CO7219

INTERNET AND CLOUD COMPUTING

Cloud System Design, Evaluation and Reflections

SUBMITTED BY

Student Name: Mahesh Srivinay Rayavarapu

Student ID: 199047813

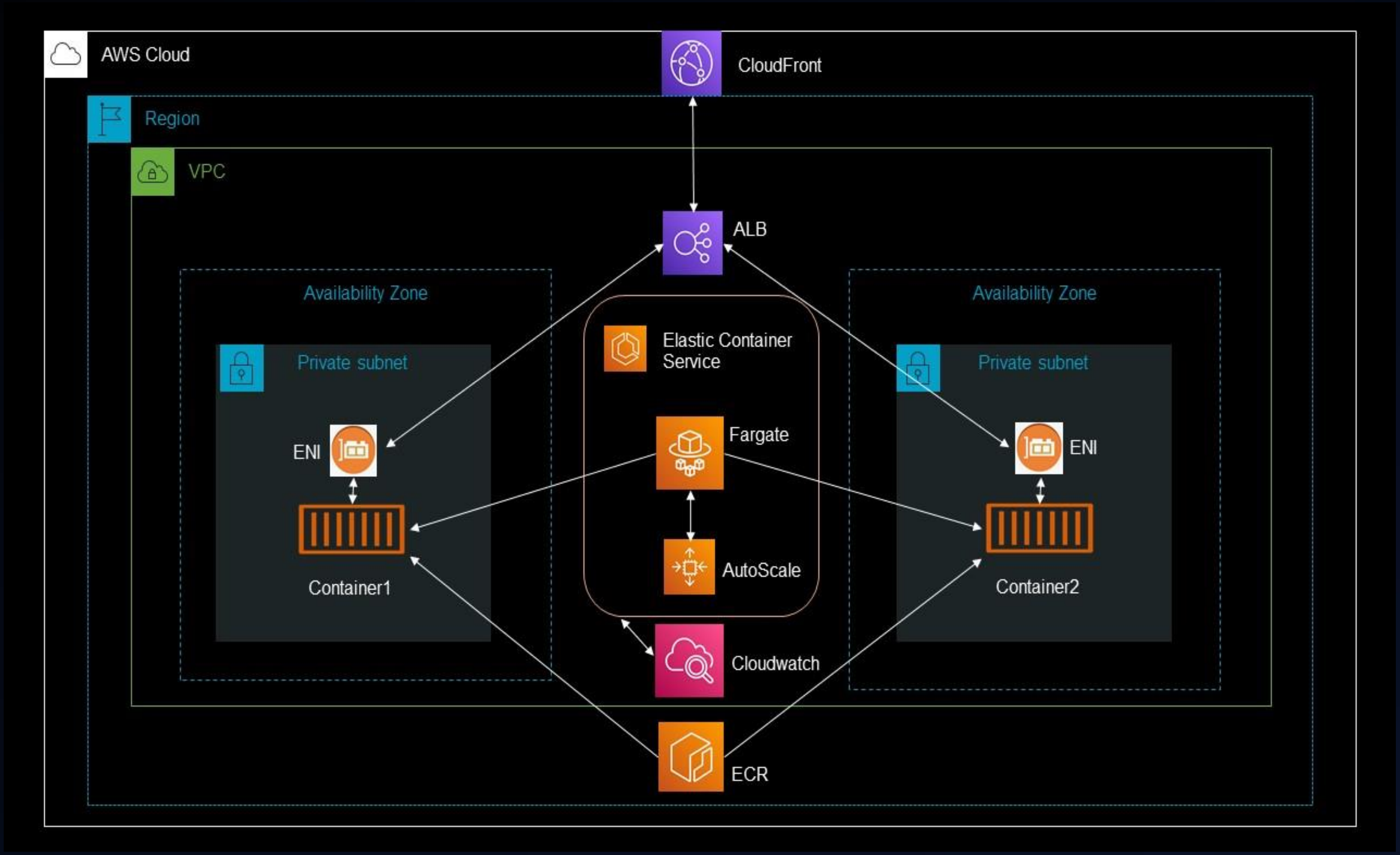
MSc in Advanced Computer science

POWER POINT PRESENTATION

Private Cloud Demonstration

MAHESH SRIVINAY RAYAVARAPU
ID: 199047813

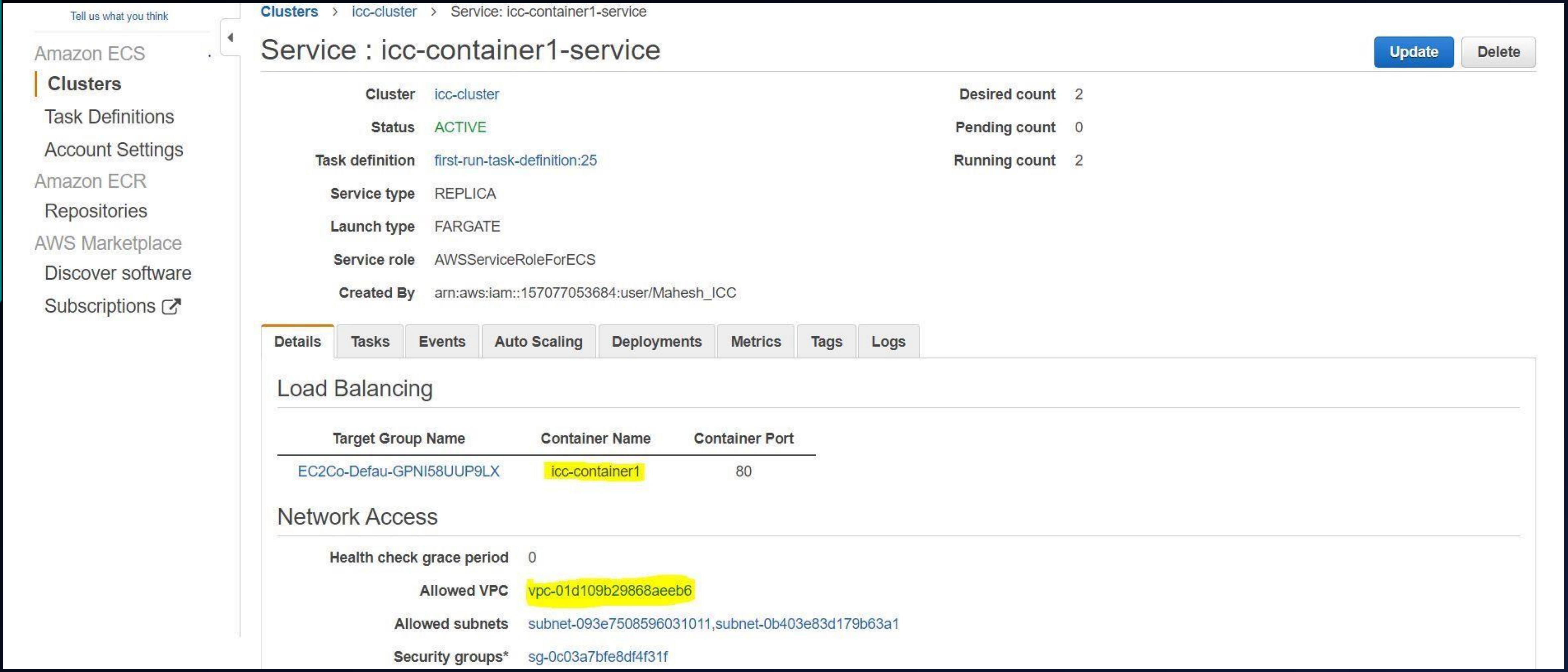
Details of Architecture Design



- Used ECR to store docker image.
- Used ECS Fargate for deploying docker image across clusters.
- VPC for hosting resources.
- CloudFront for content delivery via edge locations catching.
- ALB is used to balance load between two container replicas.

