

**Name :Roshan Saral Kumar(CLOUD-COMPUTING-INTERN)**  
**DATE:1-11-2025**

---

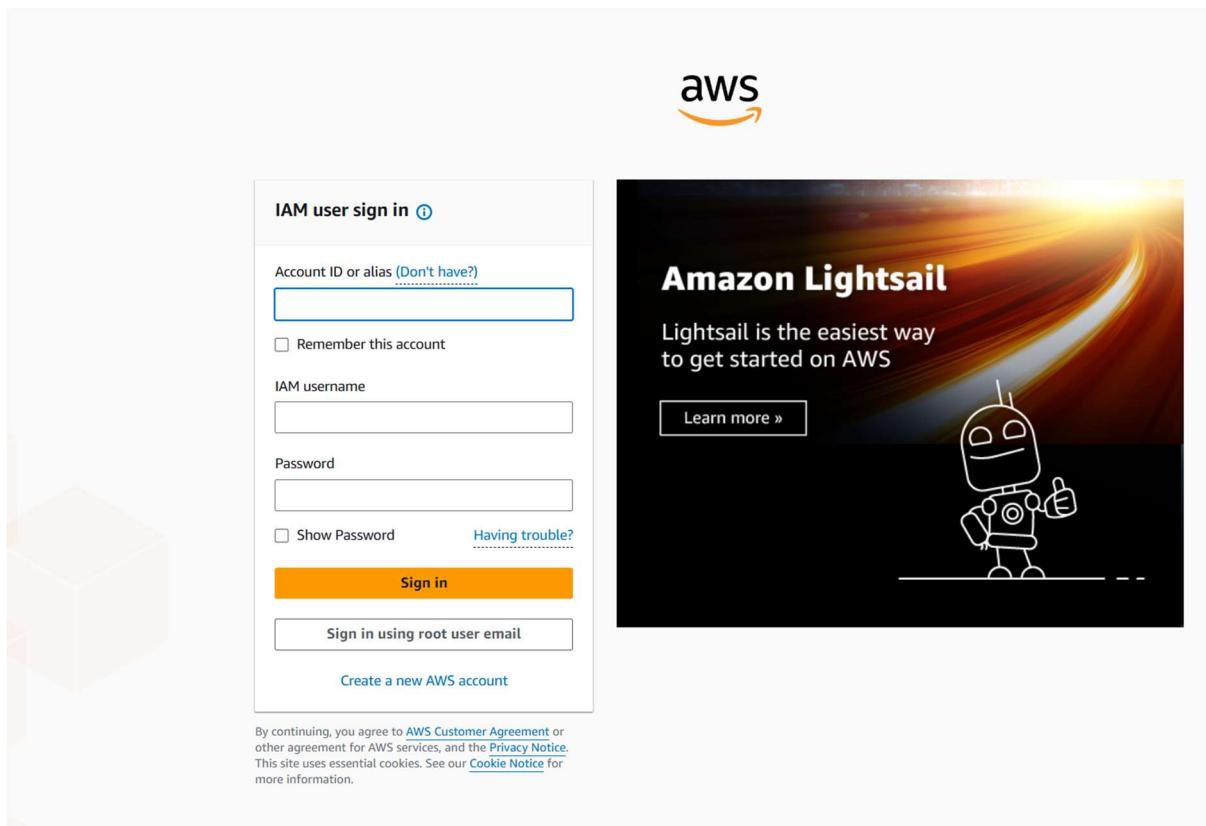
## **TASK 8**

- **Task 8: Host and Deploy a Web Application on the Cloud**
- 

- **Objective:-**
  - **To understand containerization and cloud-native deployment by creating a Docker image of a simple web application and deploying it to a cloud platform (Google Cloud Run / AWS ECS / Azure Container Instances).**
  - **This teaches how modern applications are made portable, scalable, and easy to deploy.**
- 

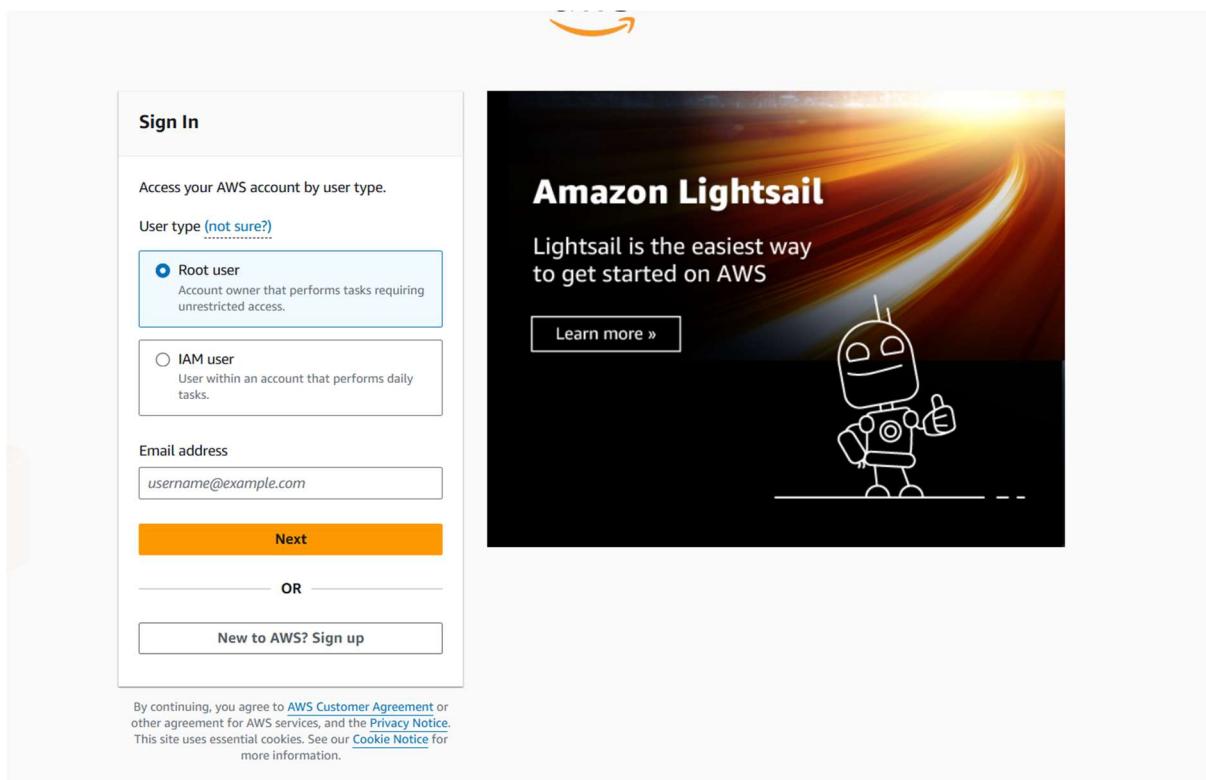
- **TOOLS USED FOR AWS (Free Tier):**
  - **Docker Desktop (or Docker CLI)**
  - **Cloud Platform:**
  - **AWS Elastic Container Service (ECS)**
  - **Programming Language: Python**
  - **Code Editor: VS Code**
  - **Terminal**
- 

