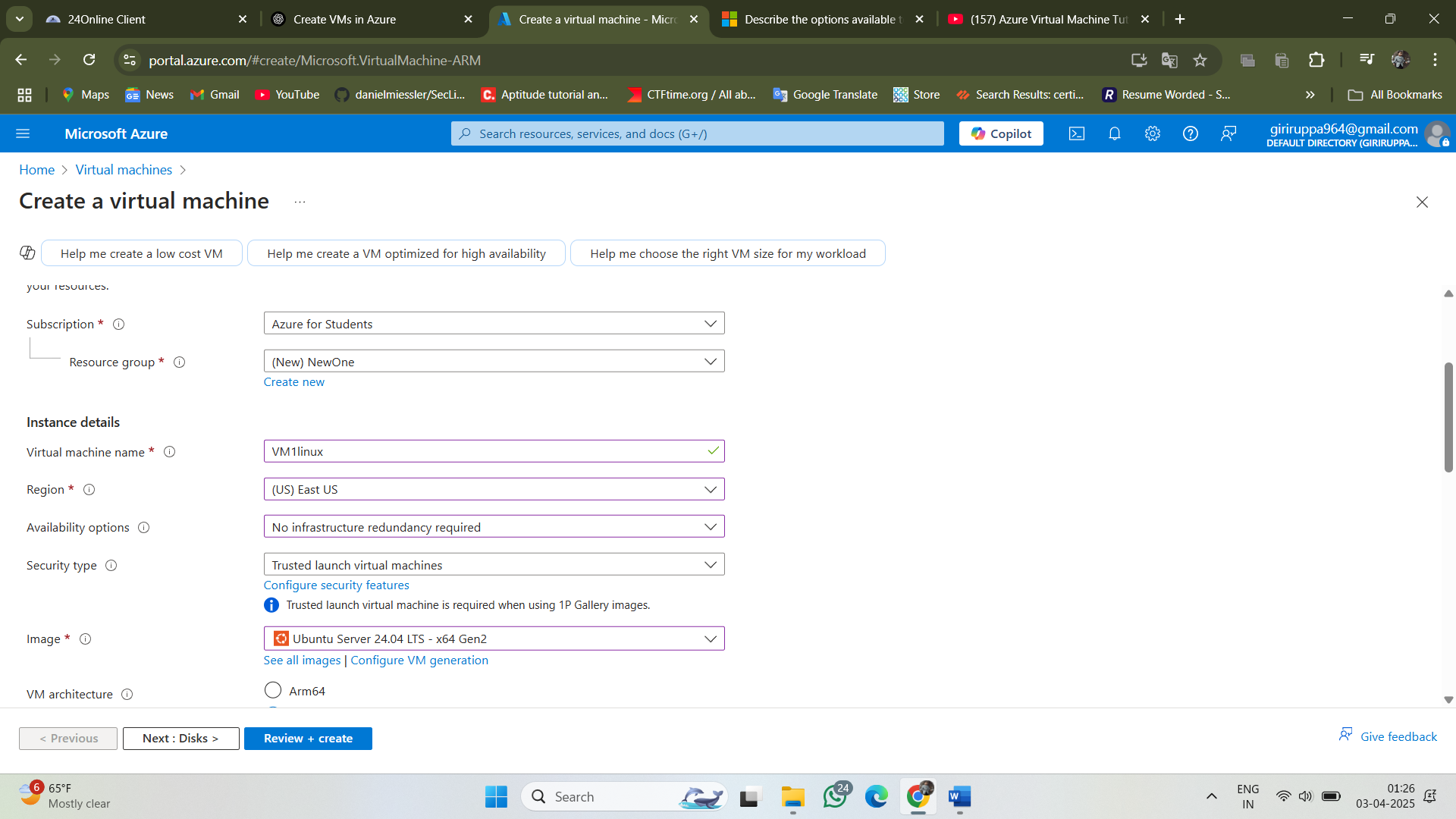
**CREATING VIRTUAL MACHINE IN AZURE**

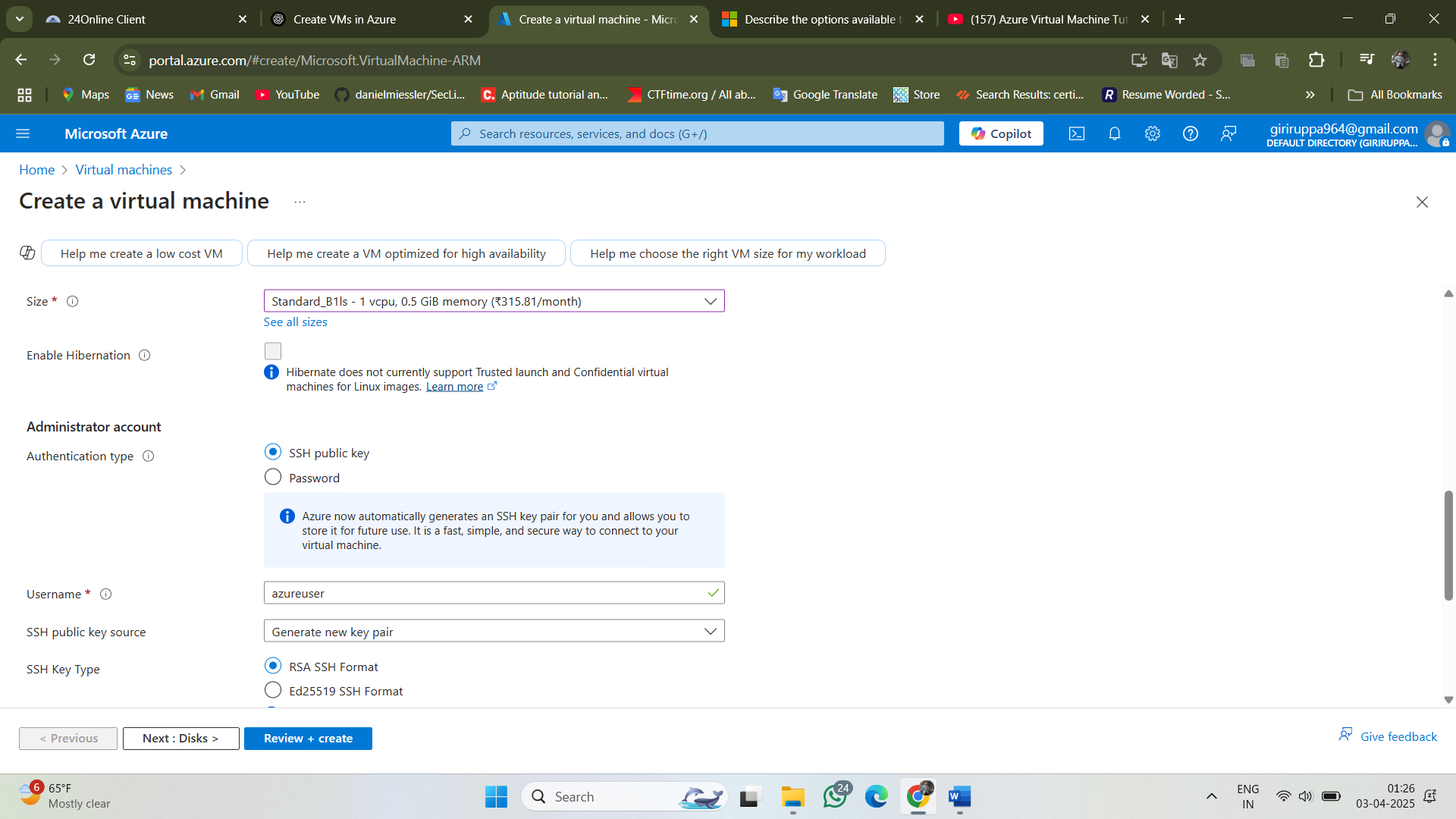
**Method 1: Using Azure Portal (GUI)**

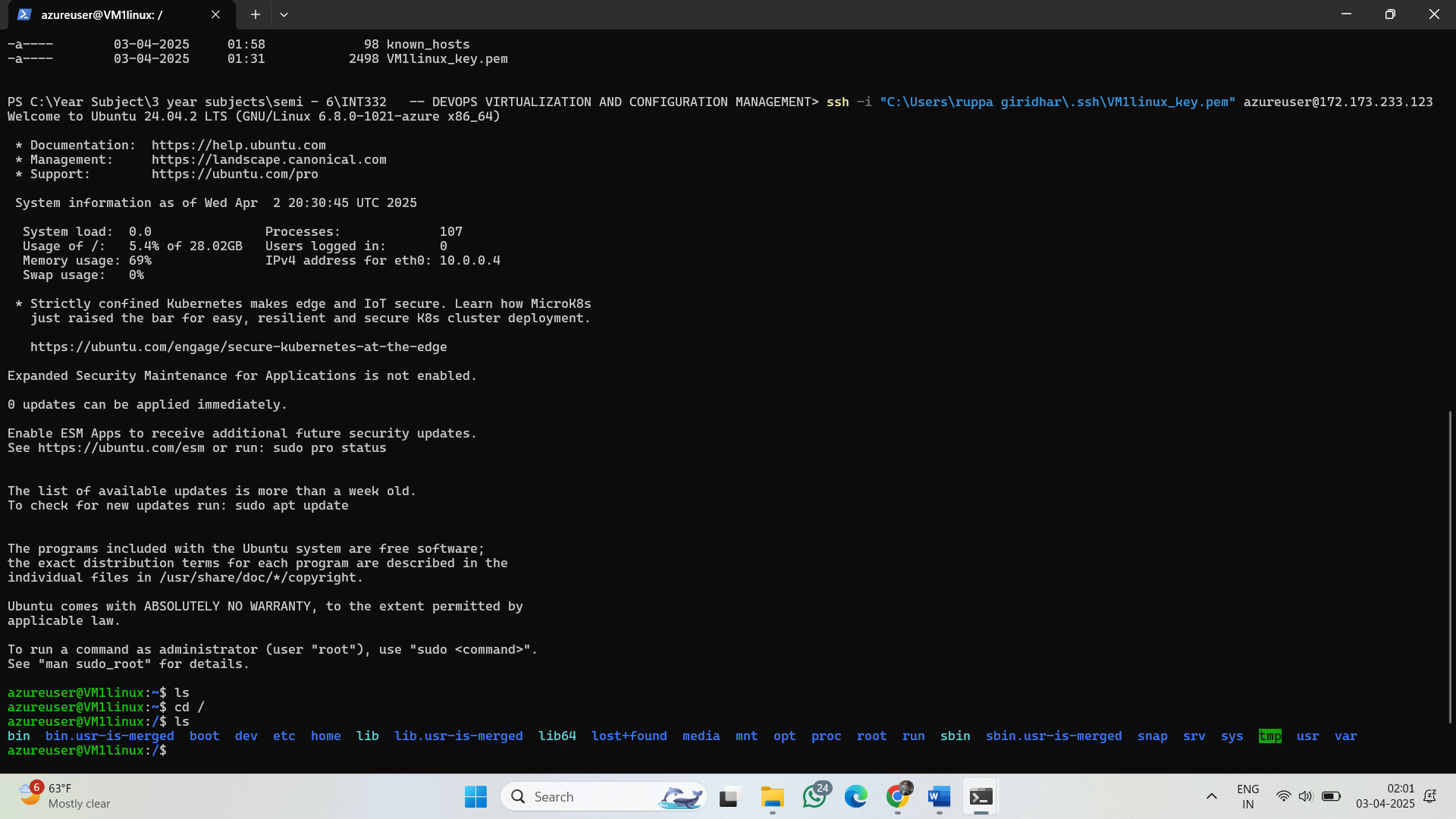
1. **Sign in to Azure Portal**
   * Go to [Azure Portal](https://portal.azure.com/)
   * Sign in with your credentials.
2. **Create the First Virtual Machine**
   * Click on **"Create a resource"** > **"Virtual Machine"**
   * Configure the following:
     + **Subscription**: Select your subscription – In my case azure for students
     + **Resource Group**: Create a new one or use an existing – mine is new one.
     + **Virtual Machine Name**: e.g., VM1
     + **Region**: Choose a region (e.g., East US) – better to choose the recommended one
     + **Image**: Choose an OS (e.g., Ubuntu, Windows Server) – I choose the Linux OS
     + **Size**: Select an appropriate VM size – select the size with respectively and check the cost per hour / per month.