ECS Cluster Creation

- ECS Cluster created by using application docker image stored in ECR.
- Used ALB and autoscaling options.



Clusters > icc-cluster > Service: icc-container1-service

Service : icc-container1-service

Cluster	icc-cluster	Desired count	2
Status	ACTIVE	Pending count	0
Task definition	first-run-task-definition:25	Running count	2
Service type	REPLICA		
Launch type	FARGATE		
Service role	AWSServiceRoleForECS		
Created By	arn:aws:iam::157077053684:user/Mahesh_ICC		

Load Balancing

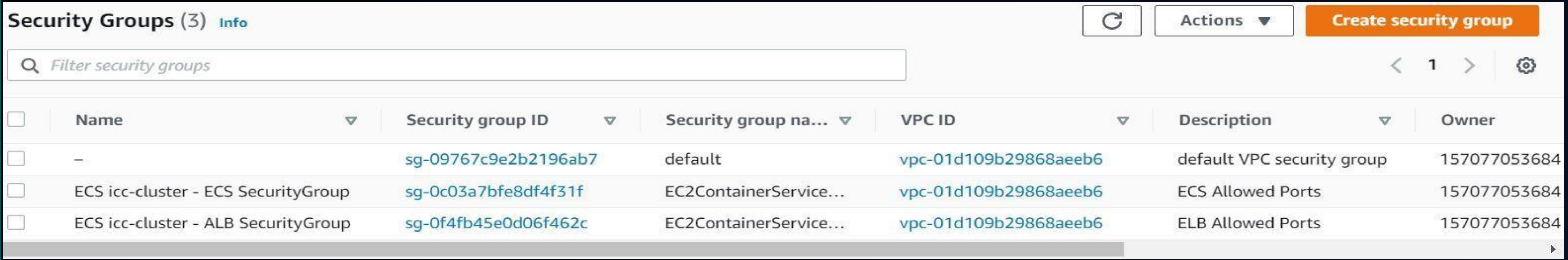
Target Group Name	Container Name	Container Port
EC2Co-Defau-GPNI58UUP9LX	icc-container1	80

