# Assessment

**Secret keys:**

Access key ID: AKIA2HVQ5OIGXXOVXKIQ

Access Key pwd: qZVbEQAoa5gKtCPDnsI5LqudkOIEDCCV5docNQ7Z

AWS VPC architecture

# To design a scalable and highly available architecture for a service using EC2 as hosts, here is the proposed plan with a focus on multiple microservices

## Architecture Plan

### High availability:

* To achieve high availability for each microservice 2 ec2 machines are created under 2 different availability zone.
* Create Auto Scaling group: Min of 2 instances need to be configured to achieve high availability

Two public subnets are created under two availability zone (there fore two ec2 machines are present under two different subnets)

* Application Load balancer is AWS managed resource, and it scales automatically with the load and availability is managed by AWS as per the SLA

**Time to start:** We have to make use of launching template and use them with minimum user data configuration which helps ec2 instances server the request quickly .

### Scalability:

This architecture is highly scalable.

**Load balancer**: Scales as per the number of requests (managed by AWS)

**EC2**: New instances are created automatically as per the scaling policy. Machines are created randomly under the chosen **availability zone**, there by making it zone redundant.

**Security:** Security group can be configured to allow the http traffic from only load balancer by updating the security group of EC2 machines with security group of LB in the ingress rules.

**Load Balancing**: Target groups allow the traffic to be distributed among the machines inside the target groups

### Security & Scalability using cloud front service

**By using Use CloudFront we achieve the following:**

* **Static Content Delivery**: To deliver cached static content (e.g., images, JavaScript, CSS) from edge locations near the users.
* **Dynamic Content Optimization**: To speed up the delivery of dynamic content (e.g., APIs) by using optimized edge locations.
* **DDoS Mitigation**: To protect your application against DDoS attacks by leveraging AWS Shield.
* **Cost Efficiency**: To reduce costs by offloading traffic from your origin servers to CloudFront edge locations.
* **Scalability**: When expecting unpredictable spikes in traffic and need to serve content globally without latency issues.

A diagram of a computer

Description automatically generated

Figure 1 : High level design

Implementation: ***Following infra has been implemented to validate the design using terraform.***

1. Target Group for Each Microservice

- Create a dedicated target group for each microservice to isolate and manage traffic effectively.

2. Application Load Balancer (ALB)

- Deploy an Application Load Balancer with the following configurations:

a. Create a dedicated listener for each microservice.

b. Map each listener to its corresponding target group.

c. Configure health checks for each target group to ensure service availability.

d. Attach a security group to the ALB to allow traffic only on required ports (e.g., HTTP/HTTPS).

3. VPC Design

- Set up a robust VPC architecture with:

a. At least \*\*two public subnets\*\*, each in a different availability zone, to ensure high availability.

b. Security groups:

- Allow incoming traffic on required ports (e.g., 80, 443) and port 22 for SSH access.

- Allow traffic from the security group associated with the load balancer.

4. EC2 Configuration

- Create a \*\*Launch Template\*\* based on an existing and fully configured EC2 instance running the required microservice.

5. Auto Scaling Group

- Implement an Auto Scaling Group (ASG) with the following configurations:

a. Ensure the ASG spans a minimum of \*\*two availability zones\*\*.

b. Use the \*\*Launch Template\*\* created in Step 4.

c. Associate the target group created in Step 2 with the ASG, enabling the load balancer to dynamically register instances as backends and distribute traffic evenly.

d. Configure auto-scaling policies based on performance metrics, such as triggering a scale-out action when CPU utilization exceeds 80%.

This architecture ensures scalability, fault tolerance, and high availability while maintaining an organized and professional approach to managing microservices.

**Note**:

ELB is completely managed by the AWS, and it scales as per the incoming traffic.

EC2 scaling should be managed by us with help of auto scaling group.

### Demo Screen shots implementing above design

Note: Below infra is create using terraform.

## Terraform

**Structure**:

Source code: [**https://github.com/srikanthmalta/aws\_assessment/tree/main/terraform**](https://github.com/srikanthmalta/aws_assessment/tree/main/terraform)

Ideally, we create a module for each service based on our design requirements and use modules to create infra. Due to time constraints i used resource blocks to create infra.