     + **Administrator Account**: Set username & password/SSH key(better to choose the key ) - After choose ssh it will download one file in your system .pem extension . This private key will use up coming steps. Like when your connecting to the Virtual machine in windows or Linux or mac.
     + **Networking**: Choose/Create a Virtual Network

**For beginners(practice purpose) we need to assign the all options . But it required in industry (better to choose the best pratices)**

* + Click **Next** and configure other settings (Storage, Monitoring, etc.), then click **Review + Create**
  + Click **Create** and wait for deployment.







**Method 2: Using Azure CLI (Command Line)**

1. Open **Azure Cloud Shell** or run these commands from your local machine (if Azure CLI is installed).
2. **Login to Azure**

az login

1. **Create a Resource Group** (If not already created)

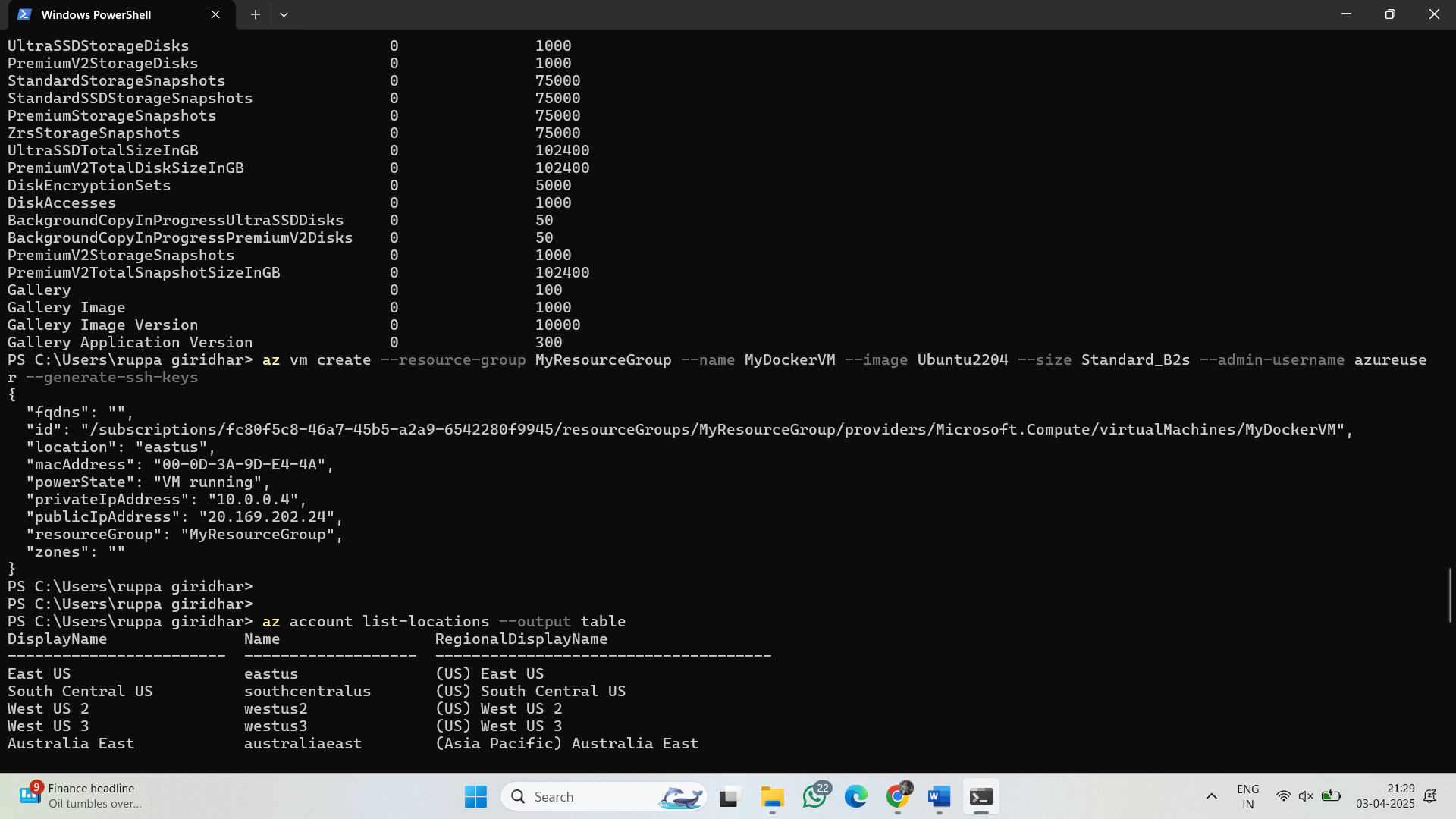
az group create --name MyResourceGroup --location eastus