- **THIS IS THE SIGN IN CONSOLE PAGE AND IAM USING SIGN USING ROOT USER EMAIL (ROOT USER MEANS I HAVE ACCESS TO ALL THE SERVICES OF AWS IF IT IS AN IAM (IDENTITY AND ACCESS MANAGEMENT USER THEN THE USER WILL HAVE ACCESS TO ONLY A LIMITED NUMBER OF SERVICES THAT AWS OFFERS AND THE IAM USERS WILL HAVE IAM POLICIES ATTACHED TO THEM SO ONLY THOSE SERVICES THEY WILL BE ABLE TO USE) CURRENTLY LOGGING IN AS ROOT USER EMAIL.**



The image shows the AWS IAM User Sign In page. At the top right is the AWS logo. Below it is a sidebar with the title "IAM user sign in ⓘ". The main form contains fields for "Account ID or alias (Don't have?)", "Remember this account", "IAM username", "Password", and "Show Password". There is also a link "Having trouble?". A large orange "Sign in" button is centered. Below the button are links for "Sign in using root user email" and "Create a new AWS account". At the bottom of the page is a small legal notice about cookie usage.

- AFTER CLICKING ON SIGN IN USING ROOT USER EMAIL I WILL GET THIS PAGE WHERE I HAVE TO ENTER MY EMAIL-ID. AND AS WE CAN SEE ROOT USER IS SELECTED BY DEFAULT. WE CAN ALSO SELECT IAM USER IN THIS PAGE IN CASE IF THERE IS A CHANGE.



The image shows the AWS Sign In page. At the top right is the AWS logo. Below it is a sidebar with the title "Sign In". The main form asks "Access your AWS account by user type." It has two options: "Root user" (selected) and "IAM user". Below these are fields for "Email address" (containing "username@example.com") and a large orange "Next" button. To the right of the form is a sidebar for "Amazon Lightsail" with the text "Lightsail is the easiest way to get started on AWS" and a "Learn more »" button. At the bottom of the page is a small legal notice about cookie usage.

- AFTER ENTERING OUR EMAIL WE NEED TO GIVE OUR PASSWORD

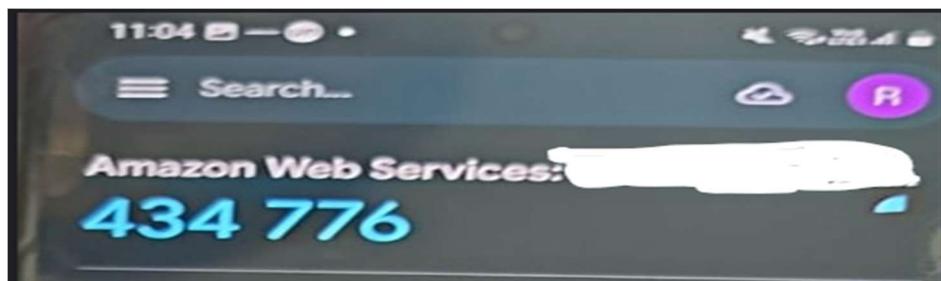
The image shows two side-by-side screenshots. The left screenshot is the 'Root user sign in' page from the AWS Management Console. It has a title bar, a password input field with placeholder text 'Enter the password for [REDACTED] (not you?)', a 'Password' label, a 'Show password' checkbox, a 'Forgot password?' link, a large orange 'Sign in' button, a 'Sign in to a different account' link, and a 'Create a new AWS account' link. The right screenshot is the 'Amazon Lightsail' landing page. It features a dark background with a bright orange and yellow swoosh graphic. The 'Amazon Lightsail' logo is at the top, followed by the text 'Lightsail is the easiest way to get started on AWS'. Below this is a 'Learn more »' button and a cartoon illustration of a white robot giving a thumbs up.

- SINCE I HAVE AN AWS ACCOUNT ALREADY CREATED I ACTUALLY MADE AN MFA FOR MY ROOT USER . MFA STANDS FOR (MULTI-FACTOR AUTHENTICATION CODE) IT ACTS AS A DOUBLE LAYER PROTECTION FOR OUR ROOT USER OTHERWISE WE CAN GET HACKED SO ALREADY PROVIDE AN EXTRA LAYER OF PROTECTION TO BE ON THE SAFER SIDE.
- MFA CODE IS GIVEN BY TOTEM TOKENS OR U CAN ALSO USE THE “GOOGLE AUTHENTICATOR APP” IN YOUR MOBILE WHICH IS MOSTLY PREFERRED.

