Aws full document

Aws Networking

Aws Ec2

Aws S3

Aws IAM

Aws RDS

Aws Cloudwatch

Aws Certificate manager

Aws Rout53

Aws Lambda overview

AWS Networking

Overview of AWS Networking

Amazon Web Services (AWS) Networking encompasses a broad set of services and capabilities that allow you to build highly scalable, reliable, and secure networking environments in the cloud. These services are designed to provide connectivity between AWS resources, between AWS and your on-premises data centers, as well as to and from the Internet.

Key Components of AWS Networking

1. Amazon VPC (Virtual Private Cloud)
    - Amazon VPC is the foundational networking service that allows you to create a logically isolated network within the AWS Cloud. VPC enables you to define your own IP address ranges, create subnets, route traffic between instances, and configure security settings like firewalls.
    - With VPC, you can:
        - Create Subnets: Segregate your network into subnets (e.g., public and private subnets) to organize resources in a way that suits your application.
        - Private and Public IP Addresses: Assign private IPs to your EC2 instances and optionally, public IPs for internet-facing instances.
        - Route Tables: Control how traffic is routed within your VPC and to external networks.

- Security Groups and Network ACLs: Control inbound and outbound traffic to/from resources using security groups (stateful) and network ACLs (stateless).
- VPC Peering: Connect multiple VPCs to share resources and services between them.

2. Elastic IP (EIP)
    - Elastic IP is a static, public IP address designed for dynamic cloud computing. You can associate an EIP with an EC2 instance or a network interface, providing a fixed address that doesn't change, even if you stop and start your instance.
    - Useful for high availability and failover configurations, as you can remap the EIP to another instance in case of failure.

Key Networking Concepts in AWS

1. VPC Subnets:
    - Subnets are segments of your VPC's IP address range, where you place your AWS resources like EC2 instances. You can create public subnets (which can route traffic to/from the internet) and private subnets (isolated from the internet).
2. Route Tables:
    - A route table defines how traffic is routed within a VPC and to/from the internet. Each subnet in a VPC must be associated with a route table.
3. Security Groups and Network ACLs:
    - Security Groups are virtual firewalls that control inbound and outbound traffic at the instance level. They are stateful, meaning that if you allow inbound traffic, the response is automatically allowed.

- Network ACLs (Access Control Lists) are stateless firewalls that apply at the subnet level. Unlike security groups, network ACLs require explicit rules for both inbound and outbound traffic.

4. NAT Gateway and NAT Instance:
    - NAT Gateway allows instances in a private subnet to connect to the internet for software updates and other purposes, while preventing inbound internet traffic from directly accessing the private instances.
    - NAT Instance is a similar concept, but with more flexibility and management responsibilities.
5. Elastic Network Interfaces (ENI):
    - Elastic Network Interface is a virtual network interface that you can attach to an EC2 instance or Lambda function. It enables communication between AWS resources, allowing more advanced network configurations (e.g., multi-homed instances).

Security in AWS Networking

1. VPC Security:
    - Security groups and network ACLs help protect your VPC and resources by controlling traffic at both the instance and subnet levels.
    - VPC Flow Logs: Capture detailed logs of network traffic in and out of your VPC, which can be useful for troubleshooting, monitoring, and auditing.
2. Private Communication:
    - PrivateLink and VPC Peering allow private communication between VPCs and AWS services without routing traffic over the public internet.
3. Encryption:

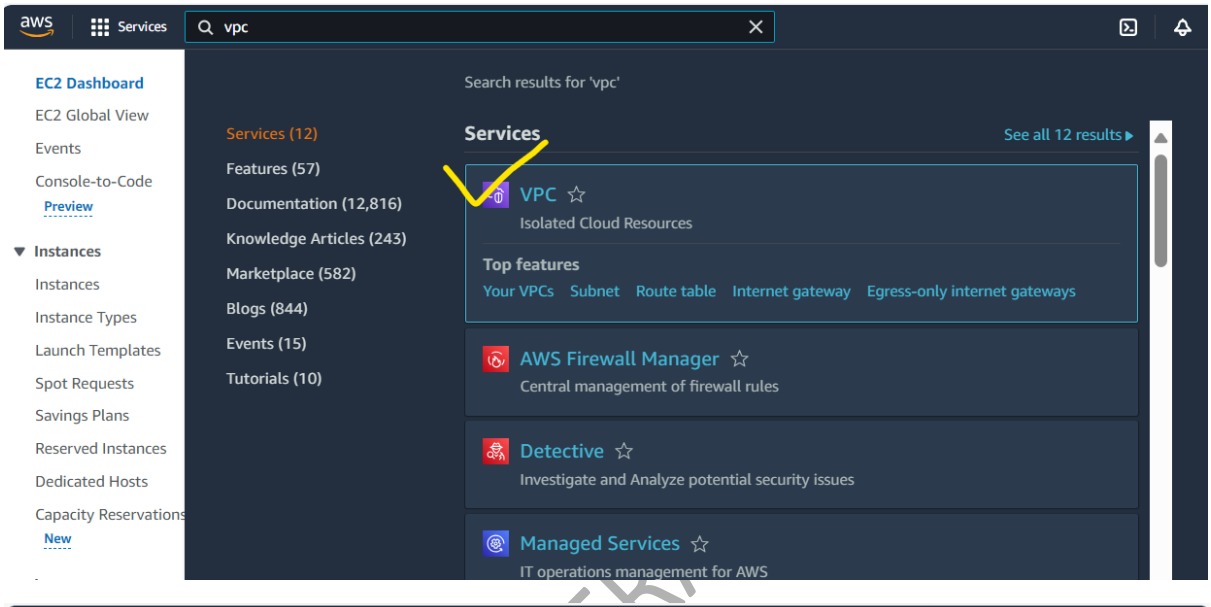
- Encryption is supported for all traffic that moves between VPC resources. You can enable encryption for services like Amazon EFS, S3, and CloudFront, as well as traffic between EC2 instances.

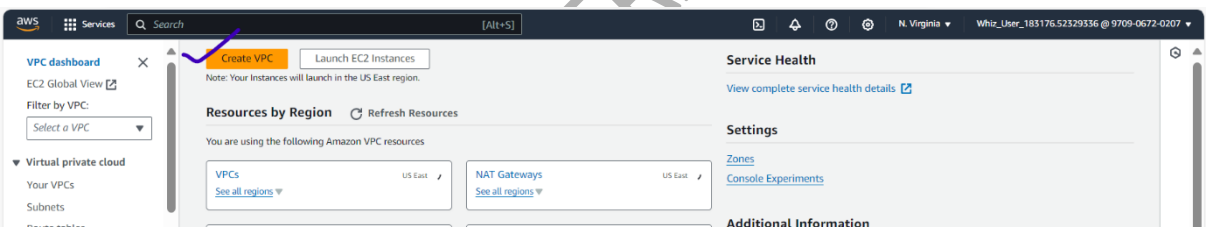
Benefits of AWS Networking

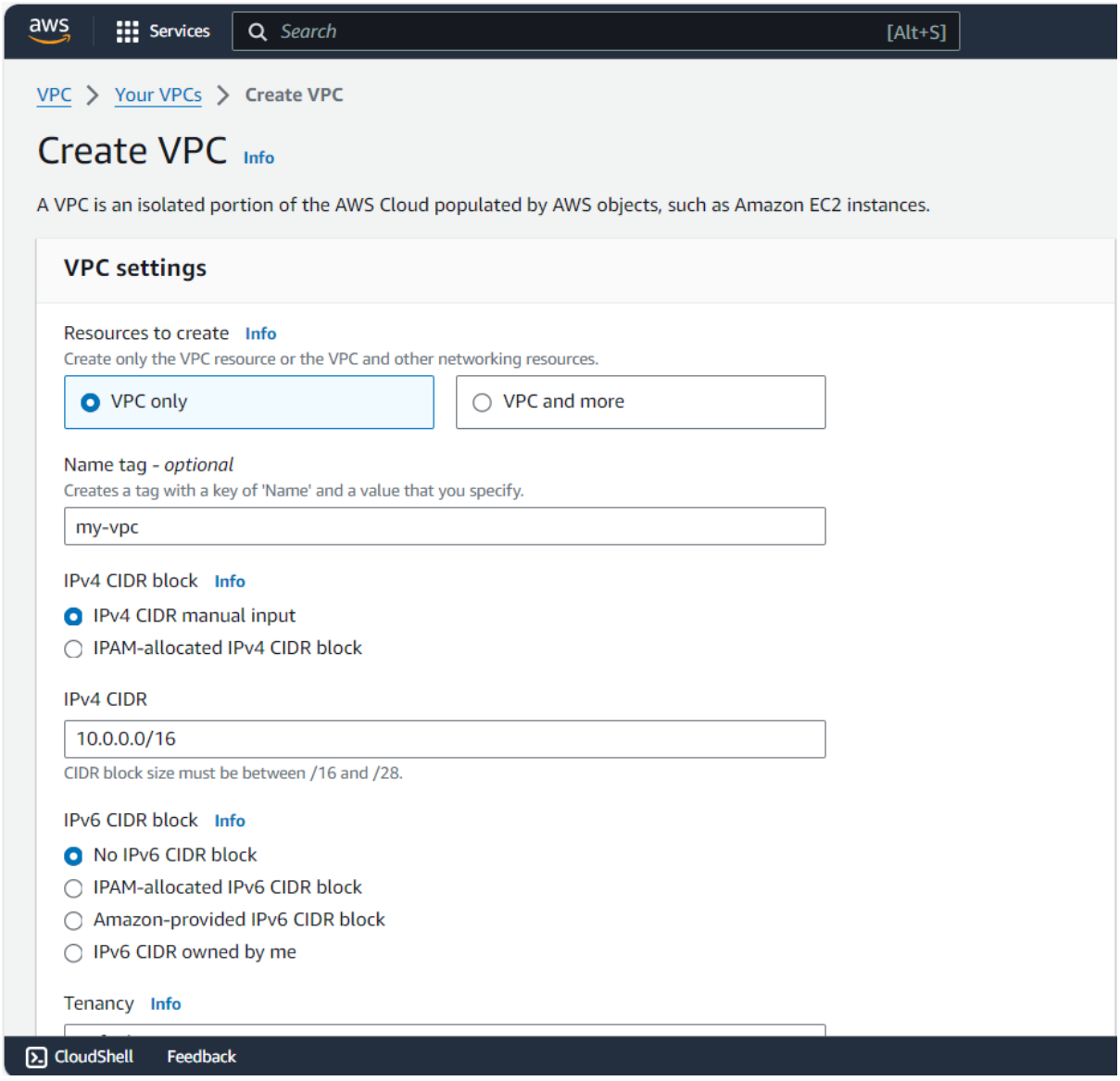
1. Scalability and Flexibility:
    - AWS networking services are designed to scale with your applications. Whether you're deploying a small website or a large enterprise system, AWS can scale to meet the demand.
2. Global Reach:
    - AWS offers global infrastructure, so you can deploy applications closer to your users with low-latency connections via services like CloudFront, Global Accelerator, and Direct Connect.
3. Security:
    - AWS provides a range of security features, including encryption, firewalls, and DDoS protection (via AWS Shield), helping you secure your network and applications.
4. Reliability:
    - AWS networking components like Elastic Load Balancing and AWS Direct Connect ensure high availability and fault tolerance for your applications.
5. Cost-Effectiveness:
    - Pay-as-you-go pricing models allow you to optimize costs based on usage. AWS helps you minimize operational costs by abstracting much of the complexity involved in network management.

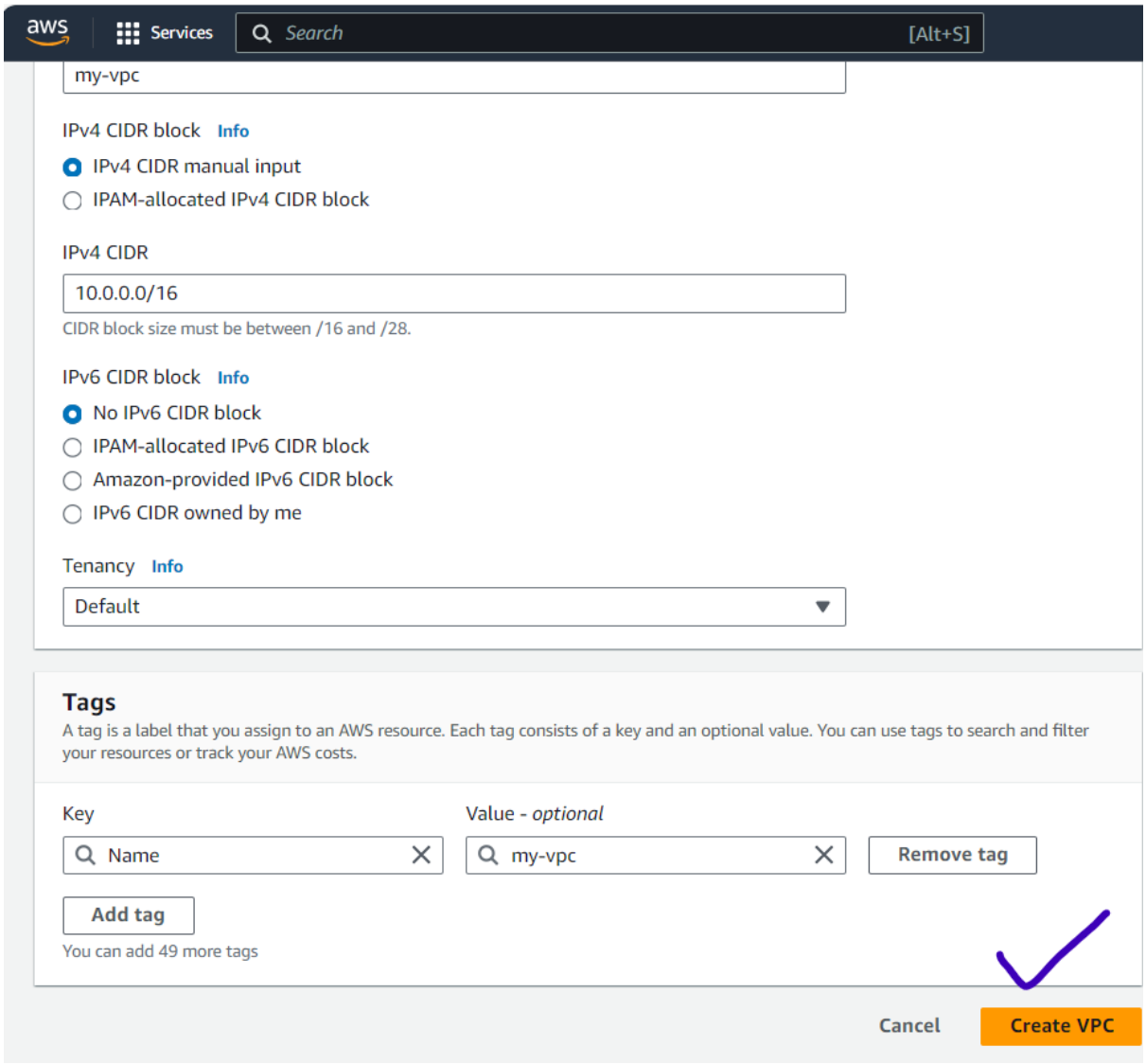
1. Create VPC Network

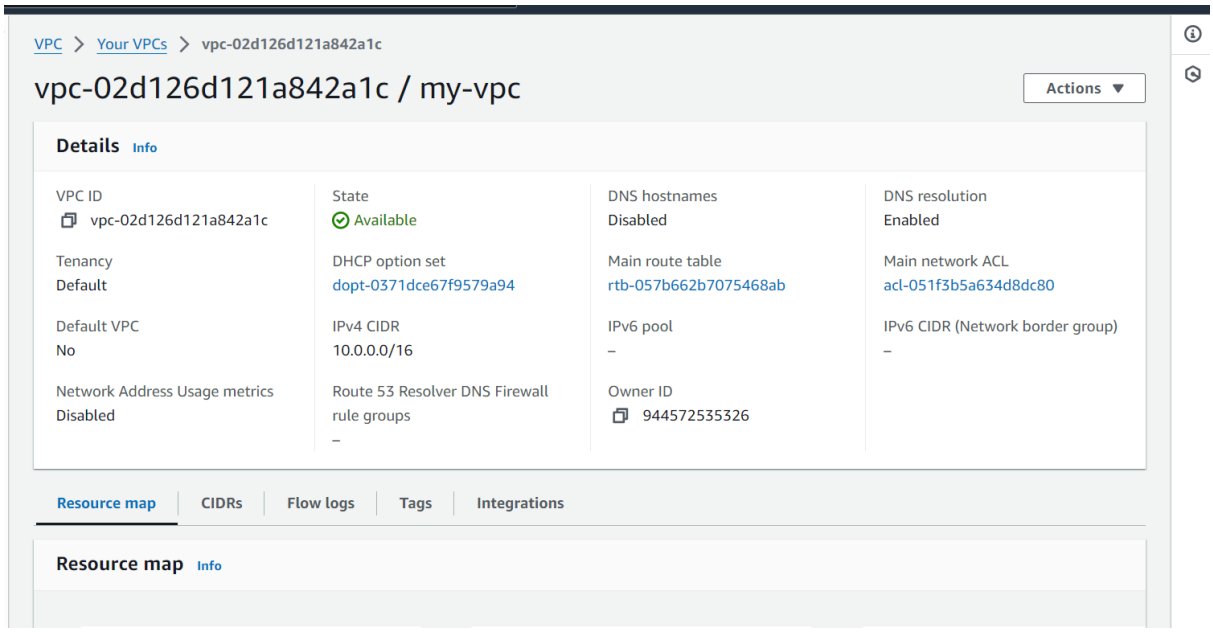

vpc


aws
Services
vpc
EC2 Dashboard
EC2 Global View
Events
Console-to-Code
Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
New
Search results for 'vpc'
Services (12)
Features (57)
Documentation (12,816)
Knowledge Articles (243)
Marketplace (582)
Blogs (844)
Events (15)
Tutorials (10)
Services
See all 12 results
VPC
Isolated Cloud Resources
Top features
Your VPCs
Subnet
Route table
Internet gateway
Egress-only internet gateways
AWS Firewall Manager
Central management of firewall rules
Detective
Investigate and Analyze potential security issues
Managed Services
IT operations management for AWS


aws
Services
Search
[Alt+S]
N. Virginia
VPC dashboard
EC2 Global View
Filter by VPC:
Select a VPC
Virtual private cloud
Your VPCs
Subnets
Create VPC
Launch EC2 Instances
Note: Your Instances will launch in the US East region.
Resources by Region
Refresh Resources
You are using the following Amazon VPC resources
VPCs
See all regions
NAT Gateways
See all regions
Service Health
View complete service health details
Settings
Zones
Console Experiments
Additional Information

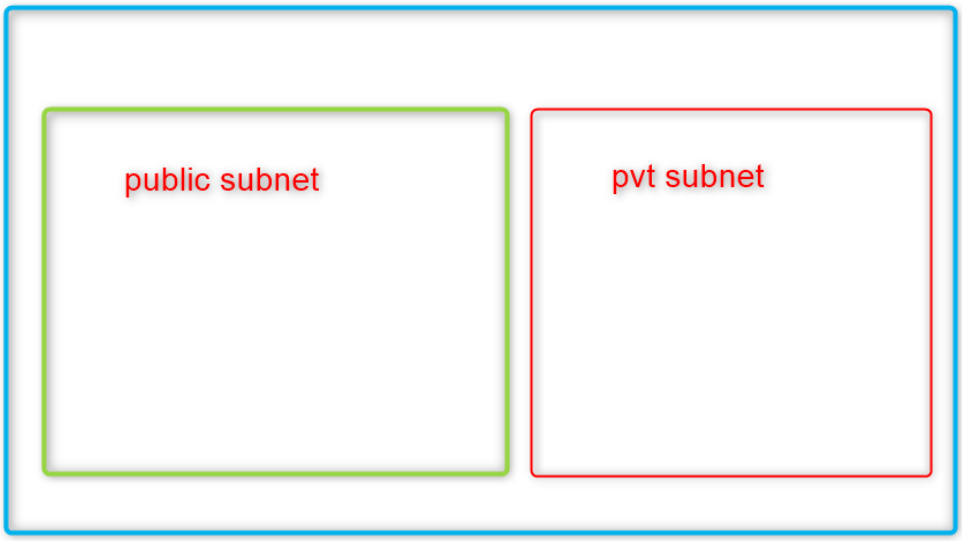

aws
Services
Search
[Alt+S]
VPC > Your VPCs > Create VPC
Create VPC Info
A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon EC2 instances.
VPC settings
Resources to create Info
Create only the VPC resource or the VPC and other networking resources.
VPC only
VPC and more
Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify.
my-vpc
IPv4 CIDR block Info
IPv4 CIDR manual input
IPAM-allocated IPv4 CIDR block
IPv4 CIDR
10.0.0.0/16
CIDR block size must be between /16 and /28.
IPv6 CIDR block Info
No IPv6 CIDR block
IPAM-allocated IPv6 CIDR block
Amazon-provided IPv6 CIDR block
IPv6 CIDR owned by me
Tenancy Info
CloudShell
Feedback

