Aaro7 - Helping MSMEs Rise On Time

Job Title: Cloud Engineer – AWS Infrastructure (Fintech Platform)

<u>Screening Test</u>

Section 1: EC2 + App Hosting

1. (Hands-on) Spin up an EC2 instance and deploy a simple FastAPI app with one route (/ping → {"message": "pong"})

Host it using Gunicorn and Nginx
Ensure it is accessible on http://your-ec2-ip/ping

Attach screenshots of:

- Your Nginx config
- Running process (Gunicorn)
- Browser output of the endpoint

Step 1 : Launch EC2 Instance

Step 2 : Connect to EC2

```
sudo yum update -y && sudo yum install -y
python3-pip python3-venv nginx
```

Step 3 : Create FastAPI app

```
mkdir fastapi-app && cd fastapi-app
python3 -m venv venv
source venv/bin/activate
pip install fastapi uvicorn gunicorn
```

Step 4 : Create main.py

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/ping")
def ping():
    return {"message": "pong"}
```

Step 5 : Run Gunicorn

```
gunicorn main:app -k uvicorn.workers.UvicornWorker --bind 127.0.0.1:8000
```

Step 5 : Install l& Configure Nginx

```
sudo apt install nginx -y

sudo vi /etc/nginx/sites-available/fastapi

server {
    listen 80;
    server_name _;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}