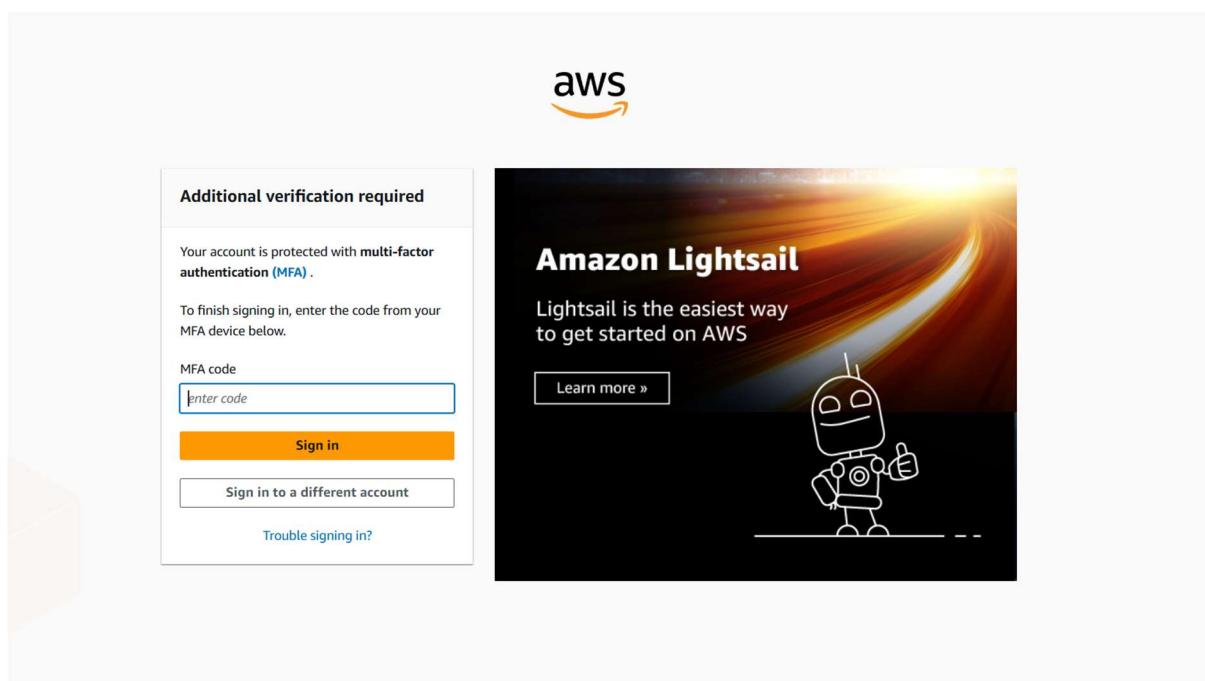


- THE SCREEN IN MY MOBLIE LOOKS LIKE THIS:-THE CONTENT WHICH IAM HIDING USING A WHITE COLOR IS MY PRIVATE INFORMATION OF MY ACCOUNT WHICH CANNOT BE SHARED BUT HIS IS HOW A SIMPLE MFA CODE

**LOOKS LIKE AND THE MFA CODE KEEPS ON CHANGING ON A DAILY BASIS.ALWAYS USE MFA TO ENSURE ENHANCED SECURITY.**



- THIS IS THE CODE THAT I HAVE TO ENTER IN MY AWS WEB BROWSER IN THE BELOW IMAGE IT IS SHOWN.



- AFTER SIGNING IN WE ENTER THE AWS MANAGEMENT CONSOLE AND THE DEFAULT REGION IS NORTH VIRGINIA THAT IS “us-east-1” WE CAN SELECT DIFFERENT REGIONS. IN THE IMAGE IT IS CLEARLY SHOWN IN THE RIGHT HAND SIDE WHICH REGION IAM CURRENTLY WORKING IN. THE BELOW IMAGE SHOWS THE SERVICES WHICH I VISITED AND THE CURRENT REGION WHICH IAM WORKING IN.
- TERMINOLOGIES:-
- Region:- It is a geothermal location which consists of a group of AZs (AVAILABILITY ZONES) and the network of a region is called a VPC(VIRTUAL PRIVATE CLOUD).
- AZ(Availability Zone):- they are a group of data centers.the network of a Az is a subnet.

- **Edge Location:-** it is a location in between various Azs or it is found at the boundary of an az In order to cache the incoming data acts like a storage for easier retrieval.

### PLS NOTE CHANGE STARTS FROM HERE BELOW:-

- **NOTE:-WE ARE WORKING IN THE PUBLIC SUBNET THAT IS “us-east-1”** if we are working in any azs it will show us—east-1a,us-east-1b and so on up to us-east-1f. We can check the azs in our aws management console itself that is given in below figure.

**SO NOW FOR THE deployment of web application SETTINGS IAM CURRENTLY WORKING IN THE NORTH VIRGINIA REGION SO IAM USING “us-east-1”.**

#### IF WE CLICK ON VPC:-

HERE WE WILL BE ABLE TO SEE THE SUBNETS IN THE VPC DASHBOARD IF WE CLICK ON SUBNETS THEN TOTALLY THERE ARE 6 SUBNETS THAT IS PRIVATE SUBNETS FROM “us-east-1a” to “us-east-1f” totally we have 6 private subnets or we can also say we have 6 azs under NORTH VIRGINIA REGION THAT IS “US-EAST-1” REGION. This is provided by aws itself by default we cannot create any Availability Zones.

Logging into aws amangement console and searching for VPC(VIRTUAL PRIVATE CLOUD)

**THIS IS THE AWS MANAGEMENT CONSOLE AFTER LOGGING IN CURRENTLY WORKING IN NORTH VIRGINIA REGION(US-EAST-1) the below image shows that**

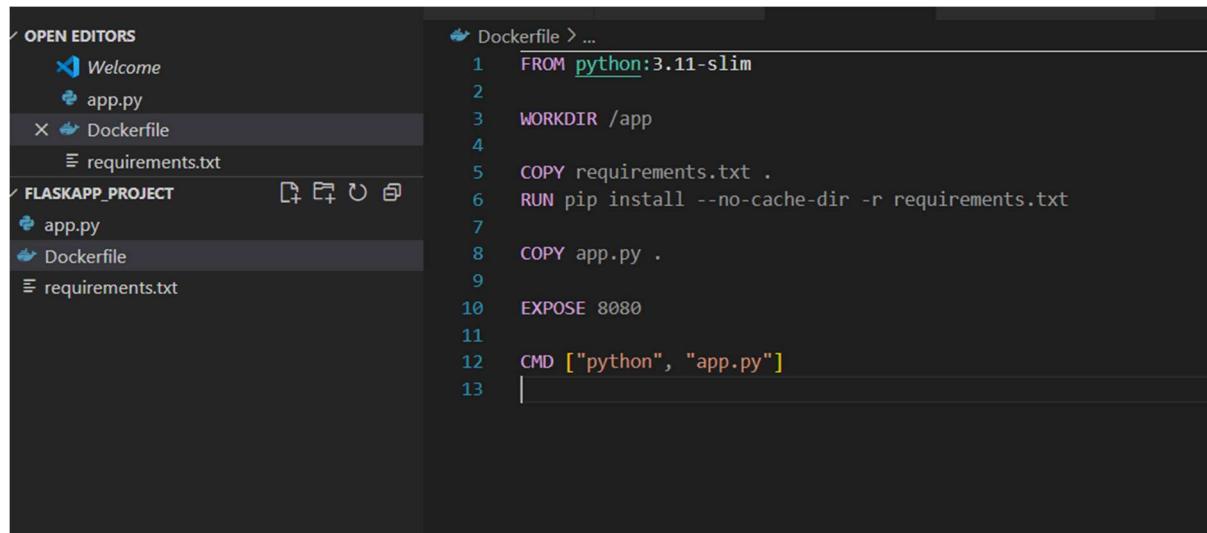
The screenshot shows the AWS Management Console Home page for the "United States (N. Virginia)" region. The top navigation bar includes the AWS logo, a search bar, and a "Console Home" link. The main content area is divided into several sections:

- Recently visited:** Lists services including S3, IAM, VPC, Aurora and RDS, EC2, CloudWatch, Lambda, and API Gateway.
- Welcome to AWS:** Includes links for "Getting started with AWS" and "Training and certification".
- AWS Health:** Shows 0 open issues and 0 scheduled changes.
- Cost and usage:** Displays credits remaining (\$158.62 USD), current month costs (\$1.34), and a note about free plan costs.
- Applications:** Shows 0 applications created.
- Regions:** A sidebar on the right lists 16 regions, with "N. Virginia" highlighted as the current region. Other regions listed include Ohio, N. California, Oregon, Hyderabad, Mumbai, Osaka, Seoul, Singapore, Sydney, Tokyo, Canada, Central, Europe, Frankfurt, Ireland, London, Paris, Stockholm, South America, São Paulo, and Days 136.

