SQL Database using Query Editor

- Opened **Azure SQL Database Query Editor (Preview)** from the Azure Portal.
- Selected the database **Task4-AdventureWorksLT**.
- Logged in using **SQL Server / Microsoft Entra authentication**.
- Successfully connected to the Azure SQL Database.



Step 4: Execute SQL Query and View Results

- Executed the SQL query **SELECT TOP 10 * FROM SalesLT.Customer;** using the Azure SQL Database Query Editor (Preview).
- The query was run successfully without any errors.
- Customer records from the SalesLT.Customer table were displayed in the results pane.

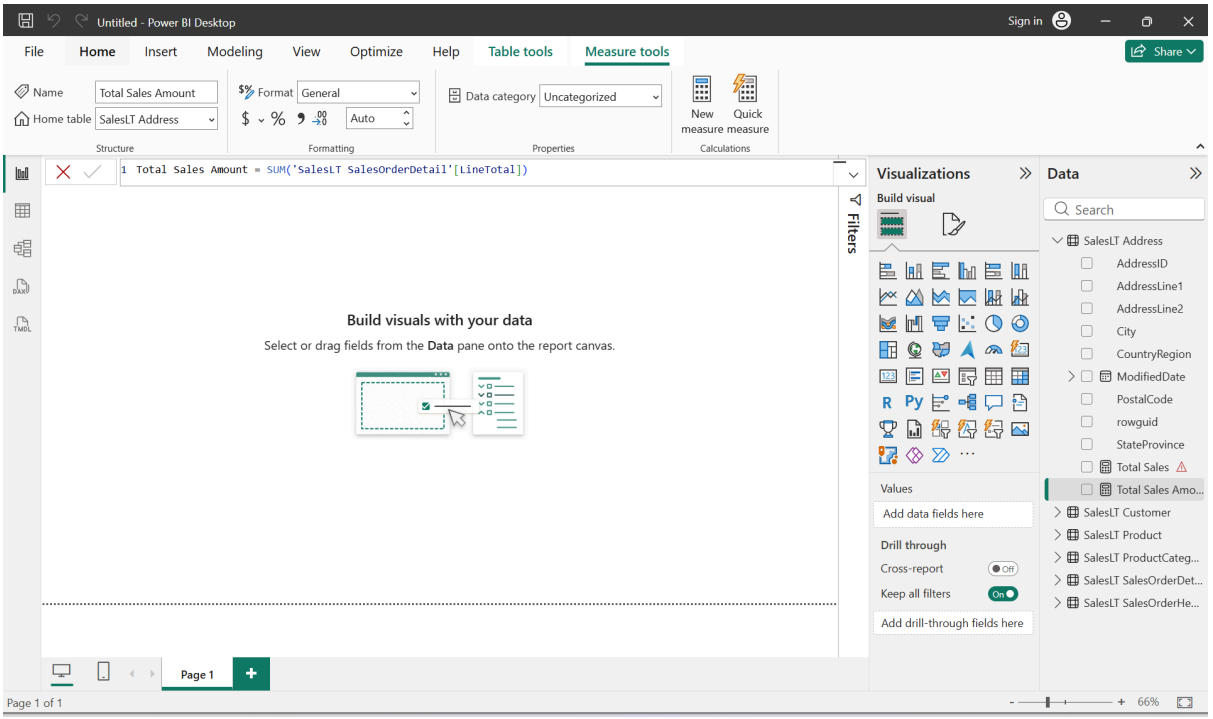


task4sqlserver-xyz.database.windows.net -- server name

Step 5: Create a Measure for Total Sales Amount in Power BI

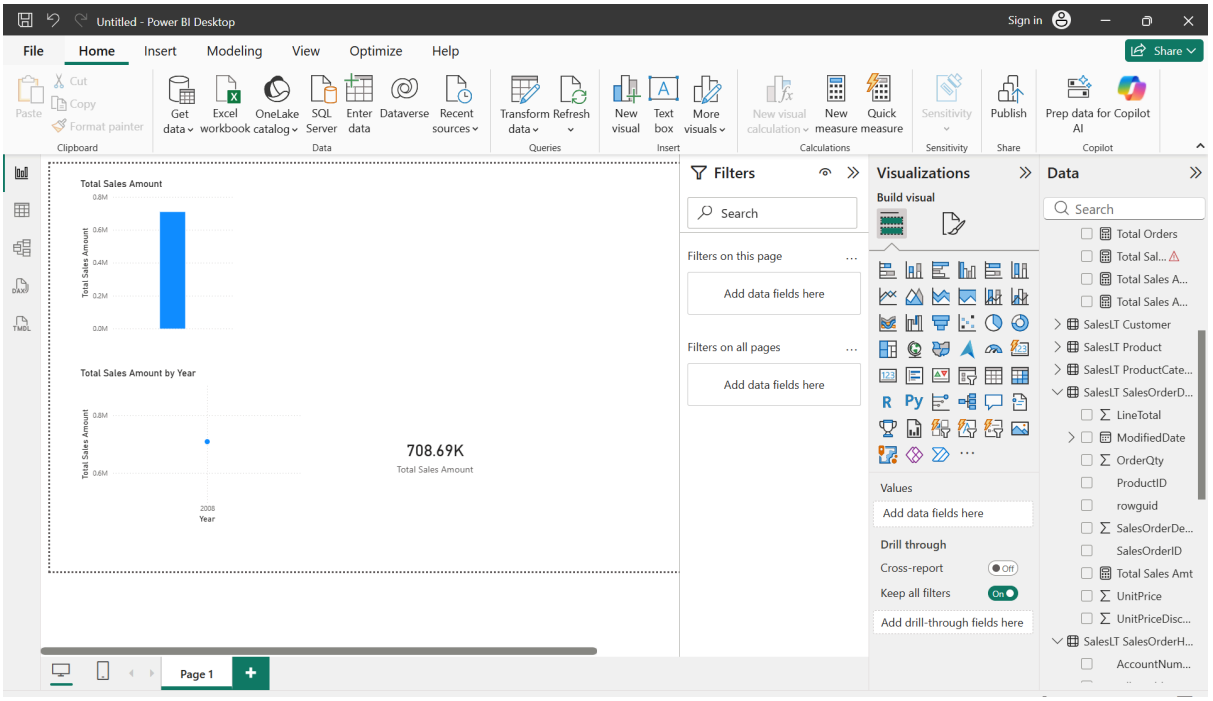