sudo amazon-linux-extras enable nginx1

sudo yum install nginx -y

sudo systemctl enable nginx

sudo systemctl start nginx

 sudo nginx -t

        nginx: configuration file /etc/nginx/nginx.conf test is successful
```

sudo vi /etc/nginx/nginx.conf

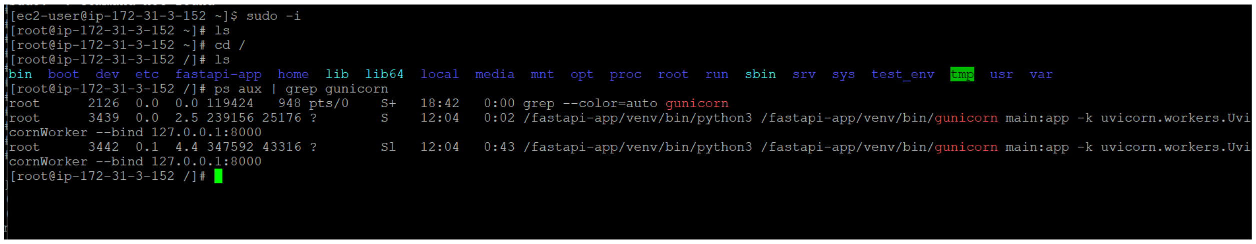
sudo systemctl restart nginx

__OUTPUT__ :

Let Check through : http://52.66.235.166/ping

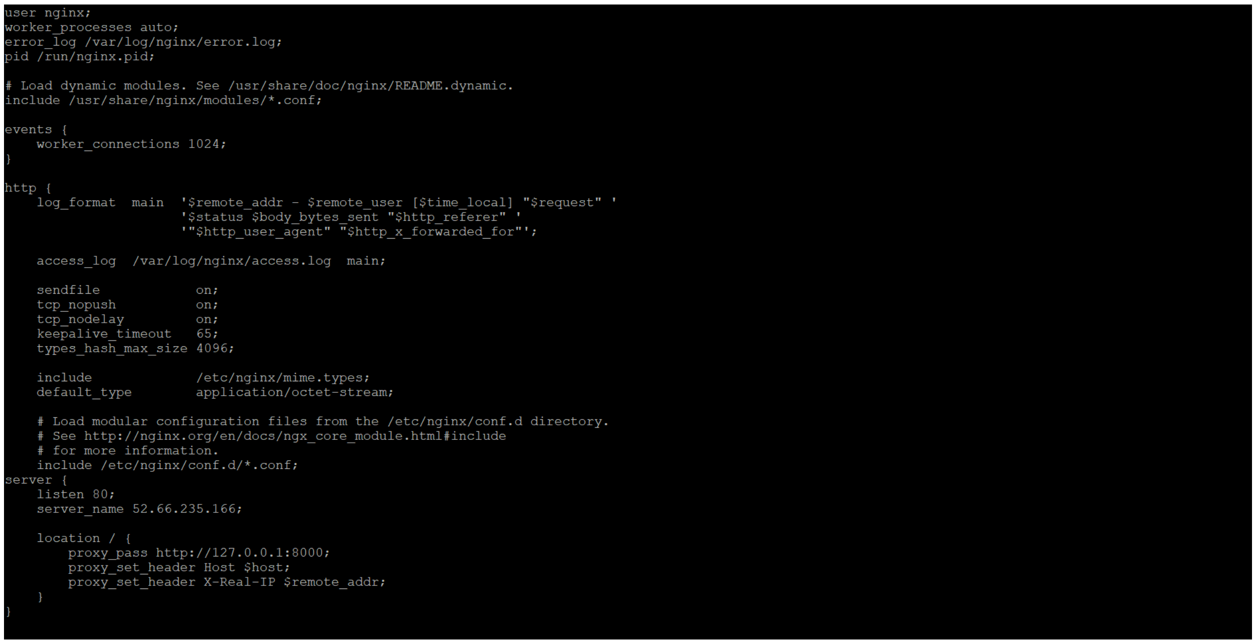
```
{
  "message": "pong"
}
```

Running process (Gunicorn)



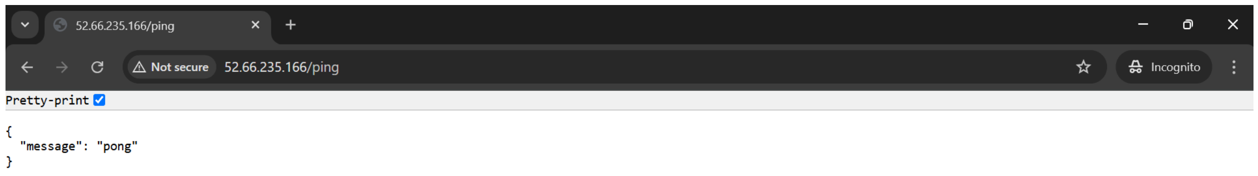
```
[ec2-user@ip-172-31-3-152 ~]$ sudo -i
[root@ip-172-31-3-152 ~]# ls
[root@ip-172-31-3-152 ~]# cd /
[root@ip-172-31-3-152 /]# ls
bin  boot  dev  etc  fastapi-app  home  lib  lib64  local  media  mnt  opt  proc  root  run  sbin  srv  sys  test_env  tmp  usr  var
[root@ip-172-31-3-152 /]# ps aux | grep gunicorn
root      2126  0.0  0.0 119424   948 pts/0    S+   18:42   0:00 grep --color=auto gunicorn
root      3439  0.0  2.5 239156 25176 ?        S    12:04   0:02 /fastapi-app/venv/bin/python3 /fastapi-app/venv/bin/gunicorn main:app -k uvicorn.workers.Uvi
cornWorker --bind 127.0.0.1:8000
root      3442  0.1  4.4 347592 43316 ?        Sl   12:04   0:43 /fastapi-app/venv/bin/python3 /fastapi-app/venv/bin/gunicorn main:app -k uvicorn.workers.Uvi
cornWorker --bind 127.0.0.1:8000
[root@ip-172-31-3-152 /]#
```

Your Nginx config



```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 4096;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;
server {
    listen 80;
    server_name 52.66.235.166;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

Browser output of the endpoint



52.66.235.166/ping

Not secure 52.66.235.166/ping

Pretty-print ✓

```
{
  "message": "pong"
}
```

2. What security measures would you take to harden an EC2 instance serving financial APIs?

1) Instance-Level Security

    Use a Hardened OS, Restrict SSH Access

2) Network Security

    Security Groups , VPC Configuration

3) Application-Level Security

    HTTPS with TLS , API Authentication & Authorization , Input Validation & Rate Limiting

4) Monitoring & Logging

    Enable Logging

5) IAM & Access Control

6) Backups & Recovery - Amazon EBS snapshots

<u>**Section 2: AWS Infra**</u>

3. Explain how you would:

Configure a PostgreSQL database using RDS
Connect it securely to the EC2 instance
Back it up regularly

Configure a PostgreSQL Database Using RDS

1) Create Database by using the **PostgreSQL Database -Standard**
2) Choose the **Available versions of pg sql**
3) Choose **Production** for high availability, or **Dev/Test** for development environments
4) Set a **DB Instance Name** Make the settings and the credential access settings
5) Choose DB Instance Class t3.micro
6) Choose the availability and durability
7) Connect to the instance and choose the vpc and private - subnets as per the instance
8) Not to grant for public access for (security)
9) Get Back ups on regular basis