As shown in below image, configuration for reach resource is present in respective file starting with prefix “aws-“

**Launch configuration is no more supported on aws for auto scaling group. AWS supports only launch template on accounts created after May 31, 2023.**

So i have create terraform code with launch template

A screenshot of a computer program

Description automatically generated

Terraform plan to create Domain terraform-test in Route 53 and add the IP address of the ec2 as record under the domain has been commented.  
**Did not apply as purchasing the domain costs us 15 USD/year.** Commented the code in the terraform file.

A screenshot of a computer program

Description automatically generated

Formatting the json response of a api response **.**

1. Installed httpd service on the ec2 machine.
2. Copied the content into /var/www/html/service.json
3. Curl command to fetch the data
   1. http://<your-server-ip>/servers.json (<http://localhost/servers.json> - when running inside the machine )
4. Script for conversion

[root@ip-10-0-1-75 ~]# cat script.sh

#!/bin/bash

# Fetch the JSON file

curl -s http://localhost/servers.json -o servers.json

# Convert JSON to the desired format

jq -r '.data[] | "Host \(.hostname)\nUser \(.username)\nHostname \(.ip)"' servers.json > servers\_formatted.txt

echo "Formatted output saved to servers\_formatted.txt"

Validated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. Script for fetching servers which are in down state

A screen shot of a computer screen

Description automatically generated

Output:  
A screen shot of a computer

Description automatically generated

3.1.1 - what method would you use to connect to server in order to troubleshoot the issue at the source?

• 3.1.2 - given that you still don’t know what the issue might be, list a number of commands you would execute to better understand what is wrong with this environment, stating what insights this command would provide to you in helping you pinpoint the issue.

• 3.1.3 - you suspect the issue might be from the nginx service. How would you figure out where the error log is located and how will you inspect this log?

• 3.1.4 - you suspect the issue might be from the php-fpm. How would you figure out where the error log is located and how will you inspect this log?

• 3.1.5 - you suspect the issue might be from a docker container. Assuming the docker container is writing logs to the default log driver, how would you inspect the logs generated by this container?

• 3.1.6 - you suspect the issue might lie somewhere else. Which logs would you investigate to try and find out what the issue is?

### Answers:

In order to connect to the server either use ssh command or use putty terminal

Ssh command : **ssh -i private.em ec2-user@<ip address>**

**Trouble shooting**

* Top command: Provides insights into CPU, memory usage, and running processes.
* df -h : To check for the disk storage
* free -h : To check for memory utilization
* ps -ef | grep <service > : to validate if the process are running
* docker commands to check if the docker containers are running fine
  + docker ps => list all running containers
  + docker logs <container id> => for container logs
* Nginix logs : look for nginx logs directly ( /var/log/nginx/\*.logs for nginx logs )
  + Tail -f to continuedly read the latest logs

## Docker

**Prerequisite:** Install docker on amazon ec2 and confirm that its working as expected.

Reference : [Installing Docker to use with the AWS SAM CLI - AWS Serverless Application Model](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/install-docker.html)

Docker command to build the Docker image

* Docker build -t express-app .

Note: Make sure we have we have all the steps to build the image in file named Dockerfile

Docker command to run image in interactive mode.

* docker run -it -p 3000:3000 express-app

Note:

* You can open the ec2 machine in different terminal and make a curl command to validate the response.
* To access from the browser, make sure ec2 machine has public IP and port 3000 open.

URL to access the from the browser

* <http://localhost:3000>
* http://<ec2 public ip>:3000

A screenshot of a computer

Description automatically generated

Command to run in detached mode

* docker run -d -p 3000:3000 express-app

Command to view the docker logs

* docker logs <container-id>

Command to run image in interactive mode while allowing to have access to source code.

* docker run -it -v /path/to/source/code:/usr/src/app -p 3000:3000 express-app

Note: By mounting source code directory of your repository (on your **host machine**) to a directory inside the Docker container. This allows you to make changes to the code on the host machine, and those changes will be immediately reflected in the container.

