# VPC 3-tier Application

## ****Overview of the Project:****

This project demonstrates a simplified 3-tier web application architecture deployed on AWS using **EC2, RDS, and S3**. The application allows users to upload an image, and it processes and stores the image along with some metadata in AWS services. Specifically:

### **1. Frontend (Web Tier):**

* Hosted on a single **EC2 instance**, serving a static HTML form where users can upload an image, and enter their name and email.
* The form is served using a **Nginx** web server.
* Upon submission, the request is **proxied to the App Server** for processing.

### **2. Backend (App Tier):**

* A **Flask-based App Server** running on another **EC2 instance** handles:
  + Receiving form submissions.
  + Uploading the image to **S3**.
  + Storing the user's name, email, and the S3 image URL in **RDS (MySQL)**.
* The Web Server uses proxy\_pass in Nginx to forward requests to the App Server via its **private IP**.

### **3. Storage:**

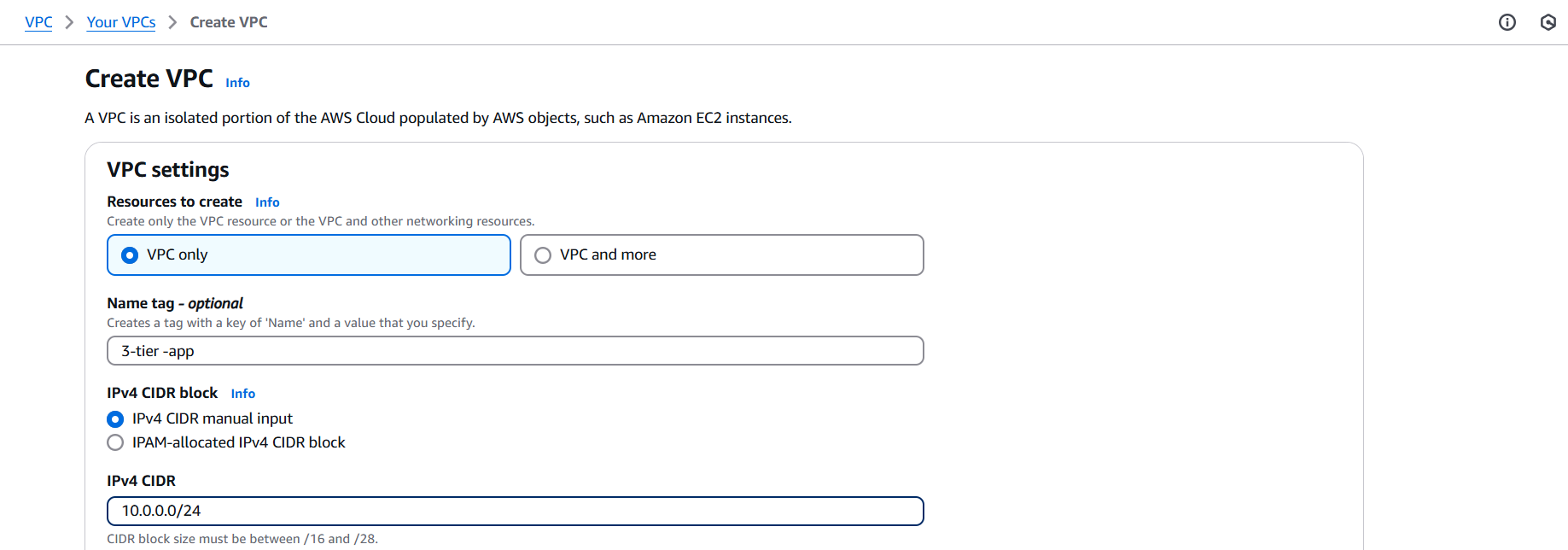
* **S3**: Stores the uploaded images.
* **RDS**: Stores metadata like user name, email, and the image's S3 URL in a **MySQL** database.