- Opened **Power BI Desktop** and loaded the AdventureWorksLT dataset.
- Created a new measure named **Total Sales Amount**.
- Defined the measure using the DAX expression
  Total Sales Amount = SUM(SalesLT_SalesOrderDetail[LineTotal])
- The measure calculates the overall sales value from the sales order details table.

- This measure is used for building sales-related visualizations.



Step 6: Build Visualizations Using the Total Sales Measure

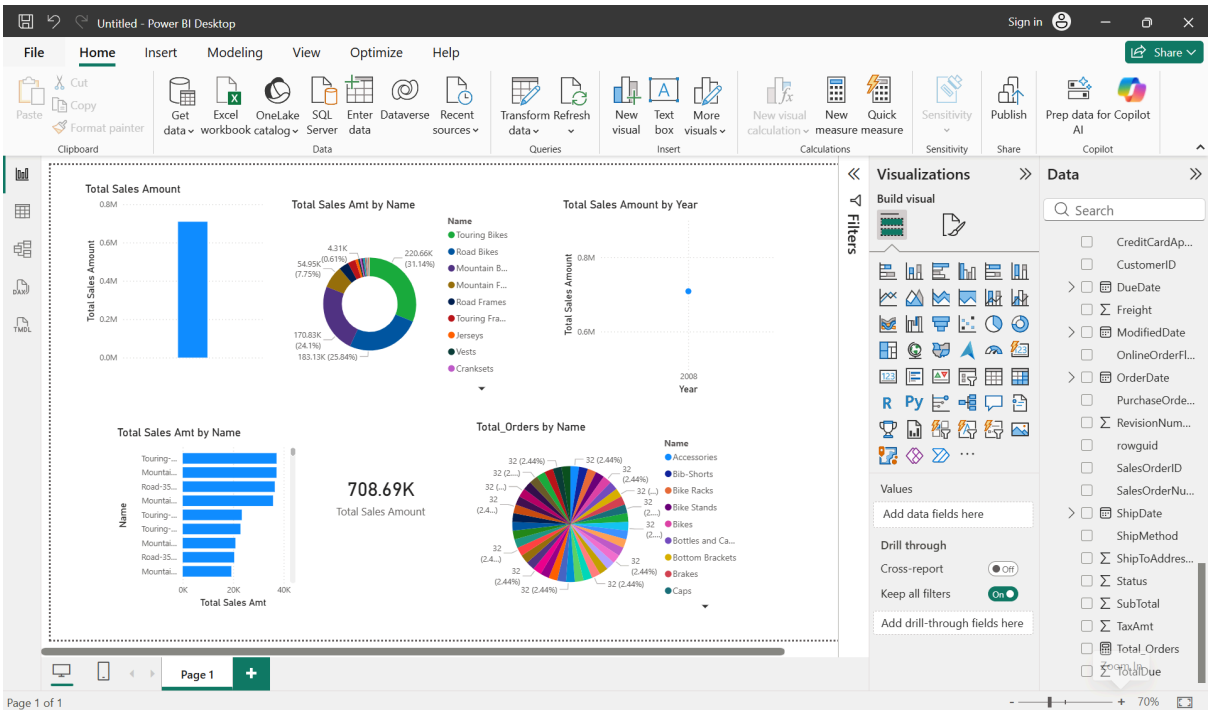
- Created visualizations using the **Total Sales Amount** measure.
- Displayed total sales using bar and card visuals.
- Visualized total sales trends (such as Total Sales Amount by Year).
- Verified that the visuals correctly reflect aggregated sales data.
- This confirms successful data modeling and visualization in Power BI.