Step 1 :- write all the dockerfile, flask app , requirements.txt files in the vs code

Code:-

Docker file:-



The screenshot shows the VS Code interface with the 'Dockerfile' tab selected in the left sidebar under the 'OPEN EDITORS' section. The right pane displays the Dockerfile code:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
EXPOSE 8080
CMD ["python", "app.py"]
```

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

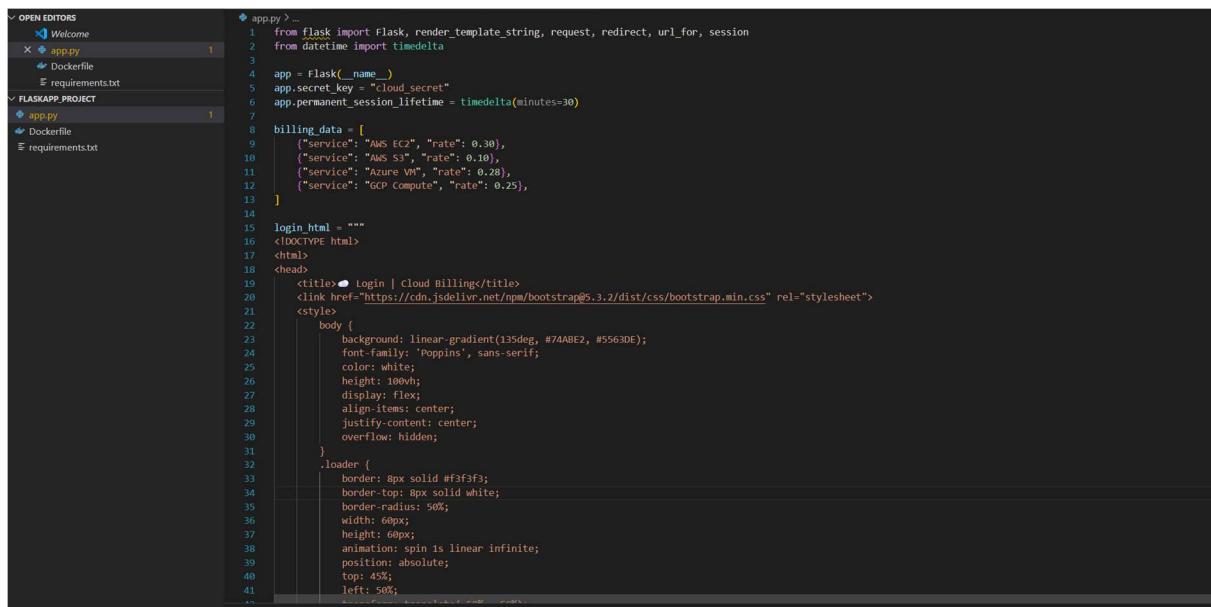
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py .

EXPOSE 8080

CMD ["python", "app.py"]

## PYTHON FILE WHICH CONTAINS THE FLASK APPLICATION:-



```
OPEN EDITORS
  Welcome
  app.py 1
  Dockerfile
  requirements.txt

FLASKAPP_PROJECT
  app.py 1
  Dockerfile
  requirements.txt

app.py > ...
1  from flask import Flask, render_template_string, request, redirect, url_for, session
2  from datetime import timedelta
3
4  app = Flask(__name__)
5  app.secret_key = "cloud_secret"
6  app.permanent_session_lifetime = timedelta(minutes=30)
7
8  billing_data = [
9      {"service": "AWS EC2", "rate": 0.30},
10     {"service": "AWS S3", "rate": 0.10},
11     {"service": "Azure VM", "rate": 0.28},
12     {"service": "GCP Compute", "rate": 0.25},
13 ]
14
15  login_html = """
16  <!DOCTYPE html>
17  <html>
18  <head>
19      <title>>Login | Cloud Billing</title>
20      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
21  <style>
22      body {
23          background: linear-gradient(135deg, #74ABE2, #5563DE);
24          font-family: 'Poppins', sans-serif;
25          color: white;
26          height: 100vh;
27          display: flex;
28          align-items: center;
29          justify-content: center;
30          overflow: hidden;
31      }
32      .loader {
33          border: 8px solid #f3f3f3;
34          border-top: 8px solid white;
35          border-radius: 50%;
36          width: 60px;
37          height: 60px;
38          animation: spin 1s linear infinite;
39          position: absolute;
40          top: 45%;
41          left: 50%;
```

```
from flask import Flask, render_template_string, request, redirect, url_for, session
from datetime import timedelta
```

```
app = Flask(__name__)
app.secret_key = "cloud_secret"
app.permanent_session_lifetime = timedelta(minutes=30)
```

```
billing_data = [
    {"service": "AWS EC2", "rate": 0.30},
    {"service": "AWS S3", "rate": 0.10},
    {"service": "Azure VM", "rate": 0.28},
    {"service": "GCP Compute", "rate": 0.25},
]
```

```
login_html = """
<!DOCTYPE html>
<html>
<head>
    <title>>Login | Cloud Billing</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
<style>
    body {
```

background: linear-gradient(135deg, #74ABE2, #5563DE);  
font-family: 'Poppins', sans-serif;