## ****Key Details:****

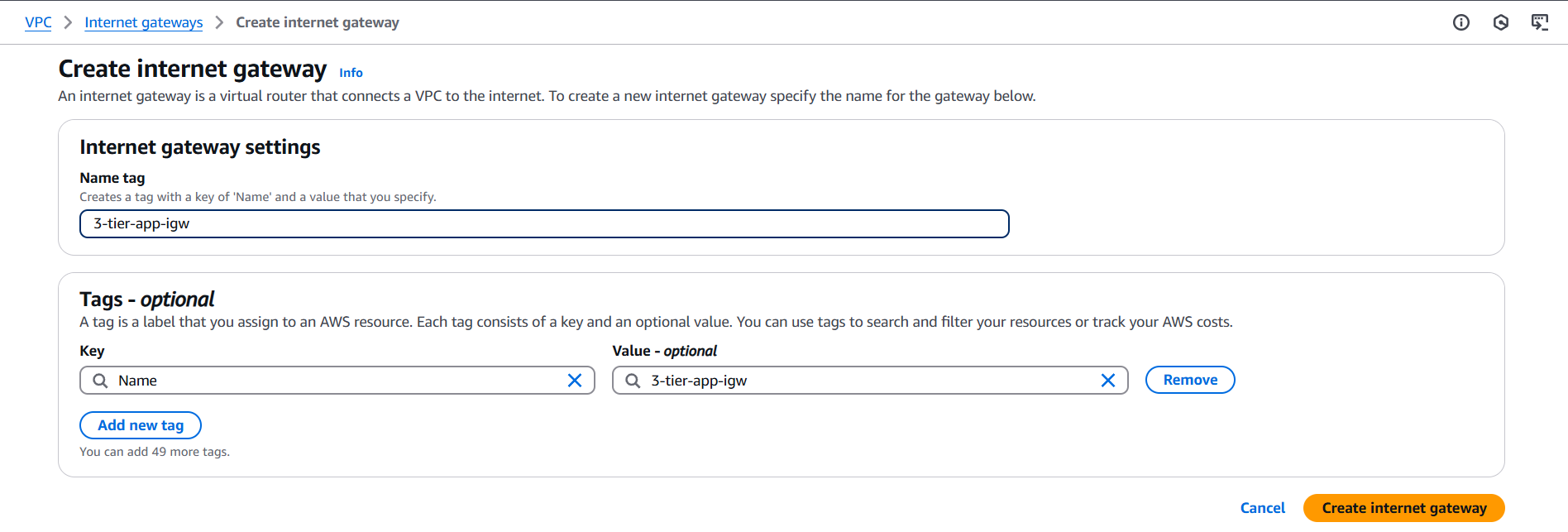
* **Public Subnets Only**: Both the **Web Server** and **App Server** are launched in **public subnets** to simplify the setup and avoid NAT gateway charges.
* **No Load Balancers or Auto Scaling**:
  + The setup uses **individual EC2 instances** for both the Web and App Tiers.
  + There are **no ALBs or ASGs** involved in this deployment for simplicity.
* **IAM Role Usage**:
  + An IAM role with permissions for **S3** and **RDS** is attached to the EC2 instances to enable secure interaction with AWS services.
* **Nginx Proxy Configuration**:
  + Nginx on the Web Server is configured to **proxy requests** to the App Server's private IP on a specific route (e.g., /upload).

### Step 1 - Create VPC architecture:

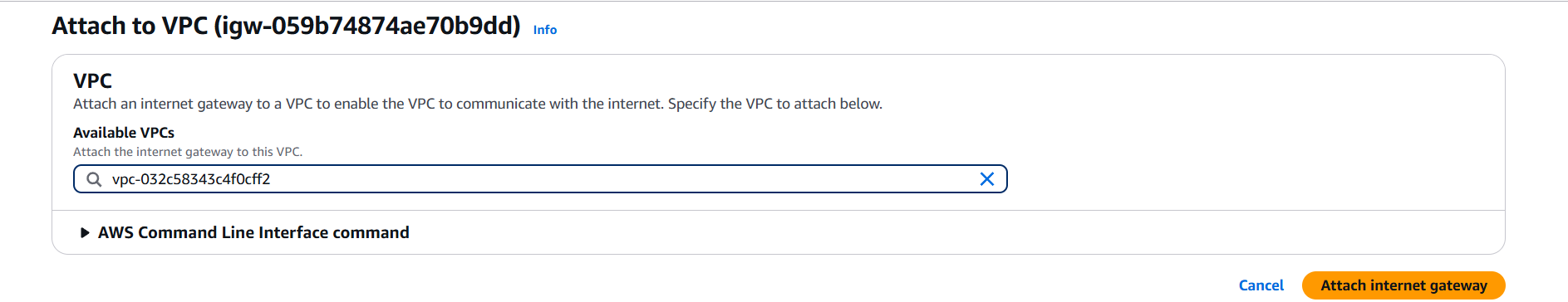
* Create a custom VPC.



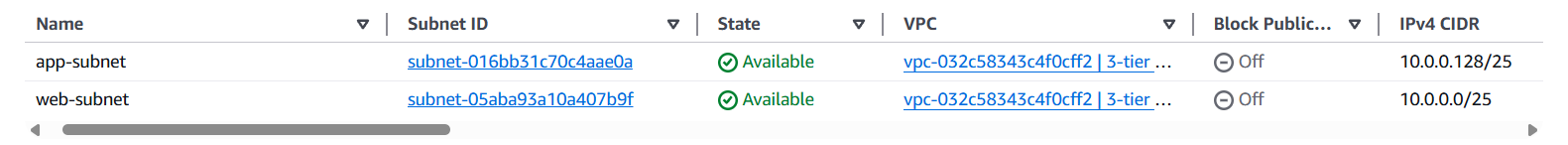
* Create an internet gateway.