Network Access

Health check grace period	0
Allowed VPC	vpc-01d109b29868aeeb6
Allowed subnets	subnet-093e7508596031011,subnet-0b403e83d179b63a1
Security groups*	sg-0c03a7bfe8df4f31f

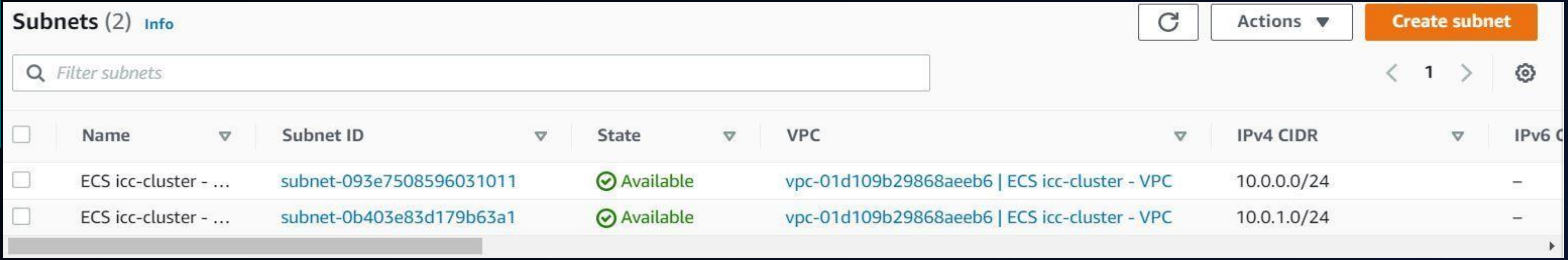
Networking

- ECS cluster is hosted on VPC with two subnets, two security groups and internet gateway.



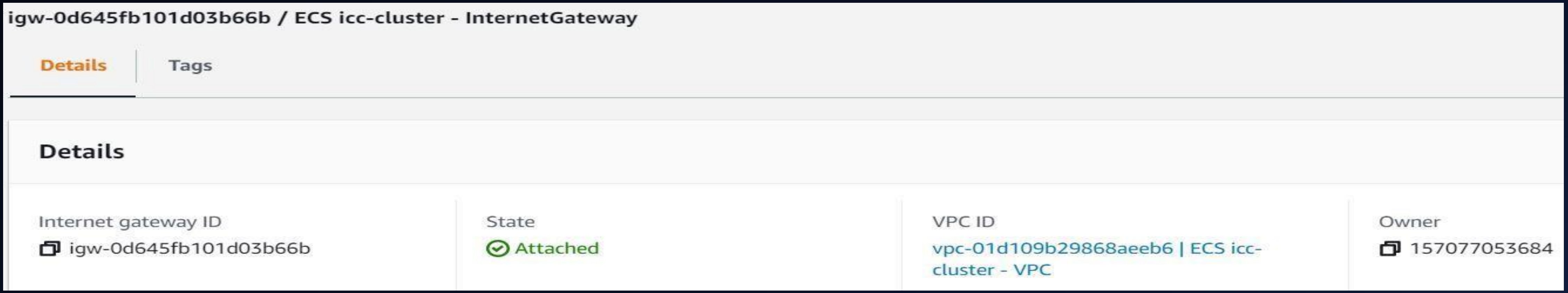
Security Groups (3) Info

Name	Security group ID	Security group na...	VPC ID	Description	Owner
–	sg-09767c9e2b2196ab7	default	vpc-01d109b29868aeeb6	default VPC security group	157077053684
ECS icc-cluster - ECS SecurityGroup	sg-0c03a7bfe8df4f31f	EC2ContainerService...	vpc-01d109b29868aeeb6	ECS Allowed Ports	157077053684
ECS icc-cluster - ALB SecurityGroup	sg-0f4fb45e0d06f462c	EC2ContainerService...	vpc-01d109b29868aeeb6	ELB Allowed Ports	157077053684



Subnets (2) Info

Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 C
ECS icc-cluster - ...	subnet-093e7508596031011	Available	vpc-01d109b29868aeeb6 | ECS icc-cluster - VPC	10.0.0.0/24	–
ECS icc-cluster - ...	subnet-0b403e83d179b63a1	Available	vpc-01d109b29868aeeb6 | ECS icc-cluster - VPC	10.0.1.0/24	–