Step 7: Create required visualizations in Power BI

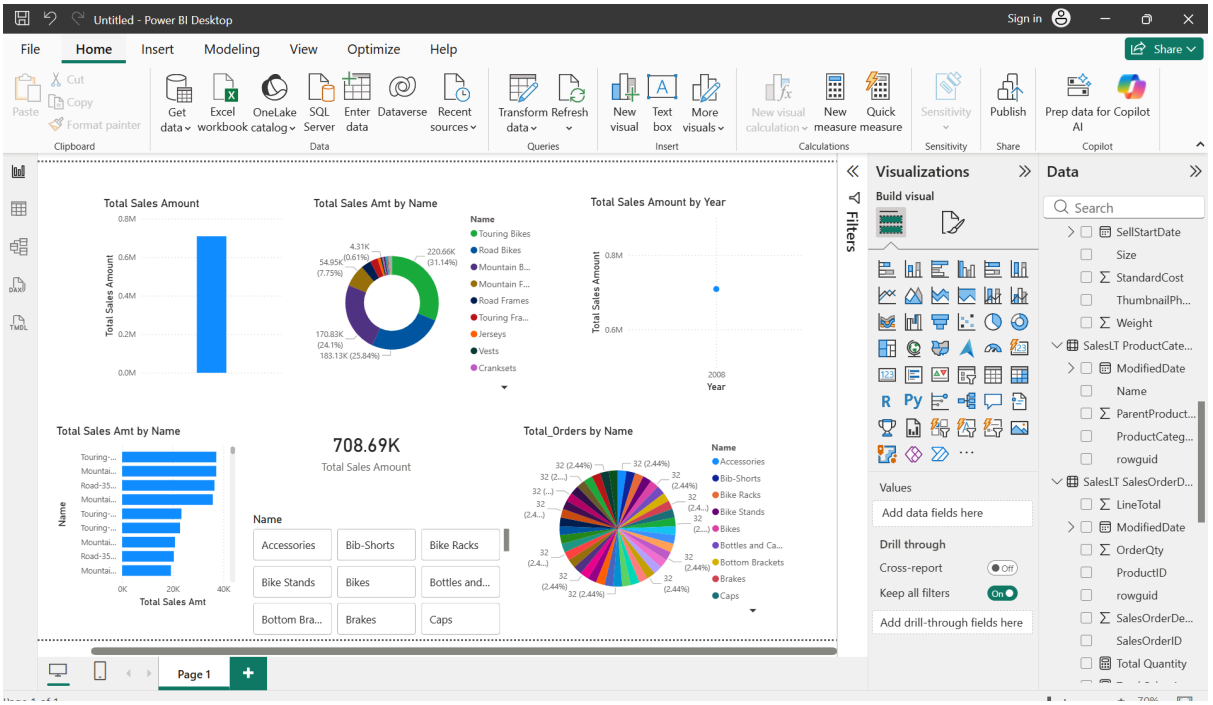
In Power BI Desktop, use the loaded AdventureWorksLT tables and created measures to build visuals. Add the following visuals to the report canvas:

- A column chart showing **Total Sales Amount**
- A donut chart showing **Total Sales Amount by Product Name**
- A bar chart showing **Total Sales Amount by Product Name**
- A scatter or line chart showing **Total Sales Amount by Year**
- A pie chart showing **Total Orders by Product Name**
- A card visual displaying **Total Sales Amount**



Step 8: Format and finalize the Power BI dashboard

Adjust the layout and formatting of all visuals for clarity and presentation. Apply titles, legends, and labels to each chart. Align visuals neatly on the canvas and ensure consistent colors and sizing. Add slicers if required (for example, Product Name or Category) to enable interactive filtering. Save the Power BI report file (.pbix) after verifying that all visuals display correct data.



CONCLUSION

This project successfully demonstrates how cloud-based databases and business intelligence tools can be integrated to analyze and visualize business data effectively. Azure SQL Database enables secure and scalable data storage, while Power BI transforms the data into meaningful insights through interactive dashboards.

By applying SQL querying, data modeling, and visualization techniques, the project provides clear insights into sales trends and product performance. Overall, the project highlights the importance of cloud analytics in modern business environments and shows how Azure and Power BI can support informed decision-making.