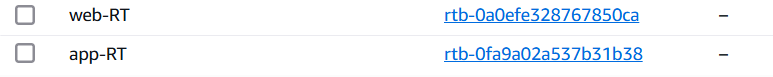
* Attach to custom VPC.



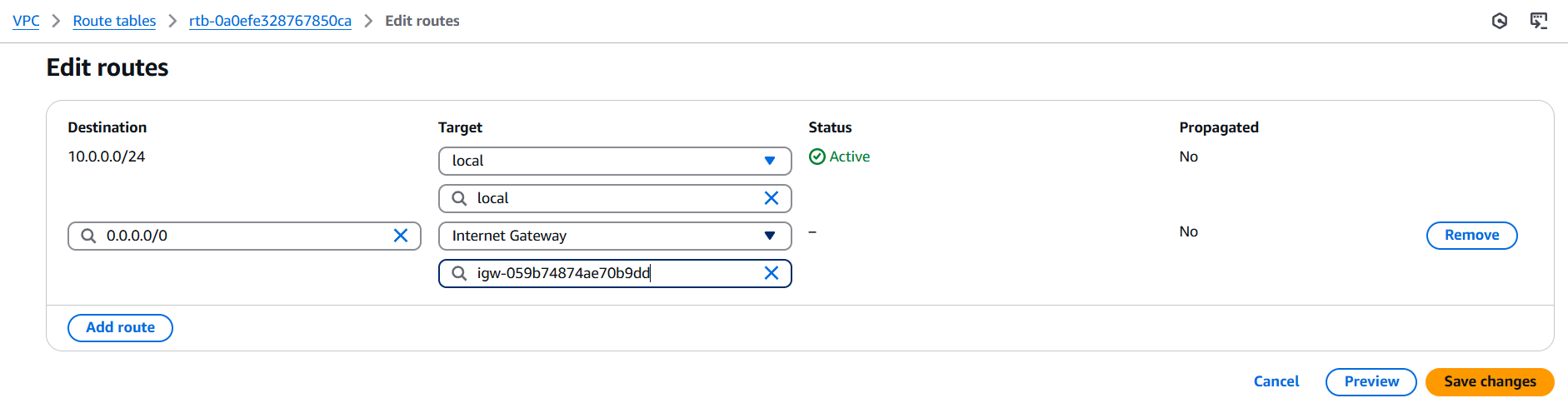
* Create 2 subnets in it. Both in diff. AZs.



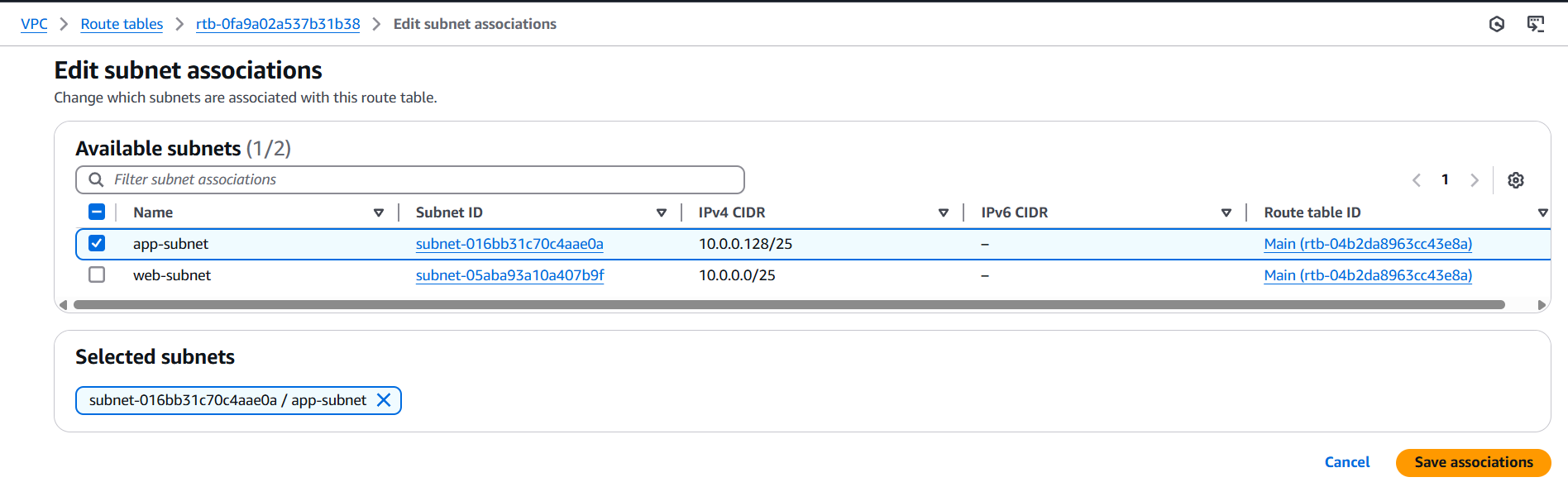
* Create two route tables each for webserver and appserver.