```

        color: white;
        height: 100vh;
        display: flex;
        align-items: center;
        justify-content: center;
        overflow: hidden;
    }
    .loader {
        border: 8px solid #f3f3f3;
        border-top: 8px solid white;
        border-radius: 50%;
        width: 60px;
        height: 60px;
        animation: spin 1s linear infinite;
        position: absolute;
        top: 45%;
        left: 50%;
        transform: translate(-50%, -50%);
    }
    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
    .card {
        display: none;
        animation: fadeIn 1s ease-in;
    }
    @keyframes fadeIn {
        from { opacity: 0; transform: translateY(-20px); }
        to { opacity: 1; transform: translateY(0); }
    }
</style>
</head>
<body>
    <div class="loader"></div>
    <div class="card bg-light text-dark p-4 rounded shadow" style="width: 22rem;">
        <h4 class="mb-3 text-center"> Cloud Billing Login</h4>
        {% if error %}
            <div class="alert alert-danger">{{ error }}</div>
        {% endif %}
        <form method="POST">
            <input type="text" name="username" class="form-control mb-2" placeholder="Username" required>
            <input type="password" name="password" class="form-control mb-3" placeholder="Password" required>
            <button type="submit" class="btn btn-primary w-100">Login</button>
        </form>
    </div>
    <script>
        setTimeout(() => {
            document.querySelector('.loader').style.display = 'none';
            document.querySelector('.card').style.display = 'block';
        }, 2000);
    </script>

```

```

</script>
</body>
</html>
"""

dashboard_html = """
<!DOCTYPE html>
<html>
<head>
    <title>☁ Dashboard | Cloud Billing</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background: linear-gradient(135deg, #e0ecff, #ffffff);
            font-family: 'Poppins', sans-serif;
        }
        .table-container {
            margin-top: 40px;
        }
        .total {
            font-size: 1.2rem;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <nav class="navbar navbar-dark bg-primary px-4">
        <span class="navbar-brand mb-0 h1">☁ Cloud Billing Dashboard</span>
        <form action="/logout" method="get">
            <button class="btn btn-light">Logout</button>
        </form>
    </nav>

    <div class="container table-container">
        <table class="table table-bordered table-hover">
            <thead class="table-primary">
                <tr>
                    <th>Service</th>
                    <th>Rate ($/hr)</th>
                    <th>Quantity</th>
                </tr>
            </thead>
            <tbody>
                {%
                    for row in billing_data %
                %}
                <tr>
                    <td>{{ row.service }}</td>
                    <td>{{ row.rate }}</td>
                    <td><input type="number" min="0" value="0" class="form-control" onchange="updateTotal()"/></td>
                </tr>
                {%
                    endfor %
                %}
            </tbody>
        </table>
    </div>

```

```

</table>
<div class="total text-center">Total Cost: $<span id="total">0.00</span></div>
</div>

<script>
const rates = {{ billing_data | toJson }};
function updateTotal() {
  let total = 0;
  const inputs = document.querySelectorAll("input[type='number']");
  inputs.forEach((input, i) => {
    const qty = parseFloat(input.value) || 0;
    total += qty * rates[i].rate;
  });
  document.getElementById("total").innerText = total.toFixed(2);
}
</script>
</body>
</html>
"""

logout_html = """
<!DOCTYPE html>
<html>
<head>
<title>👋 Goodbye</title>
<style>
body {
  background: linear-gradient(135deg, #74ABE2, #5563DE);
  font-family: 'Poppins', sans-serif;
  color: white;
  height: 100vh;
  margin: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
.cloud {
  font-size: 2rem;
  background: white;
  color: #5563DE;
  padding: 30px;
  border-radius: 50px;
  box-shadow: 0 0 20px rgba(0,0,0,0.2);
  animation: float 3s ease-in-out infinite;
  margin-bottom: 20px;
}
@keyframes float {
  0% { transform: translateY(0px); }
  50% { transform: translateY(-20px); }
  100% { transform: translateY(0px); }
}
iframe {
</style>

```

```

        border-radius: 15px;
        box-shadow: 0 0 15px rgba(0,0,0,0.3);
    }
</style>
</head>
<body>
    <div class="cloud">  Thanks for using my Cloud Billing Dashboard!</div>
    <iframe width="560" height="315"
src="https://www.youtube.com/embed/bbJUcwvKCsA?autoplay=1&loop=1&playlist=bbJUcwvKCs
A&mute=1" frameborder="0" allow="autoplay; encrypted-media" allowfullscreen></iframe>
</body>
</html>
"""

@app.route("/", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        user = request.form["username"]
        password = request.form["password"]
        if user == "admin" and password == "cloud123":
            session.permanent = True
            session["user"] = user
            return redirect(url_for("dashboard"))
        return render_template_string(login_html, error="Invalid credentials")
    return render_template_string(login_html)

@app.route("/dashboard")
def dashboard():
    if "user" not in session:
        return redirect(url_for("login"))
    return render_template_string(dashboard_html, billing_data=billing_data)

@app.route("/logout")
def logout():
    session.pop("user", None)
    return render_template_string(logout_html)

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=8080)

```

RQUIREMENTS.TXT FILE:-

```
requirements.txt
1 Flask==2.3.3
```

Flask==2.3.3

Step 2:- create a ecr repository before going for deployment.

```
roshan@Roshan-S:/mnt/c/Users/rosha/flaskapp_project$ aws ecr create-repository --repository-name flaskproductionapp --region us-east-1
{
```

TO CREATE THE REPOSITORY IN ECR (ELASTIC CONTAINER REGISTRY):-  
I CANNOT SHARE THE OUTPUT BCZ IT CONSISTS OF PERSONAL DETAILS.

THE BELOW SHOWS THE COMMAND USED:-

**aws ecr create-repository --repository-name flaskproductionapp --region us-east-1**

**AUTHENTICATING DOCKER TO AWS ECR COMMAND:-**

**aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <account id>.dkr.ecr.us-east-1.amazonaws.com/<repo-name>**

**OUTPUT:-**

```
roshan@Roshan-S:/
Login Succeeded
```

NOW AFTER THIS WE HAVE TO BUILD OUR DOCKER IMAGE(COMMAND USED IS):-

**docker build -t flaskproductionapp .**

output:-

