**HOSTING STATIC WEBSITES ON AWS S3 AND EC2**

**S3 BUCKET STEPS**

**1. Create an S3 Bucket**

* **Go to AWS Management Console → S3 → Create bucket.**
* **Enter a globally unique bucket name (e.g., mywebsite-bucket).**
* **Choose a region (closest to your users).**
* **Leave default settings for now → click Create bucket.**

**2. Upload Website Files**

* **Open the bucket you just created.**
* **Click Upload → add your index.html, style.css, script.js, etc.**

**3. Enable Static Website Hosting**

* **Go to Properties tab of your bucket.**
* **Scroll to Static website hosting → click Edit.**
* **Select Enable.**
* **Enter:**
  + **Index document = index.html.**
  + **Error document = error.html (optional).**
* **Save changes.**

**👉 You’ll get a bucket website endpoint URL (like http://mywebsite-bucket.s3-website-us-east-1.amazonaws.com).**

**4. Set Public Read Access**

**By default, S3 blocks public access. To allow visitors to see your site:**

1. **Go to Permissions → Bucket Policy.**
2. **Add this policy (replace mywebsite-bucket with your bucket name):**

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Sid": "PublicReadGetObject",**

**"Effect": "Allow",**

**"Principal": "\*",**

**"Action": "s3:GetObject",**

**"Resource": "arn:aws:s3:::mywebsite-bucket/\*"**

**}**

**]**

**}**

1. **Save policy.**
2. **In Block Public Access settings, uncheck Block all public access (be careful, only for static website buckets).**

**5. Test the Website**

* **Copy the Website endpoint URL from the bucket Properties.**
* **Paste it in your browser.**
* **You should see your index.html load. 🎉**

**6. (Optional) Use Custom Domain with Route 53 & CloudFront**

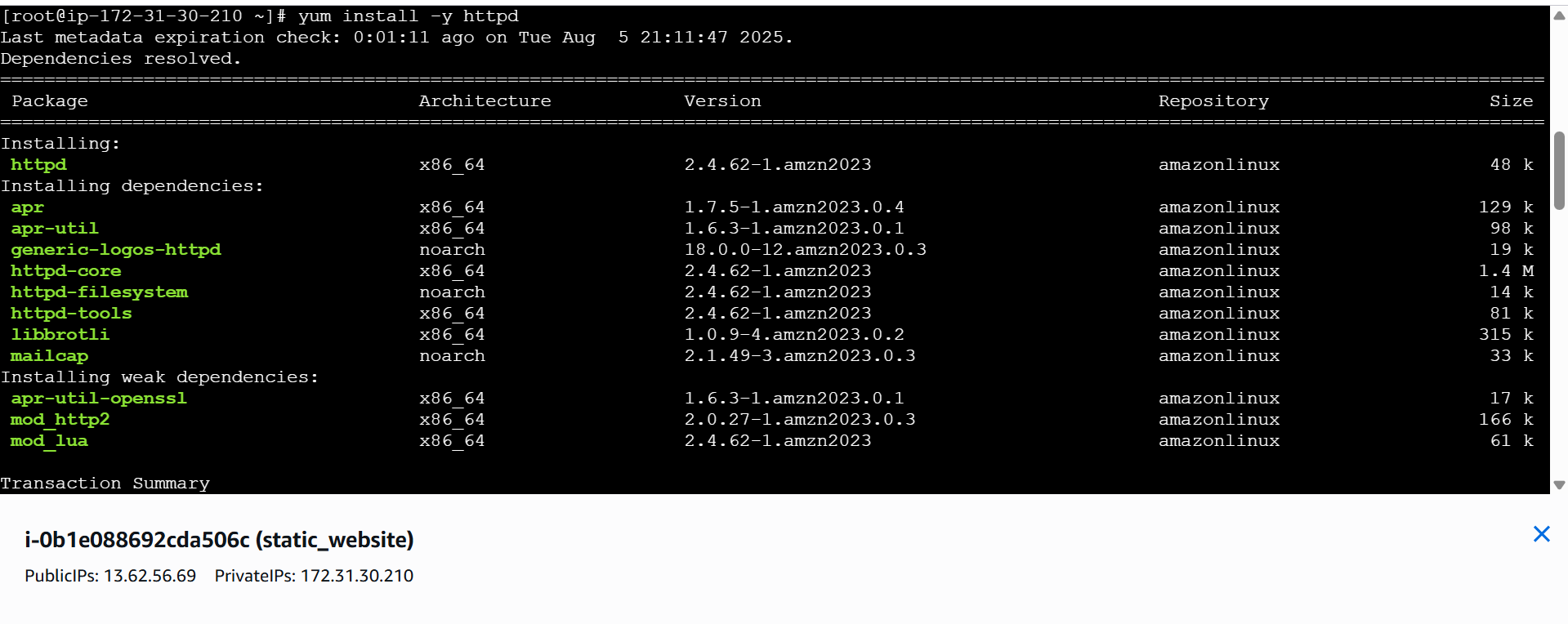
* **Register domain in Route 53.**
* **Point DNS records to your S3 static site.**
* **Use CloudFront for HTTPS (since S3 website endpoints don’t support HTTPS directly).**