* Add igw in routes of both the tables as we are using public servers so no need of NAT gateway and private subnet or RT.



* Associate respective subnets to them. App-subnet to app-RT and web-subnet to web-RT.



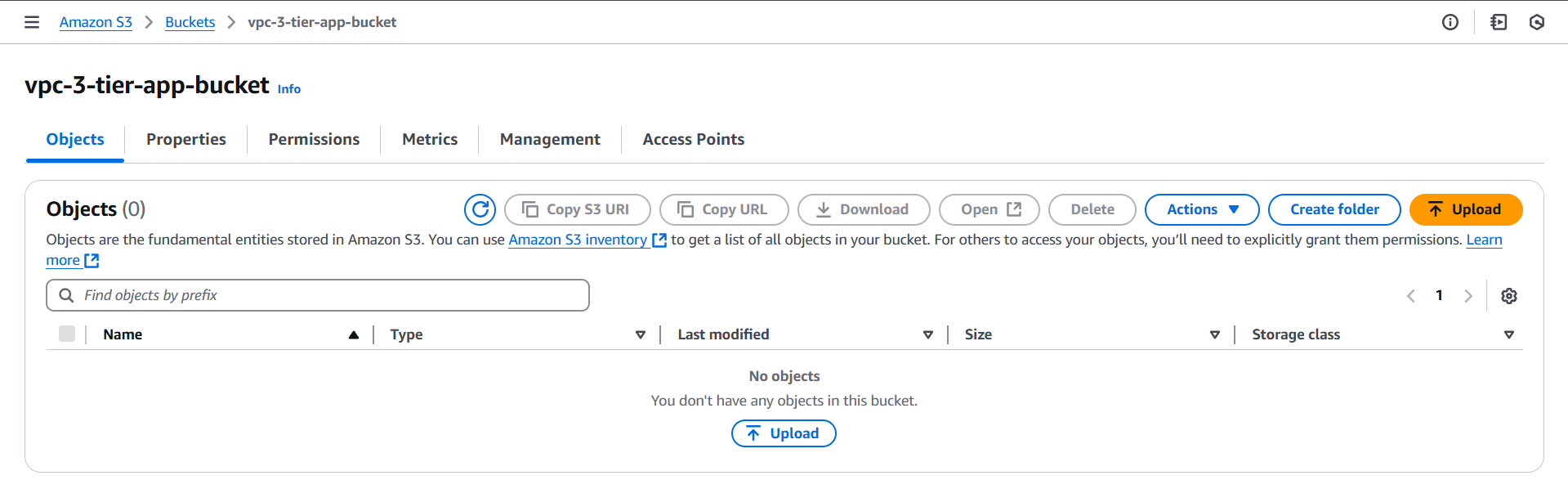
### Step 2 – Create an IAM role:

* Go to IAM role creation from IAM dashboard to create a role with S3 and RDS full access.
* Click on create role to create custom role.
* Trusted entity type: AWS service.
* Use case: EC2.
* Add permission policies:
  + [AmazonRDSFullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonRDSFullAccess)
  + AmazonS3FullAccess
* Role name: EC2-S3-RDS-Access
* Review and create the role.