```
Logfile successfully created.  
roshan@Roshan-S:/mnt/c/Users/rosha/flaskapp_project$ docker build -t flaskproductionapp .  
[+] Building 2.6s (11/11) FINISHED  
=> [internal] load build definition from Dockerfile  
=> => transferring dockerfile: 217B  
=> [internal] load metadata for docker.io/library/python:3.11-slim  
=> [auth] library/python:pull token for registry-1.docker.io  
=> [internal] load .dockerignore  
=> => transferring context: 2B  
=> [1/5] FROM docker.io/library/python:3.11-slim@sha256:8eb5fc663972b871c528fef04be4eaa9ab8ab4539a5316c4b8c133771214a617  
=> => resolve docker.io/library/python:3.11-slim@sha256:8eb5fc663972b871c528fef04be4eaa9ab8ab4539a5316c4b8c133771214a617  
=> [internal] load build context  
=> => transferring context: 7.24kB  
=> CACHED [2/5] WORKDIR /app  
=> CACHED [3/5] COPY requirements.txt .  
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt  
=> CACHED [5/5] COPY app.py .  
=> exporting to image  
=> => exporting layers  
=> => exporting manifest sha256:e0e085a62264f3e901164fa40caaf5ac62972665120fb4c59a620eeb3a86c0f7  
=> => exporting config sha256:069aeeb355bf0a56aa4cf986e012930d774cb7e86fc2d0d8f2e3c7824cb6d82e  
=> => exporting attestation manifest sha256:8c3539a9e46fbaa946907dc7a59ff9bef16c6e661cf6794ebcef591813d9b2ed  
=> => exporting manifest list sha256:4275e4a71fa50572a9f36ab1e80addcc114572a8448e64128918ddb0e8752f89  
=> => naming to docker.io/library/flaskproductionapp:latest  
=> => unpacking to docker.io/library/flaskproductionapp:latest
```

NEXT I WILL TAG THE IMAGE FOR ECR:-

```
docker tag flaskproductionapp:latest <your-account-id>.dkr.ecr.us-east-1.amazonaws.com/flaskproductionapp:latest
```

next I will push the image to ecr:-

```
docker push <your-account-id>.dkr.ecr.us-east-1.amazonaws.com/flaskproductionapp:latest
```

OUTPUT:-