Connect RDS PostgreSQL Securely to EC2

1) Security Group Configuration - inbound rule - port (5432)

2) Connect to Terminal

```
sudo yum install postgresql -y
psql -h <endpoint> -U <username> -d <db-name> -W
```

3) Use Secrets Manager
4) Use SSL certificates from AWS RDS documentation

Back It Up Regularly

1) Set up backup retention period
2) AWS automatically performs daily snapshots

3) Ensures high availability by replicating data to a standby instance in a different AZ
4) Use snapshot copy to replicate backups across regions for disaster recovery
5) Use **AWS CloudWatch** to monitor backup operation
6) Set alarms for failed backups or storage thresholds

4. If your FastAPI service uploads files to S3:

How would you give it minimum permissions to upload to only a specific folder?

1) Create an IAM Role
2) Attach this role to your EC2 instance running FastAPI.
3) Limit Permissions :
    a) allow s3:PutObject to the specific folder like `uploads/`
    b) allow s3:ListBucket need to verify object existence

Share a sample IAM policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowPutObjectOnlyToSpecificFolder",
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::mys3bucket1705/uploads/*"
    },
    {
      "Sid": "AllowListBucketForUploadsPrefix",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3::::mys3bucket1705",
      "Condition": {
        "StringLike": {
          "s3:prefix": "uploads/*"
        }
      }
    }
  ]
}
```

Boto3 for S3 access

```
import boto3

s3 = boto3.client('s3')
bucket_name = “mys3bucket1705"

def upload_file_to_s3(file, filename):
    s3.upload_fileobj(file, bucket_name, f"uploads/{filename}")
```

Section 3: Monitoring & Automation

5. What tools would you use to:
Monitor uptime and errors?

To monitor uptime and detect errors in a **production-grade FastAPI application on EC2**

1) Amazon CloudWatch
2) Prometheus + Grafana

Trigger alerts if memory or CPU usage goes beyond 80%?

To trigger alerts when system resources exceed a threshold -

1) Amazon CloudWatch Alarms

   set EC2 **CPUUtilization** metri

   Configure **threshold** at > `80%` for CPU or memory

   Trigger **SNS notification** to email, SMS, or Lambda.

2) CloudWatch Agent - Install on EC2 to collect **custom metrics** like memory, disk, and processes.

6. Create a systemd unit file that ensures the Gunicorn-based FastAPI app starts on reboot and restarts on failure

1) Create the Unit File

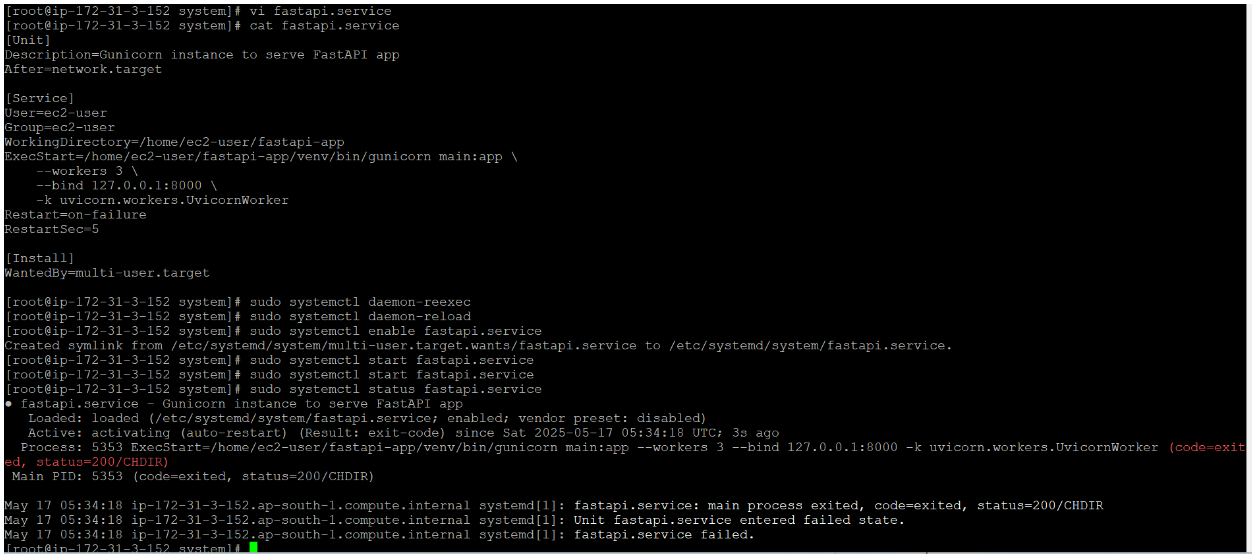
```
sudo vim /etc/systemd/system/fastapi.service