### S3 bucket:

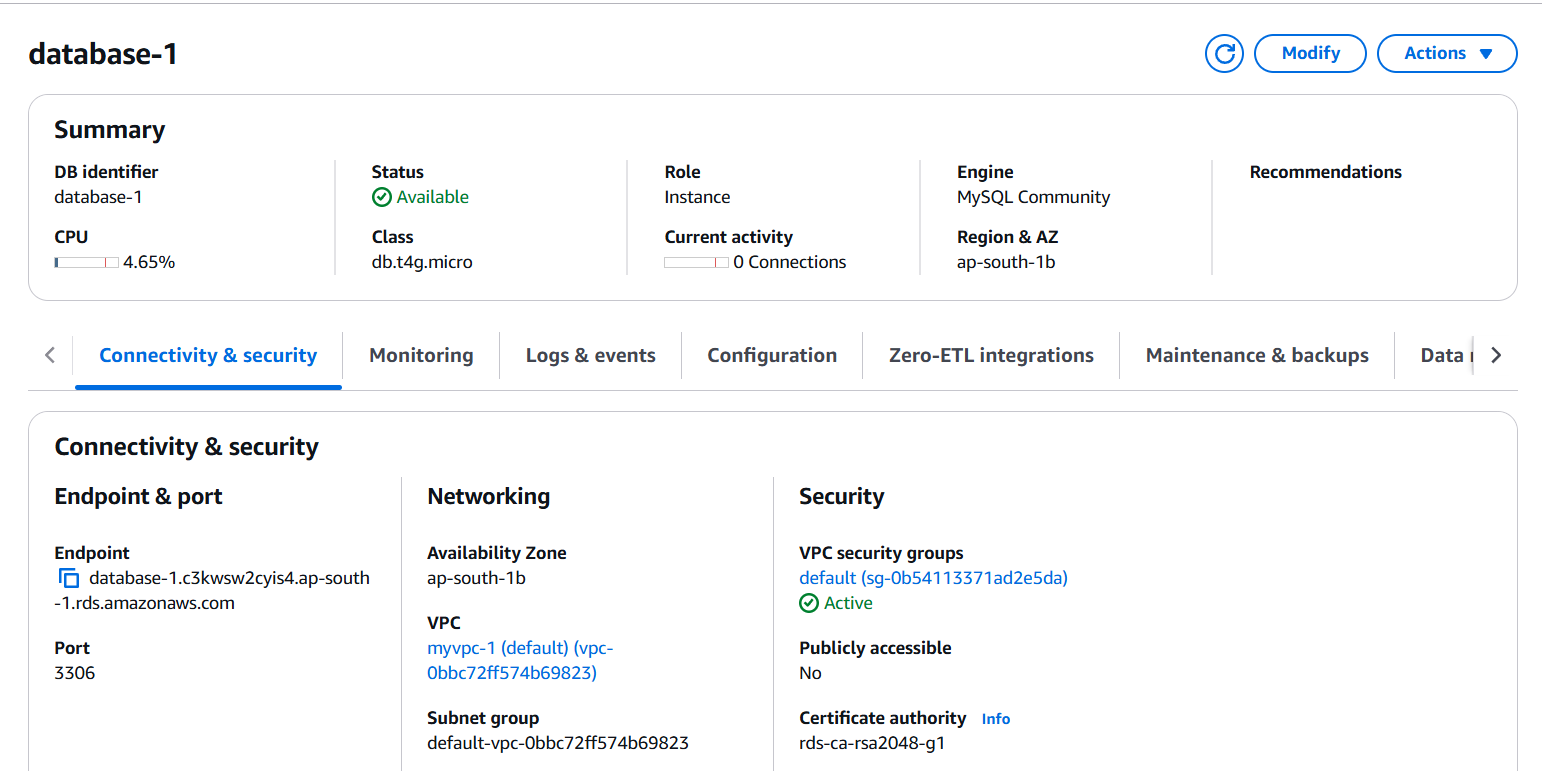
Create an ACL enabled and public-access bucket of globally unique name.

Add a policy of S3:geObject to read and view the image on site to everyone (Principle: \*).