1. **Create the First VM (VM1)**

az vm create --resource-group MyResourceGroup --name VM1 --image UbuntuLTS --admin-username azureuser --generate-ssh-keys --size Standard\_B1s

1. **Check the Running VMs**

az vm list --resource-group MyResourceGroup --output table



**Method 3: Using Terraform (Infrastructure as Code)**

**Using Terraform (Infrastructure as Code) in a Real-World Scenario**

**Scenario: Scaling Infrastructure for Black Friday Sales**

Black Friday and other major sales events, like Amazon’s Great Indian Festival or Flipkart’s Big Billion Days, create massive spikes in user traffic. To handle these spikes, cloud infrastructure needs to scale dynamically.

**Challenges Faced by E-Commerce Companies:**

1. **Sudden Traffic Surge:** Millions of users visit the website simultaneously, causing a huge load on servers.
2. **Performance Issues:** If not managed well, slow page loading or server crashes can lead to revenue loss.
3. **Cost Optimization:** Companies don’t want to pay for expensive infrastructure when traffic is low.
4. **Rapid Scaling:** Infrastructure must scale up before the event and scale down afterward to optimize costs.

**Solution: Using Terraform for Automated Infrastructure Scaling**

Terraform, an Infrastructure as Code (IaC) tool, allows companies to **provision, manage, and scale** their cloud infrastructure dynamically. Here’s how it works:

**Step-by-Step Implementation using Terraform**

1. **Defining Cloud Resources:**
   * Companies use Terraform to define their cloud infrastructure in code (AWS, Azure, GCP).
   * Example: Amazon or Flipkart can create a Terraform script to deploy multiple EC2 instances, databases, and load balancers.
2. **Auto Scaling for Peak Traffic:**
   * Using **Terraform + AWS Auto Scaling**, they define rules to scale instances automatically when CPU utilization exceeds 70%.
   * **Example:** If 500,000 users suddenly log in, more EC2 instances launch to handle traffic.
3. **Load Balancer for Even Traffic Distribution:**
   * Terraform provisions an **AWS Elastic Load Balancer (ELB)** to distribute incoming traffic across multiple servers.
4. **Infrastructure Monitoring & Logs (Observability):**
   * Terraform deploys **CloudWatch or Splunk** to monitor real-time traffic and detect failures.
   * If a server crashes, Terraform auto-recreates it.
5. **Database Scaling:**
   * Using **Terraform + AWS RDS Auto Scaling**, the database scales up during high traffic and scales down afterward.
6. **Rollback & Disaster Recovery:**
   * Terraform allows rolling back changes instantly if any issue occurs.
   * Example: If a faulty update causes server failure, Terraform can redeploy the last stable version.

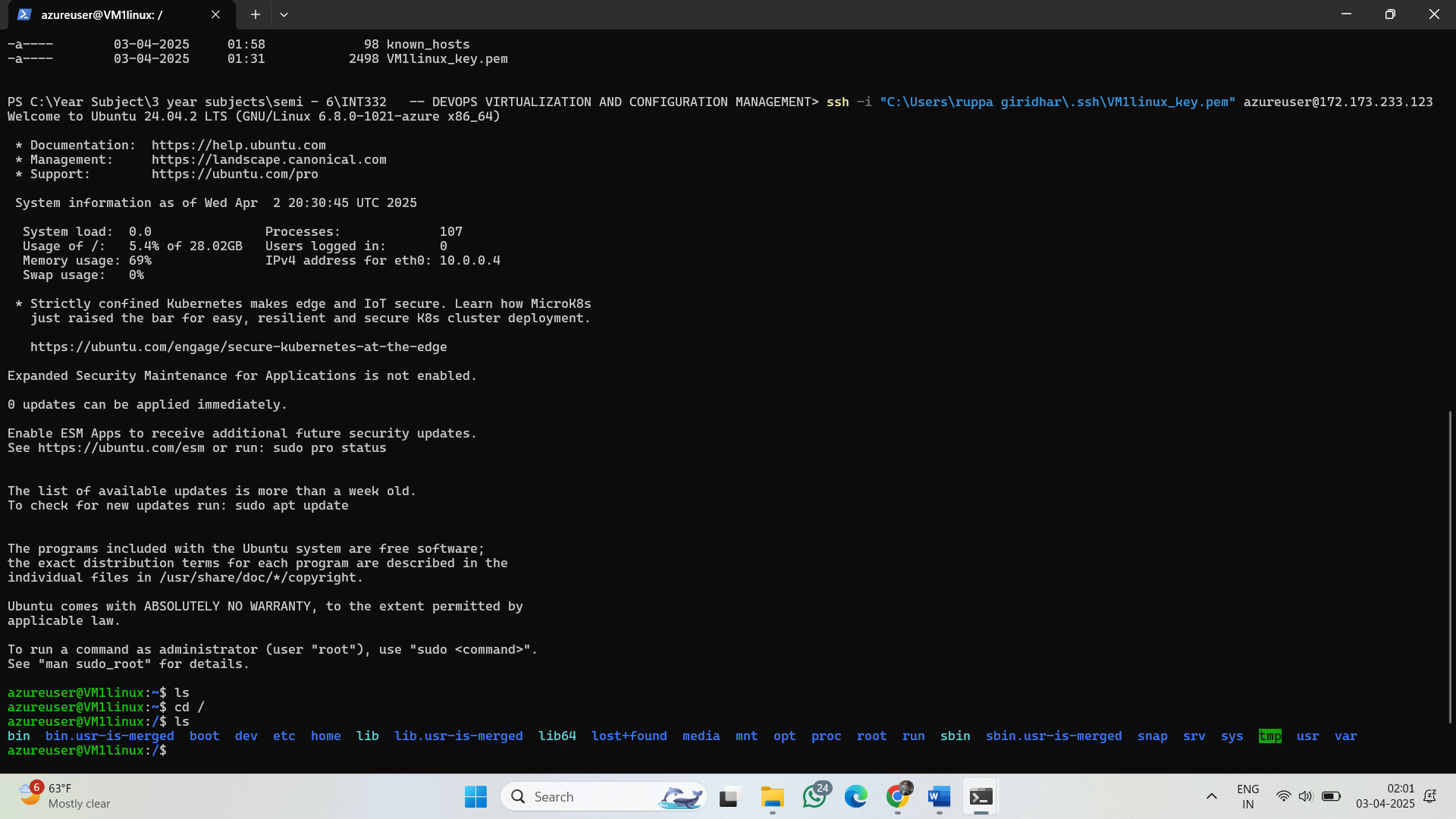
By implementing Terraform, companies like **Amazon and Flipkart** can handle Black Friday traffic spikes efficiently without manual intervention, ensuring a seamless shopping experience for millions of customers.