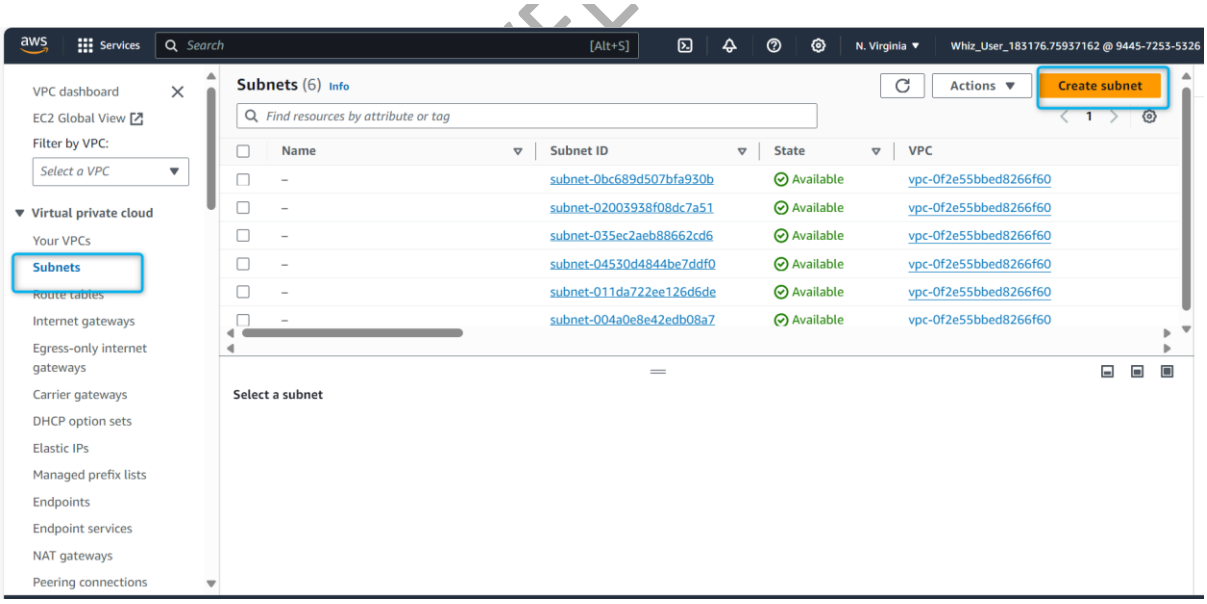


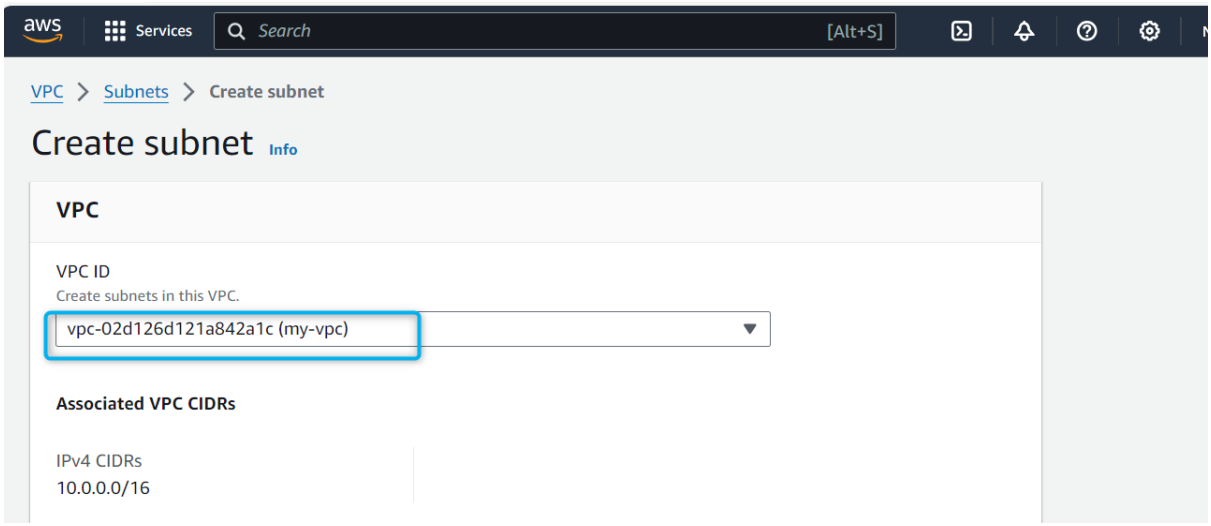


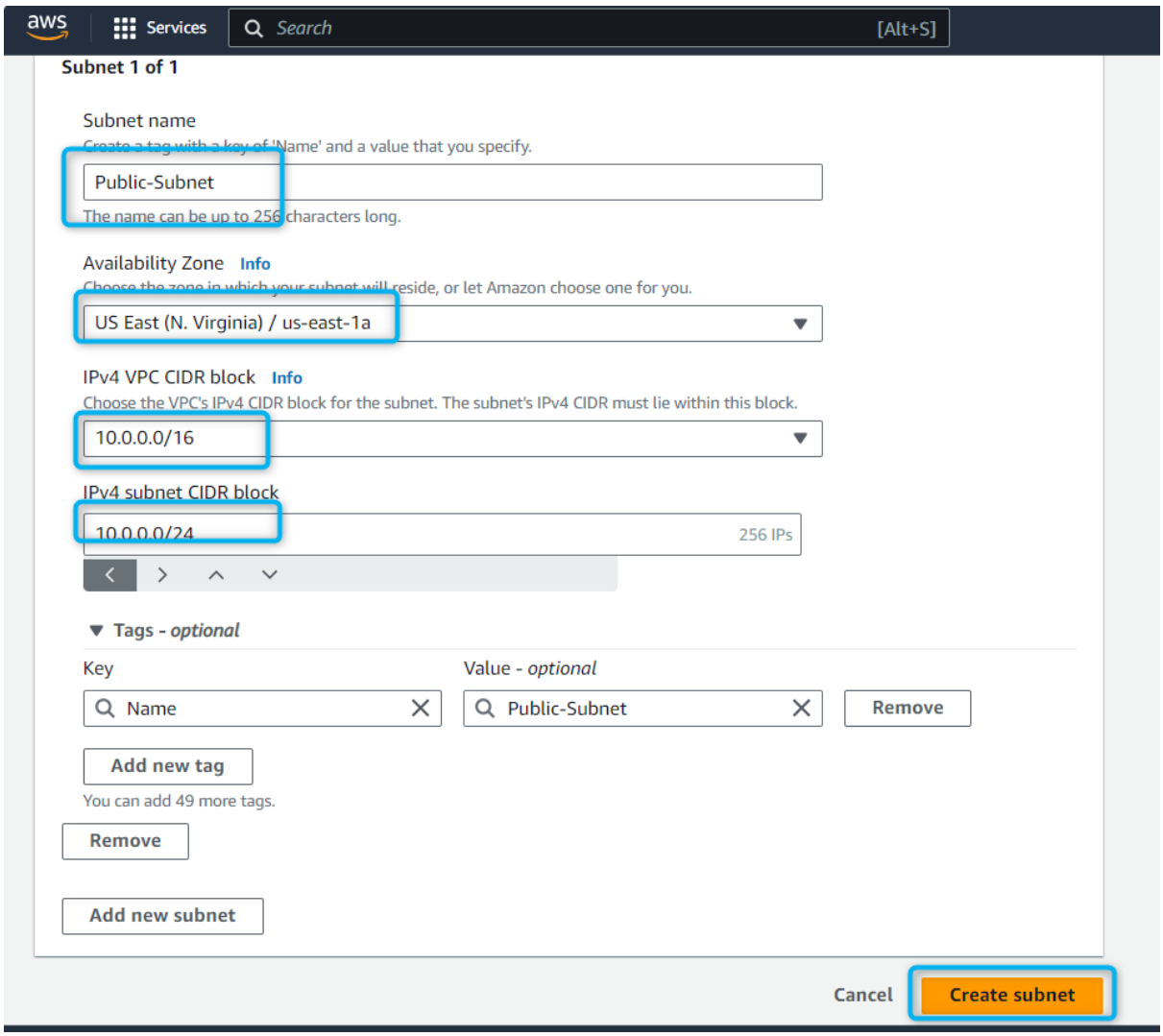
Now vpc created.

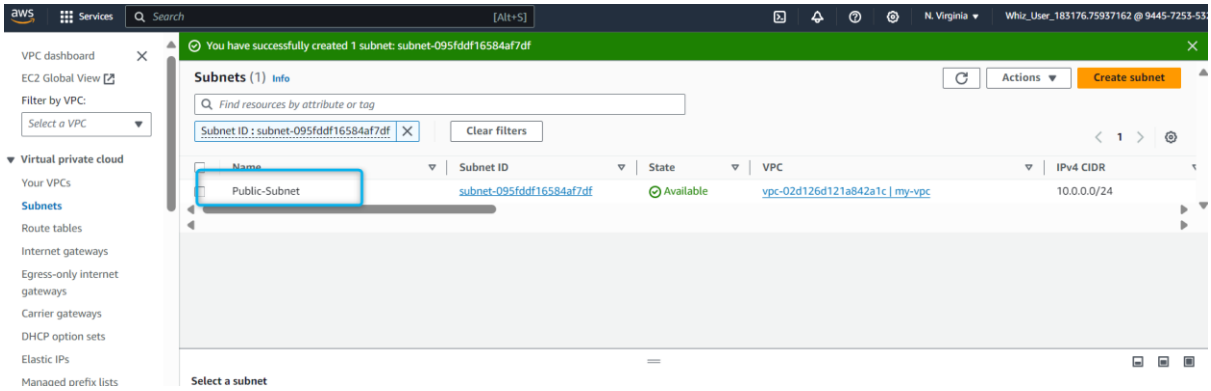
2. Create Subnet (Public Subnet and Private Subnet)

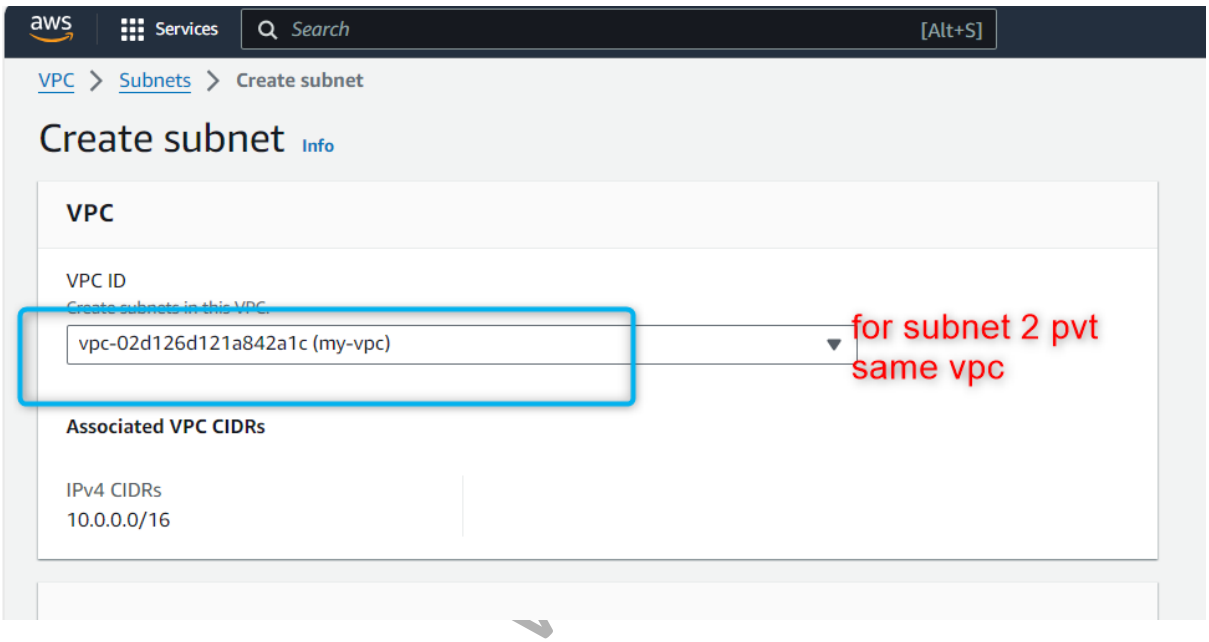


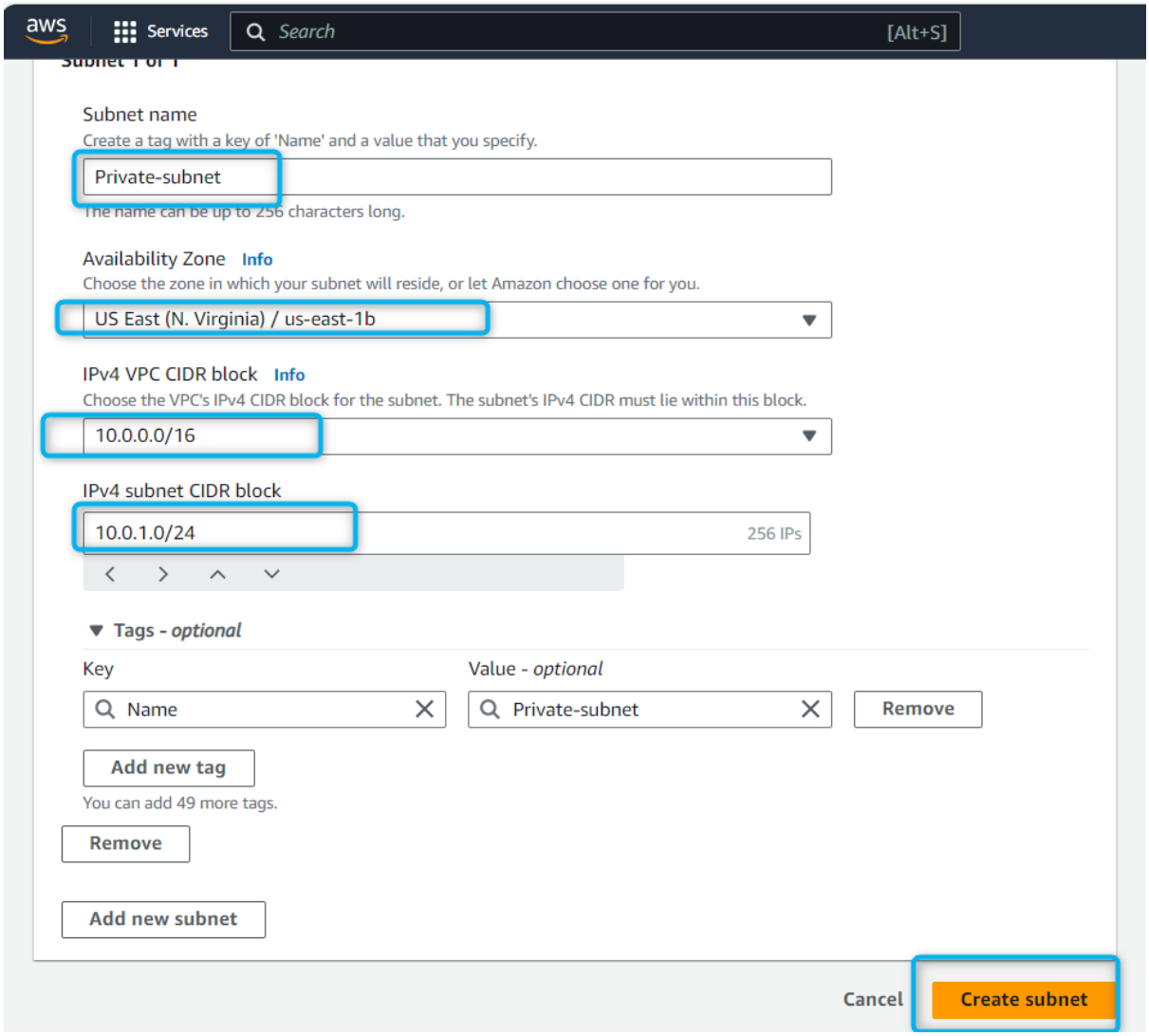


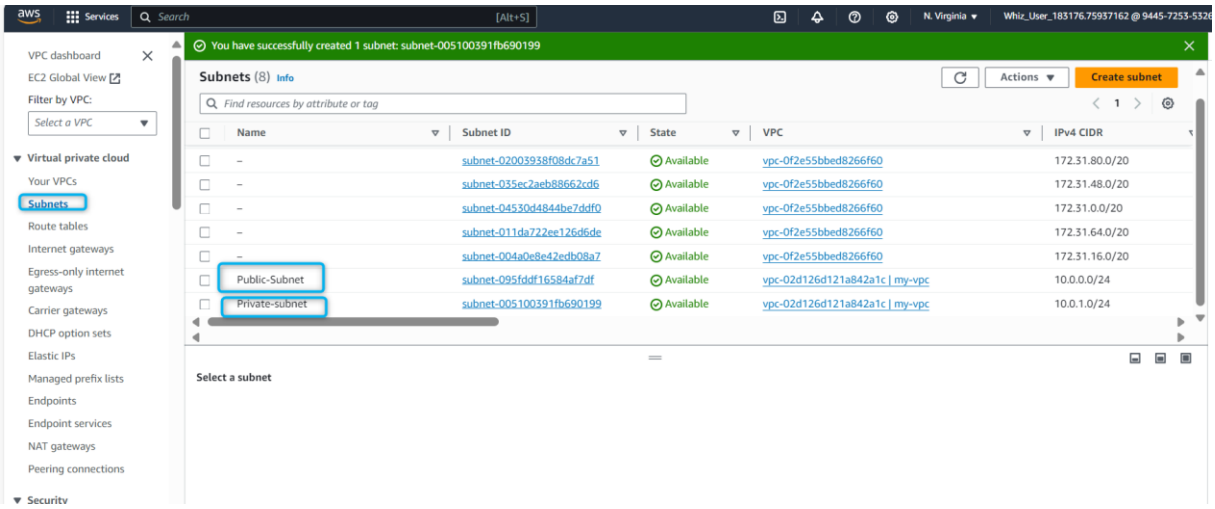

aws
Services
Search
[Alt+S]
VPC > Subnets > Create subnet
Create subnet Info
VPC
VPC ID
Create subnets in this VPC.
vpc-02d126d121a842a1c (my-vpc)
Associated VPC CIDRs
IPv4 CIDRs
10.0.0.0/16


aws
Services
Search
[Alt+S]
Subnet 1 of 1
Subnet name
Create a tag with a key of 'Name' and a value that you specify.
Public-Subnet
The name can be up to 256 characters long.
Availability Zone Info
Choose the zone in which your subnet will reside, or let Amazon choose one for you.
US East (N. Virginia) / us-east-1a
IPv4 VPC CIDR block Info
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.
10.0.0.0/16
IPv4 subnet CIDR block
10.0.0.0/24
256 IPs
Tags - optional
Key
Value - optional
Name
Public-Subnet
Remove
Add new tag
You can add 49 more tags.
Remove
Add new subnet
Cancel
Create subnet

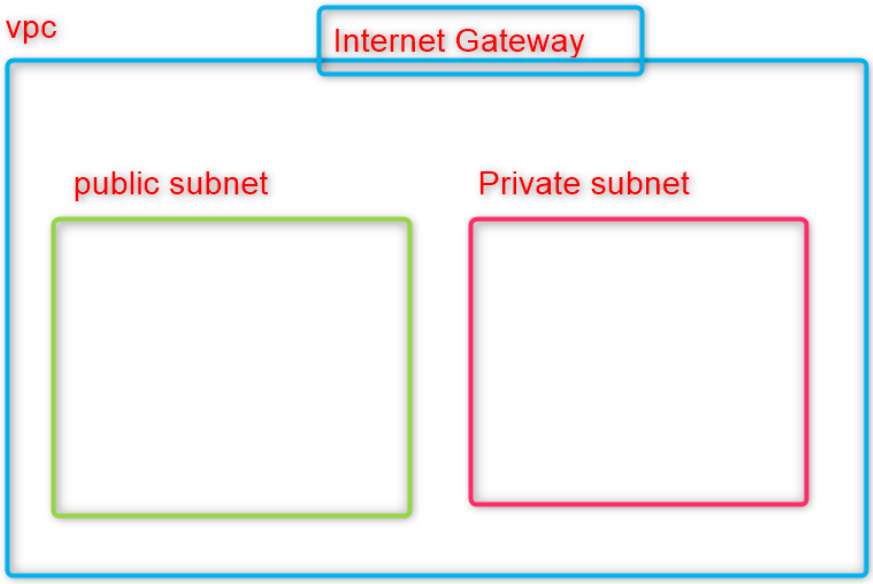

You have successfully created 1 subnet: subnet-095fddf16584af7df
VPC dashboard
EC2 Global View
Filter by VPC:
Select a VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Subnets (1) Info
Find resources by attribute or tag
Subnet ID : subnet-095fddf16584af7df
Clear filters
Actions
Create subnet
Name
Subnet ID
State
VPC
IPv4 CIDR
Public-Subnet
subnet-095fddf16584af7df
Available
vpc-02d126d121a842a1c | my-vpc
10.0.0.0/24
Select a subnet

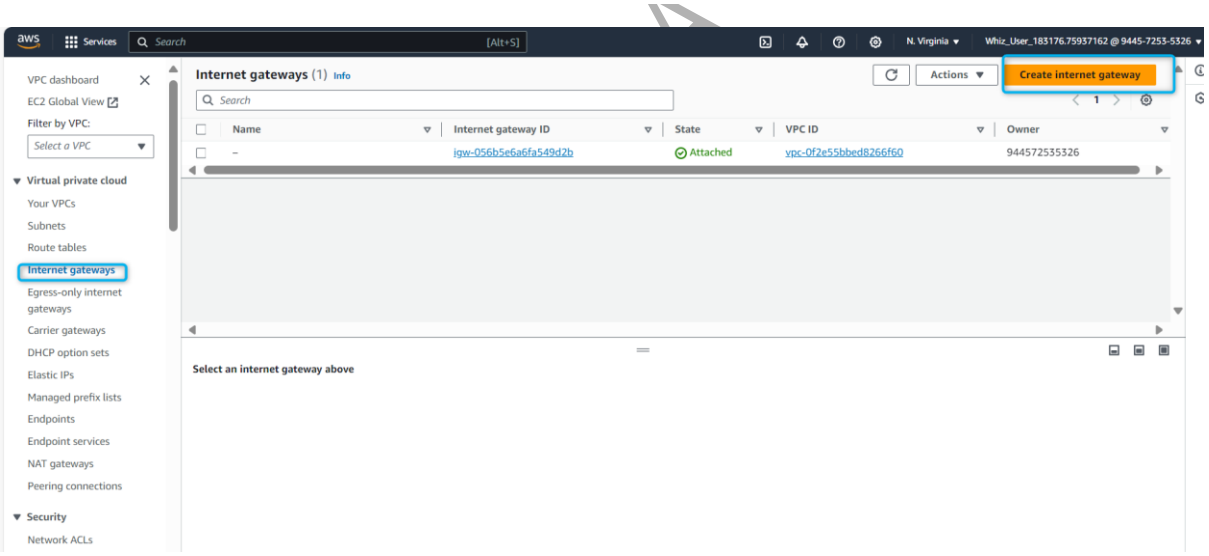

aws
Services
Search
[Alt+S]
VPC
Subnets
Create subnet
Create subnet Info
VPC
VPC ID
Create subnets in this VPC.
vpc-02d126d121a842a1c (my-vpc)
for subnet 2 pvt
same vpc
Associated VPC CIDRs
IPv4 CIDRs
10.0.0.0/16

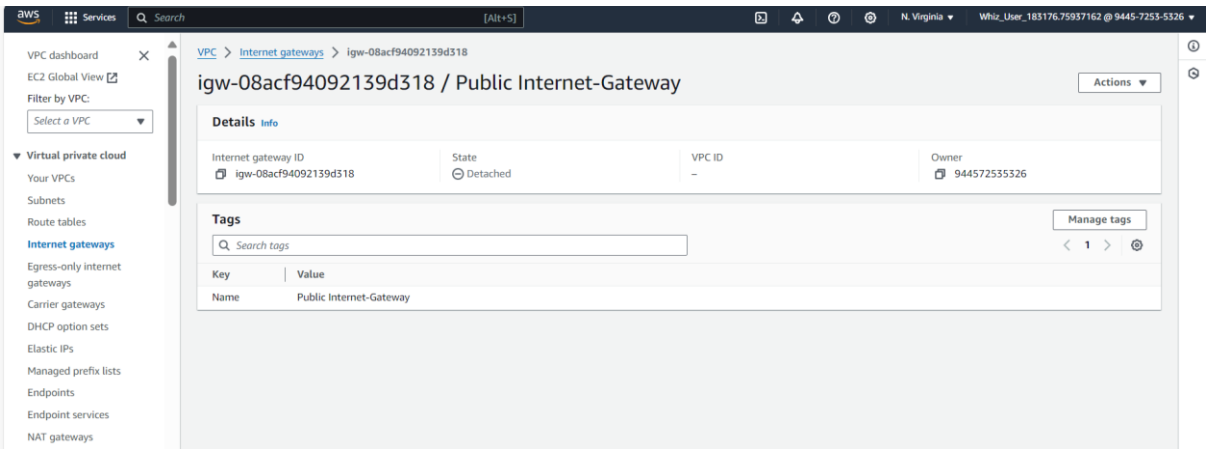




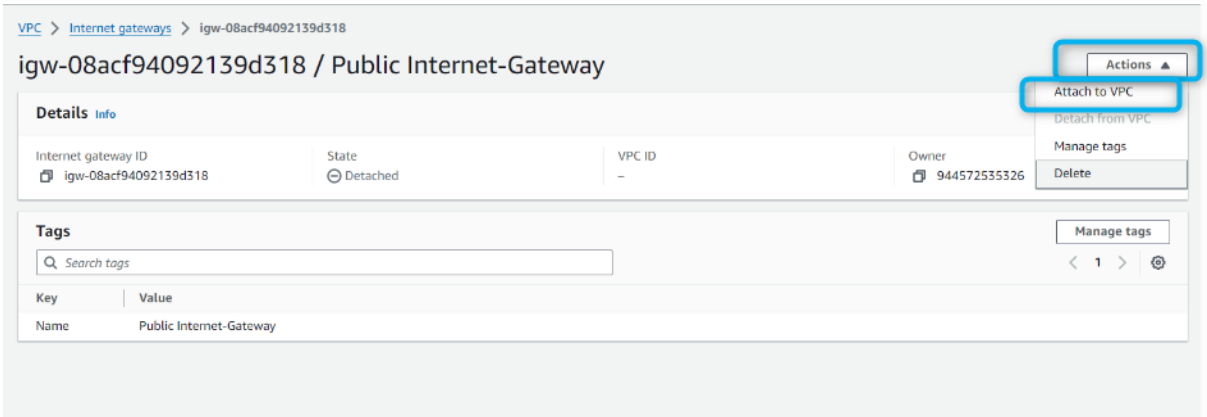
3. Create Internet Gateway

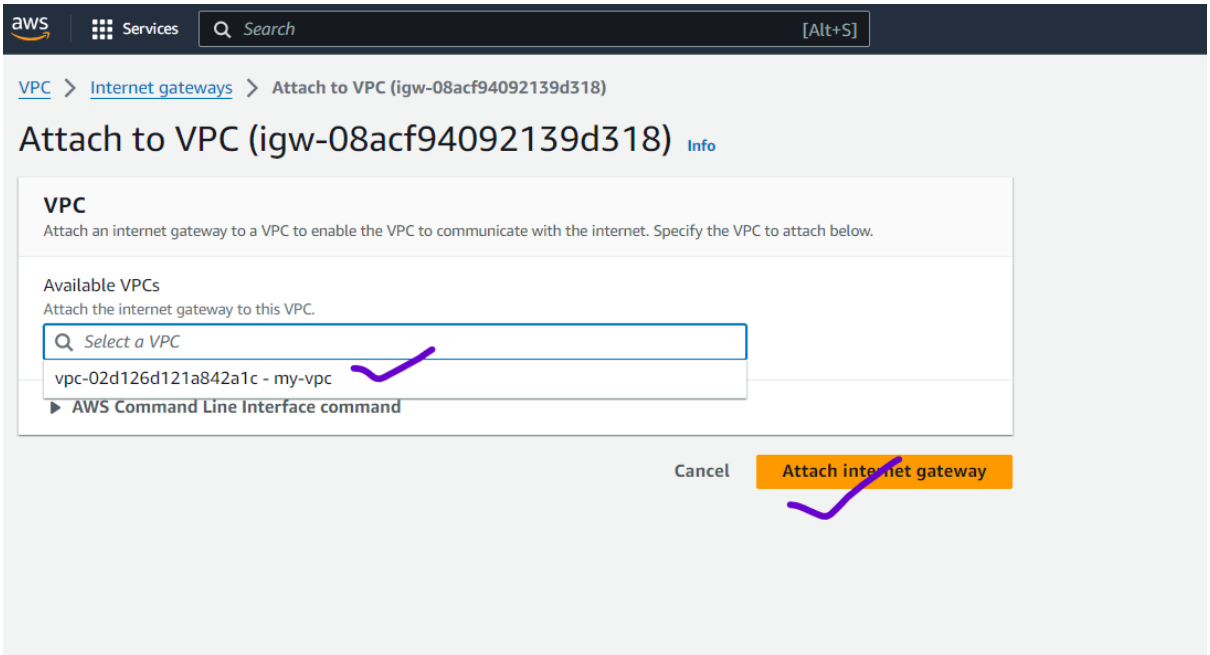

vpc
Internet Gateway
public subnet
Private subnet


Internet gateways (1) Info
Actions
Create internet gateway
Name
Internet gateway ID
State
VPC ID
Owner
igw-056b5e6a6fa549d2b
Attached
vpc-0f2e55bbed8266f60
944572535326
Select an internet gateway above
VPC dashboard
EC2 Global View
Filter by VPC:
Select a VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
Peering connections
Security
Network ACLs