### RDS: (After appserver creation)

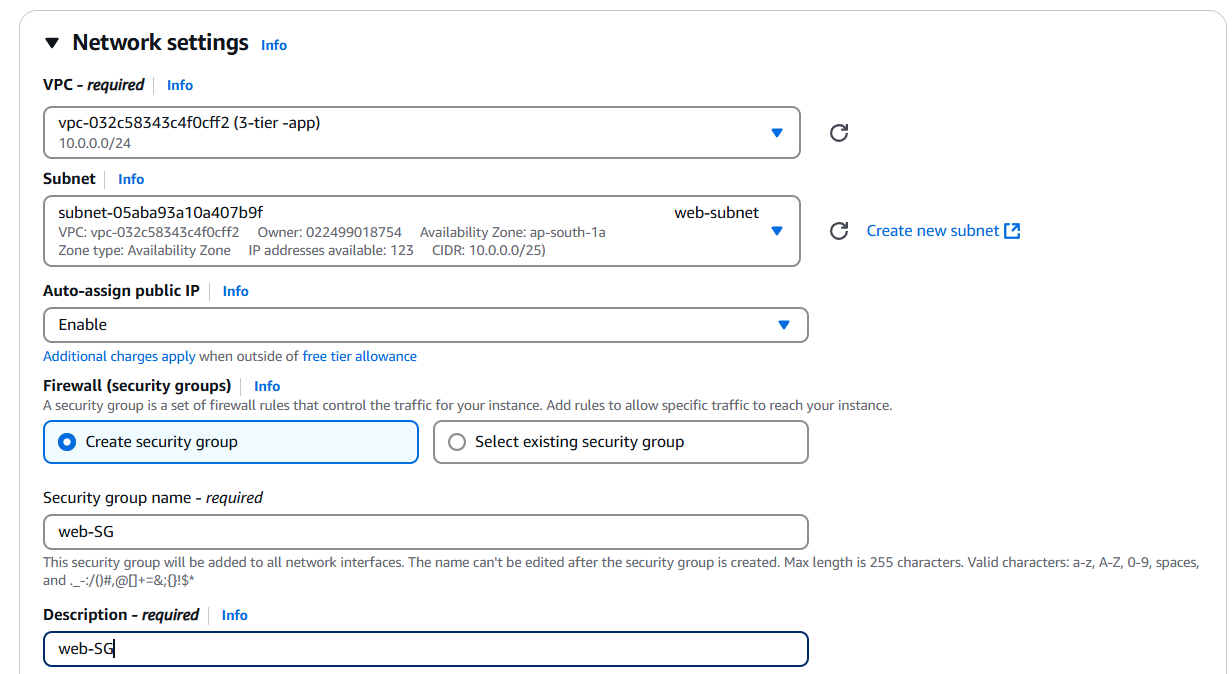
* Create a database from Aurora and RDS. You need DB subnet group use both subnets.
* Database creation method: Standard.
* Configuration: MySQL.
* DB instance size: Free-tier.
* Credential manager: self-managed.
* Enter a password and enter VPC and subnet group.
* Create the database.



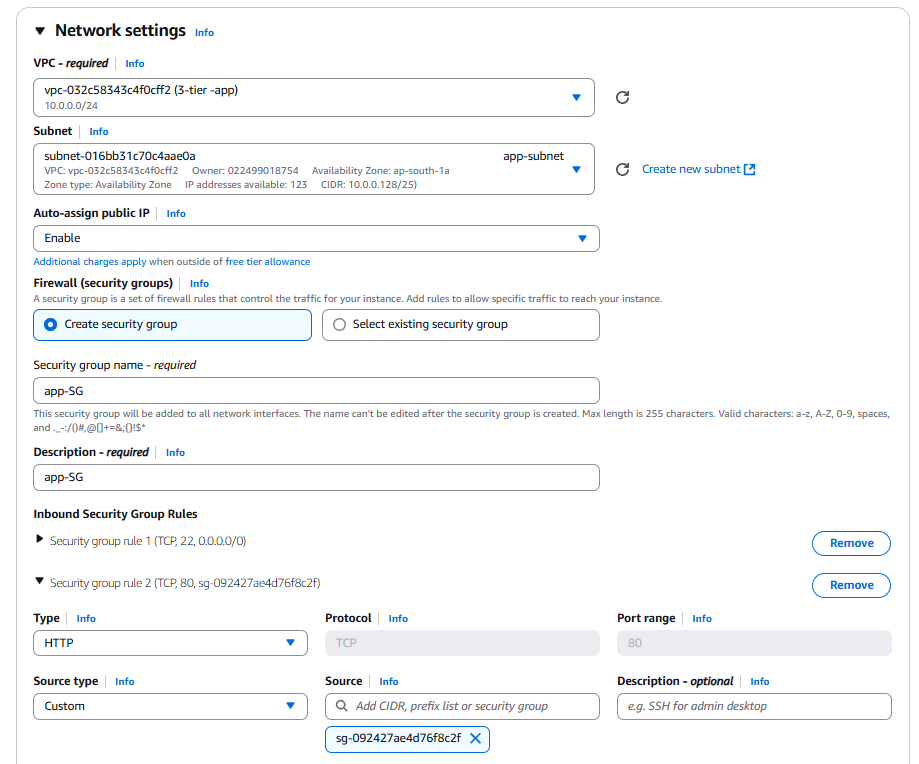
* Perform EC2 connection with appserver.

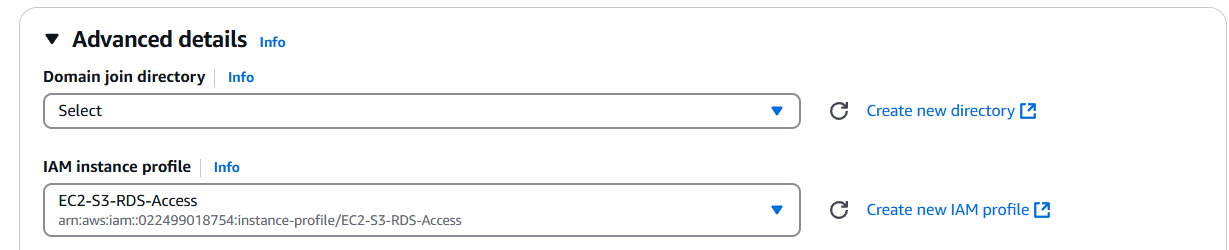
### EC2 instances:

* Launch an EC2 instance (Amazon Linux) named webserver with HTTP in inbound rule and below settings without fail.