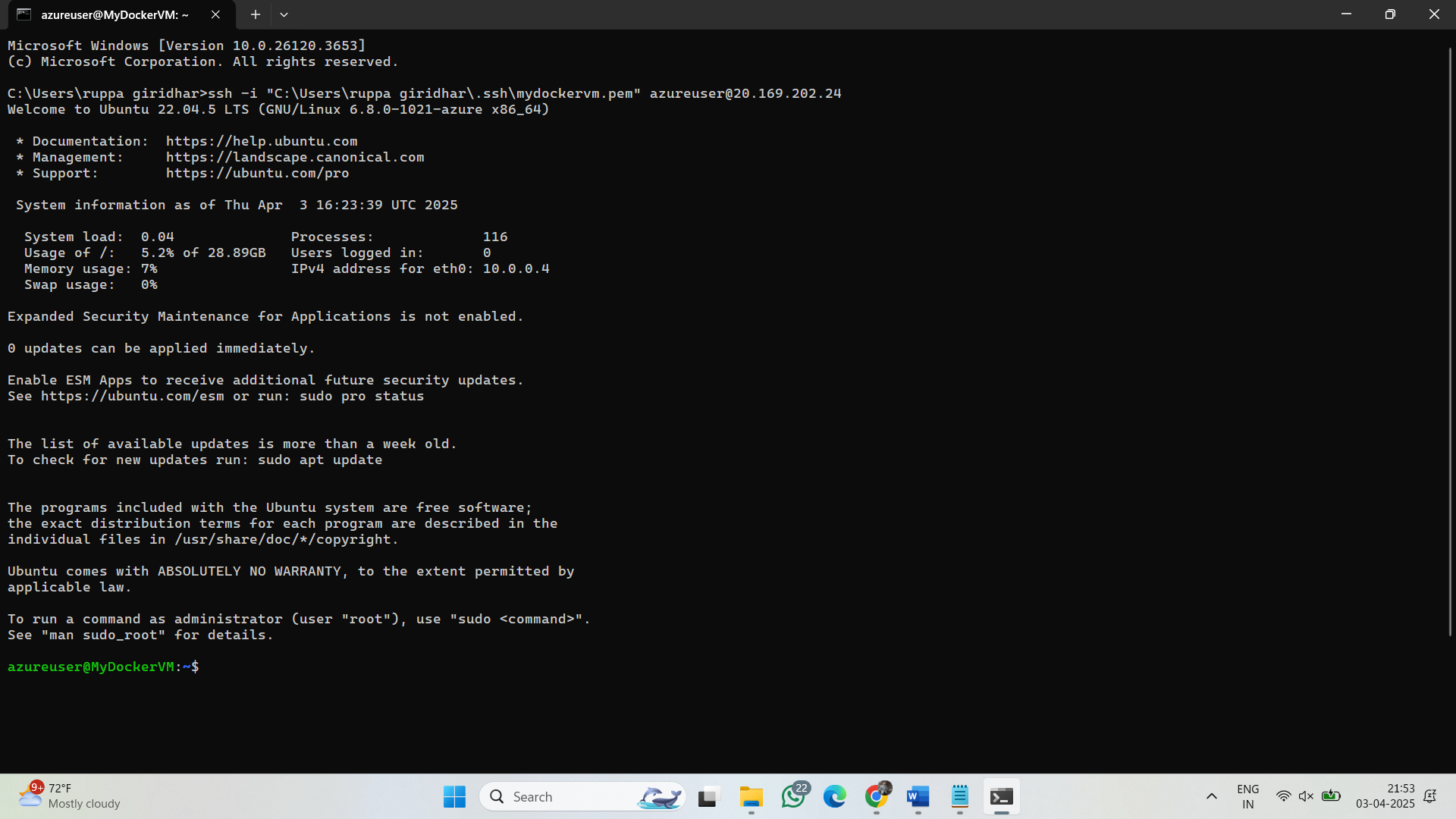
**2)LOGIN THE VM IN YOUR HOST MACHINE(WINDOWS)**

Through your PowerShell you can login your Virtual machine (Linux) in windows

Needs – public IP , RSA key,username

* + - 1. Vm1 – creation by GUI
      2. Vm2 -- creation by CMD





**3)After logging into your Azure Ubuntu VM, follow these steps to install Docker:**

**🔹 Step 1: Update the System**

Run the following command to update the package list and install required dependencies:

sudo apt update && sudo apt upgrade -y

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

**🔹 Step 2: Add Docker’s Official GPG Key**

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

**🔹 Step 3: Add Docker Repository**

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Go through above 3 steps first then below 2 steps



**🔹 Step 4: Install Docker**

sudo apt update

sudo apt install -y docker-ce docker-ce-cli containerd.io

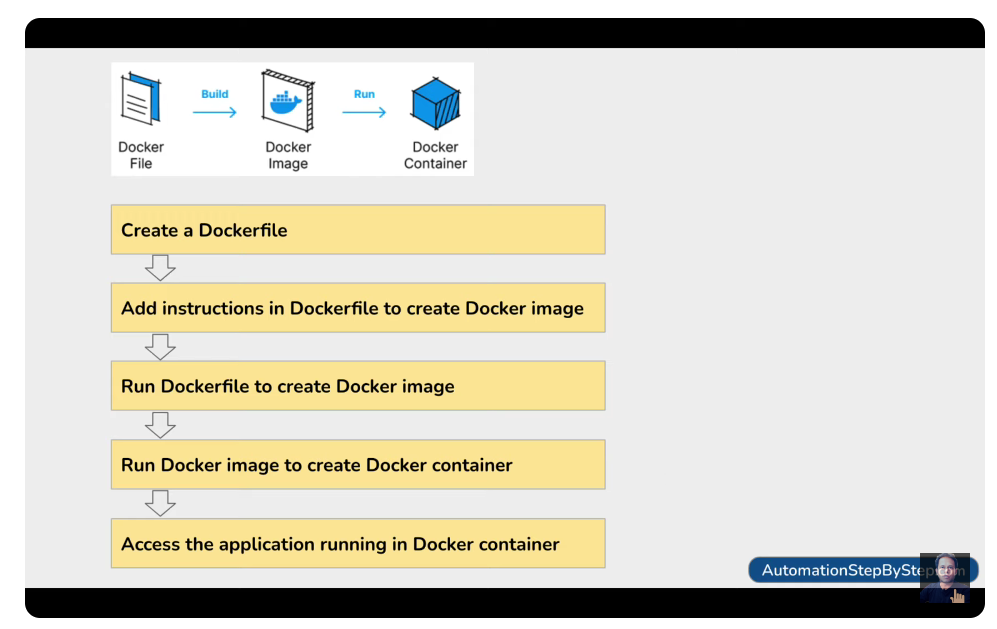
**🔹 Step 5: Verify Docker Installation**

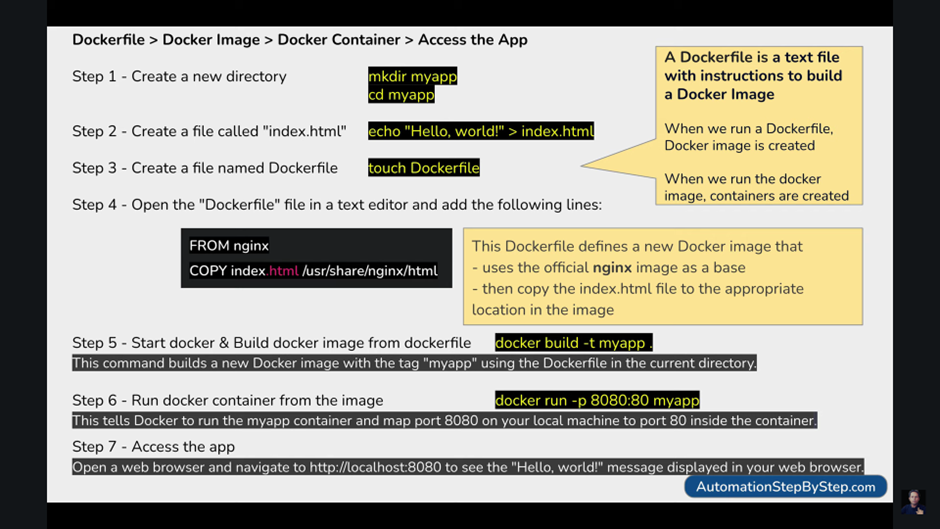
Check if Docker is installed and running:

docker --version

sudo systemctl status docker

**4) Docker:**





Above one is showing steps from creation to access

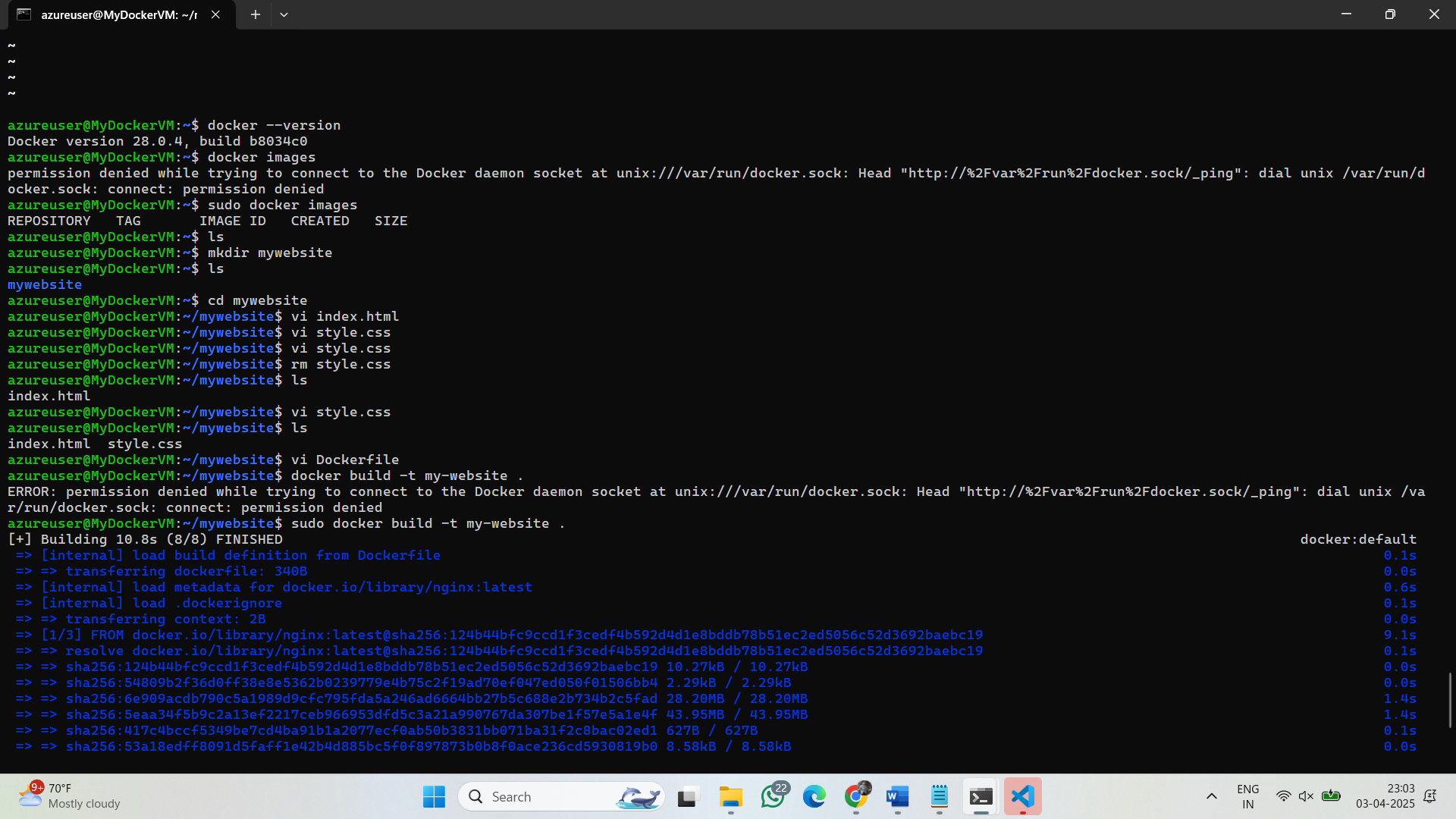
### **Steps to Run Your HTML-CSS Project in Docker and Access Remotely**

**Step 1: Create a Project Directory**

Ensure your project files are in a single directory, e.g., my-website/:

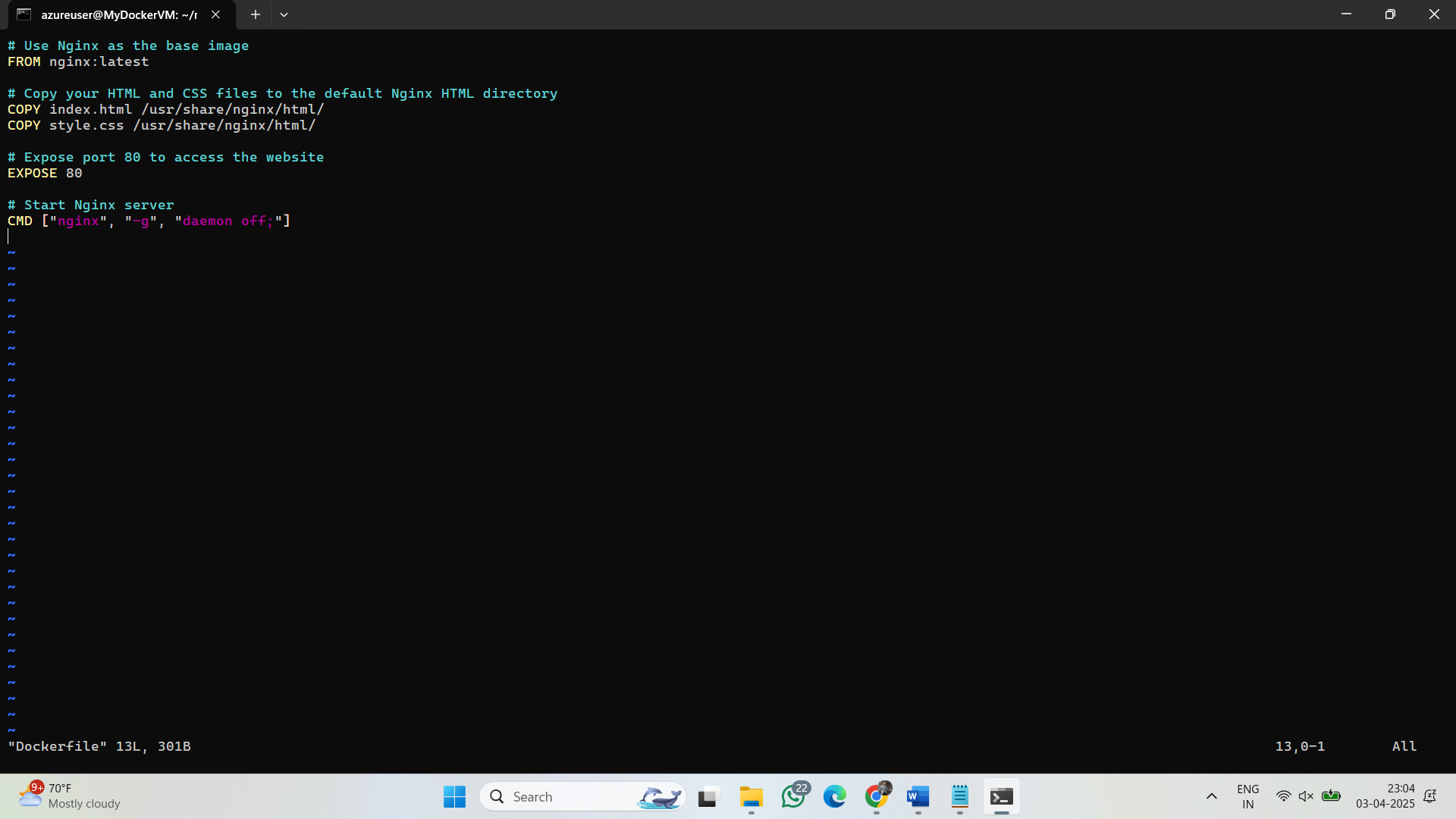
my-website/

│── index.html

│── style.css 

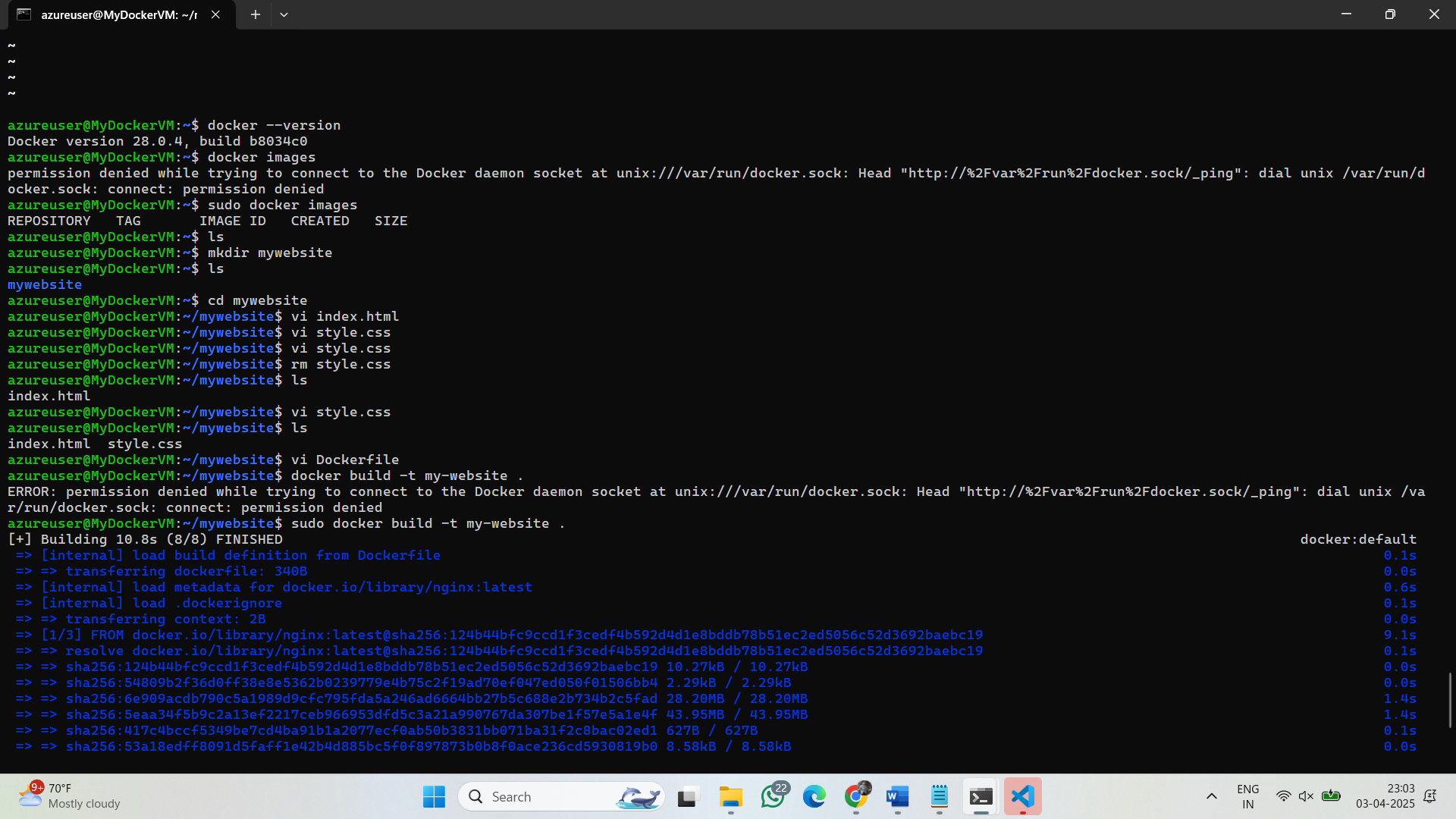
**Step 2: Create a Dockerfile**

Inside the my-website/ directory, create a file named **Dockerfile** with the following content:

dockerfile 

**Step 3: Build the Docker Image**

Navigate to your project directory and build the Docker image using this command:

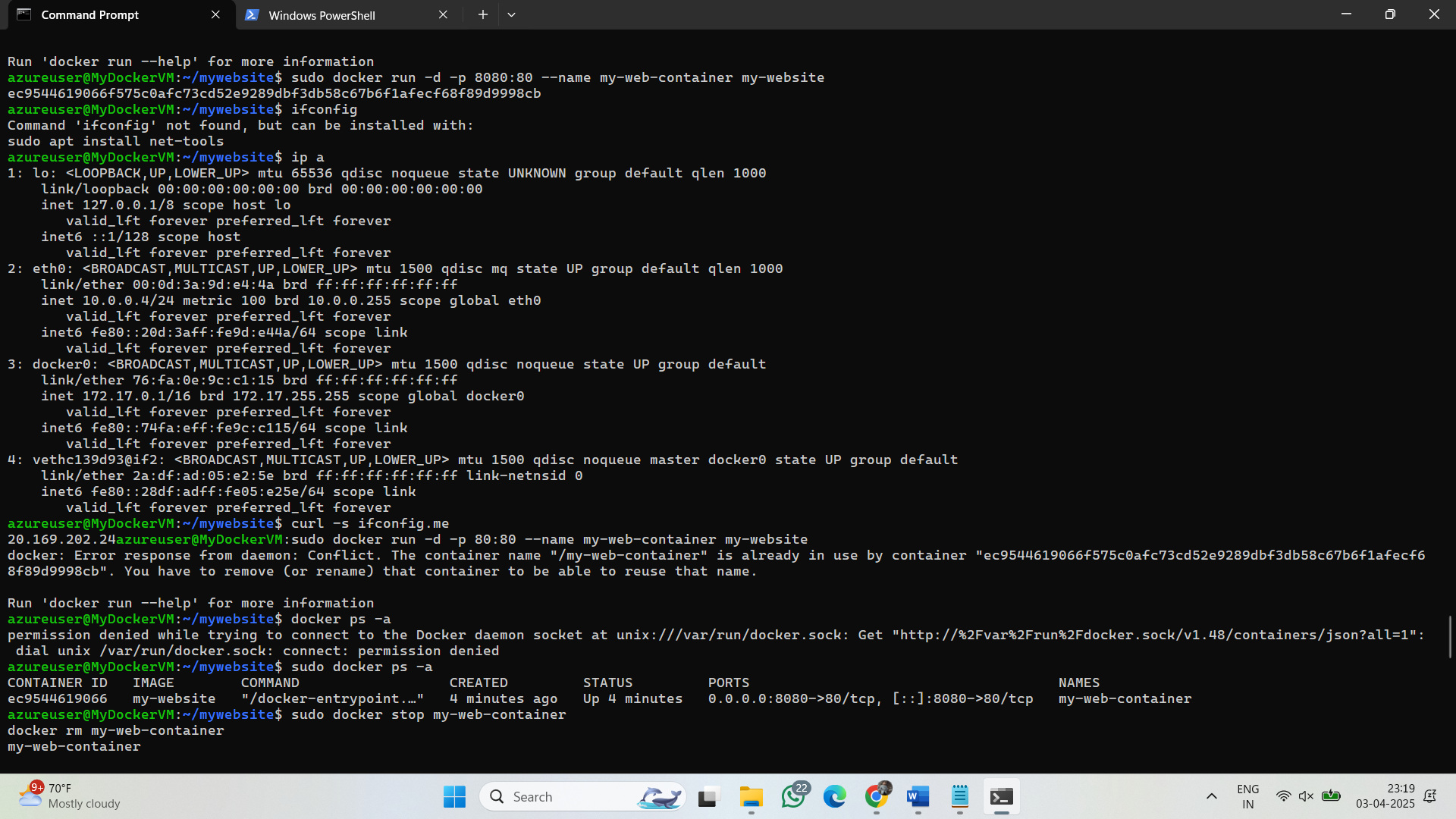
docker build -t my-website . 

**Step 4: Run the Docker Container**

Run the container and map port 80 of the container to port 8080 of your local machine:

docker run -d -p 8080:80 --name my-web-container my-website

Now, your website will be accessible on **http://localhost:8080**.



**Step 5: Access Remotely**

To access it remotely:

* If you're using **Docker on a cloud server (e.g., AWS, DigitalOcean, etc.)**, replace localhost with your **public IP**.
* If using Docker locally, find your system’s IP and access it using http://your-ip:8080.