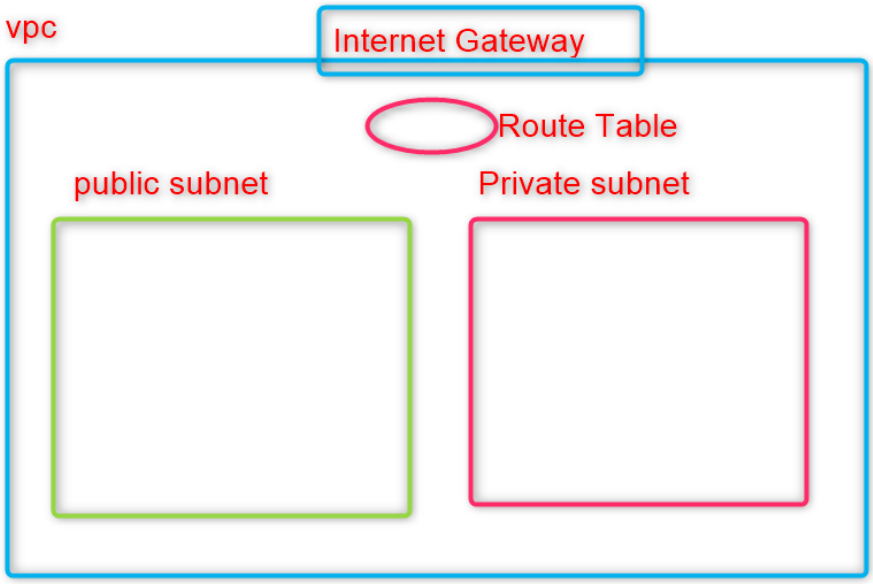

VPC > Internet gateways > igw-08acf94092139d318
igw-08acf94092139d318 / Public Internet-Gateway
Actions
Details Info
Internet gateway ID
igw-08acf94092139d318
State
Detached
VPC ID
Owner
944572535326
Tags
Manage tags
Search tags
Key
Value
Name
Public Internet-Gateway
VPC dashboard
EC2 Global View
Filter by VPC:
Select a VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways

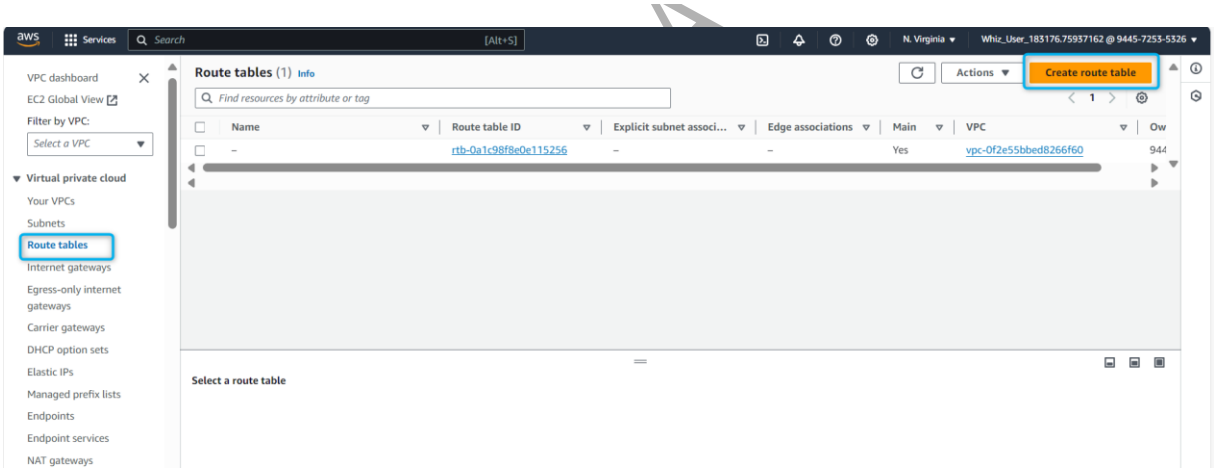
Internet gateway created now attached this internet gateway to vpc.

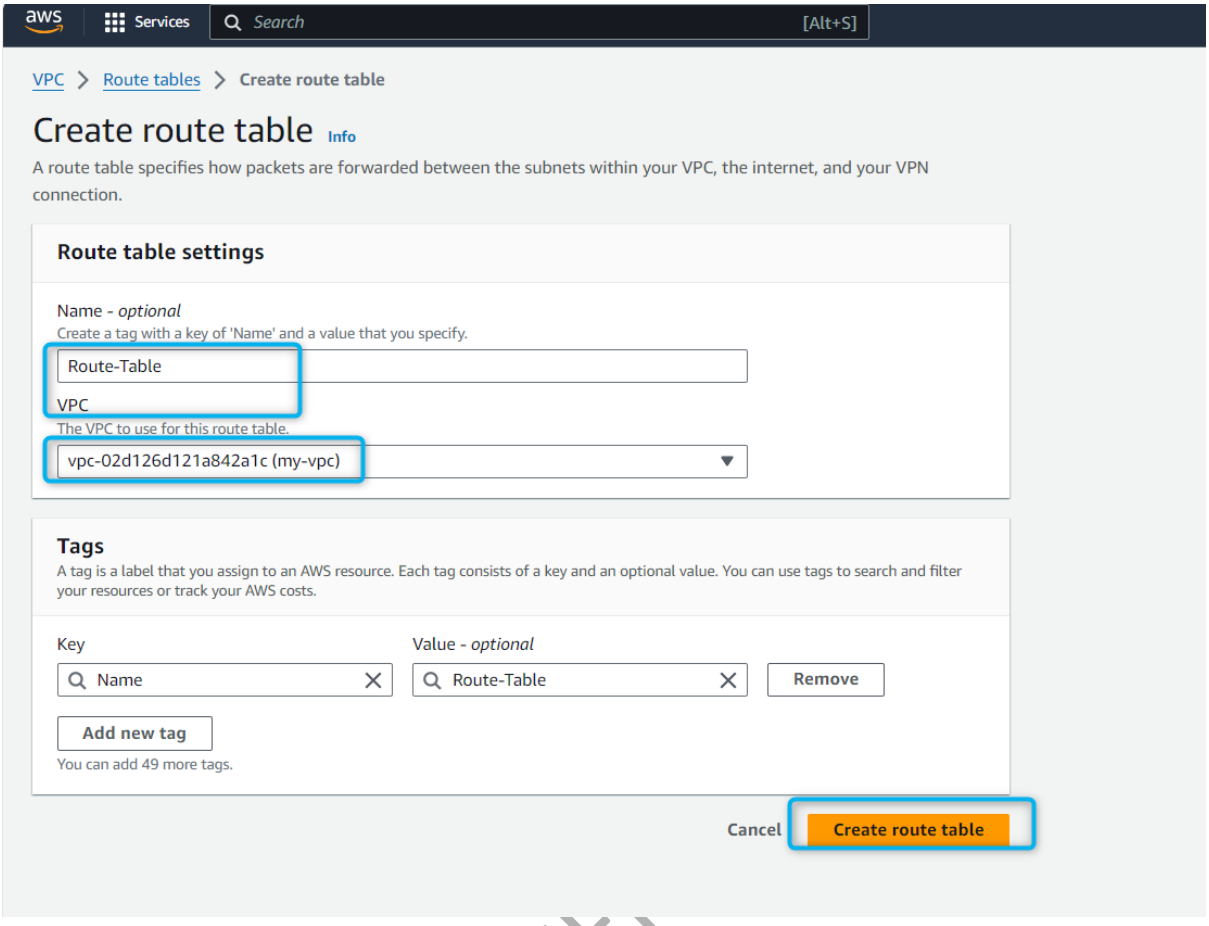


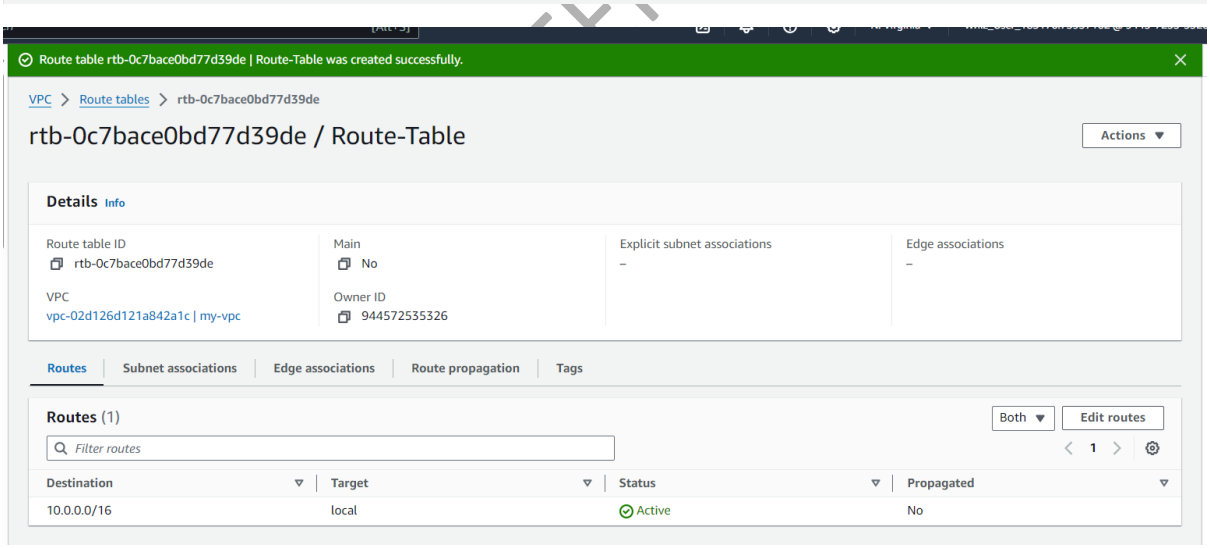


4.Create Route Table

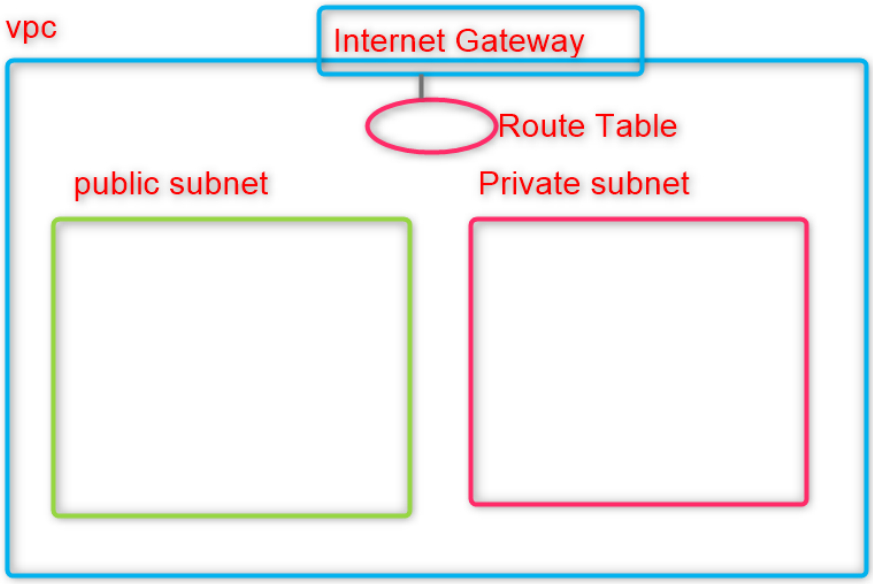

vpc
Internet Gateway
Route Table
public subnet
Private subnet

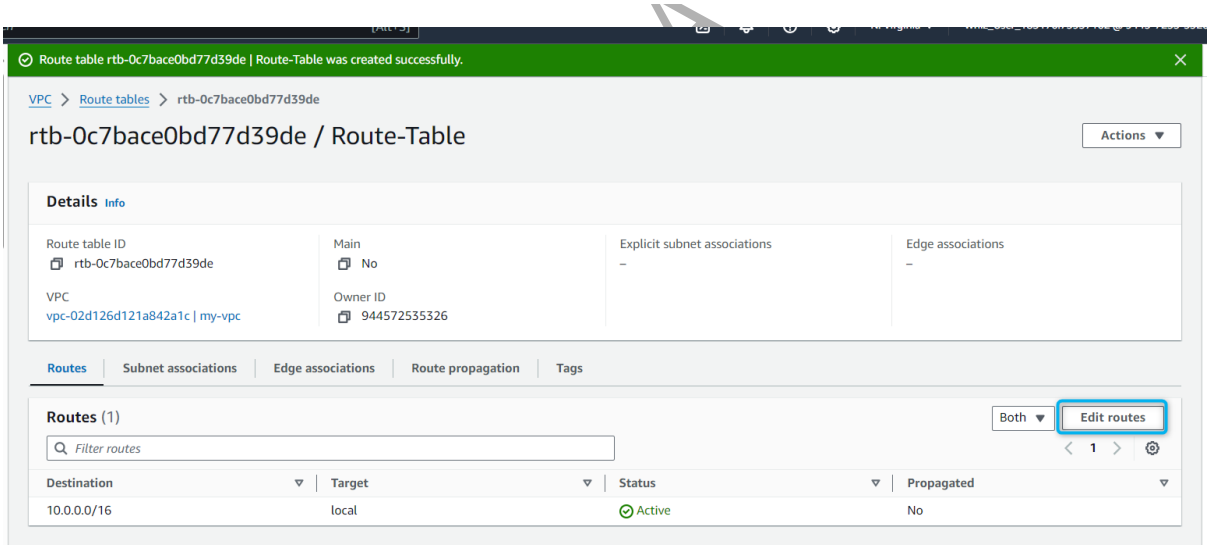

aws
Services
Search
[Alt+S]
N. Virginia
Whiz_User_183176.75937162 @ 9445-7253-5326
VPC dashboard
EC2 Global View
Filter by VPC:
Select a VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
Route tables (1) Info
Find resources by attribute or tag
Actions
Create route table
Name
Route table ID
Explicit subnet associ...
Edge associations
Main
VPC
rtb-0a1c98f8e0e115256
Yes
vpc-0f2e55bbed8266f60
Select a route table

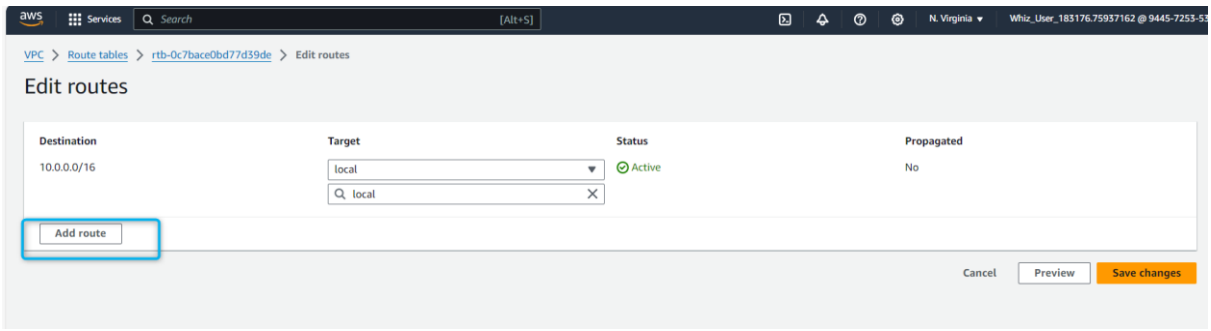


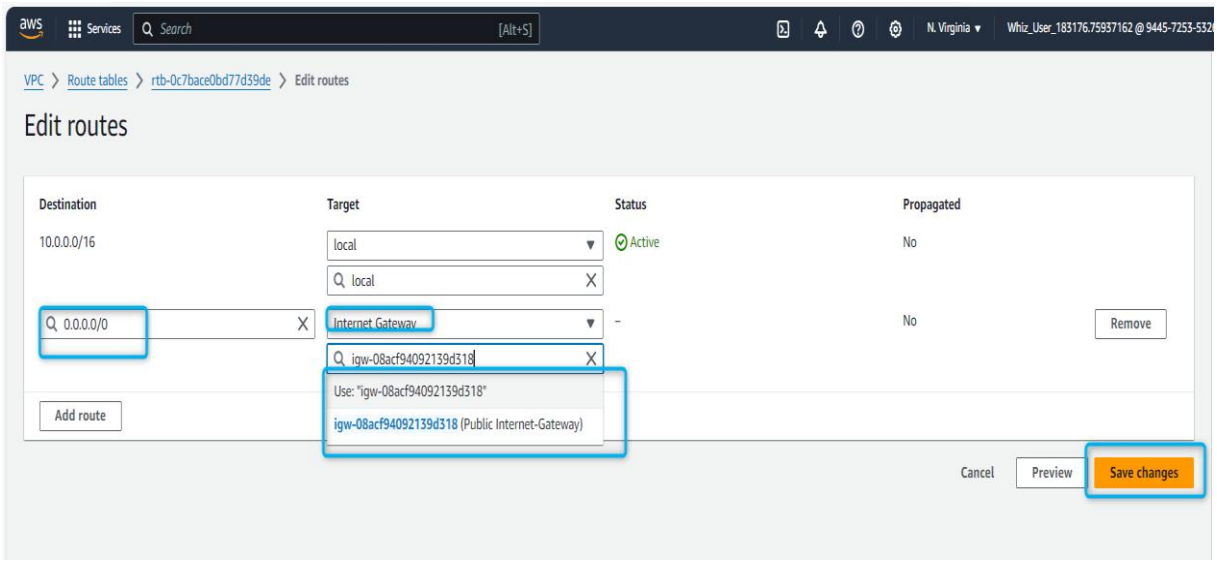


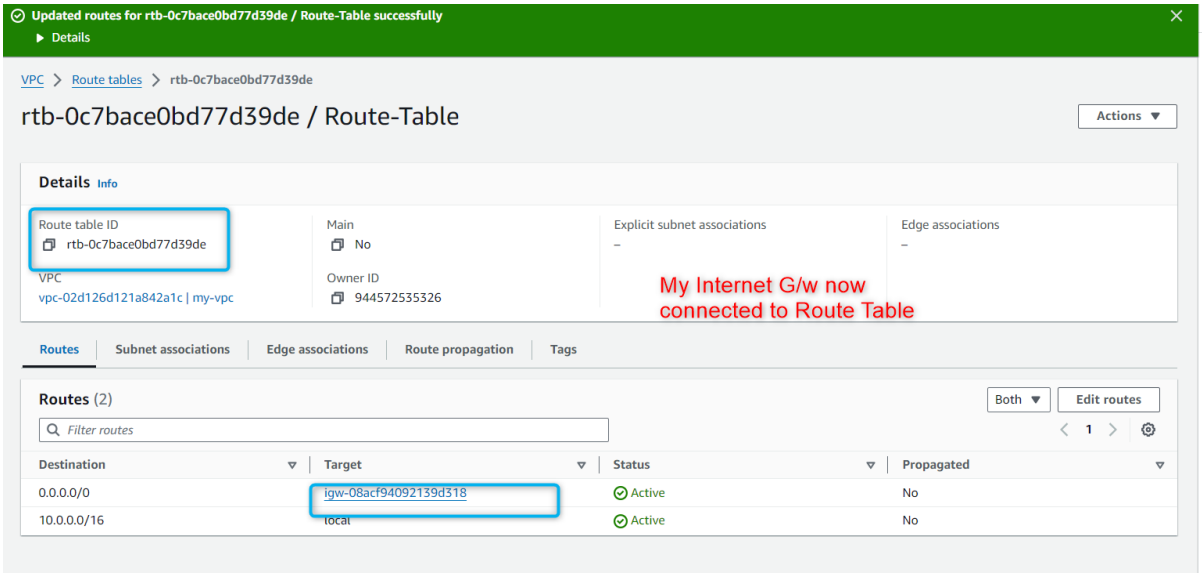
- Attach this route table to the internet Gateway.


vpc
Internet Gateway
Route Table
public subnet
Private subnet

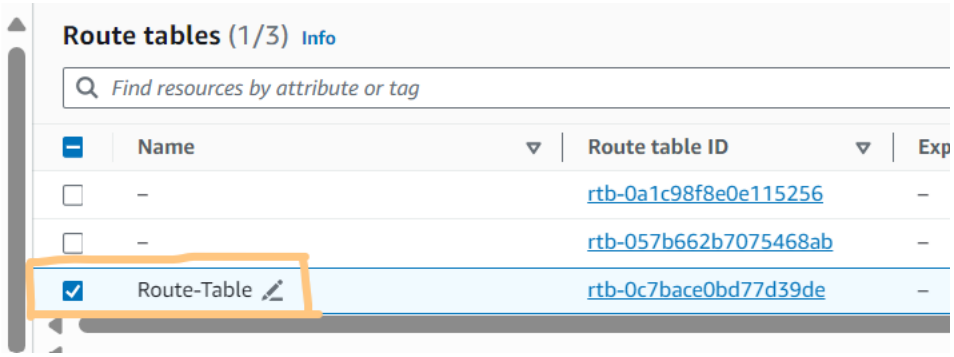

Route table rtb-0c7bace0bd77d39de | Route-Table was created successfully.
VPC > Route tables > rtb-0c7bace0bd77d39de
rtb-0c7bace0bd77d39de / Route-Table
Actions
Details Info
Route table ID
rtb-0c7bace0bd77d39de
Main
No
Explicit subnet associations
–
Edge associations
–
VPC
vpc-02d126d121a842a1c | my-vpc
Owner ID
944572535326
Routes
Subnet associations
Edge associations
Route propagation
Tags
Routes (1)
Both
Edit routes
Filter routes
1
Destination
Target
Status
Propagated
10.0.0.0/16
local
Active
No

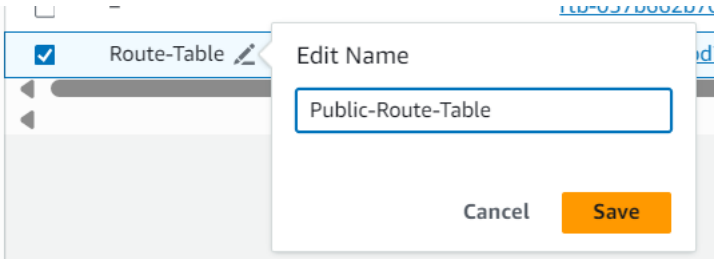

Services
Search
[Alt+S]
N. Virginia
VPC > Route tables > rtb-0c7bace0bd77d39de > Edit routes
Edit routes
Destination
Target
Status
Propagated
10.0.0.0/16
local
Active
No
local
Add route
Cancel
Preview
Save changes