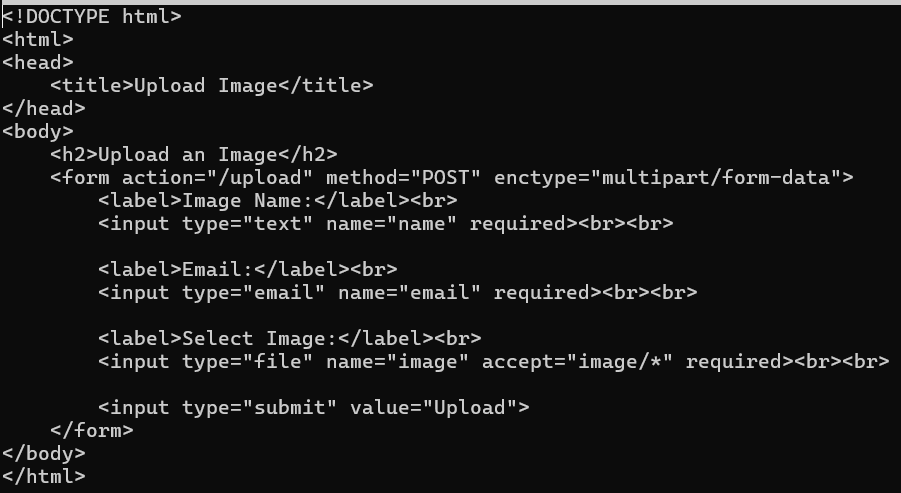
* Launch an EC2 instance named appserver with following settings with ssh and source of HTTP as Web-SG.



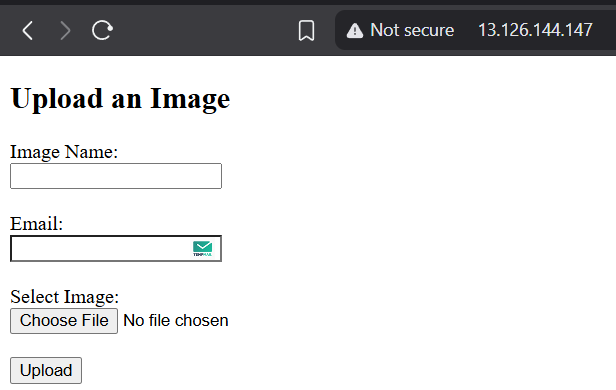


#### Webserver:

* Connect to webserver via ssh.
* Run below commands:
  + sudo yum update
  + sudo yum install nginx -y
  + sudo systemctl start nginx
  + sudo systemctl enable nginx
  + cd /usr/share/nginx/html
  + ls
* Open index.html => sudo nano index.html
* Put HTML code in that file.



* Restart nginx and check for page by putting the public IP in browser. (sudo systemctl restart nginx, <http://public_IP>).