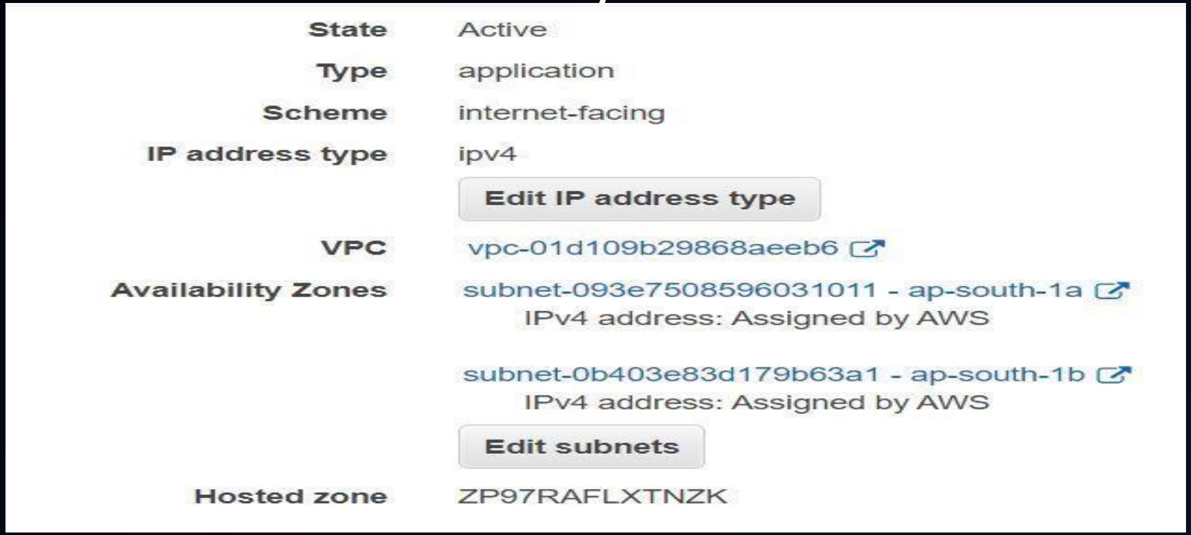
igw-0d645fb101d03b66b / ECS icc-cluster - InternetGateway

Details

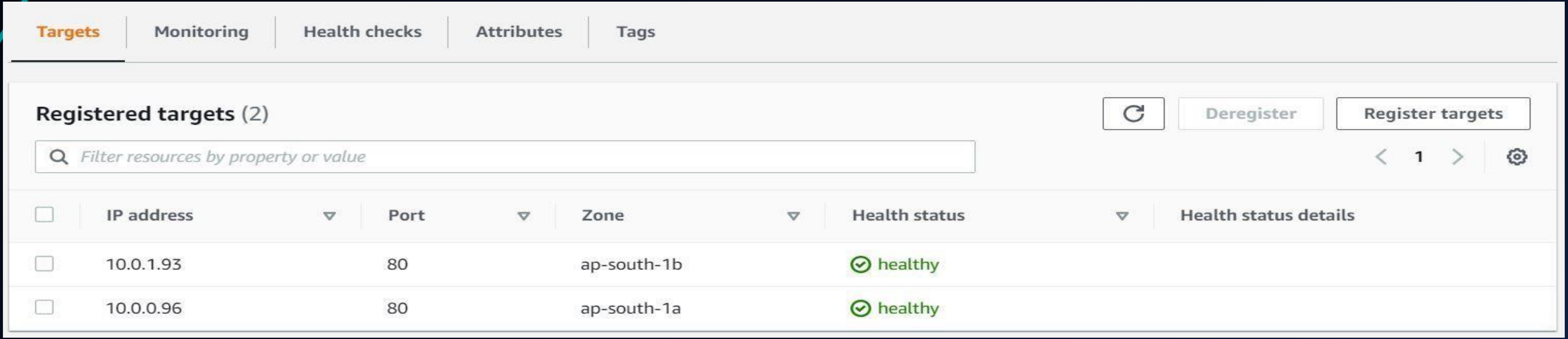
Internet gateway ID	State	VPC ID	Owner
igw-0d645fb101d03b66b	Attached	vpc-01d109b29868aeeb6 | ECS icc-cluster - VPC	157077053684

Load Balancing

- ALB balances load between two container replicas hosted on two subnets residing in two different availability zones.