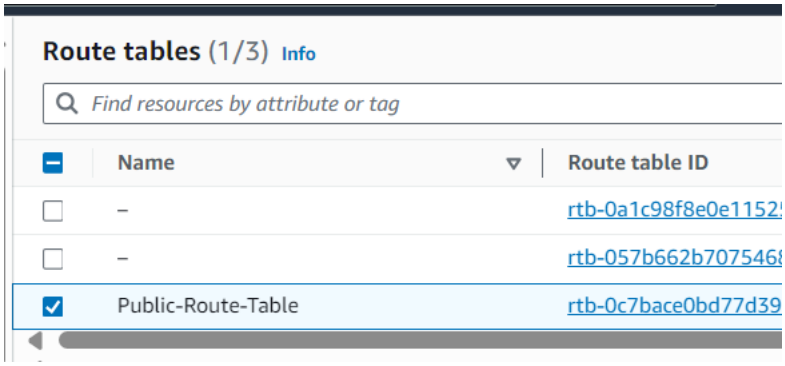




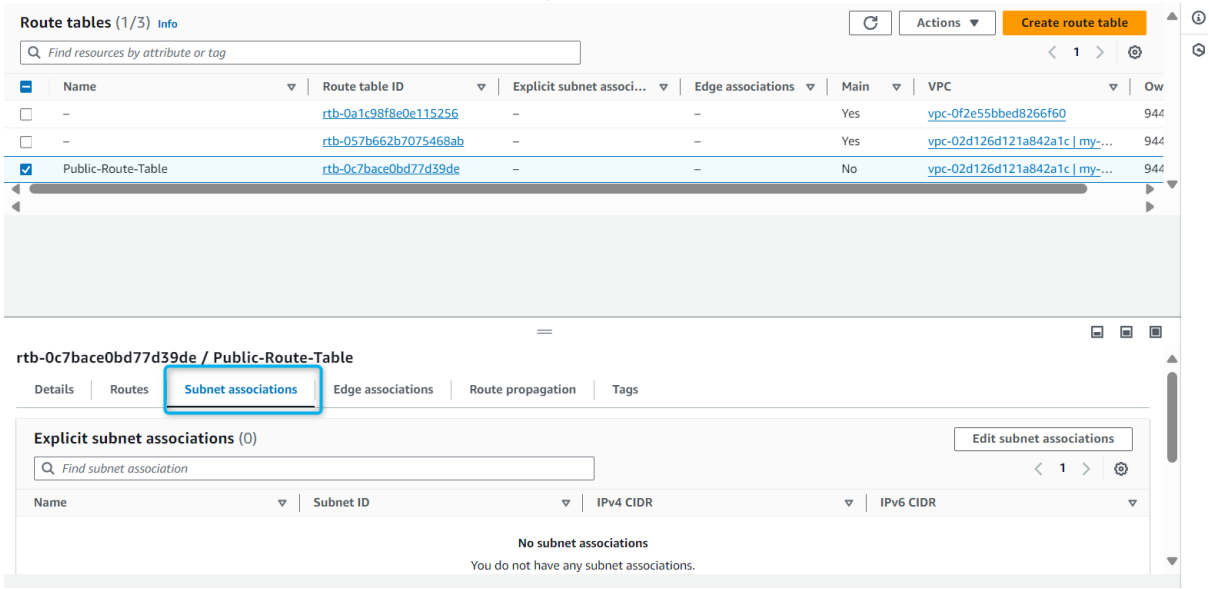
- Now let me rename my Route Table name to Public Route table.

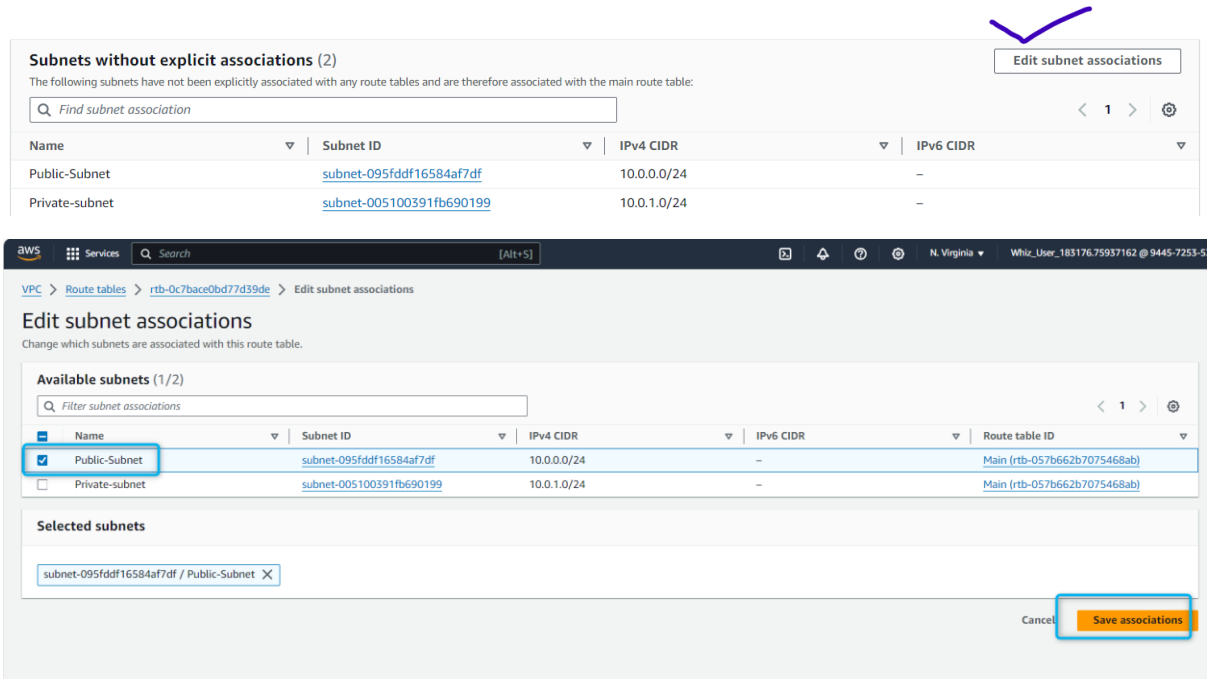


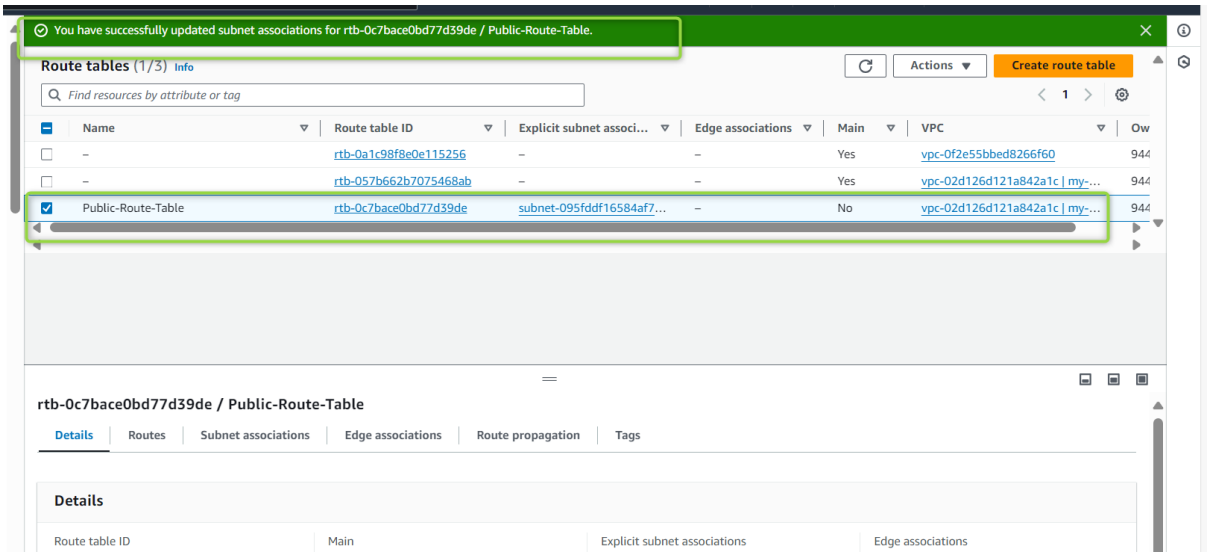




- Now as my Public Route Table is attached with internet gateway and getting internet into it. I can attach my Public Route Table with my Public Subnet so that my Public Subnet will also able to get Internet Access.







So my public subnet can able to get internet from Route table.

Now we will use this same above custom network to connect our EC2 instance with internet so that we can able to ping goole.com with help of internet into ec2 server after configuring this custom network.

Amazon Web Services – Security Group vs NACL

Security groups and NACL both act as virtual firewalls which control the traffic from Inbound and Outbound. In this article, we will discuss the difference between Security Groups and NACL on Amazon Web Services.

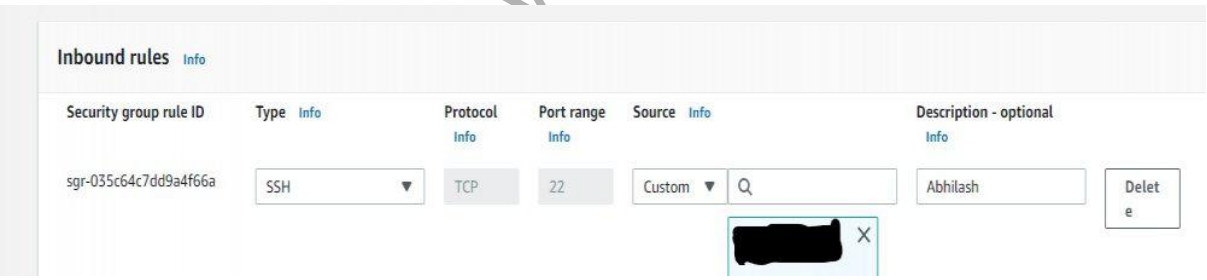
Security Group:

Security groups are virtual shields or protectors of EC2 instances. Unless specifically allowed By default all Inbound traffic is blocked whereas all Outbound traffic is allowed from the Instance.

We can edit inbound and outbound rules after creating the Security Group. Here is an example of default outbound rules which allow all traffic for all protocols.

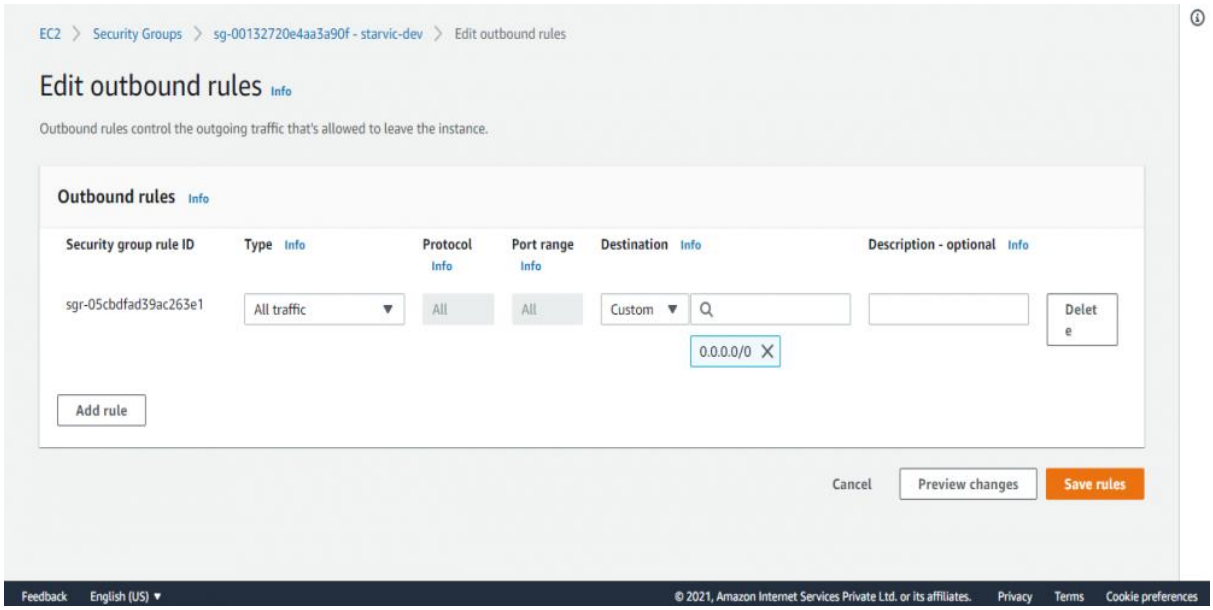
It is virtual firewall for your EC2 instances to control incoming and outgoing traffic

Here, we are adding inbound rules for protocol SSH with the default port of 22 for our current IP address here.



1. Above picture is for Inbound rule

In SG we can block traffic at inbound level only



2. Outbound

The above picture there is no provision to block outbound rules so in SG there by default outbound allowed

In the security groups, we cannot block a specific IP address because it doesn't have any DENY rule just like ALLOW rule

Limits of Security Groups :

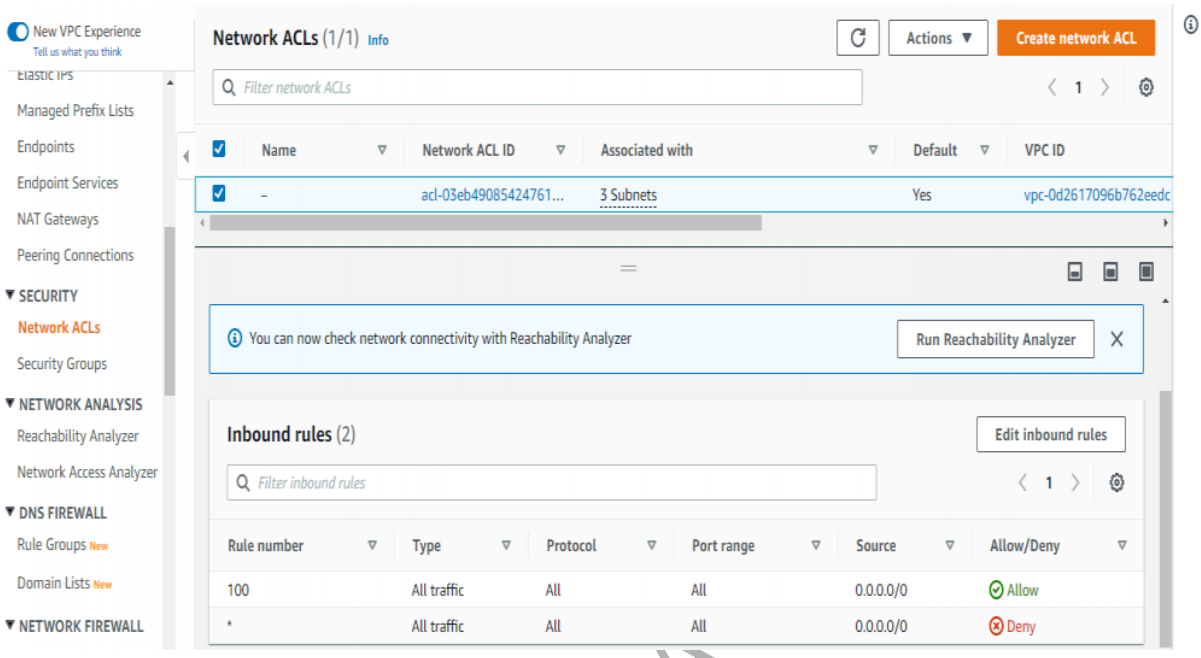
For a specific Security Group, the maximum Inbound and Outbound rules is 60

Network Access Control List (NACL):

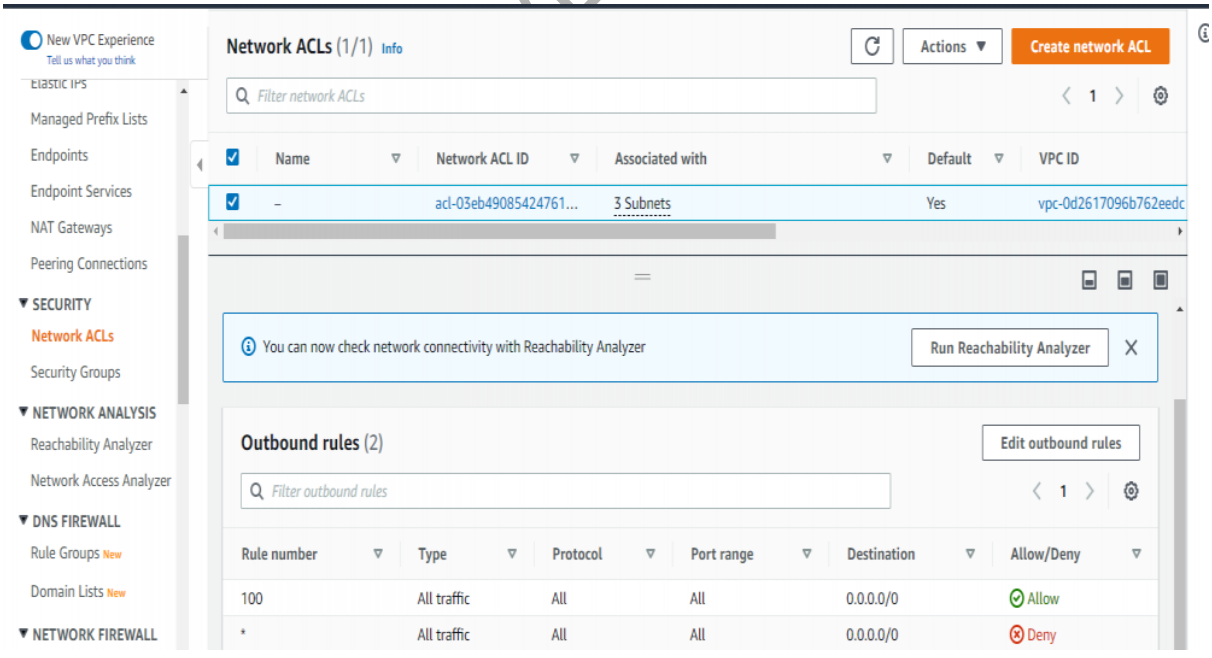
Network Access Control List is also a virtual firewall for subnets, which controls the Inbound and Outbound traffic of Subnets. After the creation of VPC, a Default NACL will be associated and allow all Inbound and Outbound Traffic.

In NACL just like Security Groups, it contains set of Inbound and Outbound Rules , that can either allow or deny Traffic into or out of subnets. Since we

have option to allow or deny traffic the order of the rules becomes important so that AWS uses a concept of rule number.



1. Inbound rule



2. Outbound Rule

Amazon EC2 (Elastic Compute Cloud)

- Introduction and Purpose of having EC2
- Instances
- Instance Types
- Regions and Availability Zones
- Purchasing Options
- AMI
- Key Pairs
- Launch Templates
- Placement Groups
- Elastic Block Store (EBS) and Utilizing
- Snapshots
- Elastic IPs
- Load Balancers
- Elastic Load Balancer (ELB)
- Target Groups

CloudWatch Monitoring

Auto Scaling Groups (ASG)

Created by Athira KK

A Brief Introduction to EC2 Instance:

An EC2 instance, short for Elastic Compute Cloud instance, is a virtual server offered by Amazon Web Services (AWS) that provides resizable compute capacity in the cloud. Essentially, it allows users to rent virtual servers to run their applications or host their websites without the need to invest in physical hardware.

- EC2 instances offer a variety of configurations for CPU, memory, storage, and networking, catering to diverse workload requirements.
- Users can select from optimized instance types, such as general-purpose, memoryintensive, or high-performance computing, to suit their specific needs.
- Deployment and management are simplified through AWS Management Console, CLI, or SDKs, facilitating easy provisioning, monitoring, and scaling of instances.
- EC2 serves a wide range of use cases, from basic web hosting to complex data analytics, meeting the computing requirements of businesses across industries and sizes.
- EC2 follows a pay-as-you-go pricing model, ensuring cost-effectiveness by billing users based on instance type, usage duration, and other factors, thus accommodating varying business needs and budget constraints.

Purpose of having an EC2 Instance

- Flexibility: With a wide range of instance types and configurations, EC2 instances cater to diverse workload requirements, from basic web hosting to highperformance computing and data analytics.
- Scalability: EC2 instances offer scalable compute capacity, allowing businesses to easily adjust resources based on demand fluctuations. This scalability ensures efficient resource utilization and cost-effectiveness.

- Global Reach: With AWS's global infrastructure, EC2 instances are available in multiple regions worldwide. This enables businesses to deploy applications closer to their target audience for improved performance and latency optimization.
- Security and Reliability: EC2 instances provide a secure and reliable environment for running applications, with features like built-in encryption, firewalls, and monitoring tools to safeguard data and ensure high availability.
- Ease of Management: EC2 instances can be quickly deployed and managed through AWS Management Console, CLI, or SDKs, streamlining the process of provisioning, monitoring, and scaling instances as needed.

Amazon EC2 offers a variety of instance types tailored for diverse needs. These types feature different configurations of CPU, memory, storage, and networking capacity, providing flexibility in selecting resources for your applications. Additionally, each instance type includes multiple sizes, enabling easy scaling according to your workload's demands.

EC2 Instance Types Overview:

General Purpose:

- These instances offer a balanced mix of compute, memory, and networking resources.
- Ideal for applications like web servers and code repositories that require equal proportions of these resources.

Burstable Performance:

- Also known as the T instance family, these instances provide baseline CPU performance with the ability to burst above it when needed.
- Refer to the Amazon EC2 User Guide for Linux Instances for more details.

Compute Optimized:

- Designed for compute-intensive applications requiring high-performance processors.
- Suitable for tasks such as batch processing, media transcoding, and highperformance web servers.

Memory Optimized:

- Tailored for workloads processing large datasets in memory.
- Deliver fast performance for applications with substantial memory requirements.

Storage Optimized:

- Intended for workloads demanding high, sequential read and write access to large data sets on local storage.
- Optimized to provide high IOPS for applications requiring rapid random I/O operations.

Accelerated Computing:

- Utilizes hardware accelerators or co-processors for efficient processing of functions like floating-point calculations and graphics processing.
- Enhances performance compared to software-based processing on CPUs.

High-Performance Computing (HPC):

- Purpose-built for running HPC workloads at scale on AWS, offering optimal priceperformance.
- Ideal for complex simulations and deep learning tasks requiring high-performance processors.

Previous Generation:

- AWS continues to support previous generation instance types for users who have optimized their applications around them.
- While current generation instance types offer better performance, previous generation types are still available for legacy applications.

Understanding AWS Regions, Availability Zones, and Deployment Strategies

Regions:

- AWS divides its global infrastructure into geographical regions, such as North America, Europe, Asia Pacific, etc.
- Each region is completely isolated from others and consists of multiple Availability Zones.
- Users select a region when deploying resources like EC2 instances. The choice of region affects latency, compliance, and data residency.

Availability Zones (AZs):

- Availability Zones are distinct data centers within a region, designed for fault tolerance and high availability.
- They are physically separate from each other and are connected through highspeed, low-latency links.
- Deploying resources across multiple AZs enhances fault tolerance. If one AZ experiences an issue, applications can continue running in others without interruption.

Deployment Strategy:

- To ensure high availability and fault tolerance, it's recommended to distribute resources (like EC2 instances) across multiple AZs within a region.
- This strategy protects against failures or outages in a single AZ and ensures that applications remain accessible even if one AZ becomes unavailable.

Scalability and Performance:

- Deploying resources across multiple AZs allows for better scalability and performance.
- Load balancers can distribute incoming traffic evenly across multiple AZs, optimizing resource utilization and improving response times for users.

Data Residency and Compliance:

- Different regions may have varying data residency regulations and compliance requirements.
- Users need to select the appropriate region and AZs to ensure compliance with local regulations and keep data within designated geographic boundaries.

Cost Considerations:

- Costs for AWS services may vary across regions, so users should consider pricing differences when selecting a region for deployment.
- Additionally, data transfer costs may apply when transferring data between regions or AZs.

EC2 Major Purchasing options:

1. **On Demand Instances** — Pay for the compute capacity per second , no up- front payment.We can decide when to launch,terminate,start.stop,or reboot.
Use cases: short-term work-loads , irregular workloads that cannot be interrupted.