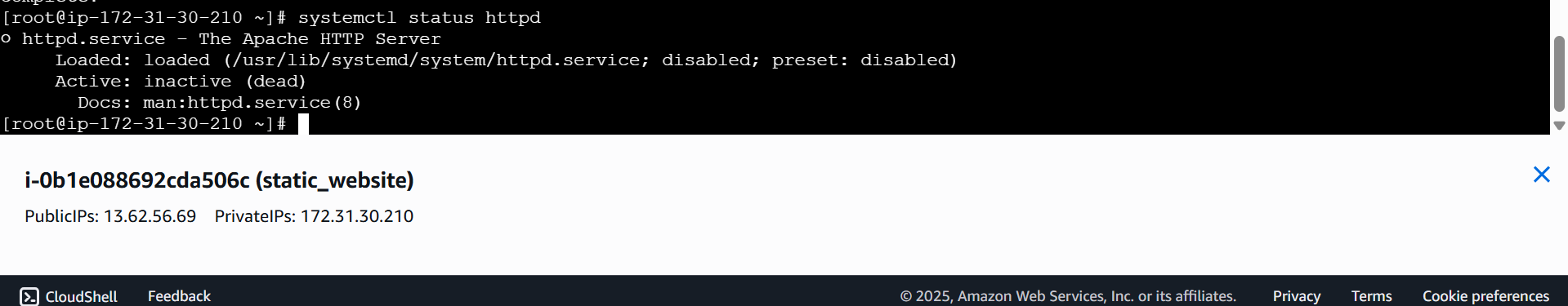
**EC2 BUCKET STEPS(PLEASE FOLLOW THE ASSESSMENT)**