State	Active
Type	application
Scheme	internet-facing
IP address type	ipv4
	Edit IP address type
VPC	vpc-01d109b29868aeeb6
Availability Zones	subnet-093e7508596031011 - ap-south-1a IPv4 address: Assigned by AWS
	subnet-0b403e83d179b63a1 - ap-south-1b IPv4 address: Assigned by AWS
	Edit subnets
Hosted zone	ZP97RAFLXTNZK

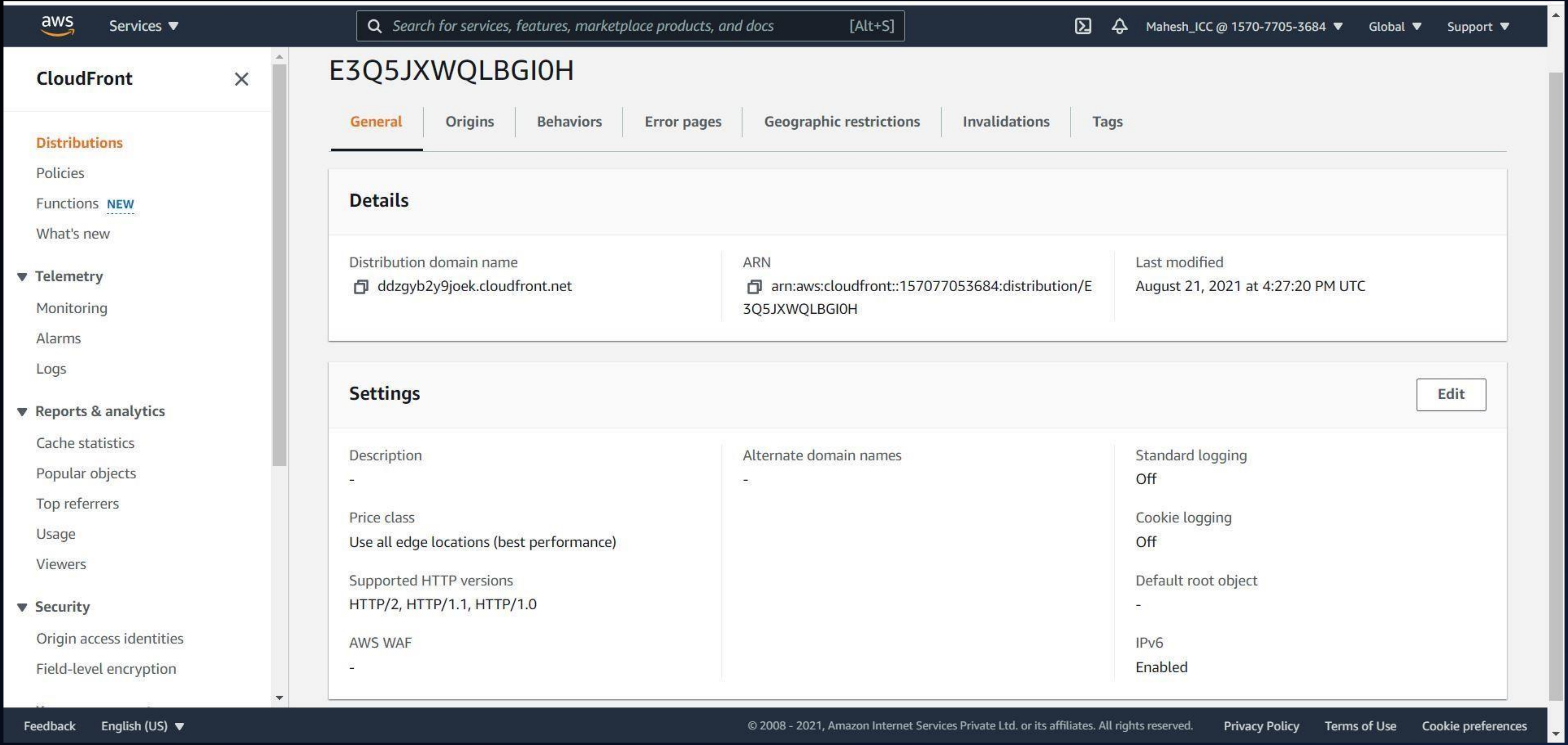


Registered targets (2)

IP address	Port	Zone	Health status	Health status details
10.0.1.93	80	ap-south-1b	✓ healthy	
10.0.0.96	80	ap-south-1a	✓ healthy	

CloudFront via ALB

- CloudFront distribution is created and it routes the traffic to ALB.
- Opted for all edge locations for low latency and high availability.



E3Q5JXWQLBGI0H

Details

Distribution domain name: ddzgyb2y9joek.cloudfront.net

ARN: arn:aws:cloudfront::157077053684:distribution/E3Q5JXWQLBGI0H

Last modified: August 21, 2021 at 4:27:20 PM UTC

Settings

Description: -

Alternate domain names: -

Standard logging: Off

Price class: Use all edge locations (best performance)