```
roshan@Roshan-S:/mnt/c/Users/r  
he push refers to repository  
eca3b45c2a0: Pushed  
daa4f9aedb5: Pushed  
352dfa3f357: Pushed  
c645de2d9ac: Pushed  
db477cf87de: Pushed  
73850a50582: Pushed  
9fb8589da02: Pushed  
9ffe18d7fdb: Pushed  
8513bd72563: Pushed  
atest: digest: sha256:4275e4a
```

It has pushed all the layers of the image to the ecr or the repo which I created for ecr

Step 3:- AFTER THIS WE HAVE TO SEARCH FOR ECS(ELASTIC CONTAINER SERVICE) AND WE HAVE TO CREATE A CLUSTER FOR FLASKAPPPRODUCTION.

Cluster name is :- flask-cluster and iam using fargate only.next iam going to create it.

New! ECS Managed Instances  
Get started with ECS Managed Instances with flexible hardware capabilities and reduced operational overheads. Configure in the infrastructure section below. Your cluster is automatically configured for AWS Fargate (serverless) with Fargate and Fargate Spot. [Learn more](#)

**Cluster configuration**

Cluster name: flask-cluster  
Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

► [Service Connect defaults – optional](#)

▼ **Infrastructure - advanced** Info  
Configure the manner of obtaining compute resources that will be used to host your application.

Select a method of obtaining compute capacity  
Your cluster is automatically configured for AWS Fargate (serverless), but you may choose to add Amazon EC2 instances (servers).

- Fargate only**  
Serverless – you don't think about creating or managing servers. Great for most common workloads.
- Fargate and managed instances** New  
Managed instances – Amazon ECS will manage patching and scaling on your behalf while giving you configurability about the types of instances. Great for more advanced workloads.
- Fargate and Self-managed instances**  
Self-managed instances - you must ensure the instances are patched and scaled properly, and you have full control over the instances.

▼ **Monitoring - optional**  
Configure observability, encryption and logging options to maintain compliance and operational visibility of your container environment.

**Container Insights** Info  
CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices.

Select the level of observability you want to achieve with Container Insights  
Container Insights is turned off by default in your account settings. You can override that here at the cluster level.

- Container Insights with enhanced observability** Recommended  
Provides detailed health and performance metrics at task and container level in addition to aggregated metrics at cluster and service level. Enables easier drill downs for faster problem isolation and troubleshooting.
- Container Insights**  
Provides aggregated metrics at cluster and service level. You can run deep dive analysis with logs insights analytics.
- Turned off**

Specify the infrastructure requirements for the task definition.

**Launch type** | [Info](#)  
Selection of the launch type will change task definition parameters.

**AWS Fargate**  
Serverless compute for containers.

**Managed instances - new**  
Use if you have specific hardware constraints, such as GPU accelerators, CPU instruction sets, or network-optimized hardware (offloads scaling, patching, and instance management to AWS).

**Amazon EC2 instances**  
Self-managed infrastructure using Amazon EC2 instances.

**OS, Architecture, Network mode**  
Network mode is used for tasks and is dependent on the compute type selected.

**Operating system/Architecture** | [Info](#)      **Network mode** | [Info](#)

Linux/X86\_64      awsbatch

**Task size** | [Info](#)  
Specify the amount of CPU and memory to reserve for your task.

CPU	Memory
.5 vCPU	1 GB

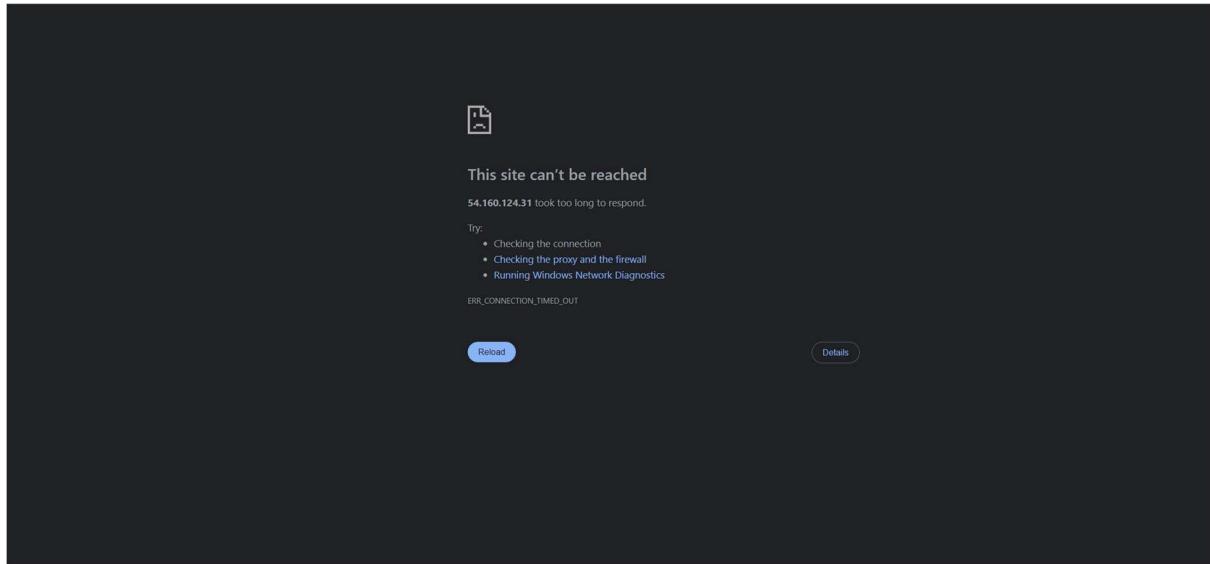
The memory and cpu gives a huge cost so always take the minimum that is 0.5vcpu and 1gb memory. To reduce the cost.

Here some few things to consider one is the security group handling was very important which I was facing as a challenge but I was able to resolve it and was able to allow port 8080 traffic initially the application was not running bcz there was nothing open to port 80 instead the entire application needed to run in port 8080.

So always choose the right container and application port otherwise the application will not work and this scenario happened to me yesterday today I fixed it.

The below figure was the issue and I fixed it using security groups and allowing inbound rule access

for port 8080.



This is the ecs cluster which is created.

A screenshot of the AWS CloudWatch Metrics interface. The top navigation bar shows "Clusters (1)" and "info". On the right, there are buttons for "Last updated" (1 November 2025, 09:00 (UTC+5:30)), "Create cluster", and a refresh icon. Below the navigation is a search bar labeled "Search clusters". The main table has columns: Cluster, Services, Tasks, Container instances, CloudWatch monitoring, and Capacity provider strategy. One row is shown for "flask-cluster": Services 0, Tasks 0 pending | 1 running, Container instances 0 EC2, CloudWatch monitoring Default, Capacity provider strategy No default found.

The security group was not mentioned in my tasks that is why the issue persisted.

This is the flask-task which was created for the ecs.

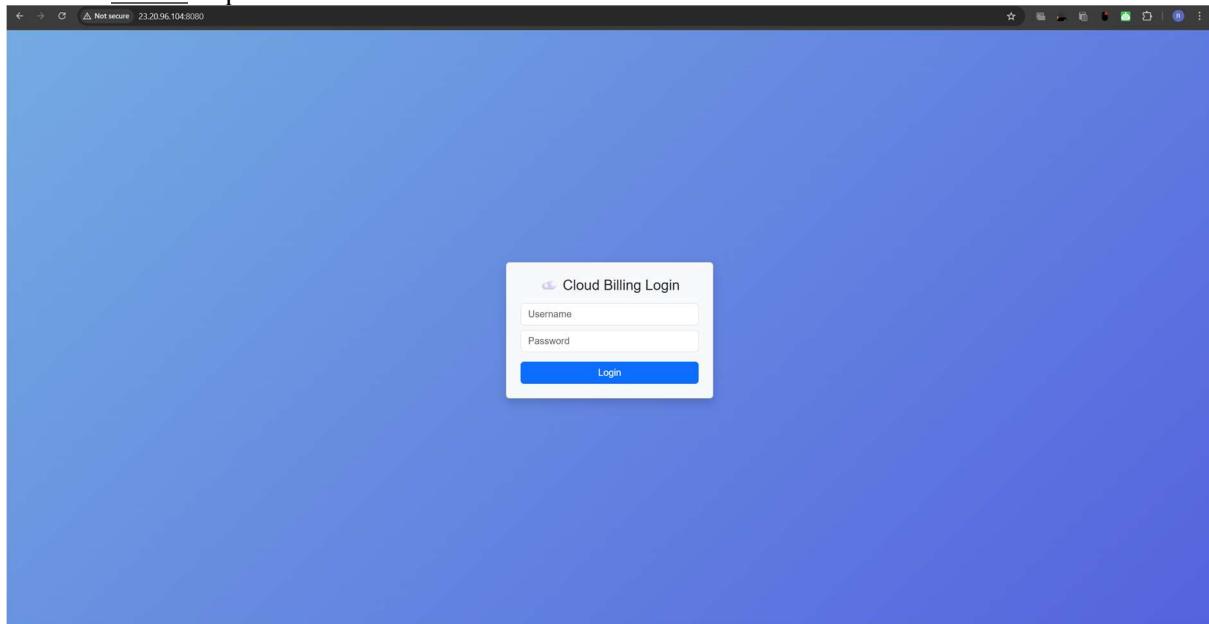
A screenshot of the AWS CloudWatch Metrics interface. The top navigation bar shows "flask-task (1)" and "Info". On the right, there are buttons for "Last updated" (1 November 2025, 09:05 (UTC+5:30)), "Deploy", "Actions", "Create new revision", and a refresh icon. Below the navigation is a search bar labeled "Filter task definition revisions by value" and a dropdown for "Filter status" set to "Active". The main table has columns: Task definition: revision and Status. One row is shown for "flask-task:1": Status ACTIVE.