-it = Interactive mode

-v = mounts source code directory

-p = exposed the port

This approach is very useful for **development environments** because:

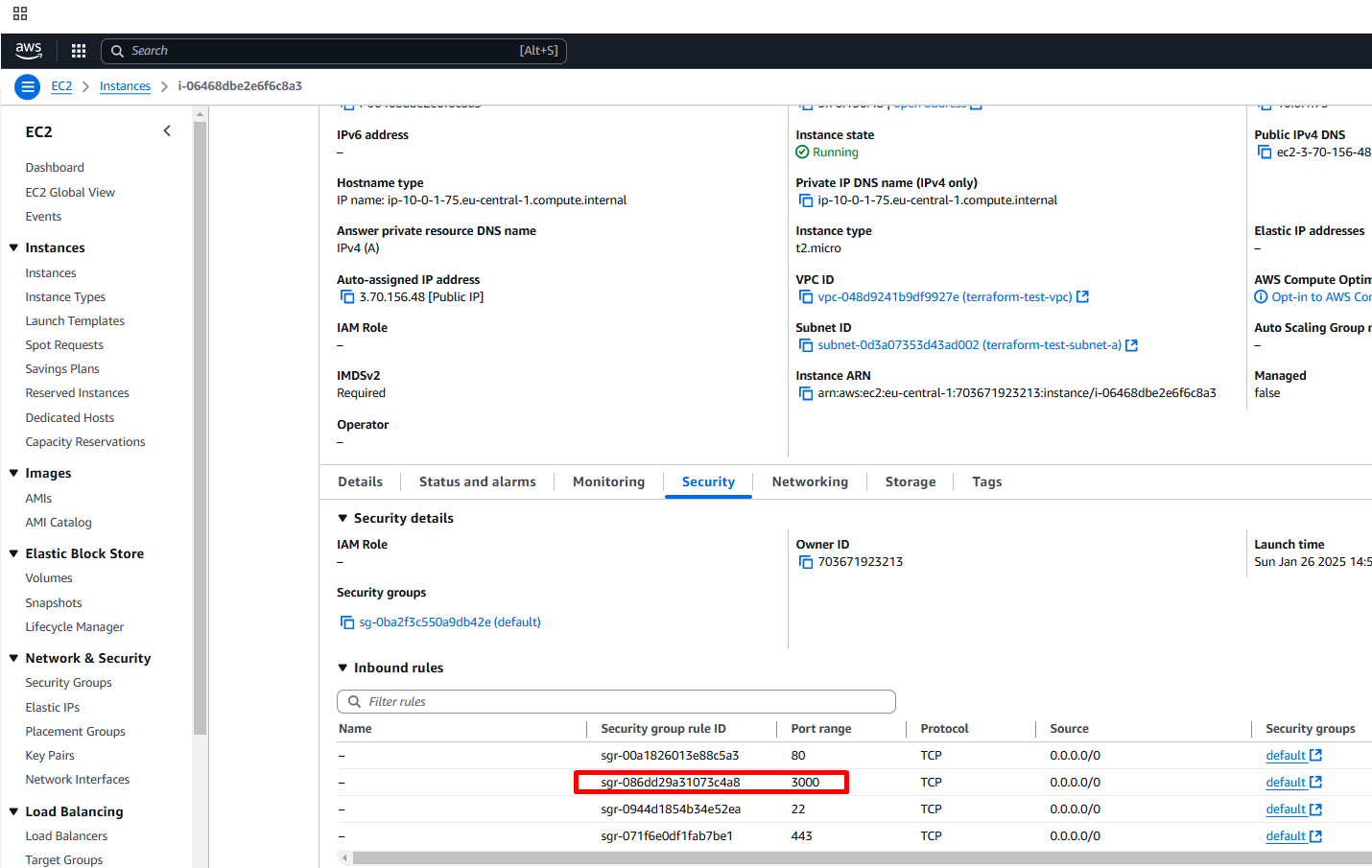
1. You don’t need to rebuild the Docker image after every code change.
2. You can test your changes in real time by refreshing your browser.

## Docker implementation validation:

A screen shot of a computer

Description automatically generated

Open port on security group of ec2 to access the endpoint from the public internet via browser by using public IP of ec2 machine



A screenshot of a computer

Description automatically generated