Cookie logging: Off

Supported HTTP versions: HTTP/2, HTTP/1.1, HTTP/1.0

Default root object: -

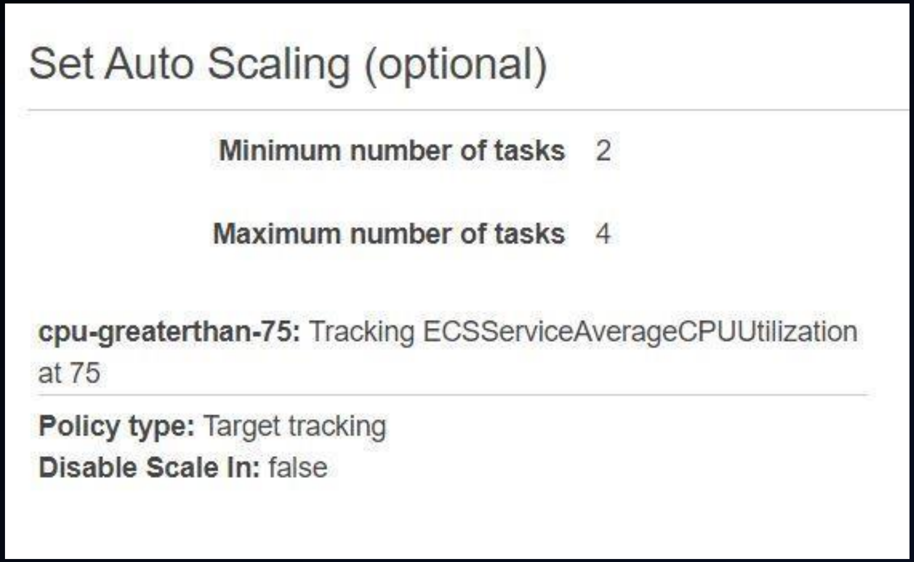
AWS WAF: -

IPv6: Enabled

© 2008 - 2021, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved.

CloudWatch Monitoring

- CPU usage is monitored using CloudWatch. Autoscaling policy uses this metric for creating additional container replicas.

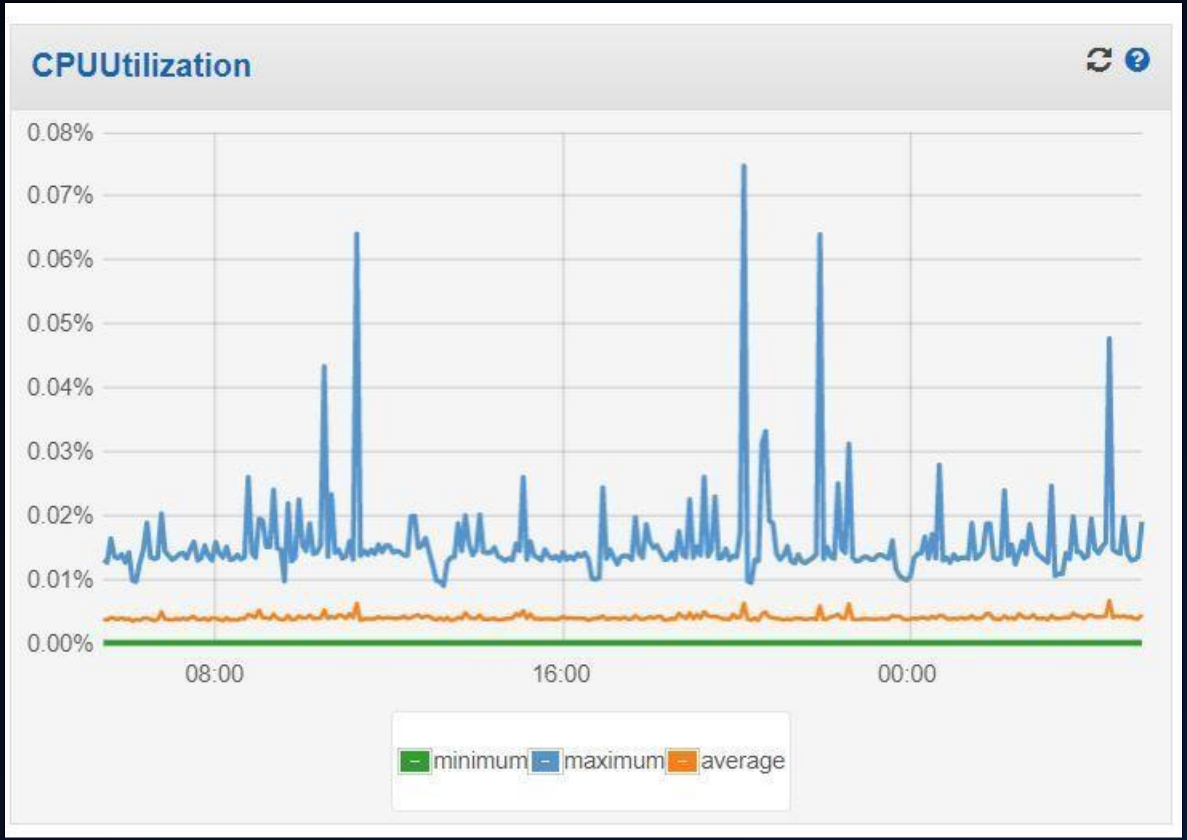


Set Auto Scaling (optional)