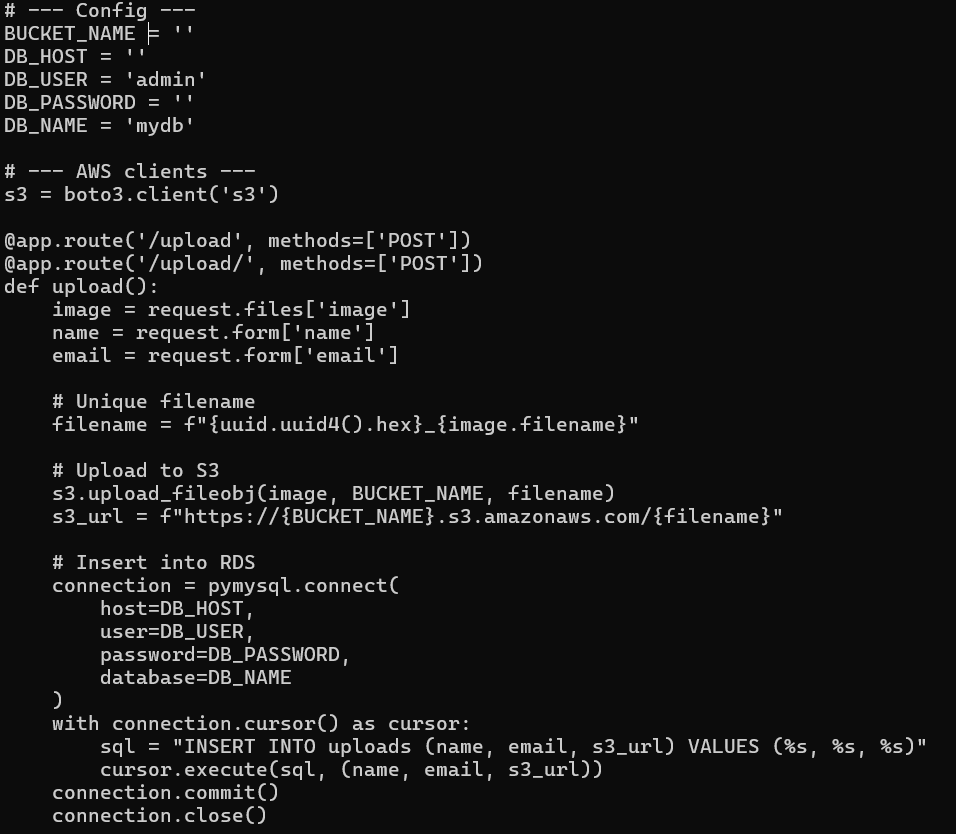
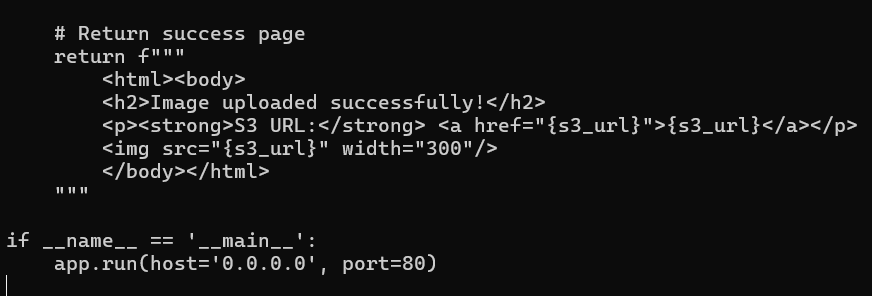
* Go to nginx.conf file to add proxy\_pass to appserver (sudo nano /etc/nginx/nginx.conf).
* Add code below the line.



As:

#### Appserver:

* Connect to appserver via ssh.
* Run below commands:
  + sudo yum update -y
  + sudo yum install python3 git -y
  + sudo yum install python3-pip -y
  + python3 -m pip install --upgrade pip
  + sudo yum install mariadb105-server -y
  + mkdir ~/app && cd ~/app
  + nano app.py
* Add app.py code.
* Put below code:

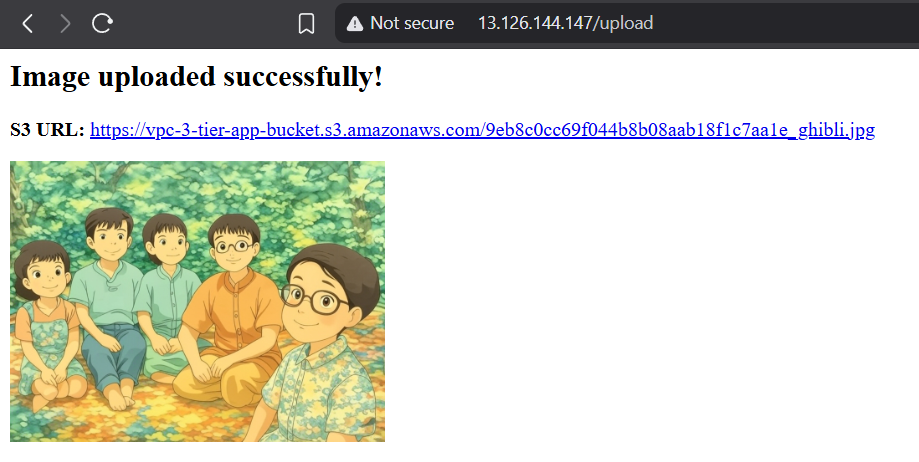
* Run the app: sudo python3 app.py.

### Check the working:

* Hit public IP of web server.
* Enter details.



* Click on submit button to upload image to the S3 bucket and view on the next page.



For this 3-tier app to run properly most important things that I needed to take care of and had some errors in some of them are:

1. Creating RDS in custom VPC and a DB subnet group with 2 or more subnets from 2 different AZs.
2. Correct path to proxy\_pass.
3. Correct routing in app.py.
4. Post method in index.html to /upload.
5. Remember to add Web-SG to app-SG so that connection happens.
6. Create and connect RDS to EC2 only after creating appserver to avoid hassle that may happen if we try to do it via SG editing way.
7. Errors do come and try to solve them on your own.