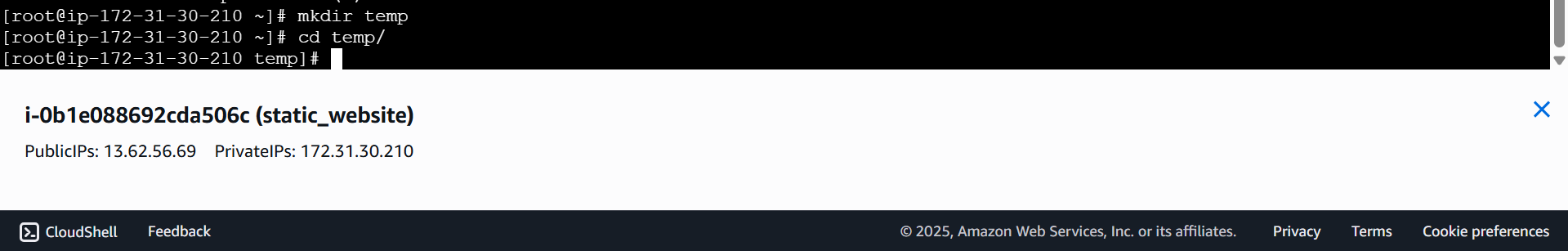
**1. Launch an EC2 Instance**

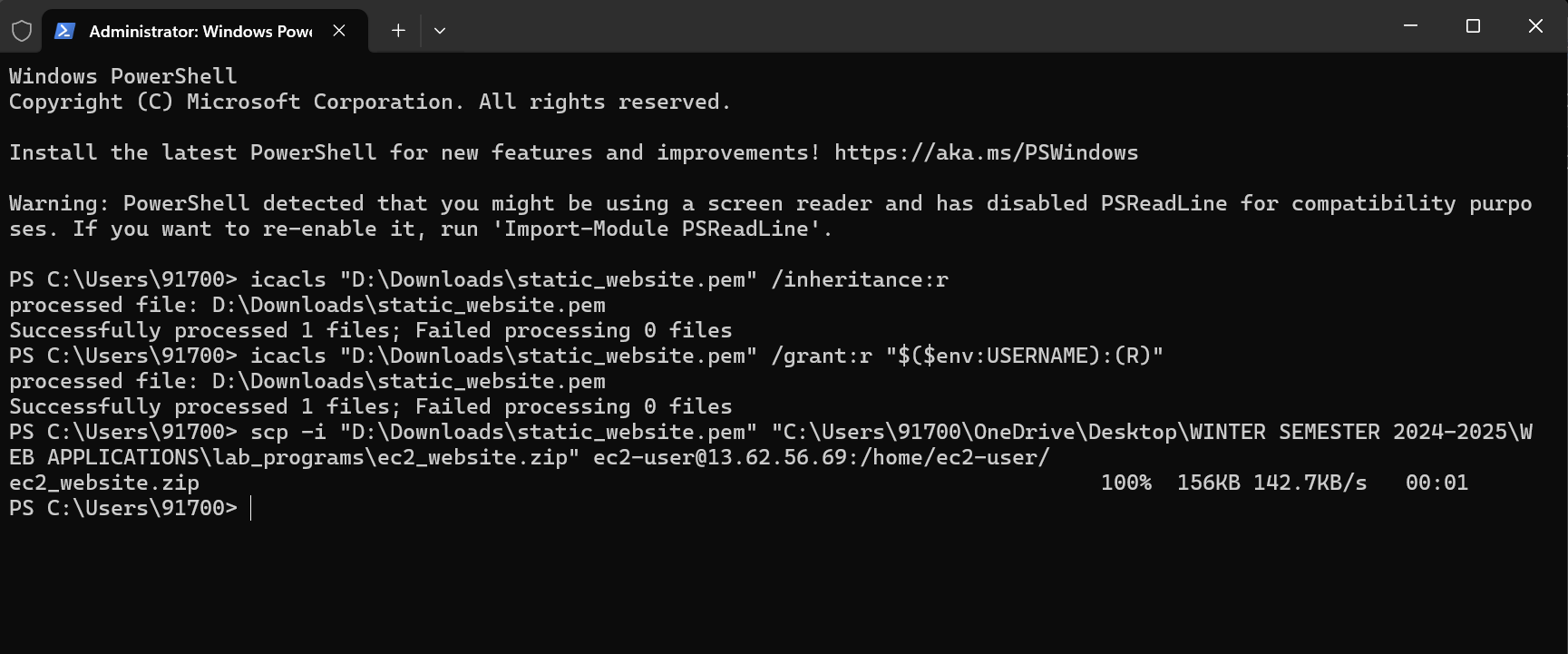
* **Go to AWS Console → EC2 → Launch Instance.**
* **Choose Amazon Linux 2 or Ubuntu AMI.**
* **Select t2.micro (free-tier eligible).**
* **Configure key pair (so you can SSH).**
* **In Security Group, allow:**
  + **SSH (22) → your IP.**
  + **HTTP (80) → Anywhere (0.0.0.0/0).**
* **Launch the instance.**
* **LAUNCHING A STATIC WEBSITE USING EC2 INSTANCE:**