2. **Reserved Instances** — We can reserve instances in terms of 1year and 3year. Provide up to 70% discount compared to On-Demand. Payment method can be all upfront, partial upfront,no upfront. If you purchase Reserved Instances of a specific type in an availability zone and the same type is available in same zone then automatically charge at lower rate.
Use cases — Long-term workloads , Applications with steady state or predictable usage.

3. **Spot Instances** — Most cost effective , We can request for unused instances up to 90% discount. But if someone is paying more than of yours, then instance will get loosed. So it is less reliable.
Use cases — Image processing , non-critical applications.

4. **Dedicated Hosts**– Physical server completely dedicated for your use. More expensive and can be purchase as on demand or reserved. Billing will be per-host.
Use cases — Companies that have strong regulatory or compliance needs.

5.**Dedicated Instances** — Instances running on hardware that's dedicated to a single user. It can share hardware with other instances in same account. Billing will be per-instance.

6.**Savings Plans** — This will assist the user to lower the EC2 costs if they agree to keep it for a specific period — ideally, one or three years. The number of active instances are unrestricted.

AMI (Amazon Machine Image):

- An AMI is a template used to create virtual machines (instances) within the Amazon EC2 service.

- It contains the operating system, application server, and applications necessary to launch an instance.
- You can create custom AMIs or use pre-built AMIs provided by AWS and other users.

AMI Images:

- AMI images are the actual files stored in Amazon S3 that represent the templates used to launch EC2 instances.
- These images include the root volume snapshot and metadata required to launch an instance.
- AMI images are versioned, allowing you to maintain multiple versions of an AMI and roll back to previous versions if needed.

Placement Groups

- Placement groups are the way of logically grouping interdependent instances together in a selected region.
- In other words, instances that are existing within a common availability zone can be grouped under placement group in order to suit workload requirements.
- By using the placement group, we will increase the performance or improve the availability.

Types

- Cluster placement group
- Spread placement group
- Partition placement group

Benefits

- Can be able to launch multiple instances within the same AZ.
- Low network latency.
- High network throughput.

Limitations

- Cannot merge placement groups.
- Cannot launch dedicated hosts in placement groups.
- Instances cannot span into multiple placement groups.

Cluster Placement Group

- All the instances are placed in a same hardware rack inside an
- availability zone.
- But if the rack fails, all instances fail at the same time.
- This type of group will enable to achieve low latency network.
- Good for High Performance applications.

Use Case — Application with low latency and high throughput.

Spread Placement Group

- All EC2 instances are placed in different hardware racks in a
- single availability zone.
- A rack failure will not affect more than one instance.

We can create up to 7 EC2 instances per availability zone in spread placement group.

- Good for high availability applications and not suitable for high performance applications.
- It can span multiple availability zones in the same region.
- **Use Case**- Critical applications that needs to be isolated from failure from each other.

Partition Placement Group

- Group of instances spread across racks, and the instances in one partition do not share the underlying hardware with instances in different partition.
- If a rack fails, it will affect on the instances within that
- particular partition only.
- Partitions can be in different availability zones in the same
- region.

It strikes a balance between high performance and high availability.

Use Case — Big data applications like HDFS, HBase, Cassandra, Kafka.

In AWS EC2, there are three main types of load balancers:

Classic Load Balancer (CLB):

- CLB is the original load balancer service provided by AWS.
- It operates at both the application and transport layers (Layer 4 and Layer 7) of the OSI model.
- CLB supports distributing incoming traffic across multiple EC2 instances in multiple Availability Zones.
- It is suitable for applications that require simple load balancing without advanced features.

Application Load Balancer (ALB):

- ALB operates at the application layer (Layer 7) of the OSI model.
-

It supports advanced routing features such as host-based routing, path-based routing, and routing based on HTTP headers.

- ALB can handle HTTP and HTTPS traffic and provides support for WebSocket and HTTP/2 protocols.

- It is ideal for modern web applications with multiple microservices or containers.

Network Load Balancer (NLB):

- NLB operates at the transport layer (Layer 4) of the OSI model.
- It provides ultra-low latency and high throughput, making it suitable for handling millions of requests per second.
- NLB supports routing traffic to targets using IP addresses and TCP ports, making it ideal for TCP and UDP-based applications.
- It is commonly used for applications that require high performance, scalability, and reliability, such as gaming, IoT, and real-time communication applications.

Target Groups are a fundamental component of Elastic Load Balancing (ELB) services in AWS, specifically associated with the Application Load Balancer (ALB) and Network Load Balancer (NLB). Here's a concise explanation:

Routing:

- When configuring a load balancer (ALB or NLB), you specify one or more target groups to route traffic to.
- Each target group defines the criteria for routing requests to its associated targets, such as based on a specific path, host, or HTTP header.

Health Checks:

- Target Groups perform health checks on their registered targets to ensure they are capable of handling requests.
- Unhealthy targets are automatically removed from the target group until they pass health checks again.

Load Balancing:

- Target Groups distribute incoming traffic evenly across their registered targets according to the configured load balancing algorithm (e.g., round-robin or least outstanding requests).

Port Configuration:

- Target Groups allow you to specify the port on which the targets receive traffic, allowing flexibility in routing requests to different ports on the same target.

Autoscaling:

- Autoscaling is a feature provided by AWS that automatically adjusts the number of EC2 instances in a fleet based on user-defined policies.
- It helps maintain the desired number of instances to handle varying levels of application traffic or workload demand.
- Autoscaling can scale both up (adding more instances) and down (removing instances) dynamically, ensuring optimal resource utilization and cost efficiency.

Autoscaling Groups:

- An Autoscaling Group (ASG) is a logical grouping of EC2 instances managed by the Auto Scaling service.
- ASGs define the scaling policies, launch configurations, and other parameters that govern the behavior of the instances within the group.
- ASGs automatically launch or terminate EC2 instances based on predefined scaling policies and health checks.
- ASGs can be configured to distribute instances across multiple Availability Zones to improve fault tolerance and high availability.

Ways to connect to EC2 instances via

- the command line: SSH connection
- EC2 Instance Connect:
- AWS CLI (Command Line Interface):
- Third-Party Tools (like PuTTY or MobaXterm) and more.....

The launch of an ec2 instance:

- Choose an AMI (Amazon Machine Image): This is a template for the virtual machine you want to create. It contains the operating system, software, configuration, and sometimes data.
- Select an Instance Type: This determines the computational resources (CPU, RAM, storage, etc.) of your EC2 instance. There are various types optimized for different use cases.
- Configure Instance Details: This includes settings like networking (VPC, subnet), IAM role, and user data (scripts or data to be run at instance launch).
- Add Storage: Configure the amount and type of storage for your instance. This is typically done using Amazon EBS (Elastic Block Store), which provides block-level storage volumes.
- Add Tags: Assign metadata to your instance in the form of key-value pairs. This helps with organization, management, and billing.
- Configure Security Group: Define firewall rules for controlling inbound and outbound traffic to your instance. Security groups act as virtual firewalls.
- Review and Launch: Review your instance configuration, make any necessary changes, select an existing key pair or create a new one for SSH access (for Linux instances), and then launch the instance.

After launching the instance, you can access it using SSH (for Linux instances) or RDP (for Windows instances) using the key pair you selected or created. Remember to properly secure your instance and follow best practices for managing AWS resources.

Let’s walk through the process of launching an EC2 instance.

An EC2 instance essentially functions as a virtual machine.

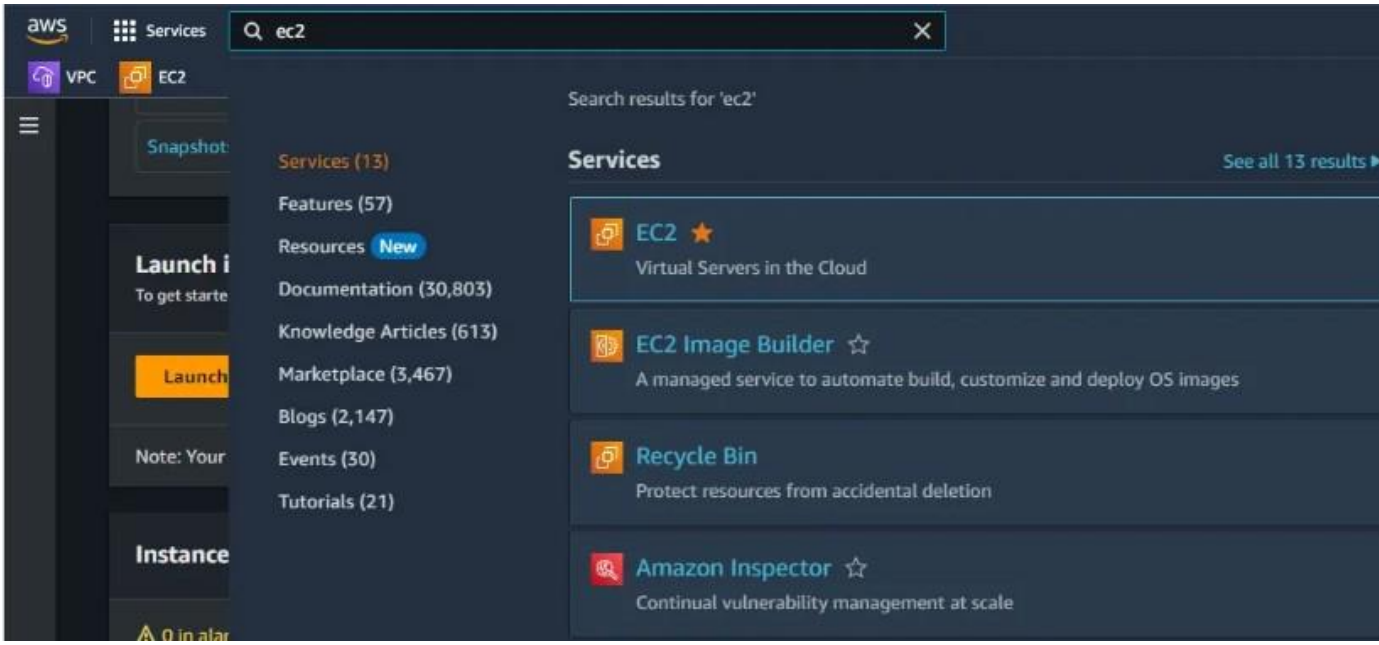
Firstly, log in to your AWS console.

Next, choose your desired region. For instance, I’ll be selecting the North Virginia region. It’s advisable to select a region based on your

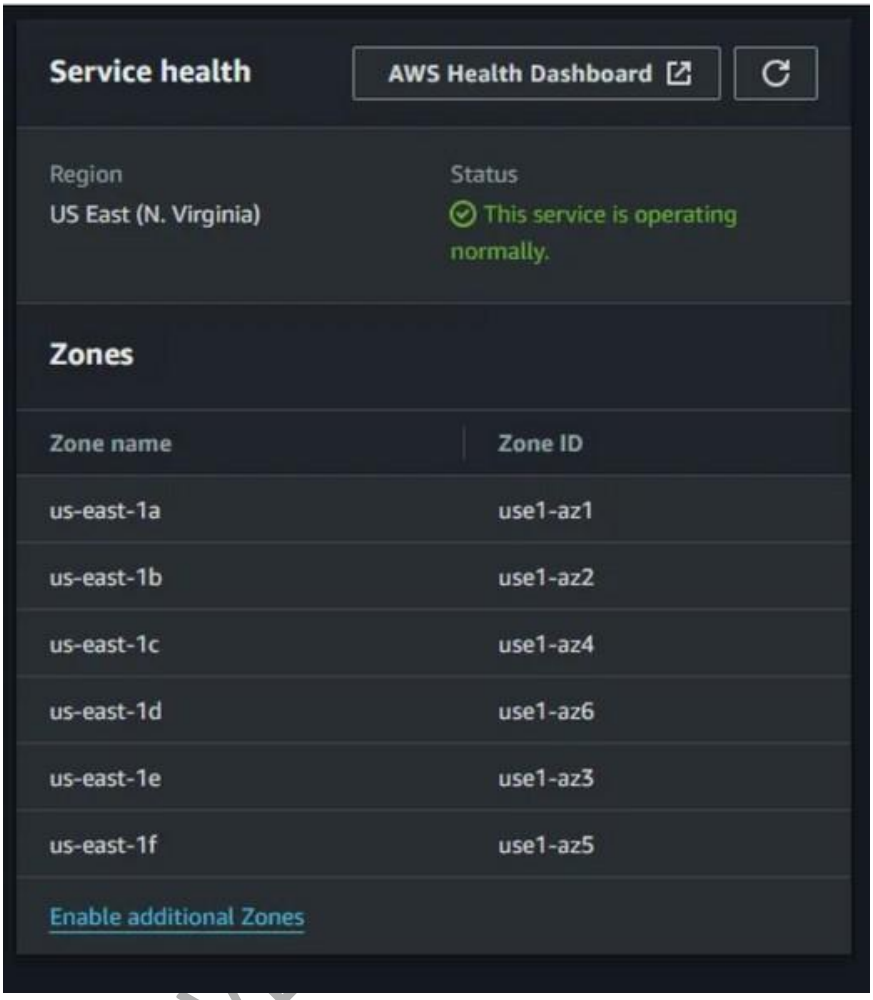
current location, as some regions may incur higher costs compared to those in the US.

Each region is identified by a code, such as US East 1 for North Virginia.

To navigate to the EC2 service, click on “Services” and locate EC2 either by browsing through all services or by directly searching for EC2.

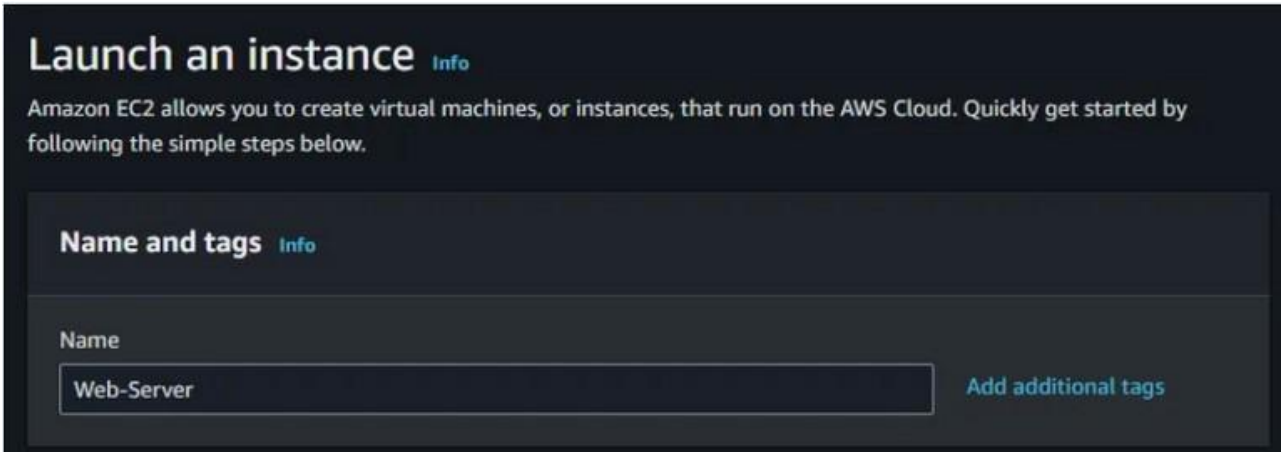


Once in the EC2 dashboard, you’ll find an overview of your instances, volumes, and other relevant information. Keep in mind that the dashboard doesn’t auto-refresh, so be sure to manually refresh after making any changes.
Scrolling down, you’ll notice zones within the region, each identified by a code like 1A, 1B, etc. Typically, a region will have at least two zones.

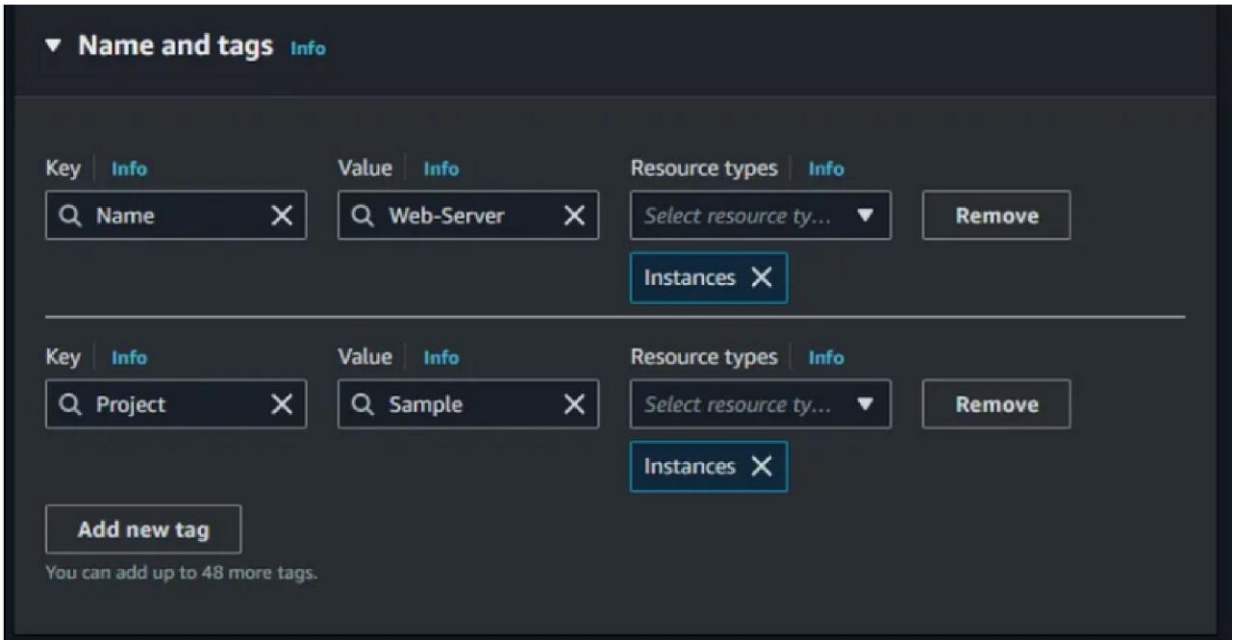


Click on “Instances” and then “Launch Instance” to initiate the instance creation process.

Before proceeding, it’s worth noting that the AWS console interface may evolve over time, but the core options and functionalities remain consistent.

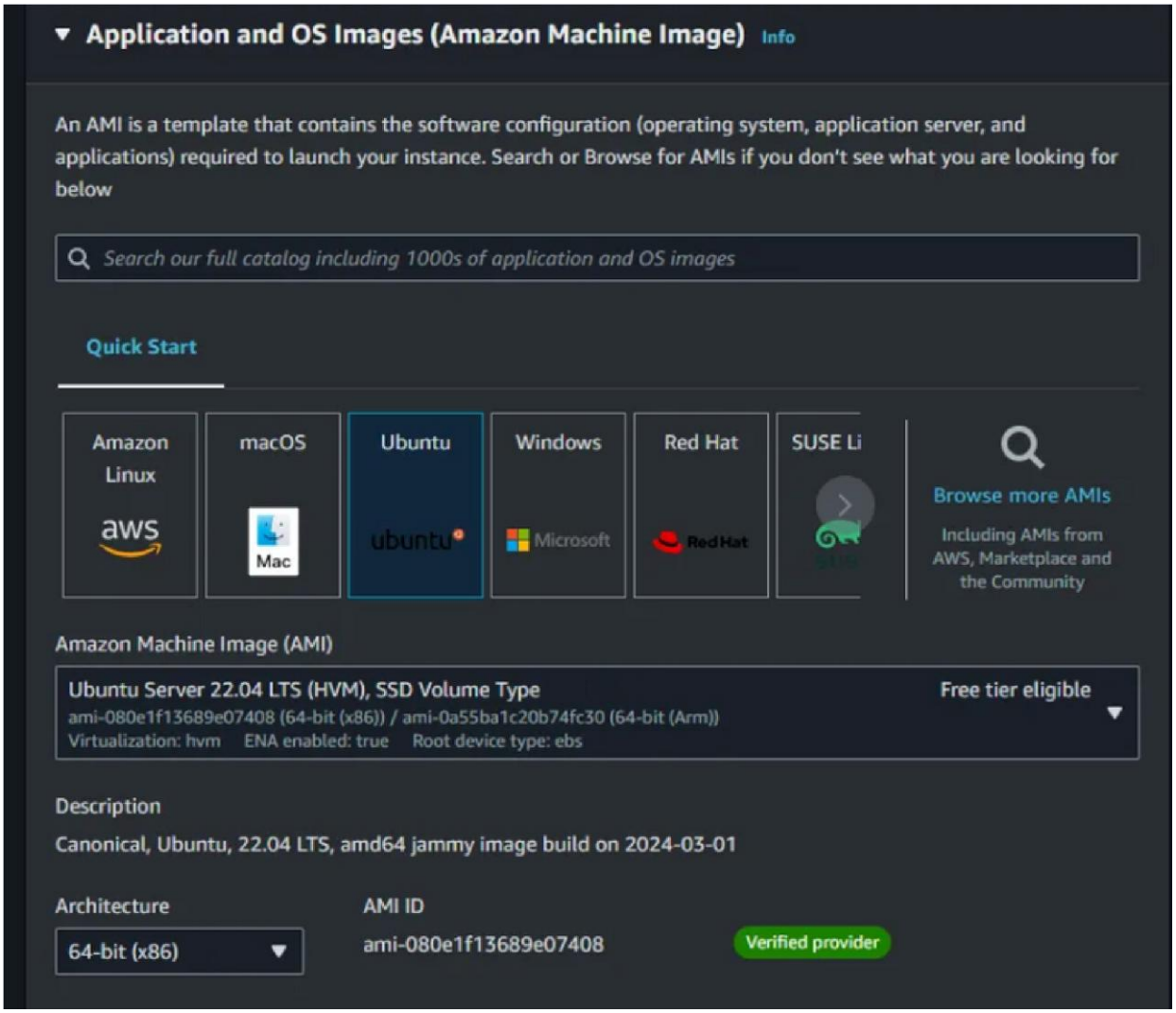


Now, let’s begin by assigning a name and optional tags to the instance. Tags can be useful for filtering and organization purposes.

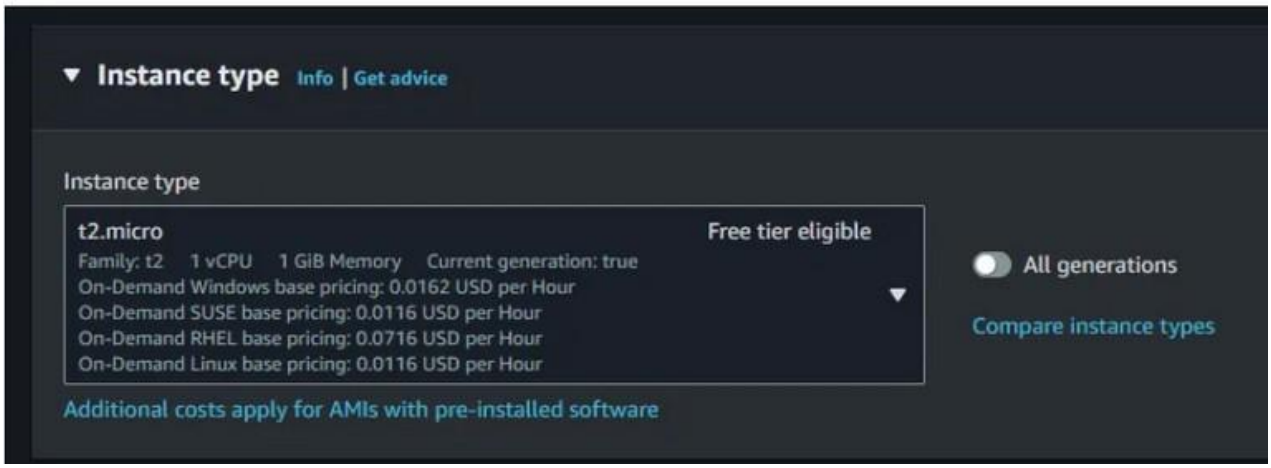


Moving on to selecting the Amazon Machine Image (AMI), you'll find a variety of options including quick start AMIs like Amazon Linux, Ubuntu, and Windows, as well as community AMIs and those from the AWS Marketplace. Carefully review the details and charges associated with each AMI.

For this example, let's choose the Ubuntu AMI provided by Amazon Web Services, which is free.