[Unit]
Description=Gunicorn instance to serve FastAPI app
After=network.target

[Service]
User=ec2-user
Group=ec2-user
WorkingDirectory=/home/ec2-user/fastapi-app
ExecStart=/home/ec2-user/fastapi-app/venv/bin/gunicorn main:app \
    --workers 3 \
    --bind 127.0.0.1:8000 \
    -k uvicorn.workers.UvicornWorker

Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable fastapi.service
sudo systemctl start fastapi.service
```

```
sudo systemctl status fastapi.service
```

OUTPUT

```
Active: active (running)
```



```
[root@ip-172-31-3-152 system]# vi fastapi.service
[root@ip-172-31-3-152 system]# cat fastapi.service
[Unit]
Description=Gunicorn instance to serve FastAPI app
After=network.target

[Service]
User=ec2-user
Group=ec2-user
WorkingDirectory=/home/ec2-user/fastapi-app
ExecStart=/home/ec2-user/fastapi-app/venv/bin/gunicorn main:app \
    --workers 3 \
    --bind 127.0.0.1:8000 \
    -k uvicorn.workers.UvicornWorker
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target

[root@ip-172-31-3-152 system]# sudo systemctl daemon-reexec
[root@ip-172-31-3-152 system]# sudo systemctl daemon-reload
[root@ip-172-31-3-152 system]# sudo systemctl enable fastapi.service
Created symlink from /etc/systemd/system/multi-user.target.wants/fastapi.service to /etc/systemd/system/fastapi.service.
[root@ip-172-31-3-152 system]# sudo systemctl start fastapi.service
[root@ip-172-31-3-152 system]# sudo systemctl start fastapi.service
[root@ip-172-31-3-152 system]# sudo systemctl status fastapi.service
● fastapi.service - Gunicorn instance to serve FastAPI app
   Loaded: loaded (/etc/systemd/system/fastapi.service; enabled; vendor preset: disabled)
   Active: activating (auto-restart) (Result: exit-code) since Sat 2025-05-17 05:34:18 UTC; 3s ago
  Process: 5353 ExecStart=/home/ec2-user/fastapi-app/venv/bin/gunicorn main:app --workers 3 --bind 127.0.0.1:8000 -k uvicorn.workers.UvicornWorker (code=exit
ed, status=200/CHDIR)
 Main PID: 5353 (code=exited, status=200/CHDIR)

May 17 05:34:18 ip-172-31-3-152.ap-south-1.compute.internal systemd[1]: fastapi.service: main process exited, code=exited, status=200/CHDIR
May 17 05:34:18 ip-172-31-3-152.ap-south-1.compute.internal systemd[1]: Unit fastapi.service entered failed state.
May 17 05:34:18 ip-172-31-3-152.ap-south-1.compute.internal systemd[1]: fastapi.service failed.
[root@ip-172-31-3-152 system]#
```

Section 4: Bonus

7. What are 2 cost optimization steps you'd recommend if this platform grows to handle 500+ MSMEs?

1. Use Auto Scaling + Spot Instances for EC2

Regular on-demand EC2 instances can be expensive at scale. So , Use **Auto Scaling Groups** to dynamically add/remove EC2 instances based on CPU or memory usage.

Mix in Spot Instances (up to 90% cheaper) for non-critical or stateless workloads like background workers or batch jobs.

2. Move to Serverless - (API + S3 + RDS)

. **Use API Gateway + Lambda** for request handling

. Store static content (files, images, docs) in S3 + CloudFront

. CloudFront speeds up content delivery to MSME users across

. Enable **RDS Auto Scaling**, turn on **storage autoscaling**