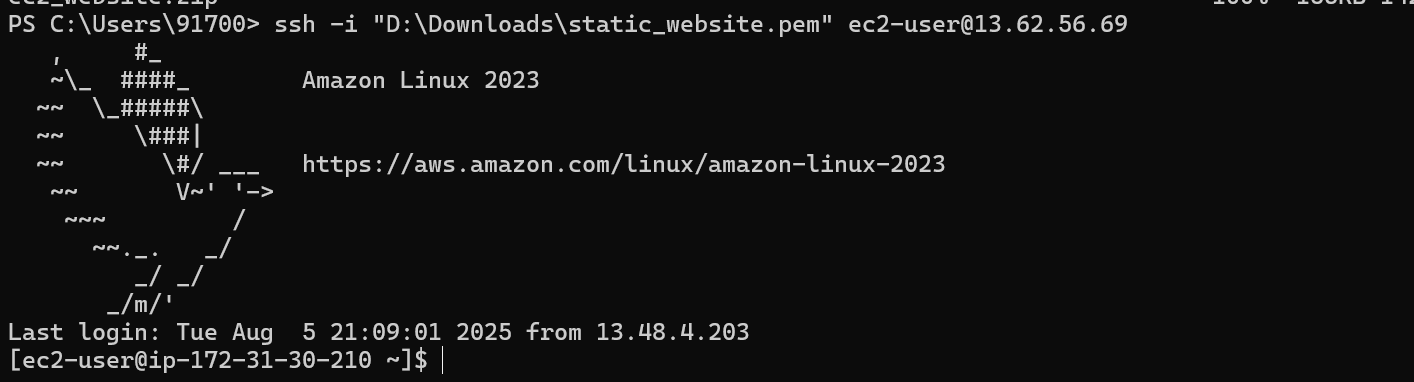


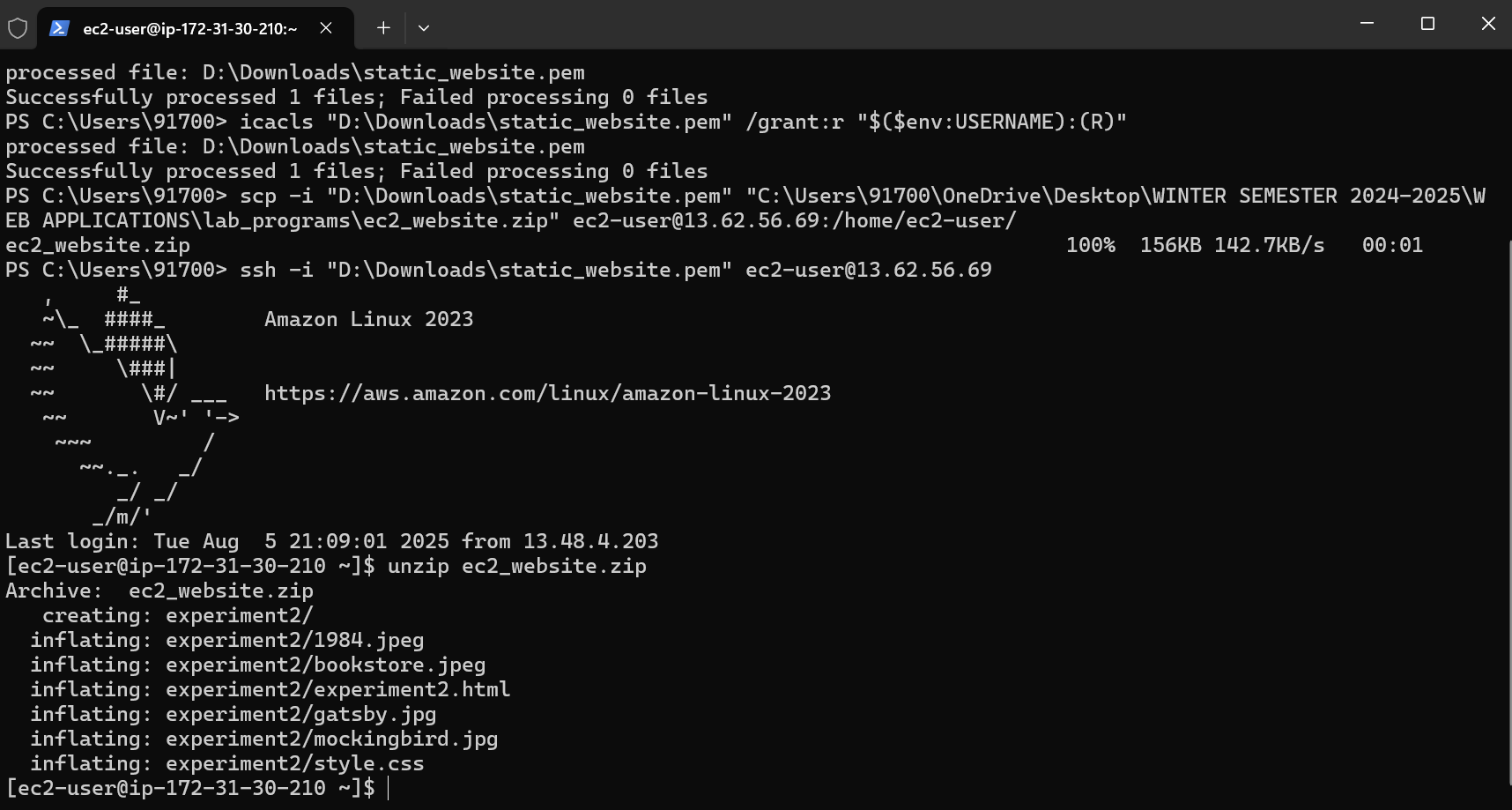


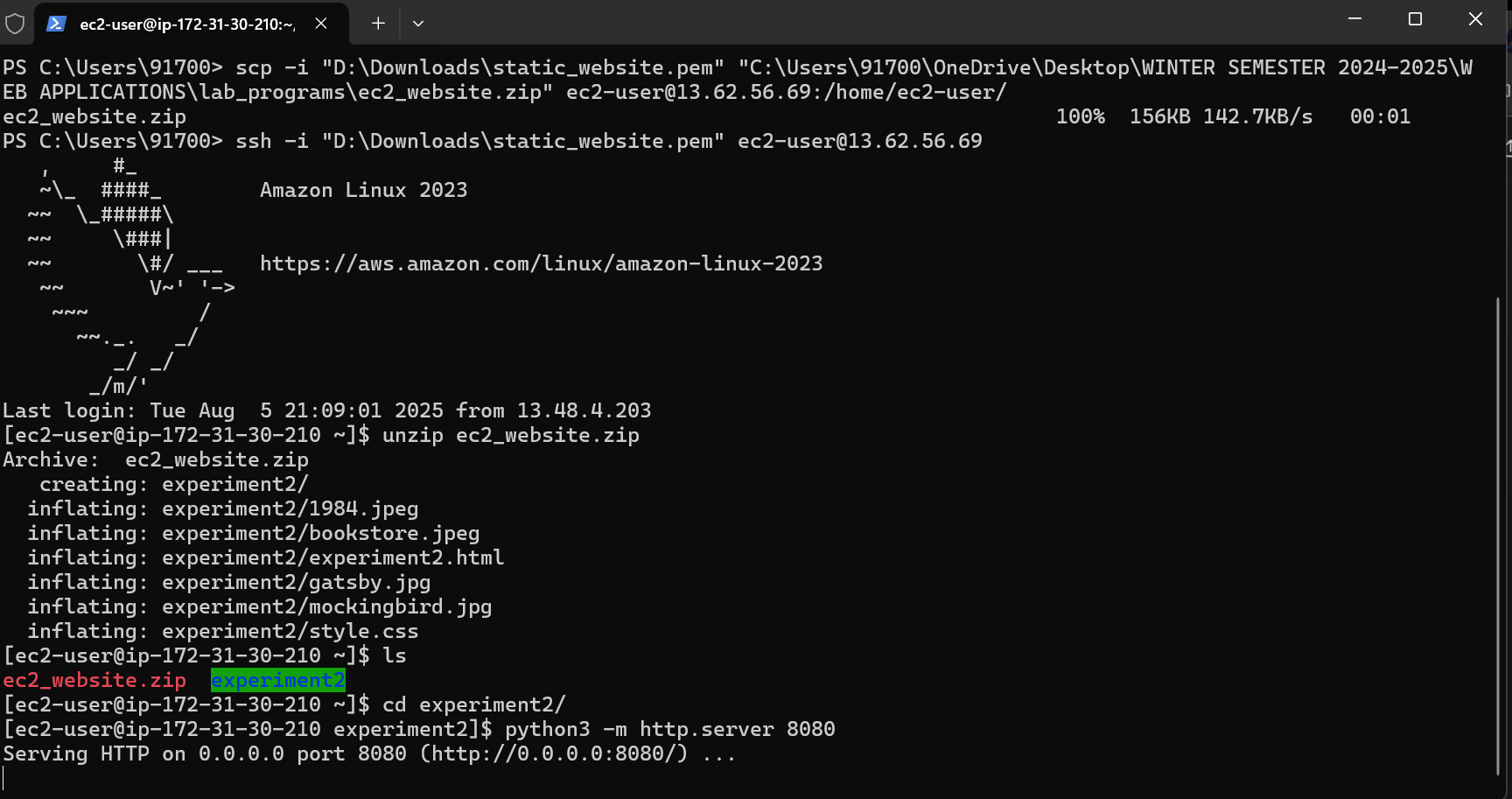


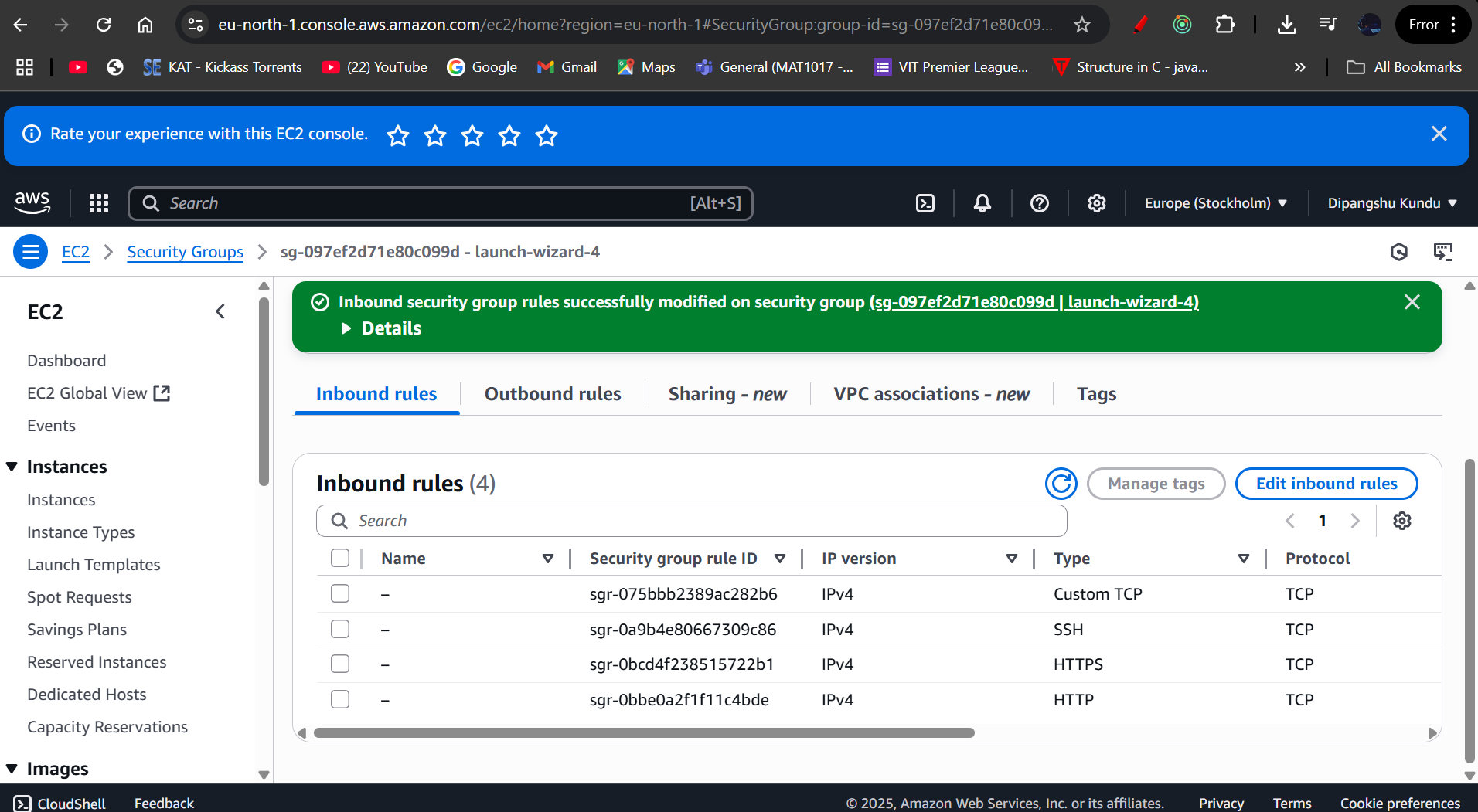












**Steps to Add Inbound Rule**

1. **In the list of Security Groups (the screen you shared), click on the Security Group ID linked to your EC2 instance (e.g., sg-xxxxxxxxx).**
2. **This will open the Security Group details page.**
3. **In the tabs, click Inbound rules.**
4. **Click Edit inbound rules (top right).**
5. **Add a new rule:**
   * **Type: Custom TCP Rule**