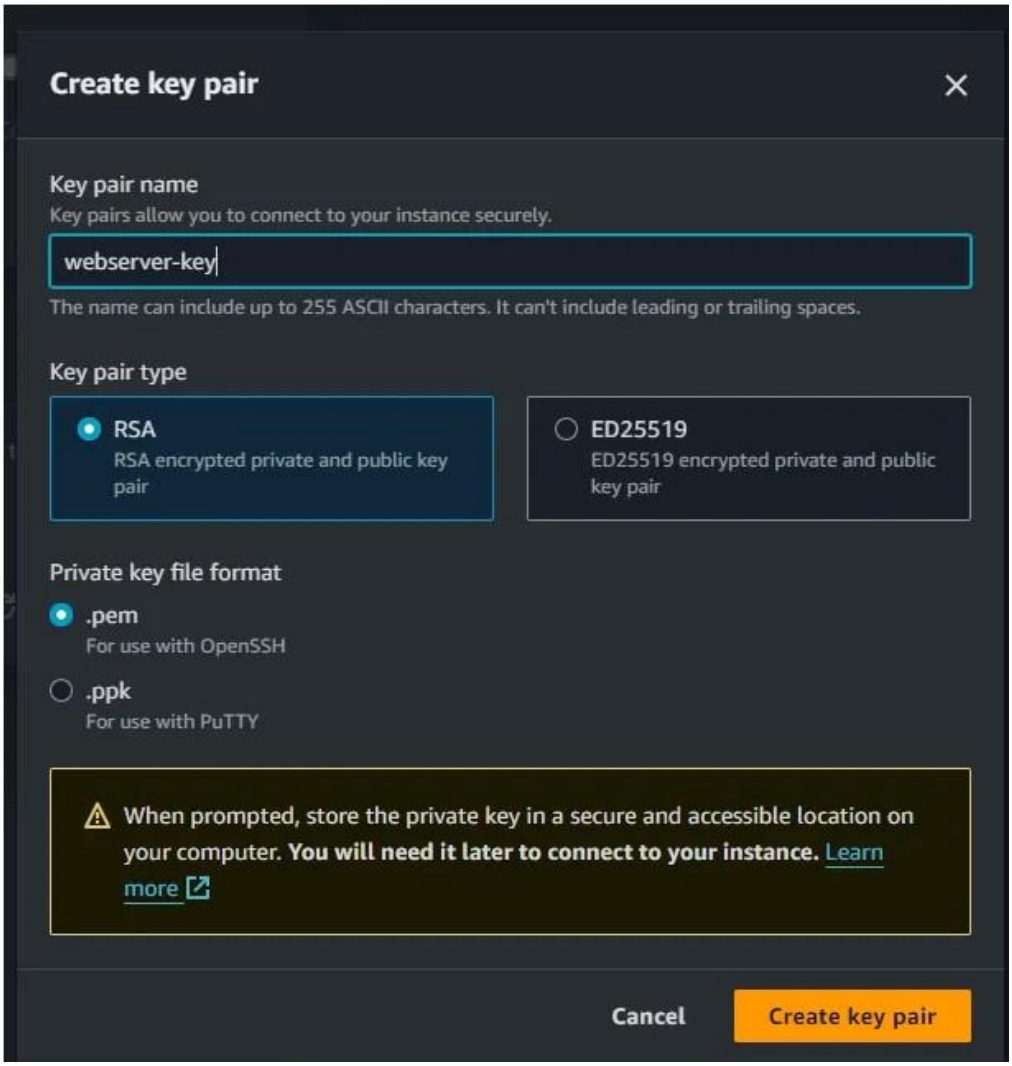
Ensure that the instance type selected aligns with your requirements. For free-tier usage, T2 micro instances are suitable for most learning purposes.



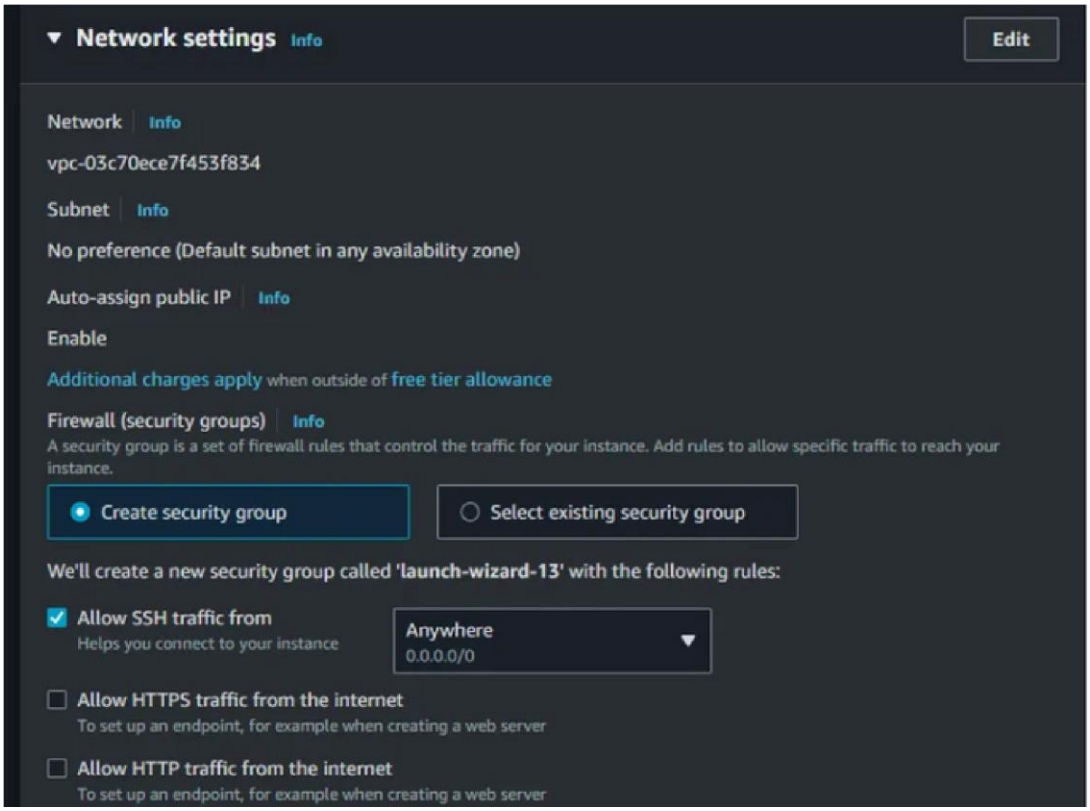
- Key pairs are used primarily for securely accessing EC2 instances. When

create an EC2 instance, you can specify a key pair to use for SSH access to the instance. The key pair consists of a public key that AWS stores, and a private key file that you download to your local machine.

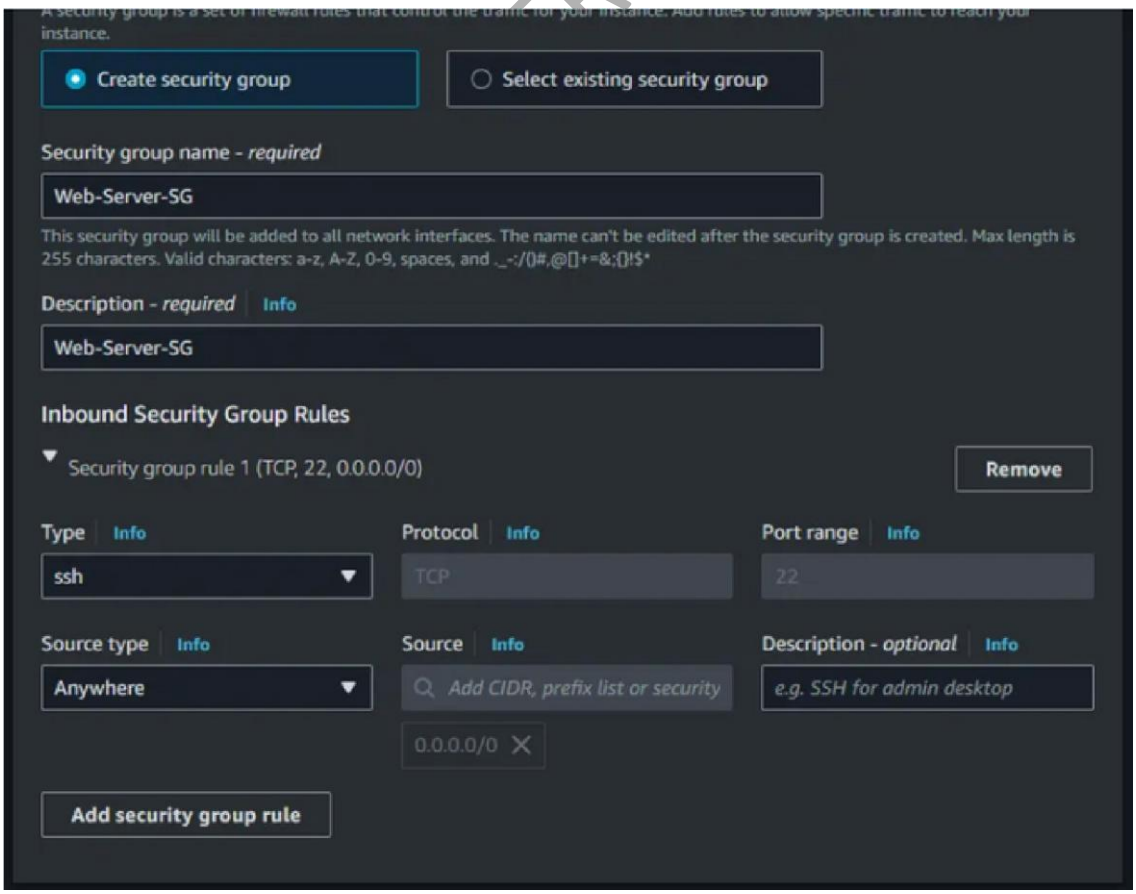
- It's important to note that AWS doesn't store your private key, so if you lose it, there's no way to retrieve it from AWS. Make sure to securely store your private keys and avoid sharing them with unauthorized users.



Proceed to configure the network settings, including security groups. Security groups act as mandatory firewalls for EC2 instances.



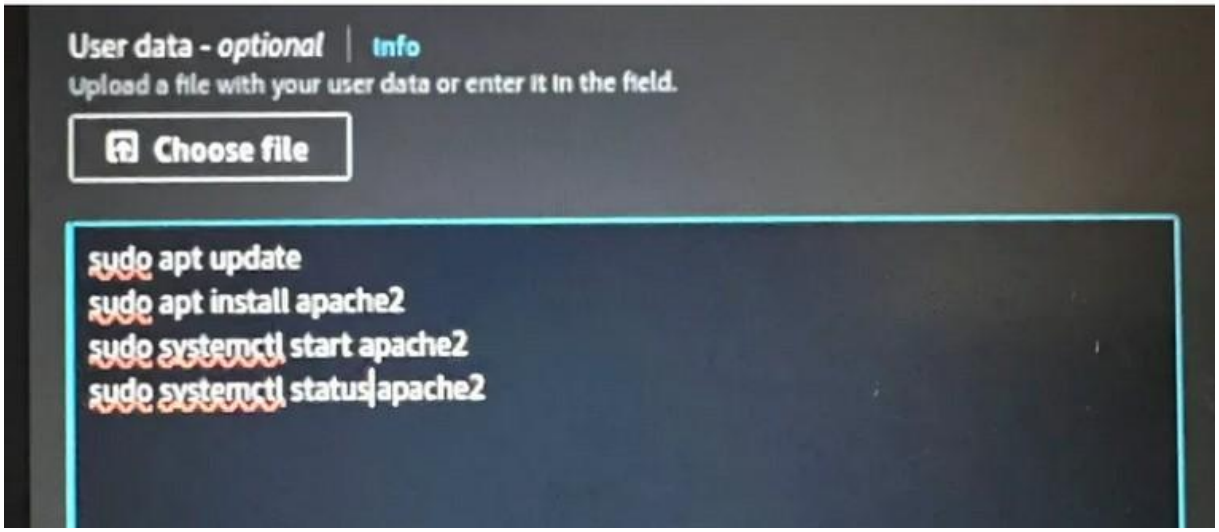
Customize the security group to allow necessary inbound and outbound traffic.



You can specify the storage size, typically defaulted to 10 GB, and further advanced details like user data, which allows for automated provisioning upon instance launch. Finally, review your configuration and launch the instance.

Upon launching, the instance may take a few moments to become available. You can monitor its status in the EC2 dashboard.
Next, establish a connection to the instance; in this example, I'm using EC2 Instance Connect.
There are two methods to install Apache2:

- By incorporating commands in the user data section before initiating the instance launch, as demonstrated below:



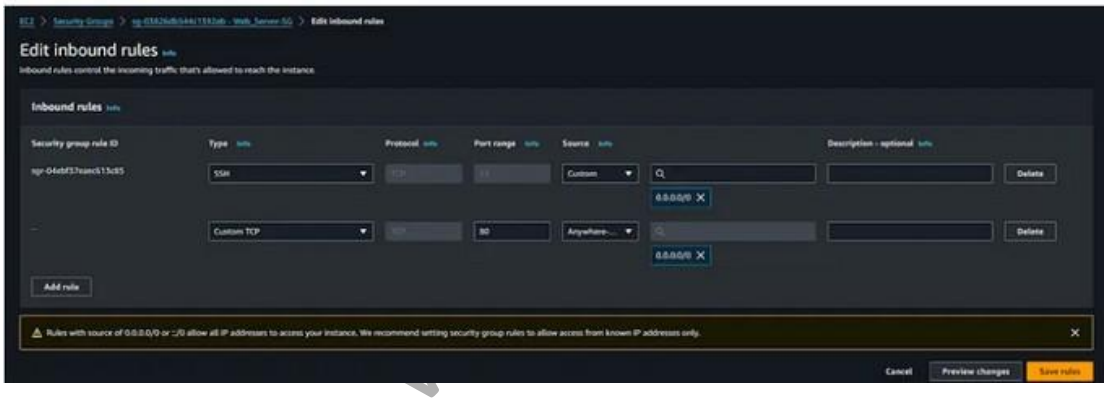
- Or, by executing the commands after connecting the ec2 instance via terminal.

- *sudo apt*
- *update sudo*
- *apt install*
- *apache2 -y*
- *sudo*

systemctl
start apache2
sudo
systemctl
enable
apache2 sudo
systemctl

status
apache2

Troubleshooting tip: If encountering connection issues, ensure the correct path to the SSH key or navigate to the directory containing the key before attempting to connect.
Once connected, you can verify the instance status, access web services like Apache, and configure firewall rules as needed to enable external access to your services. Additionally, click on the Security Group, edit inbound rules, add the following rule, and save the rules. It's important to note that allowing all IPs is solely for learning purposes and may vary depending on your project requirements.



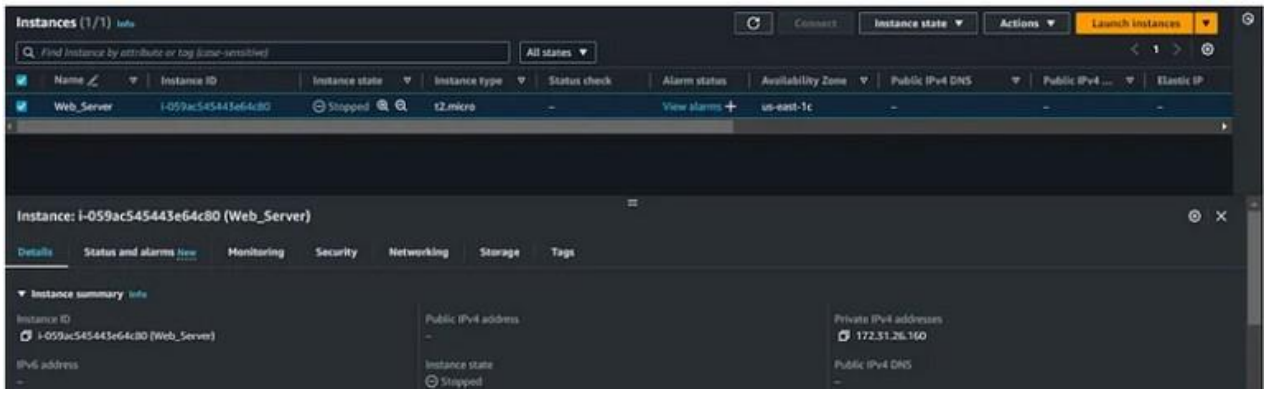
Go to the browser and copy the public IP followed by ':80' to view the Apache page.

Elastic IP

- Powering Off the EC2 Instance:
- Take note of the current public and private IP addresses of the
- instance.
- Stop the instance using the AWS Management Console.
- Observing IP Address Changes:

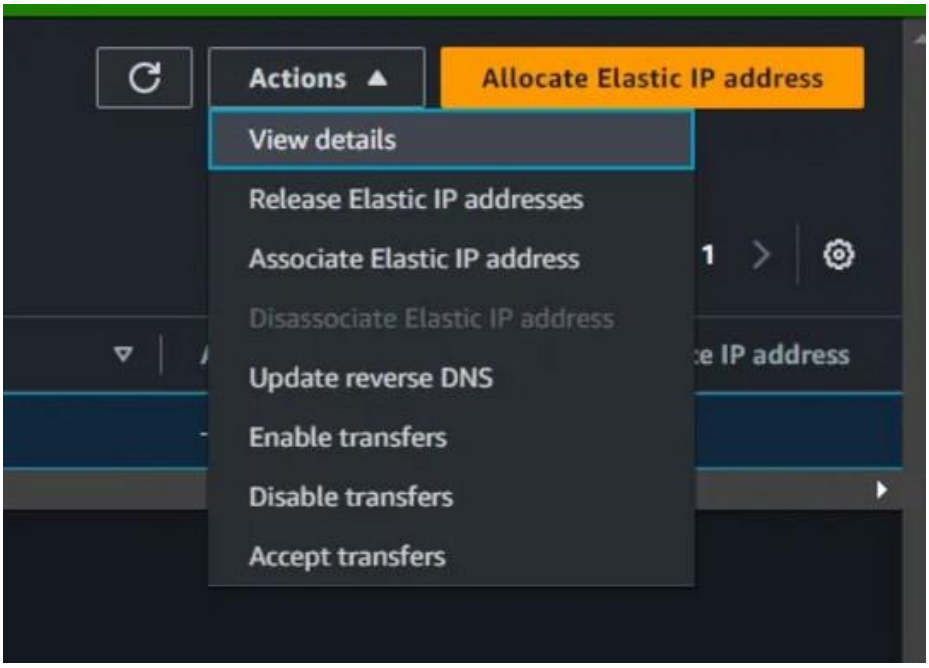
After stopping the instance, observe that the public IP address is released while the private IP address remains unchanged.

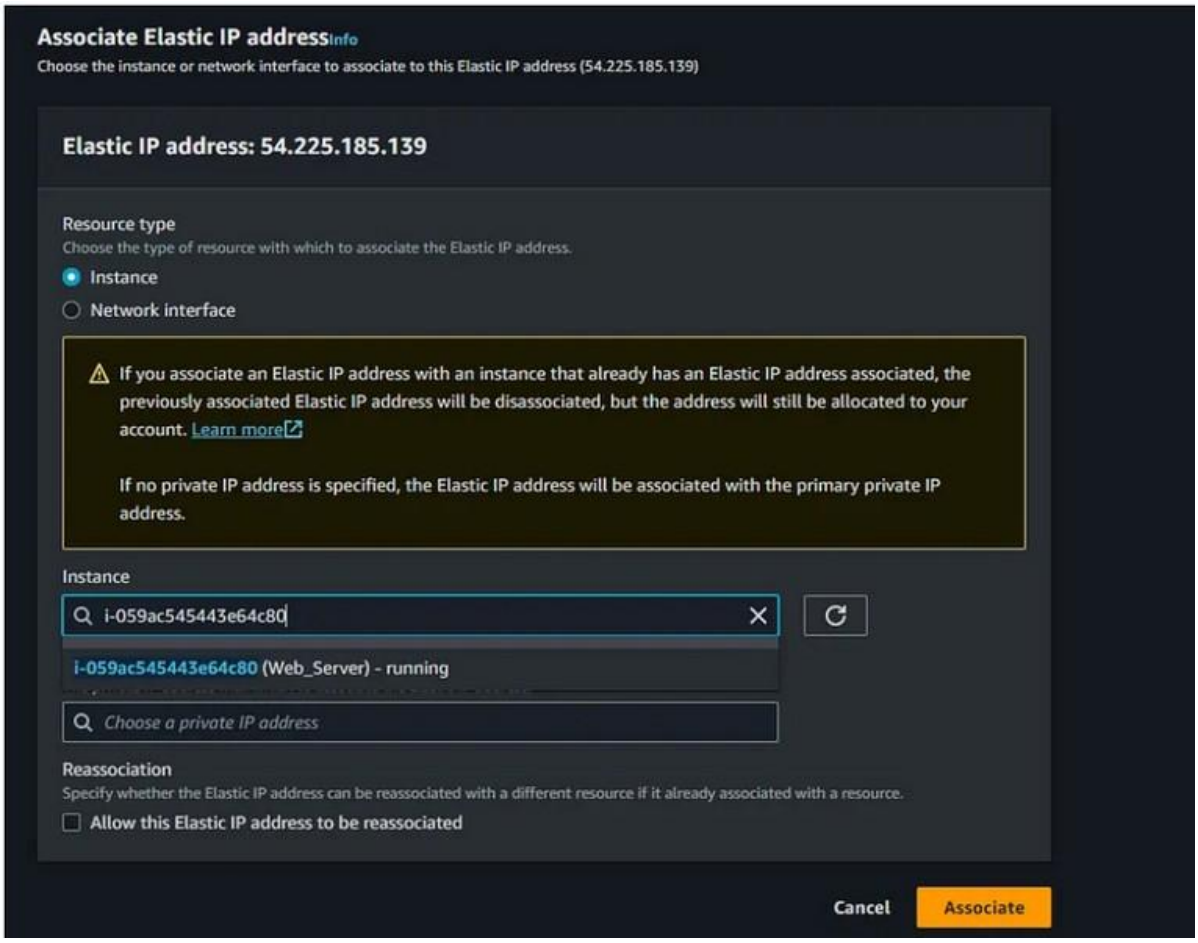
- Refresh the console to verify the changes.



Elastic IP Benefits:

- Elastic IP addresses offer the advantage of maintaining a static public IP address even in scenarios where EC2 instances are restarted or recreated.
- They provide a stable endpoint for applications or services that require a consistent public IP address.
- With Elastic IP, there's no need to update DNS records or reconfigure applications each time an instance is stopped or replaced.
- It simplifies infrastructure management and enhances reliability by ensuring uninterrupted connectivity to resources hosted on EC2 instances.
- Elastic IP addresses are particularly useful for scenarios requiring a fixed public IP address, such as hosting websites, APIs, or VPN gateways.
- Allocating an Elastic IP:
- Allocate an Elastic IP address from the AWS pool.
- Associate the Elastic IP with the desired EC2 instance.

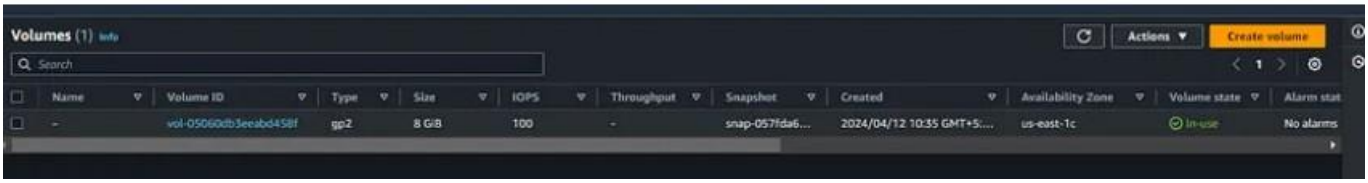




EC2 Instance Components:

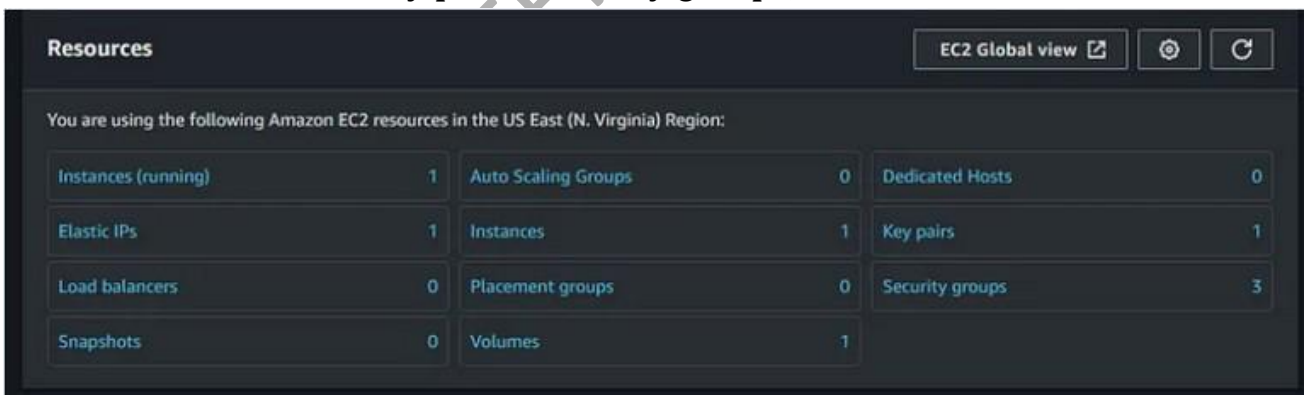
- In addition to the EC2 instance itself, other components include the network interface and volumes (virtual hard disks).