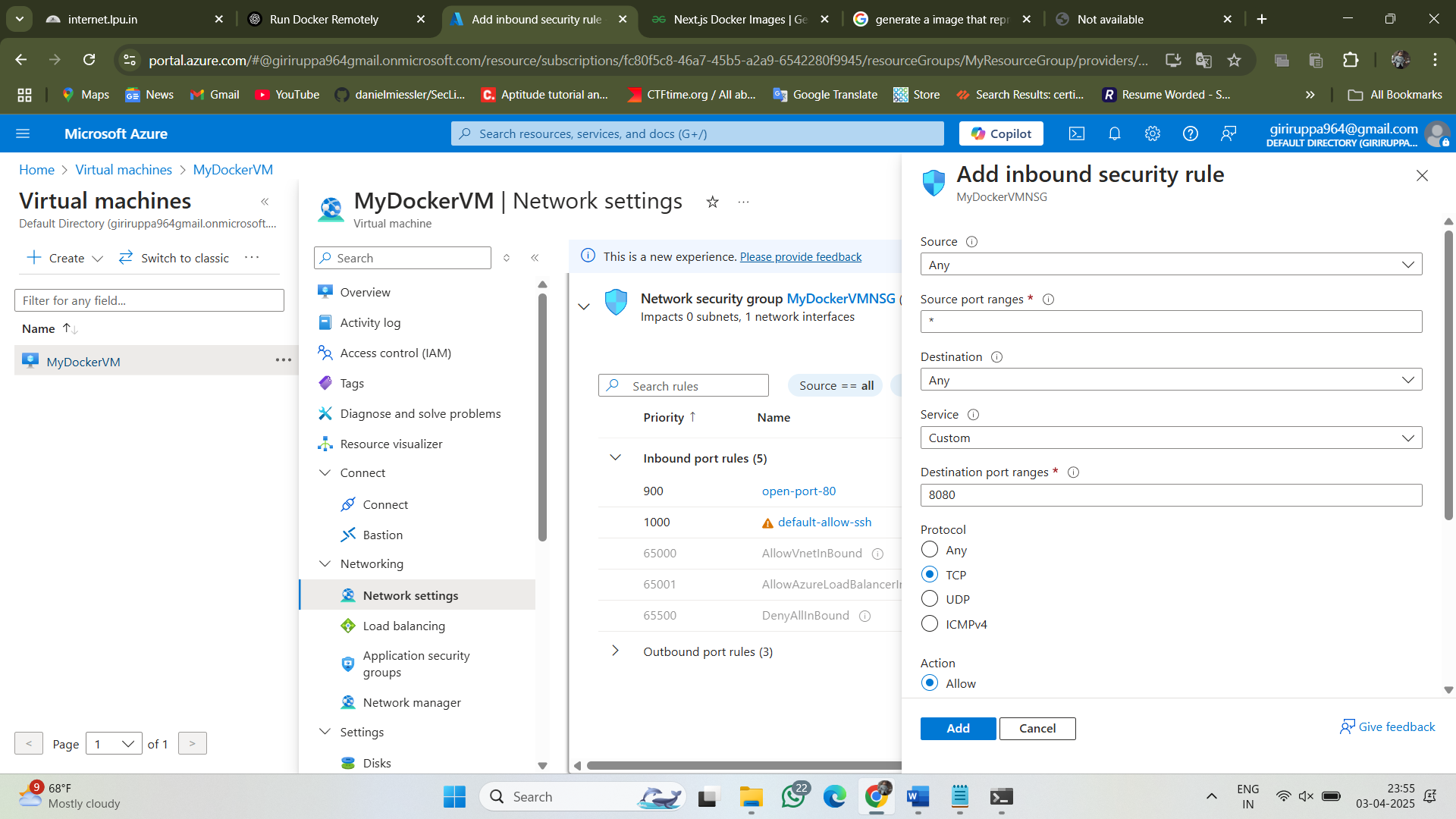
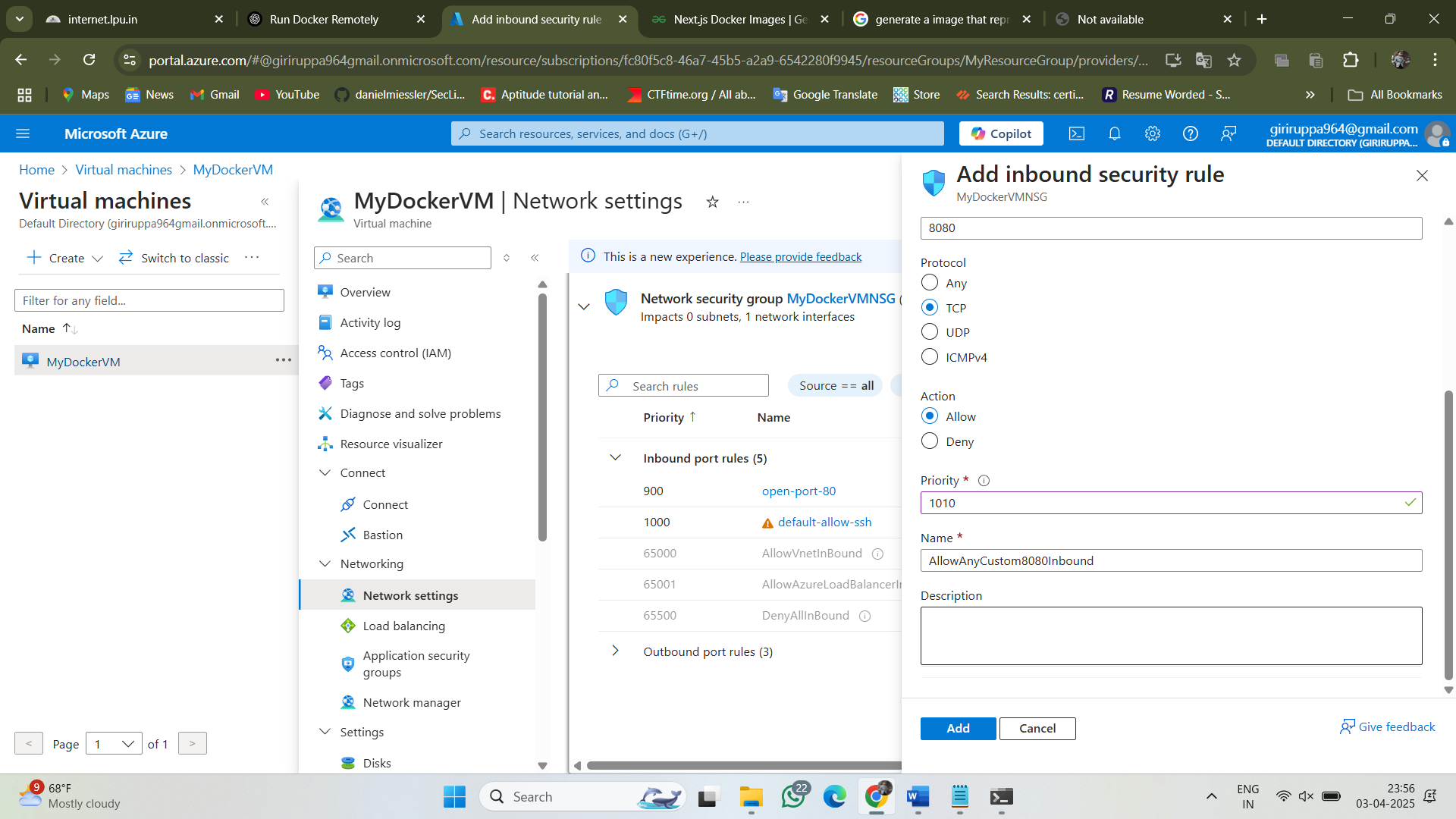
### Some issues like network releted I faced so encountered that in below

Note : If you not able to accessing through below steps  **Go to Azure Portal** → **Virtual Machines** → Click on **MyDockerVM**.