   * **Port range: 8080 (if you want to keep using Python’s server) OR 80 (if you want to use Apache).**
   * **Source: 0.0.0.0/0 (to allow from anywhere)**
   * **(Optional but safer: also add ::/0 for IPv6).**
6. **Click Save rules.**

**EC2 Setup and MySQL Database Management, including Database with Triggers and Stored Procedures**

**Got it 👍 You want to set up EC2 with MySQL database and manage it with Triggers and Stored Procedures. I’ll break this into clear steps you can follow:**

**✅ Step 1: Launch an EC2 Instance**

1. **Go to AWS EC2 Console → Launch Instance.**
2. **Choose an Ubuntu AMI.**
3. **Choose instance type → t2.micro (Free Tier).**
4. **Configure security group:**
   * **Allow SSH (22) from your IP.**
   * **Allow MySQL/Aurora (3306) from your IP (or 0.0.0.0/0 if public access, but not secure).**
   * **Allow HTTP (80) if you want to host a web app.**
5. **Launch with your .pem key.**

* **sudo apt update**
* **sudo apt install mysql-server**
* **sudo systemctl status mysql**
* **sudo mysql**
* **alter user ‘root’@’localhost’ identified with mysql\_native\_password by ‘rsvj7#2L3Y’;**
* **flush privileges;**