- The network interface manages connectivity and is associated with firewall rules, public IP addresses, and security groups.
- Volumes represent the virtual hard disks attached to the instance, storing data and operating system files.
- Understanding these components is essential for managing and configuring EC2 instances effectively.



Navigating the EC2 Dashboard:

- Encourage exploration of the EC2 dashboard to view information about resources like key pairs, security groups, and volumes.

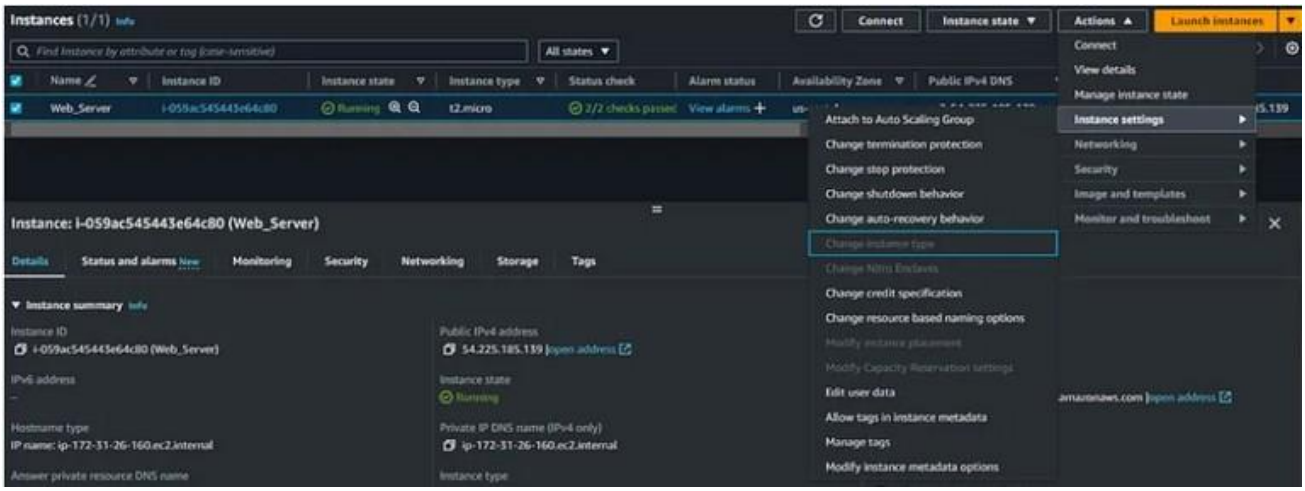


Adjusting Instance Settings:

- Demonstrate how to change instance type and other settings like Elastic IP disassociation, network interface attachment/detachment, and security group changes.

Reverting Instance Changes:

- Revert any instance changes made during the demonstration, such as changing the instance type, by powering on the instance and reverting the settings back to their original state.



Elastic Block Store (EBS)

EBS serves as the virtual hard disk for EC2 instances, offering both EBS volumes and snapshots for data storage and backup.

EBS Volumes:

- EBS volumes, akin to physical hard disks, house the operating system data of EC2 instances and can store various types of data such as databases, web servers, and files.
- During volume creation, users specify an availability zone, ensuring alignment with the instance's location.
- Data stored in EBS volumes is automatically replicated within the same availability zone, mitigating the risk of data loss due to local failures.

Snapshots:

- Snapshots, used for backing up entire volumes, provide an additional layer of data protection.
- These snapshots capture the state of the EBS volume at a specific point in time, facilitating data recovery and restoration.

Types of EBS Volumes:

- EBS volumes come in various types catering to different workload requirements and performance needs.
- General purpose volumes, SSD-based, are recommended for most workloads due to their balanced pricing and speed.

- Provisioned IOPS volumes offer enhanced performance and are ideal for large databases requiring high input-output operations per second (IOPS).
- Throughput optimized HDD volumes suit big data and data warehousing applications, while cold HDD volumes are cost-effective solutions for infrequently accessed data.
- Magnetic volumes, the cheapest option, are suitable for backups and archives but offer lower performance.

Choosing the Right Volume Type:

- The selection of an appropriate volume type depends on factors such as workload characteristics, performance requirements, and budget considerations.
- AWS documentation provides detailed information and use cases for each volume type, aiding users in making informed decisions.

After establishing a connection to the instance, execute the following commands. Please note that these commands can also be added to the instance creation process in the user data area:

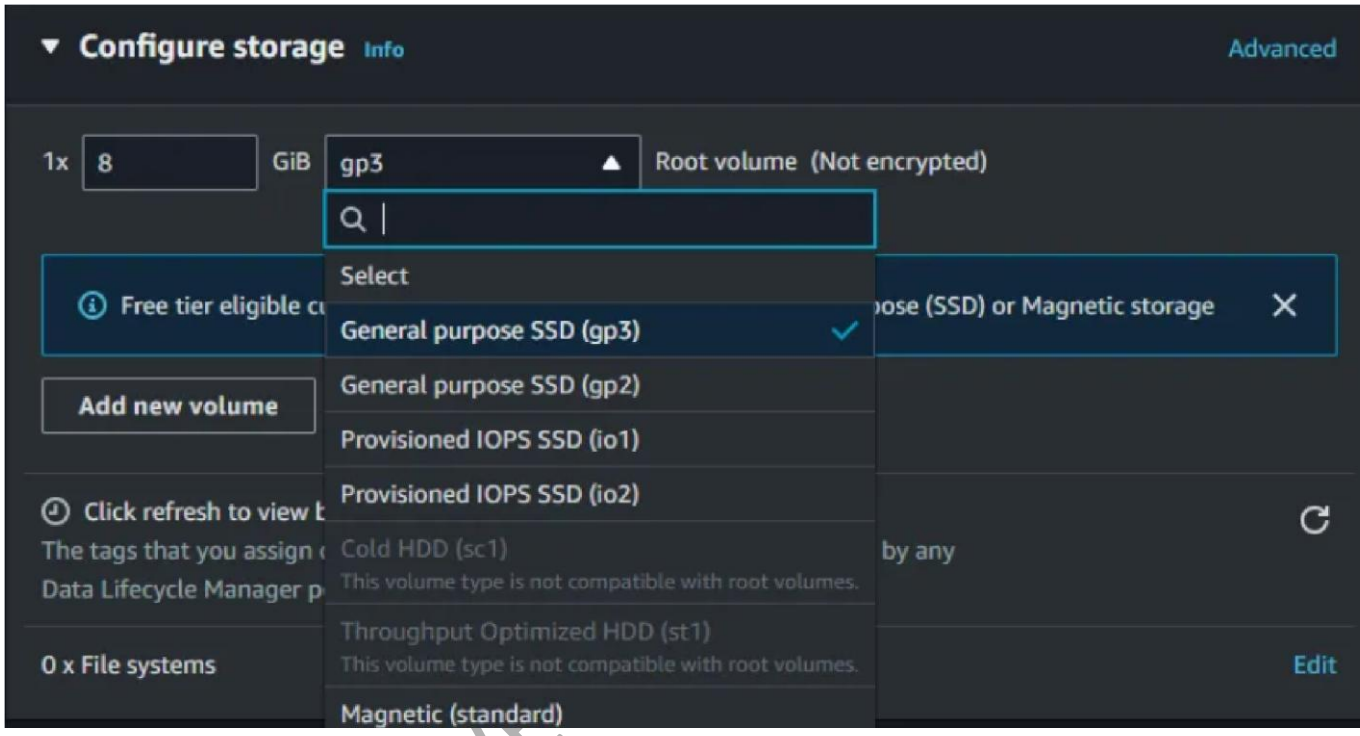
-
-
-
-
-
-

systemctl
status
apache2 cd
/tmp
wget https://www.tooplate.com/zip-templates/2119_gymso_fitness.zip unzip -o 2119_gymso_fitness.zip cp -r 2119_gymso_fitness/ /var/www/html/ systemctl restart apache2*

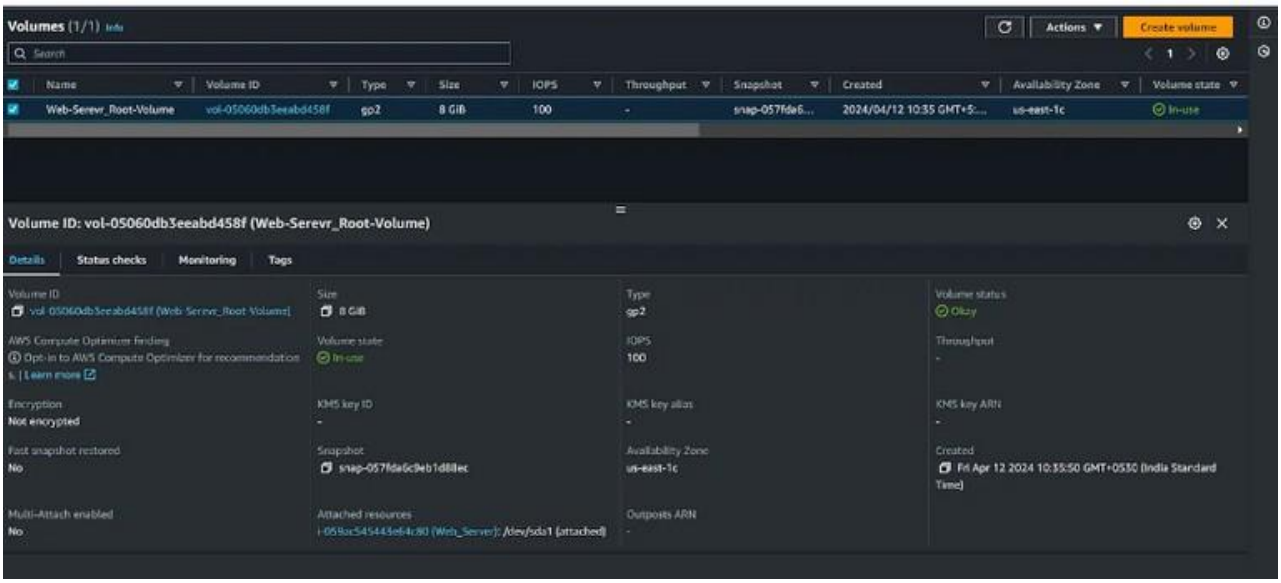
These commands are designed to check the status of the Apache web server, navigate to the temporary directory, download a ZIP file containing website templates from a specified URL, unzip the downloaded file, copy the contents to the Apache document root

directory, and finally restart the Apache service to apply the changes.

Here, we can observe the available volume types for creating an EC2 instance.



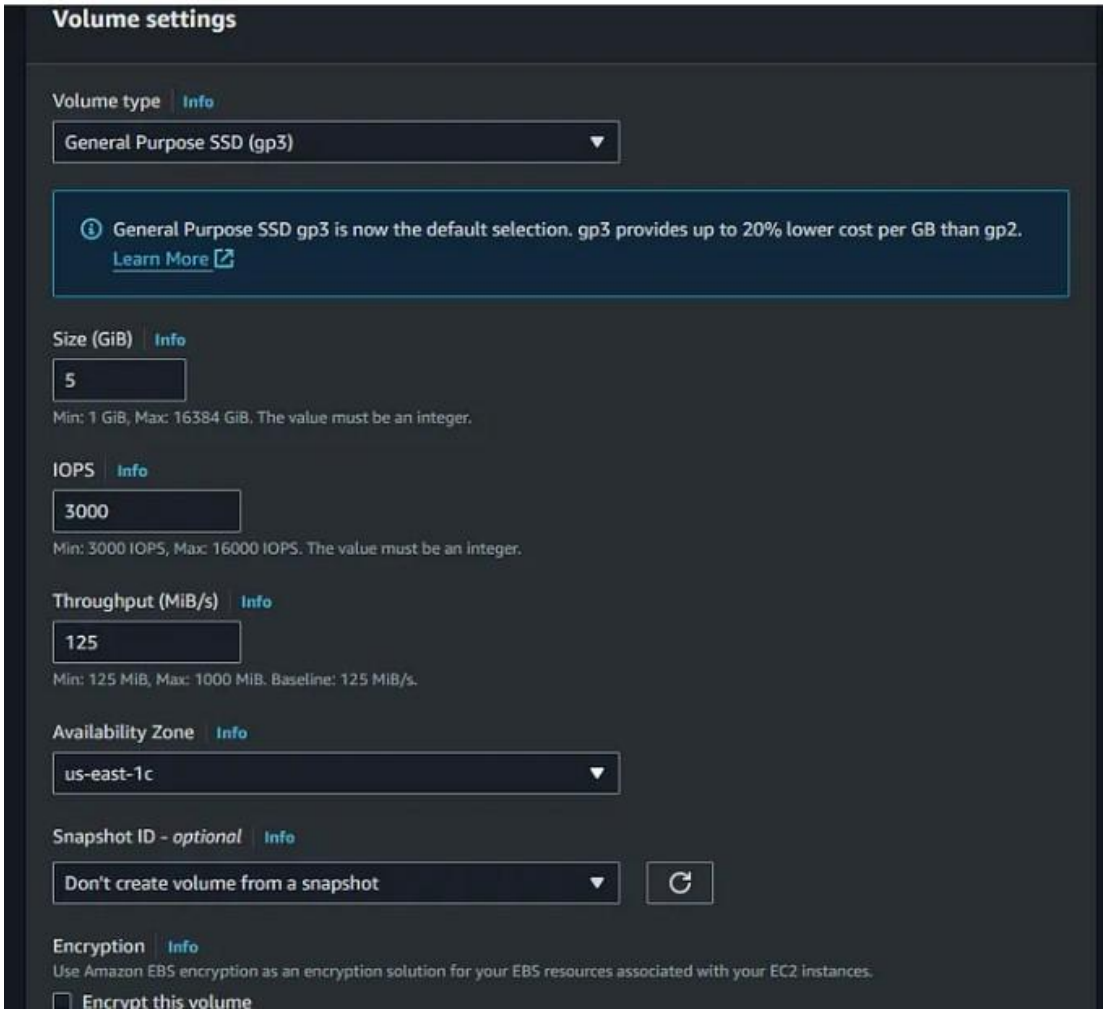
Navigate to the Volumes section located on the left-hand side of the EC2 instances page and update the name of the volume associated with the instance mentioned earlier.



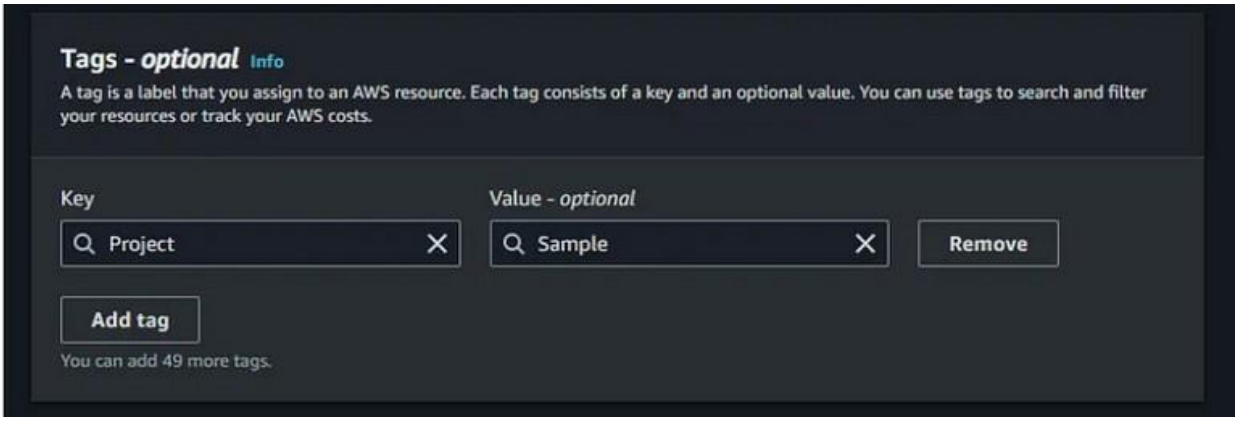
Adding extra storage to an EC2 instance, it's crucial to consider the availability zone. Both the instance and the volume need to be in the same zone to ensure connectivity.

When gathering requirements for additional storage, such as for web server images, it's important to adhere to the free tier limit of 30 GB for EBS volumes. For instance, if there's a need for five gigabytes of storage for web server images, it's imperative to ensure that the total storage remains within the free tier limit.

Click on create volume:

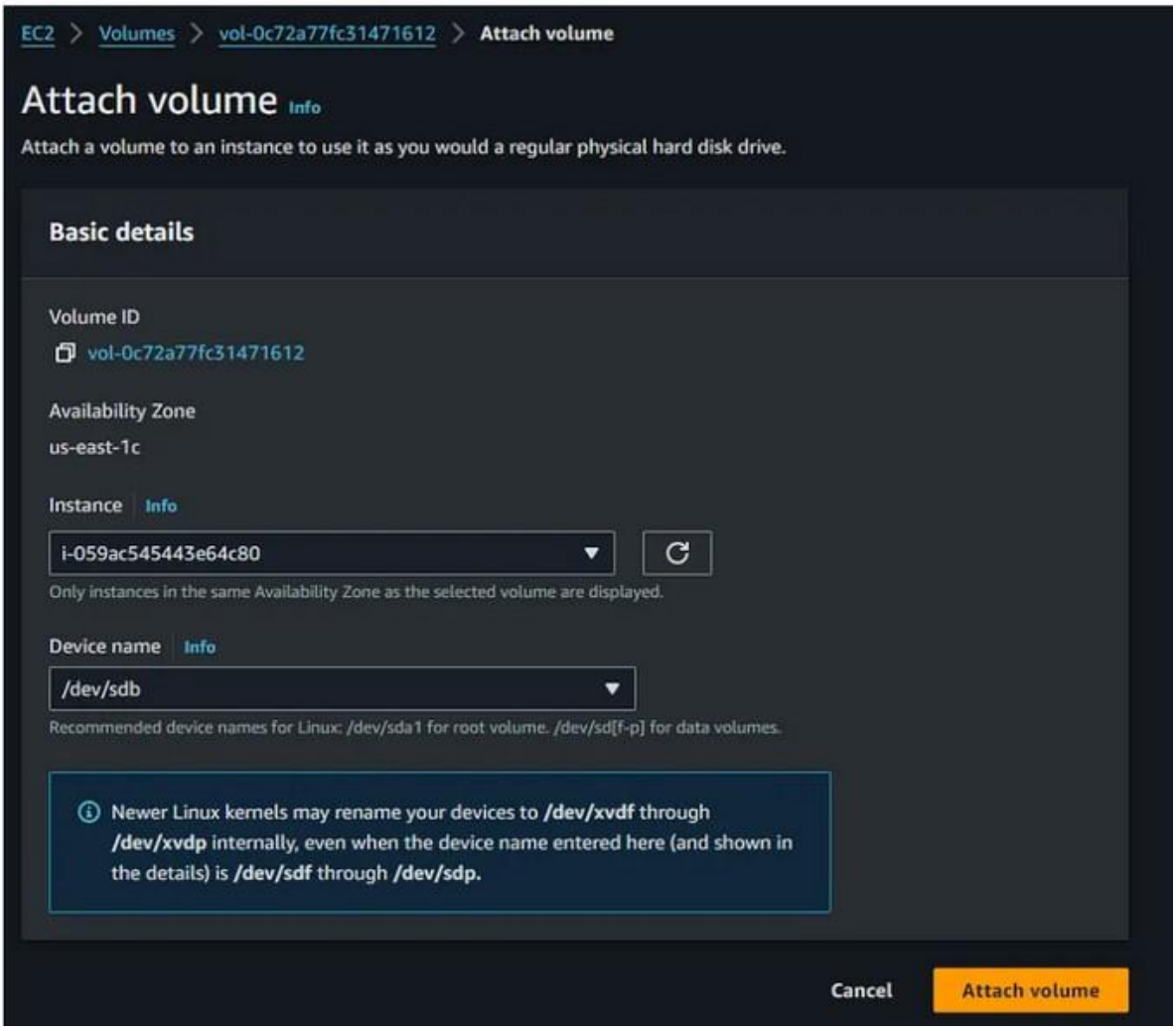


Tags, you can give upon your wish, then click on Create Volume



Choose the created volume, navigate to actions, and then opt for attaching the volume once it appears as available on the dashboard.

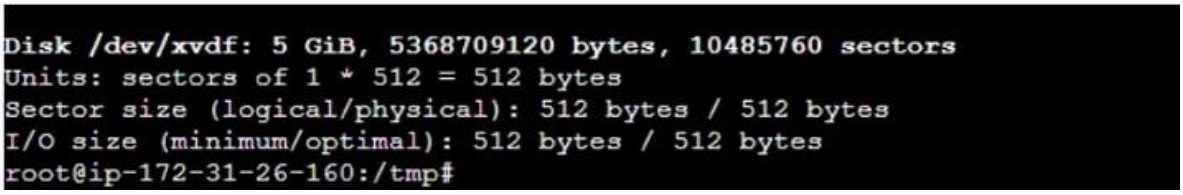
Select the instance , device name and click on attach volume.



(Please note I've selected /dev/xvdf in above screenshot as Device name). If we run the "fdisk -l" command, we could see the new volume listed:

- *fdisk -l*

List Disks: Start by running the command fdisk -l to list all disks. For instance, /dev/xvdf represents the root volume with a size of 8 GB, and it has a partition /dev/xvdaf1.



- Understand Partitions: Linux assigns partitioning numbers (e.g., xvda1) rather than names. Verify the root volume by checking its mount point using df -h. If /dev/xvda1 is mounted to the root directory, it contains the operating system data.
- Create Partition: Identify the additional disk, such as /dev/xvdf, and use fdisk to create a partition. Choose the disk path (e.g., /dev/xvdf), follow the prompts to create a new partition, and write the changes using w.

```
Command (m for help): m

Help:

  DOS (MBR)
   a   toggle a bootable flag
   b   edit nested BSD disklabel
   c   toggle the dos compatibility flag

  Generic
   d   delete a partition
   F   list free unpartitioned space
   l   list known partition types
   n   add a new partition
   p   print the partition table
   t   change a partition type
   v   verify the partition table
   i   print information about a partition

  Misc
   m   print this menu
   u   change display/entry units
   x   extra functionality (experts only)

  Script
   I   load disk layout from sfdisk script file
   O   dump disk layout to sfdisk script file

  Save & Exit
   w   write table to disk and exit
   q   quit without saving changes

  Create a new label
   g   create a new empty GPT partition table
   G   create a new empty SGI (IRIX) partition table
   o   create a new empty DOS partition table
   s   create a new empty Sun partition table


Command (m for help): n
```

```
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-10485759, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-10485759, default 10485759):

Created a new partition 1 of type 'Linux' and of size 5 GiB.

Command (m for help): p
Disk /dev/xvdf: 5 GiB, 5368709120 bytes, 10485760 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xe3548d72

Device     Boot Start      End  Sectors Size Id Type
/dev/xvdf1       2048 10485759 10483712   5G 83 Linux

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

root@ip-172-31-26-160:~#
```

- fdisk -l

```
Device     Boot Start      End  Sectors Size Id Type
/dev/xvdf1       2048 10485759 10483712   5G 83 Linux
root@ip-172-31-26-160:~#
```

Format Partition: Use the appropriate formatting utility, such as mkfs.ext4, to format the partition. For example, run mkfs.ext4 /dev/xvdf1 to format the partition with ext4 format.

```
mkfs.cramfs  mkfs.ext2    mkfs.ext3    mkfs.ext4    mkfs.fat     mkfs.minix   mkfs.msdos
```

- mkfs.ext4 /dev/xvdf1

```
root@ip-172-31-26-160:~# mkfs.ext4 /dev/xvdf1
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 1310464 4k blocks and 327680 inodes
Filesystem UUID: 18c5893b-0f5d-49f8-bb29-167b8dbff181
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

root@ip-172-31-26-160:~#
```

Temporary Mount: Mount the partition temporarily using the mount command. For instance, mount /dev/xvdf1 /var/www/html/images. Verify the mount using *df -h*.