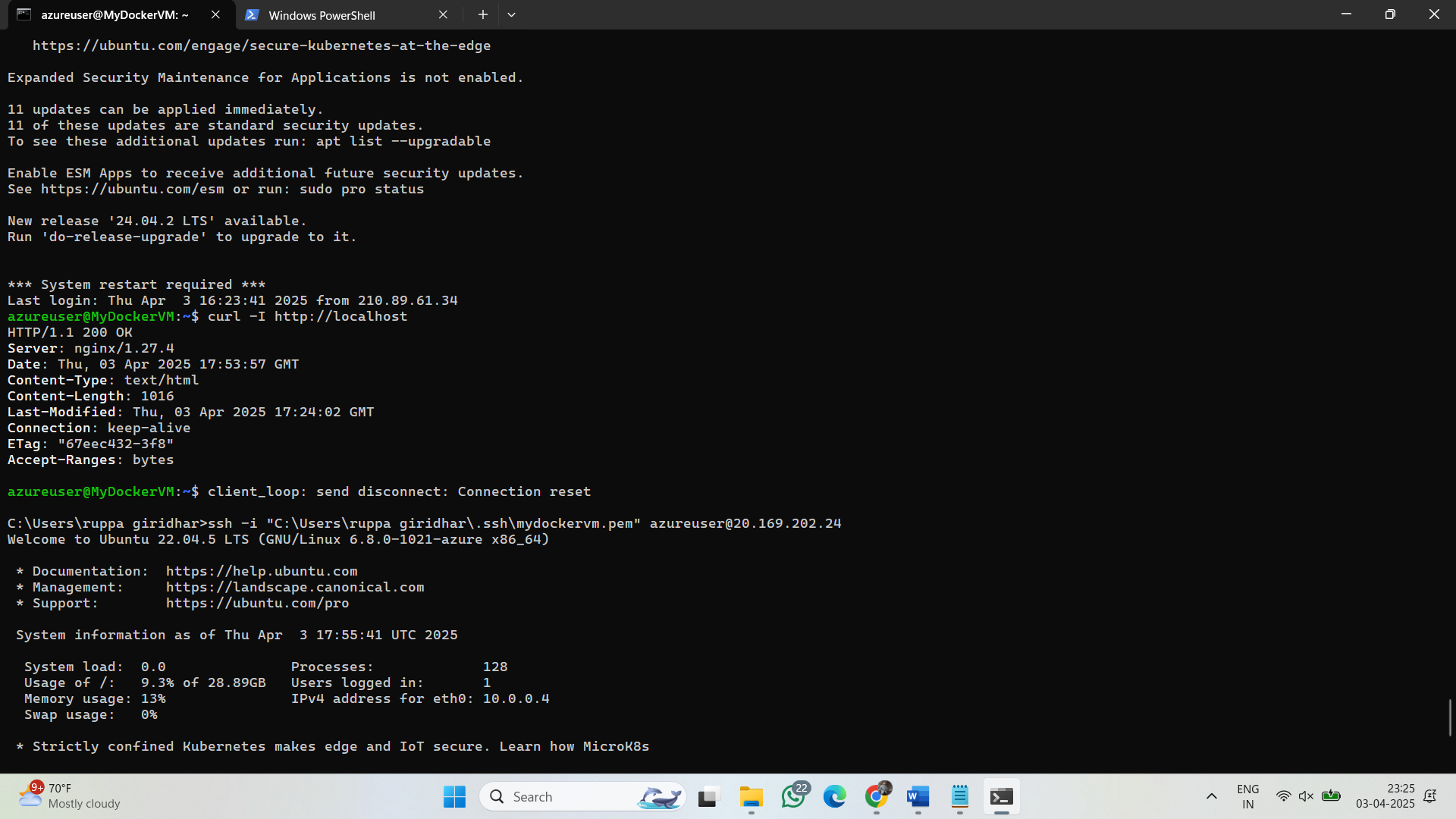
 **Open Networking** (left menu).

 **Check Inbound Rules** → Look for a rule allowing **port 80** (TCP).

 If no rule exists, **Add Inbound Rule**:

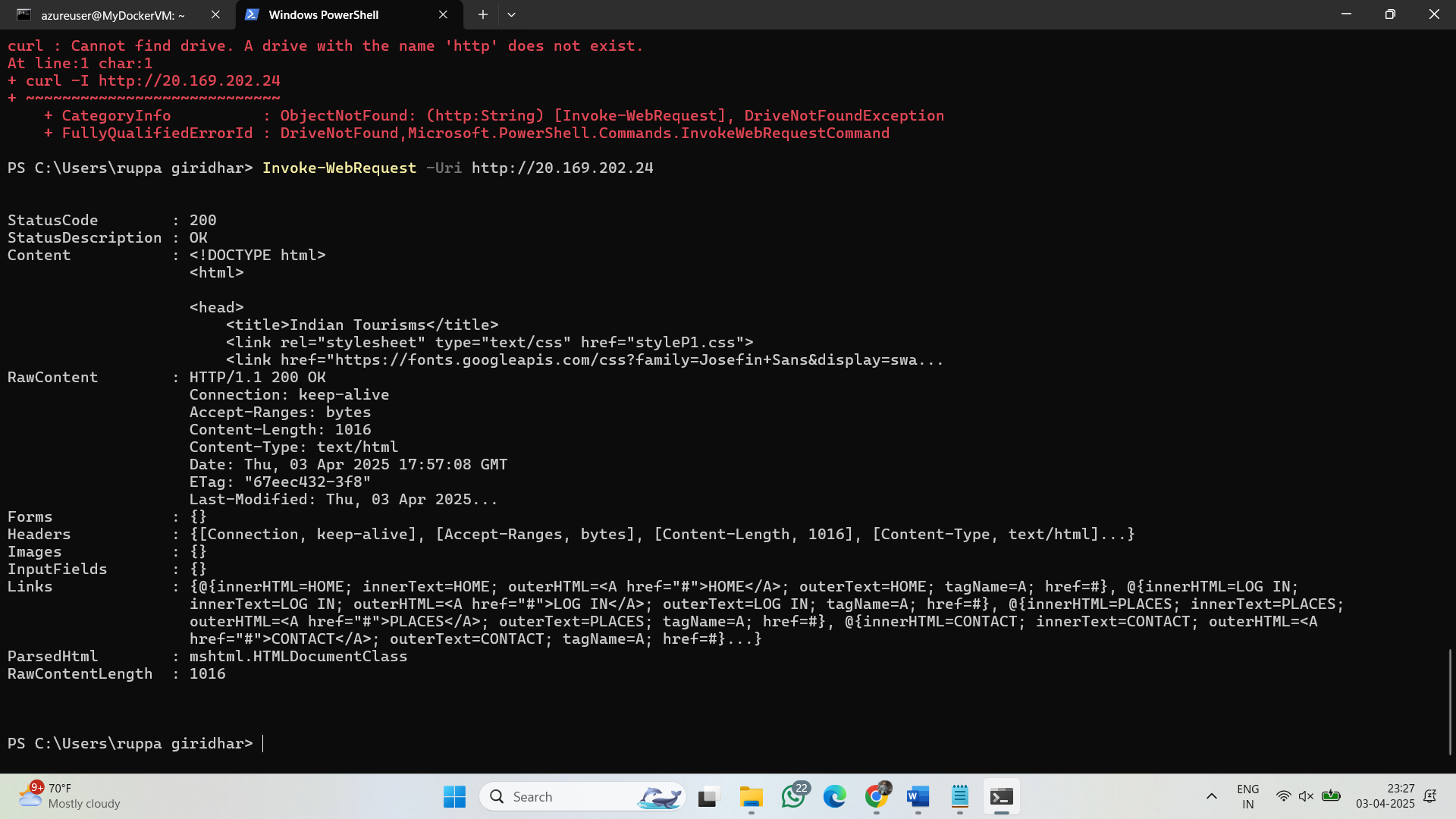
* **Source:** Any
* **Source port ranges:** \*
* **Destination:** Any
* **Destination port ranges:** **80**
* **Protocol:** TCP
* **Action:** Allow
* **Priority:** 1000
* **Name:** Allow-HTTP  
* **Locally** inside VM:

curl -I <http://localhost>



* **Remotely** from another device:

curl -I http://20.169.202.24



* **Browser access:**

http://20.168.202.24:8080