**✅ Step 6: Create Database and User**

**CREATE DATABASE companydb;**

**CREATE DATABASE companydb;**

**✅ Step 7: Create Tables**

**Example:**

**USE companydb;**

**CREATE TABLE employees (**

**emp\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100),**

**department VARCHAR(50),**

**salary DECIMAL(10,2),**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**

**);**

**✅ Step 8: Create a Stored Procedure**

**Example: Add new employee**

**DELIMITER //**

**CREATE PROCEDURE AddEmployee(IN empName VARCHAR(100), IN dept VARCHAR(50), IN sal DECIMAL(10,2))**

**BEGIN**

**INSERT INTO employees(name, department, salary) VALUES(empName, dept, sal);**

**END //**

**DELIMITER ;**

**Call it:**

**CALL AddEmployee('John Doe', 'IT', 60000.00);**

**✅ Step 9: Create a Trigger**

**Example: Log salary updates**

**CREATE TABLE salary\_audit (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**emp\_id INT,**

**old\_salary DECIMAL(10,2),**

**new\_salary DECIMAL(10,2),**

**changed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**

**);**

**DELIMITER //**

**CREATE TRIGGER after\_salary\_update**

**AFTER UPDATE ON employees**

**FOR EACH ROW**

**BEGIN**

**IF OLD.salary <> NEW.salary THEN**

**INSERT INTO salary\_audit(emp\_id, old\_salary, new\_salary)**

**VALUES(OLD.emp\_id, OLD.salary, NEW.salary);**

**END IF;**

**END //**

**DELIMITER ;**

**Test:**

**UPDATE employees SET salary = 75000 WHERE emp\_id = 1;**

**SELECT \* FROM salary\_audit;**

**✅ Step 10: Connect Remotely (Optional)**

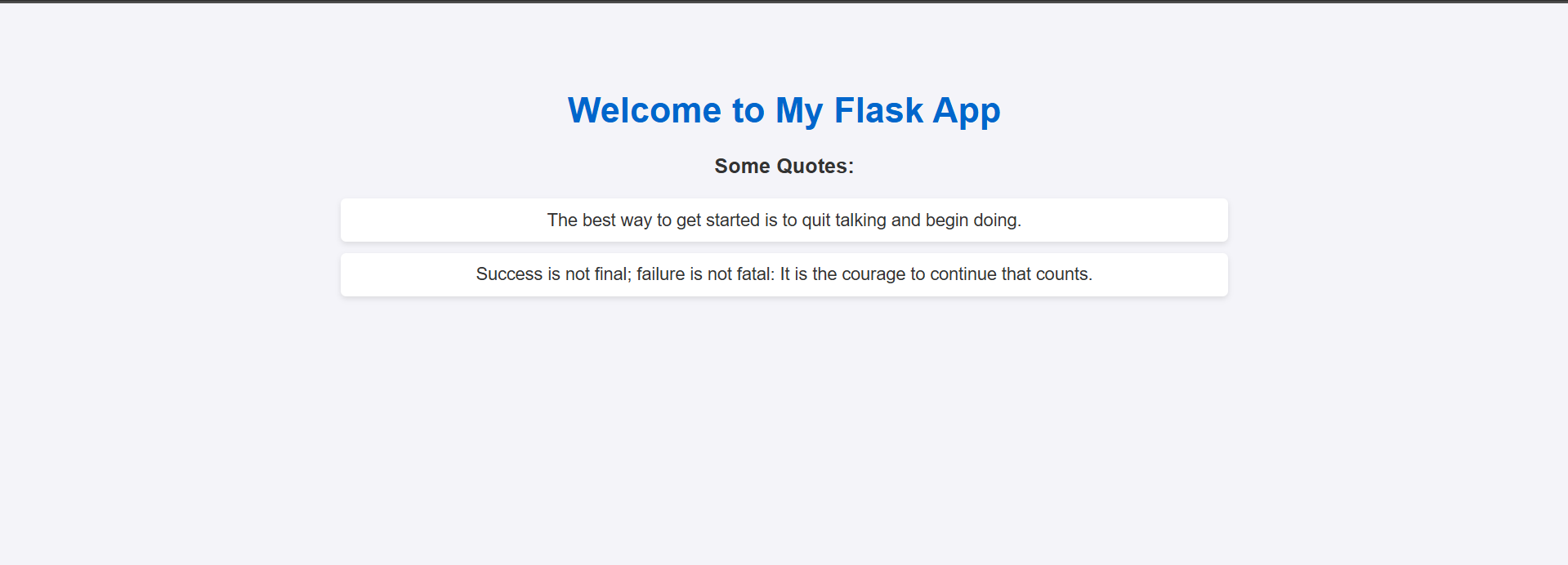
**From your local system:**

**mysql -h <EC2-Public-IP> -u dbuser -p**

**👉 Would you like me to also include steps for connecting this EC2 MySQL DB to a web app (PHP/Python Flask) so you can test triggers and procedures with a frontend?**

**Web Application Deployment using AWS Elastic Beanstalk**

**The steps are simple just create an application and just atlast create application**

****

**SERVERLESS COMPUTING USING S3 AND LAMBDA INTEGRATION**

**1. Create an S3 Bucket**

1. Go to **AWS Console → S3 → Create Bucket**.
2. Give it a unique name (e.g., my-lambda-demo-bucket).
3. Choose a region (keep it the same as Lambda).
4. Leave defaults, but **uncheck "Block all public access"** if you need public access.
5. Create the bucket.