Minimum number of tasks	2
Maximum number of tasks	4

cpu-greaterthan-75: Tracking ECSServiceAverageCPUUtilization at 75

Policy type: Target tracking
Disable Scale In: false



Video Demonstration URL:

https://www.dropbox.com/s/mxri9t6dthkhazz/ICC-Video%20Demonstartion.mp4?dl=0

Introduction:

The delivery of computing services which includes servers, networking, databases, storage etc over the cloud(internet) is known as cloud computing. This project involves the implementation and deployment of a simple notes-taking web application with Presentation layer and business logic layer in virtual private cloud. The cloud service provider used in this project is the popular Amazon Web Services.

Implementation and Architecture Design of Private Cloud:

a) Technological choice:

After evaluating the top three cloud service providers (Azure, GCP and AWS) and their offerings, I found AWS is the best suitable platform for my architecture.

Why AWS? With AWS, an application can be accessed all over the world due to its availability zones and edge locations. It has more options for serverless and ease of operations. The pay as you go pricing model and scalability factor due to the tools like Elastic load balancers and auto scaling makes AWS the most preferred cloud provider platform.

b) Tools used:

- ***Docker*** – Docker is used as the virtualization mechanism/tool due to its portability and ease of use on various platforms. Also, every known cloud or on-premise platform supports docker. And it comes with great community support. Initially, a docker image of the application was created in the local machine using Linux alpine as a base image.

- ***AWS Elastic Container Registry (ECR)*** – ECR is a container registry used to store, share and deploy the container images. After signing-in to the AWS account, navigate to the ECR service and then click on create private repository. Name the repository and push the local image to the created repository. First sign-in to the AWS CLI and then tag the image so that the image can be pushed to the repository by using docker push command. The docker image was successfully pushed to the ECR with an image URI.

- ***AWS Elastic Container Service (ECS)* –** ECS is a container orchestration service used to manage, scale and deploy docker containers across clusters with features like Elastic load balancers (ALB), security groups, cloud-watch monitoring and Auto-scaling features. The ECS task execution was managed by Fragate. Fargate is serverless and there is no need to choose EC2 instances. Using fargate, there is no need to manually scale clusters of virtual machines for running the containers.

  After pushing the docker image to ECR, create a cluster by navigate to ECS service console and create a custom container definition with image URI. In task definitions, input memory limits, port mappings and enable the CloudWatch logs for monitoring. Configured the network and enabled the autoscaling option for CPU usage exceeding 75% by adding the scaling policy with CloudWatch Metrics. We can also monitor it using the CloudWatch alarms. And in the service definition, the security groups will be automatically created and application load balancer will be enabled for managing the external traffic load. Finally, a cluster will get created with the specified name.

- ***Virtual Private Cloud (VPC)* –** The virtual network infrastructures like subnets, routing tables, security groups, functioning of network, IP address range of AWS instances can be configured quickly by using virtual private cloud. VPC is scalable based on the user needs.
  ECS cluster is hosted in VPC along with two subnets each for ECS and ALB. One security group is created to block traffic inside and outside of the VPC and it only allows the TCP traffic to ALB from CloudFront. Another security group blocks all traffic in and out of ECS containers except the traffic from ALB. These security groups thus make our resources more secure against various cyber-attacks.

- ***Application Load Balancer (ALB)*** – ALB is used in distributing the incoming traffic across multiple container replicas in one or more availability zones chosen for fault tolerance and high availability. Two different target groups were created and they tell the load balancer where the traffic should be directed.

- ***CloudWatch* –** The data and insights which are required to monitor the applications was provided by the CloudWatch. Here the CPU autoscaling metrics was monitored by CloudWatch alarms.

- ***CloudFront* –** AWS CloudFront is a global content delivery network which offers improved security. CloudFront uses edge locations for caching. So that the application will have higher availability with high performance ensuring lower latency.
  After cluster creation, navigate to CloudFront and create a distribution using ALB as origin domain. Input the network protocols and HTTP methods and in the price-class use all edge locations. So that the application is distributed across all the edge locations for lower latency and high availability. Finally, application is deployed with distribution domain name.

c) Architecture Diagram:

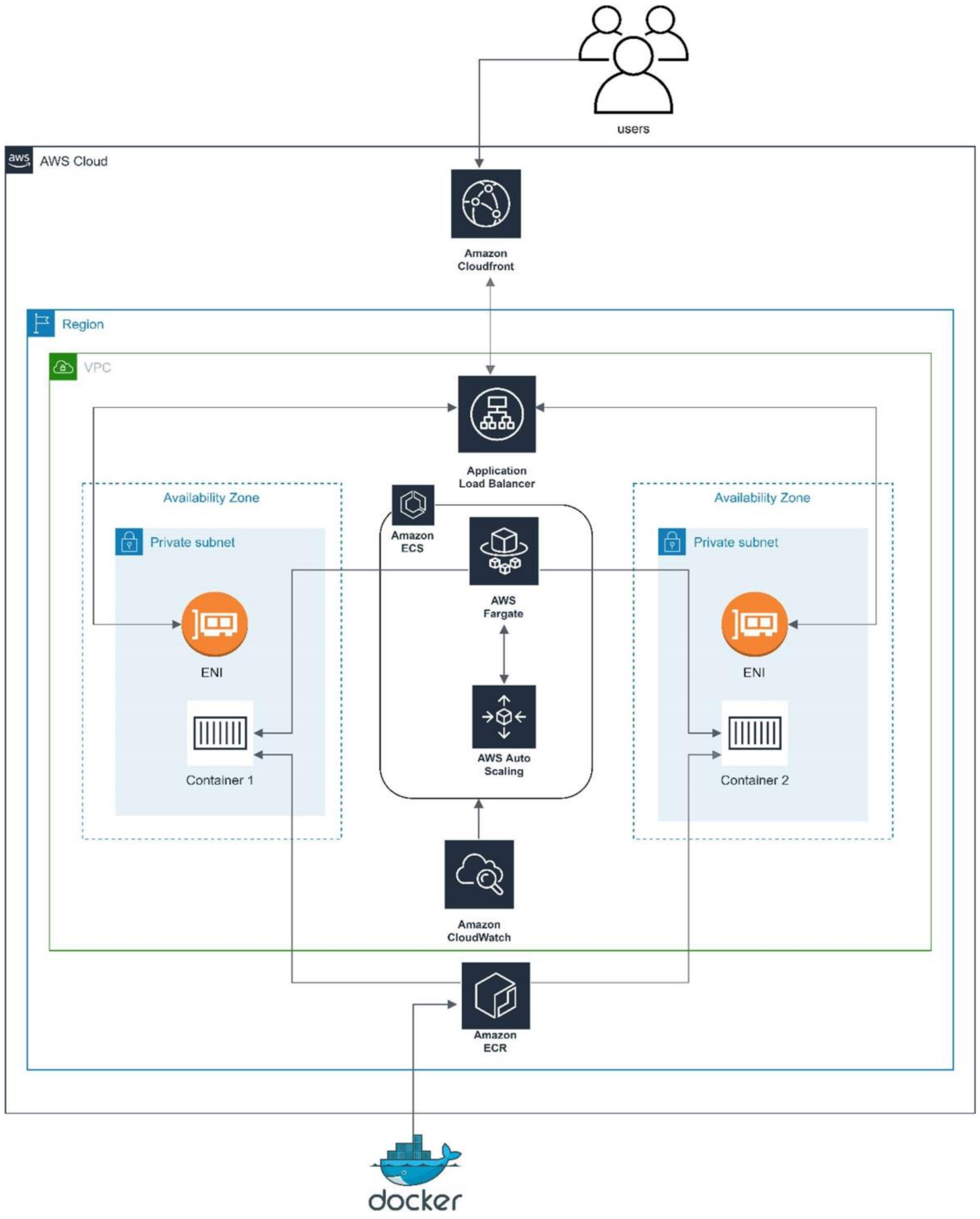


Fig 1(a)- Architecture diagram of private cloud

Weakness and Strengths of Design and Implementation:

The proposed architecture uses Fargate as container orchestrator which offers high security. Main advantage of Fargate is, we can operate the containers without having to deal with the hassles of running servers. The main reason to choose this architecture is because of its easy implementation and lower time consumption. In this architecture scalability is achieved by using the auto scaling feature in container task definition via CPU CloudWatch metrics. ALB makes sure that the incoming traffic is balanced between the networks. For achieving lower latency, the ALB is further linked with the CloudFront. In CloudFront distribution, we used all the edge locations of AWS for better performance and QOS. Coming to the cons side of architecture, it's a bit costlier for the simple application we choose to deploy. We can deploy the application on a single EC2 machine as well to reduce the costs. Right now, there are limited CloudWatch metrics that can be used to Autoscaling our containers such as CPU and memory usage. If we need to define autoscaling used on some custom policies, we should create custom metrices.

Suggestions to improve system design and Implementation:

We can improve the autoscaling by using the dynamic scaling policy designed using custom CloudWatch metrics that monitors the number of sessions used by end customers. We can implement automated notifications (SNS) to the stakeholders regarding the resource health and scaling updates. We can also improve the application features such as history of notes and session storage etc by adding RDS (database). We can also use AWS shield for blocking the DDOS attacks on our website.

Conclusion:

The above architecture and its implementation thus demonstrate the best use of the cloud tools and technologies available in market to provide an application with low latency, high availability, fault tolerance and scalability. In addition, this provides great ease of operations and maintenance to the customer.