Task details:-

Configuration			
Operating system/Architecture Linux/X86_64	Capacity provider Fargate	ENI ID <a href="#">eni-002f0d0adc7a3e570</a>	Public IP <a href="#">23.20.96.104</a>   <a href="#">open address</a>
CPU   Memory .5 vCPU   1 GB	Launch type Fargate	Network mode awsvpc	Private IP <a href="#">172.31.41.99</a>
Platform version 1.4.0	Task definition: revision <a href="#">flask-task:1</a>	Subnet <a href="#">subnet-0d5cabf48603fed34</a>	MAC address <a href="#">0e:82:64:20:5a:47</a>
Task execution role <a href="#">ecsTaskExecutionRole</a>	Task group family:flask-task		
Task role -			
Fault injection Turned off			
ECS Exec   Info			

This is the application that is developed.A “CLOUD BILLING DASHBOARD”

Website URL:- <http://23.20.96.104:8080/>



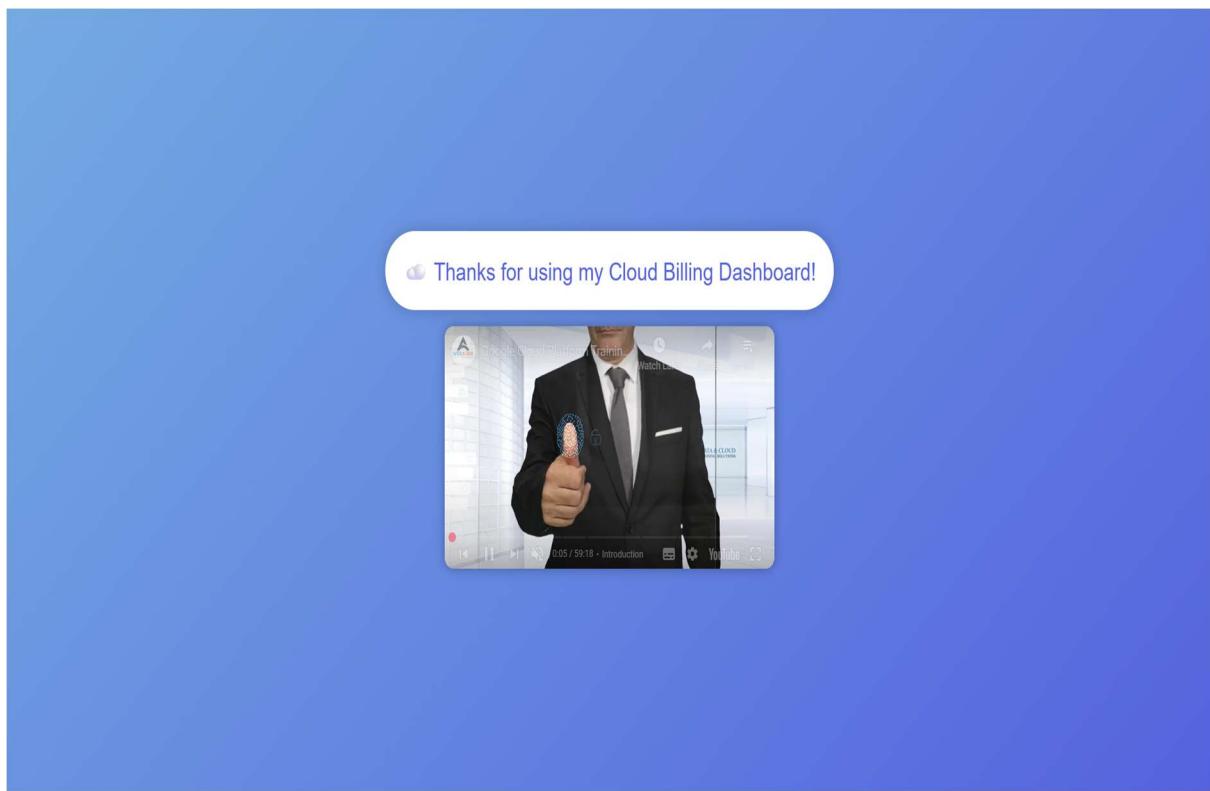
Cloud Billing Dashboard

Service	Rate (\$/hr)	Quantity
AWS EC2	0.3	13
AWS S3	0.1	16
Azure VM	0.28	11
GCP Compute	0.25	19

Total Cost: \$13.33

Can calculate the sum of quantity which is purchased we can increase or decrease the quantity as needed.

Now when iam going to logout the youtube video will play for it that is showing how we can use use cloud billing dashboard in gcp.





## THESE WERE THE CHALLENGES WHICH I FACED.

Deployment Challenges Faced:-



Encountered InvalidClientTokenId when running aws sts get-caller-identity

Root cause: expired or misconfigured credentials in `~/.aws/credentials`

Solution: regenerated IAM access keys and reconfigured using `aws configure`



Error: docker login requires at least one argument

Cause: missing ECR registry URL in the login command

Fix: added full registry URL using `--password-stdin` with account ID



Uncertainty whether image was successfully pushed to ECR

Verified using `aws ecr list-images` to confirm presence of latest tag



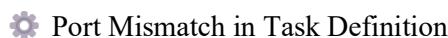
Browser showed `ERR_CONNECTION_TIMED_OUT` when accessing public IP

Cause: ECS task was running, but port access was blocked



Couldn't modify inbound rules to allow port 8080

Solution: created a new security group with custom TCP rule for port 8080 and attached it to ECS task



App was running on port 8080, but ECS container was mapped to port 80

Fix: revised task definition to match container port with app port

Feature	Docker	Amazon ECR	Amazon ECS
Purpose	Build and run containers locally	Store container images in AWS	Deploy and manage containers in AWS
Type	Container engine	Container registry	Container orchestration service
Used for	Packaging apps into images	Hosting images for deployment	Running containers on AWS infrastructure
Where it runs	Developer machine or	AWS cloud	AWS cloud (Fargate or EC2)

Feature	Docker	Amazon ECR	Amazon ECS
	server		
Key command	docker build, docker run	docker push, docker pull	aws ecs run-task, Console setup
Authentication	Local only	Requires AWS credentials	Requires AWS setup and networking
Storage	Local disk	Cloud-based image storage	No storage; pulls image from ECR
Integration	Works with ECR and ECS	Feeds images to ECS	Pulls images from ECR to run containers

#### RECENT ADVANCEMENTS IN 2025:-

Category	Use Case / Advancement	Description
AI Dashboards	Real-time ML model monitoring	Flask apps serve dashboards with live predictions, ECS auto-scales based on traffic.
Fintech	Transaction APIs & billing platforms	Flask handles secure endpoints; ECS ensures zero-downtime scaling and patching.
IoT & Edge Data	Sensor data ingestion and visualization	Flask apps stream and display real-time device data; ECS handles burst loads.
Healthcare	Patient vitals dashboard	Flask apps visualize live metrics; ECS ensures HIPAA-compliant isolation and scaling.
E-Learning	Interactive coding platforms & quizzes	Flask powers dynamic content; ECS scales with student load during exams or launches.
DevOps Tools	Internal dashboards for CI/CD pipelines	Flask apps track deployments and logs; ECS integrates with CloudWatch and IAM roles.
Retail & E-Commerce	Flash sale engines and inventory trackers	Flask apps update stock and pricing in real time; ECS handles traffic spikes.
Customer Support	Live ticketing and chatbot interfaces	Flask routes support queries; ECS ensures high availability during peak hours.
Media & Streaming	Personalized content feeds and recommendation engines	Flask serves user-specific content; ECS scales horizontally with viewer demand.

Category	Use Case / Advancement	Description
Smart Cities	Public dashboards for traffic, pollution, and utilities	Flask apps visualize civic data; ECS supports secure, multi-tenant deployments.

This is how I ran my container for the application to launch in vs code this locally from my system.

```
Debugger PIN: 521 201 727
roshan@Roshan-S:~/mnt/c/Users/rosha/flaskapp_project$ docker run -p 8080:8080 flaskproductionapp
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server inst
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://172.17.0.2:8080
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 503-056-389
172.17.0.1 - - [01/Nov/2025 03:53:47] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [01/Nov/2025 03:53:49] "GET /favicon.ico HTTP/1.1" 404 -
```

Conclusion:- could understand how to debug and fix the security groups and allow inbound traffic for port 8080 for the application developed in vs code. I have used ecr, ecs and docker desktop for running my application. And built a flask application using python language.