**2. Upload a Test File to S3**

* After bucket creation → **Upload** → choose a test file (e.g., test.txt).
* We’ll use this file to trigger Lambda.

**3. Create a Lambda Function**

1. Go to **AWS Console → Lambda → Create Function**.
2. Choose **Author from scratch**.
3. Name: MyS3LambdaFunction.
4. Runtime: **Python 3.x** (or Node.js if you prefer).
5. Permissions: Create or use an existing role with **S3 and CloudWatch access**.
6. Click **Create Function**.

**4. Add S3 Trigger to Lambda**

1. In your Lambda function → **Configuration → Triggers → Add Trigger**.
2. Select **S3**.
3. Choose your bucket (my-lambda-demo-bucket).
4. Event type: **PUT (All object create events)**.
5. Check "Enable trigger".
6. Save.

**5. Write Lambda Code**

Example: A simple function that logs file details whenever a file is uploaded:

**Python code (lambda\_function.py):**

import json

def lambda\_handler(event, context):

# Print event details

print("Event:", json.dumps(event))

# Extract bucket and object key

bucket = event['Records'][0]['s3']['bucket']['name']

key = event['Records'][0]['s3']['object']['key']

size = event['Records'][0]['s3']['object']['size']

message = f"New file uploaded: {key} ({size} bytes) in bucket {bucket}"

print(message)

return {

'statusCode': 200,

'body': json.dumps(message)

}

Upload the above code in zip file under the code section in lambda function

**6. Test the Integration**

1. Go to your **S3 bucket**.
2. Upload a new file (e.g., hello.txt).
3. Go back to **Lambda → Monitor → Logs (CloudWatch)**.
4. You should see logs showing the file name, size, and bucket

**EC2 AUTO SCALING USING LAUNCH TEMPLATES AND SCALING POLICIES**

Got it 👍  
Here’s a **clear, step-by-step rewritten version** of your lab instructions for **creating a Launch Template, Auto Scaling Group, and verifying the setup in EC2**:

**🚀 AWS Lab: EC2 Auto Scaling with Launch Template**

**Task 1: Create a Launch Template**

1. From the **AWS Management Console**, go to **Services → Compute → EC2**.
2. In the **top-right corner**, confirm which **Region** you are in (e.g., N. Virginia → us-east-1).

⚠️ Auto Scaling resources are region-specific.

1. In the **left navigation pane**, choose **Launch Templates**.
2. Click **Create launch template** and configure as follows:
   * **Launch template name:** my\_template
   * **Template version description:** Test launch template for an Auto Scaling group
   * **Auto Scaling guidance:** Select **Provide guidance**
   * **AMI:** From **Quick Start**, choose **Amazon Linux 2023 AMI (x86\_64, HVM)**
   * **Instance type:** t2.micro
   * **Key pair name:** vockey (optional for template)
   * **Subnet:** Select **Don’t include in launch template**
   * **Security groups:** Skip (will configure later)

**Advanced network configuration:**

* + Click **Add network interface** and set:
    - **Auto-assign public IP:** Enable
    - **Security groups:** Select the group containing Ec2SecurityGroup
    - **Delete on termination:** Yes

1. Click **Create launch template**.
2. On the confirmation page, choose **View launch templates**.
3. Click the link for your **Launch template ID**.
4. From the **Actions** menu, select **Create Auto Scaling group**.

**Task 2: Create an Auto Scaling Group**

1. On **Step 1: Choose launch template**:
   * **Auto Scaling group name:** my-first-asg
   * **Launch template:** Select the template created earlier (my\_template).
   * **Version:** Latest
   * Click **Next**.
2. On **Step 2: Choose instance launch options**:
   * **VPC:** Select the VPC that contains **Lab VPC**.
   * **Subnets:** Select the subnet named **Public Subnet 1**.
   * Click **Skip to review**.
3. On the **Review page**, click **Create Auto Scaling group**.

**Task 3: Verify Your Auto Scaling Group**

1. In the **left navigation pane**, go to **EC2 Dashboard → Instances (running)**.
2. Confirm that a new **EC2 instance** has launched.
   * If it does not appear, wait a minute and click the **Refresh** icon.
   * Ensure the **Instance state** shows **Running**.

✅ **Lab Complete!**

**Final Steps**

1. In the **upper-right corner**, click your username (voclabs/user).
2. Select **Sign Out**.
3. At the top of the lab page, click **End Lab → Yes** to confirm.

Would you like me to also create a **diagram/flow chart** that shows how the **Launch Template + Auto Scaling Group** works together? That would make it even easier to visualize.