Create a directory: " mkdir /tmp/image_backups", and move all data from Images to newly created one

```
root@ip-172-31-26-160:/var/www/html#
root@ip-172-31-26-160:/var/www/html# mv images/* /tmp/image_backups/
root@ip-172-31-26-160:/var/www/html#
```

Then mount the volume and verify as below:

```
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# mount /dev/xvdf1 /var/www/html/images/
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.6G  2.4G  5.3G  31% /
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  868K  190M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
/dev/xvda15     105M  6.1M   99M   6% /boot/efi
tmpfs            95M  4.0K   95M   1% /run/user/1000
/dev/xvdf1      4.9G   24K  4.6G   1% /var/www/html/images
root@ip-172-31-26-160:~#
```

Permanent Mount: Edit the /etc/fstab file to configure permanent mounting. Add an entry with the partition path, mount point, format type, and other options. Save the file and test the configuration using mount -a.

```
LABEL=cloudimg-rootfs   /        ext4   discard,errors=remount-ro       0 1
LABEL=UEFI      /boot/efi       vfat    umask=0077      0 1
/dev/xvdf1      /var/www/html/images    ext4    defaults        0 0
```

Save the file and exit.

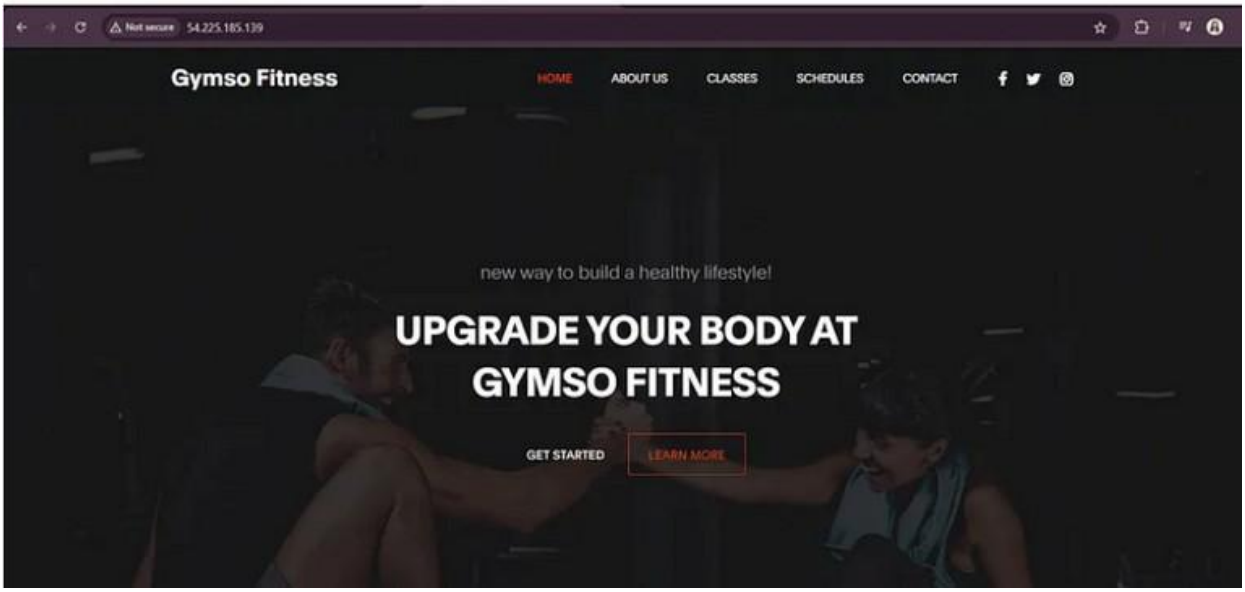
```
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# mount -a
root@ip-172-31-26-160:~#
```

```
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.6G  2.4G  5.3G  31% /
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  868K  190M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
/dev/xvda15     105M  6.1M   99M   6% /boot/efi
tmpfs            95M  4.0K   95M   1% /run/user/1000
/dev/xvdf1      4.9G   24K  4.6G   1% /var/www/html/images
root@ip-172-31-26-160:~#
```

Move Data: Move data from the backup directory to the mounted partition. For example, use mv command to move data from /tmp/img-backups to /var/www/html/images., and restart the apache2 service

```
root@ip-172-31-26-160:~# mv /tmp/image_backups/* /var/www/html/images/
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# systemctl restart apache2
root@ip-172-31-26-160:~# systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2024-04-15 00:48:20 UTC; 13s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 3770 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 3776 (apache2)
      Tasks: 55 (limit: 1121)
     Memory: 4.8M
        CPU: 28ms
     CGroup: /system.slice/apache2.service
             ├─3776 /usr/sbin/apache2 -k start
             ├─3777 /usr/sbin/apache2 -k start
             └─3778 /usr/sbin/apache2 -k start

Apr 15 00:48:20 ip-172-31-26-160 systemd[1]: Starting The Apache HTTP Server...
Apr 15 00:48:20 ip-172-31-26-160 systemd[1]: Started The Apache HTTP Server.
root@ip-172-31-26-160:~#
```

Verify: Check if the data is accessible through the browser. If not, consider disabling SELinux temporarily by modifying the /etc/selinux/config file and rebooting the system. Take the public IP and paste in the browser: you could see the page.



EBS Snapshots

Unmount the volume we created above as below:

```
root@ip-172-31-26-160:~# umount /var/www/html/images
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.6G  2.4G  5.3G  31% /
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  868K  190M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
/dev/xvda15     105M  6.1M   99M   6% /boot/efi
tmpfs            95M  4.0K   95M   1% /run/user/1000
root@ip-172-31-26-160:~#
```

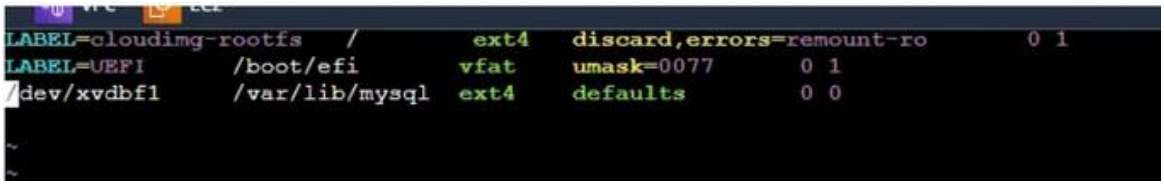
Next, navigate to the AWS console, choose the volume that was created, and proceed to detach it by selecting the respective option.

Create a new volume same as shown above and attach it to the same ec2 instance.

Follow the same steps as mentioned above to create a new partition, format , and then create a directory

```
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# mkdir -p /var/lib/mysql
root@ip-172-31-26-160:~#
```

Add the new volume to the /etc/fstab.



Then mount it.

```
root@ip-172-31-26-160:~# mount -a
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.6G  2.4G  5.3G  31% /
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  868K  190M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
/dev/xvda15     105M  6.1M   99M   6% /boot/efi
tmpfs            95M  4.0K   95M   1% /run/user/1000
/dev/xvdbf1     2.9G   24K  2.8G   1% /var/lib/mysql
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~#
```

Install mariadb-server package with the below steps:

- *apt*
- *install*
- *mariadb-*
- *server*

systemctl
start
mariadb
systemctl
enable
mariadb
systemctl
status
mariadb

List the /var/lib/mysql contents:

- *ls /var/lib/mysql/*

Snapshot functionality primarily serves as a backup and restoration tool, crucial for mitigating data loss in the event of system failures. However, snapshots offer more than just backup capabilities.

In the event of data loss, the initial step involves unmounting the partition to prevent data overwriting. If the goal is to restore the existing partition, snapshots may not be applicable, as they provide a new volume containing the data. In such cases, unmounting the partition and employing recovery tools become essential.

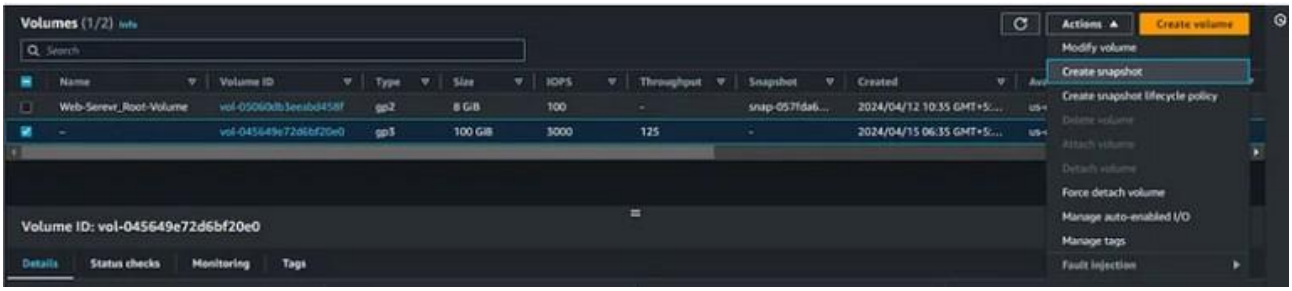
When utilizing snapshots, unmounting the partition is the preliminary action, followed by detaching the corrupted volume. Subsequently, a new volume is created from the snapshot, retaining all data and partition configurations. Upon creation, the new volume is ready for attachment and mounting, effectively replacing the corrupted volume.

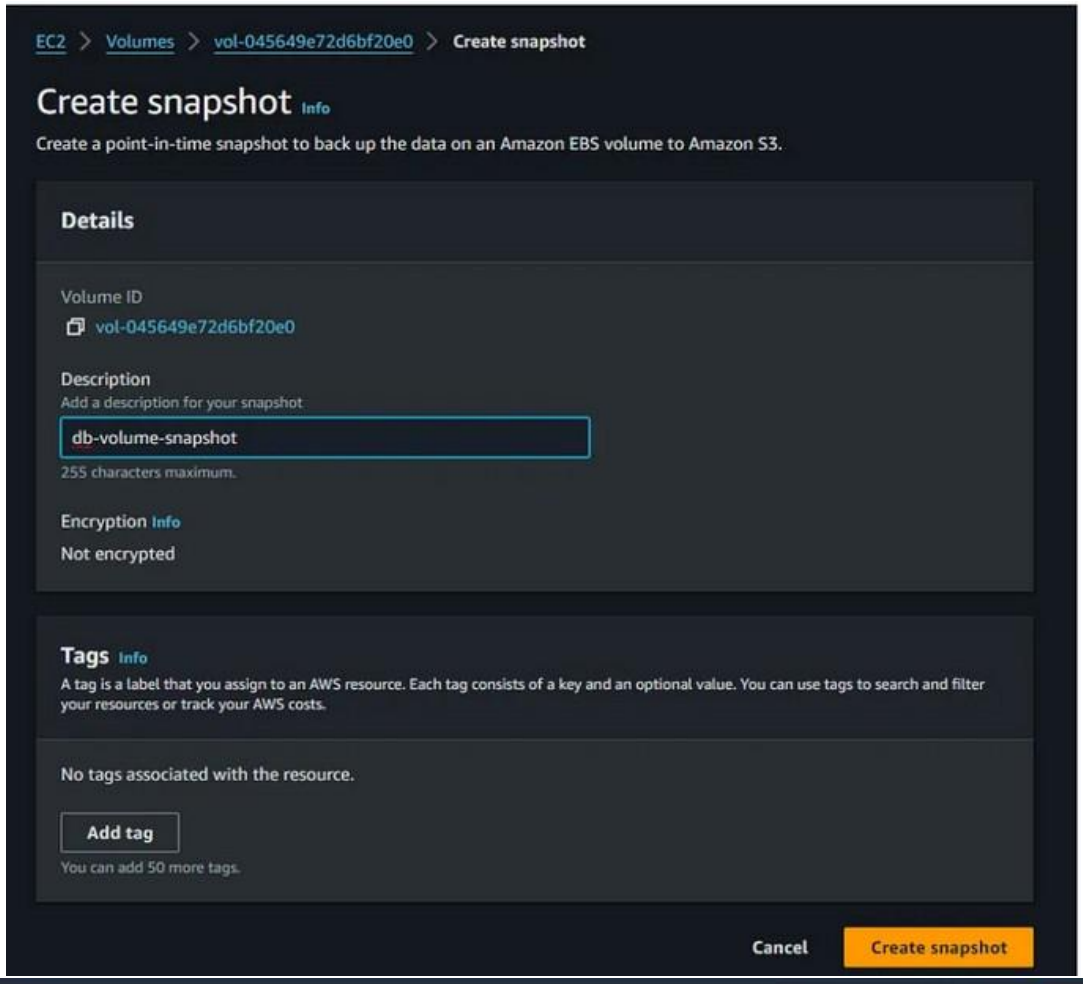
In essence, snapshots facilitate data restoration through volume replacement rather than repair, streamlining the recovery process and ensuring data integrity.

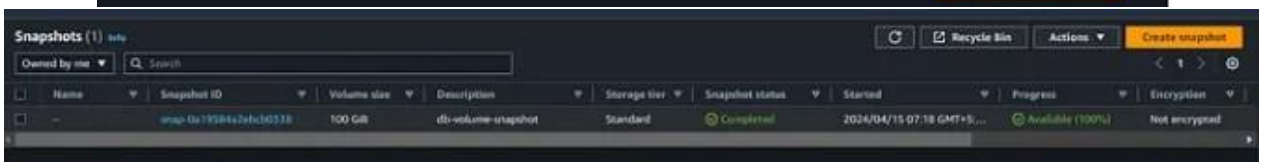
Take a Snapshot:

- Navigate to the Volume section in the AWS console.
- Select the relevant volume, then click on “Actions” and choose
- “Create Snapshot.” Provide a meaningful description and add
- tags for easy identification.

Click on “Create Snapshot” and wait for it to enter the completed state.





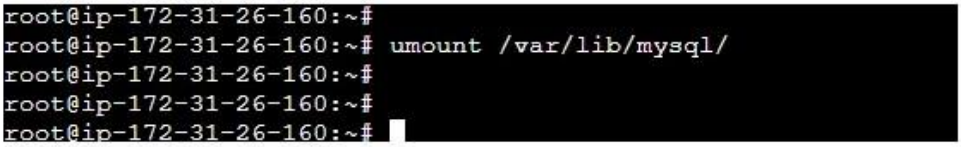


Delete Data:

- If you accidentally delete data or encounter corruption, proceed
- with data deletion.
- Access the directory containing the data you wish to remove, e.g., /var/lib/mysql. Use the command rm -rf * to remove all files and directories within the specified location.

Unmount:

- Stop any relevant services to prevent further data overwrites.
- Unmount the affected directory, e.g., /var/lib/mysql.



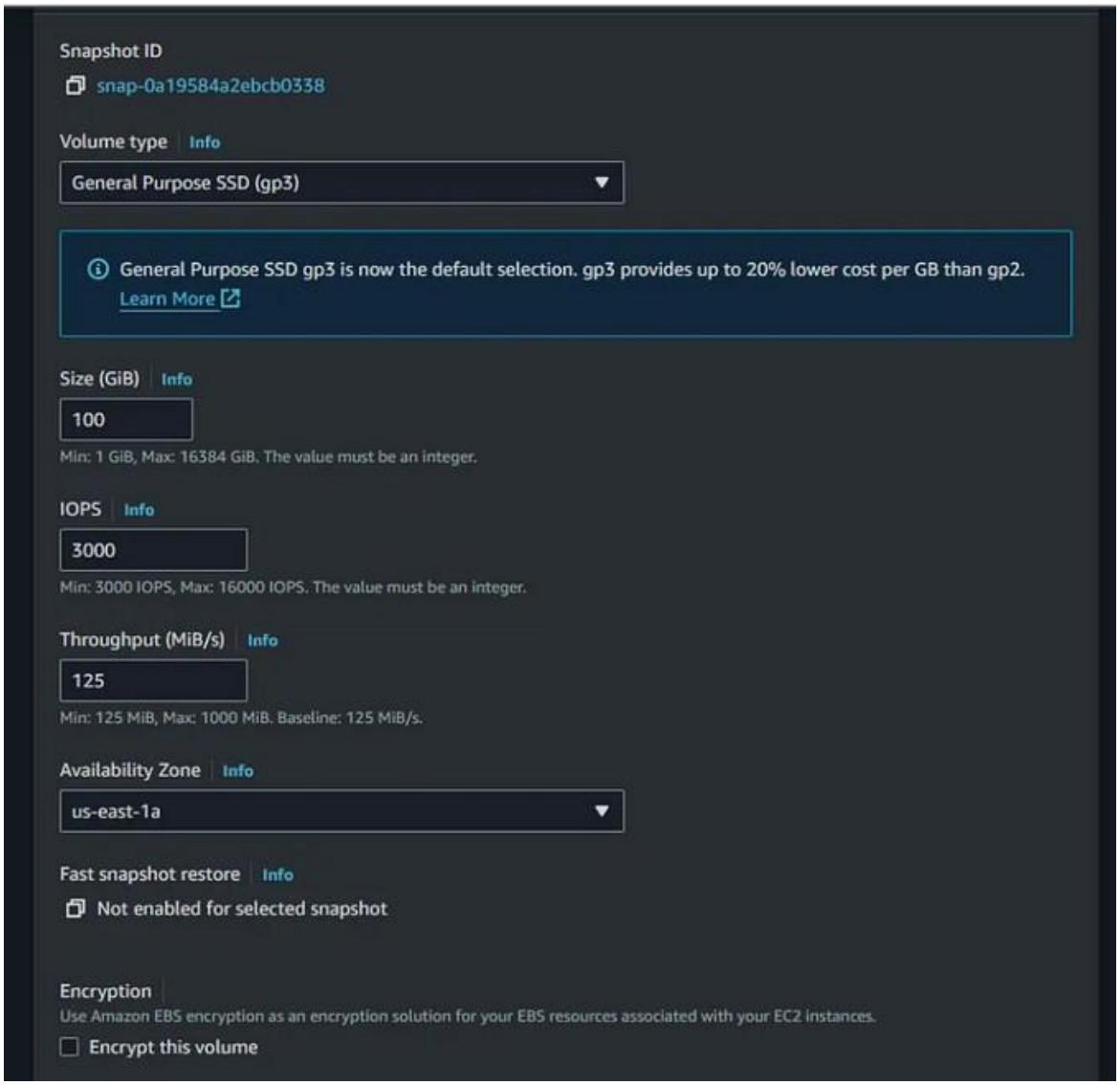
Detach Corrupted Volume:

- Navigate to the Volumes section and locate the
- corrupted volume. Rename the volume for easy
- identification, e.g., “corrupted.”

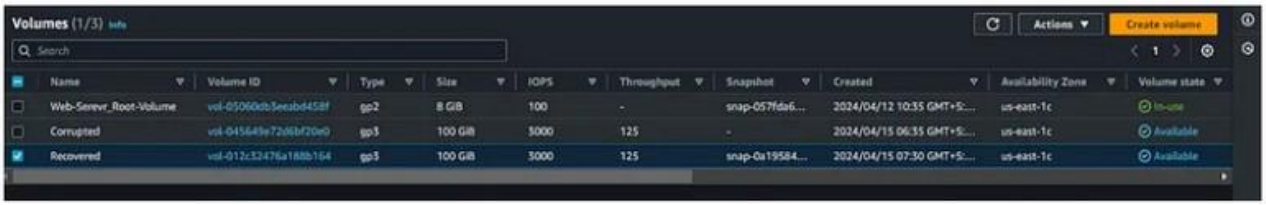
Select “Actions” and choose “Detach Volume” to disconnect it.

Recover from Snapshot:

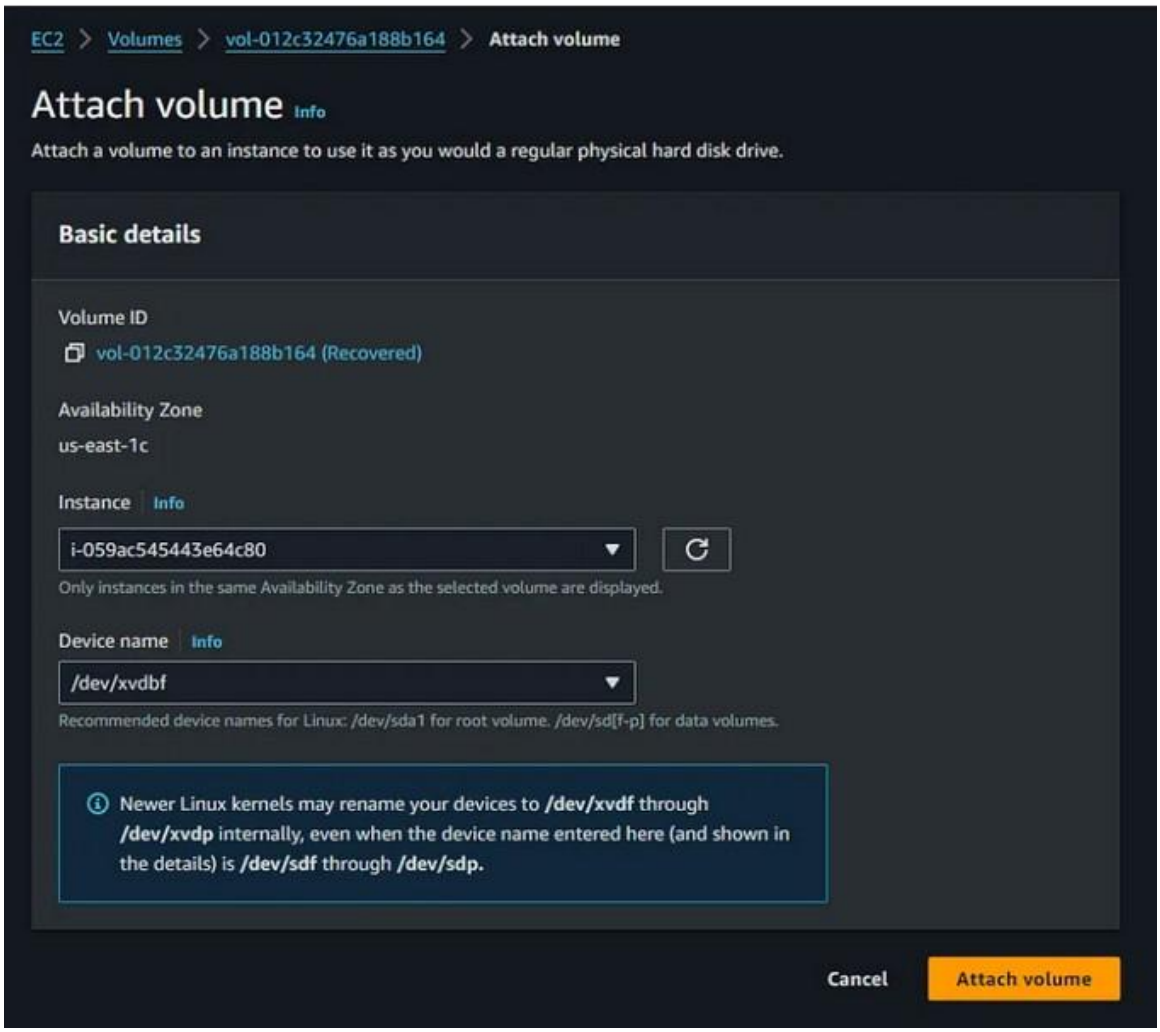
- Head to the Snapshots section and select the desired snapshot.
- Click on “Actions” and choose “Create snapshots from volume”.
- Customize volume settings such as type, size, and zone if needed.
- Add appropriate tags for clarity
- Click on “Create Volume” to finalize the recovery process.



In the Volumes console. goto Volumes and rename the new volume as “recovered”.



Goto actions and attach the volume



Below, we examined the disks and found that they were not mounted. Consequently, we executed the mount -a command to rectify the issue. Upon verification, the disks were automatically mounted, allowing us to recover the deleted data by executing ls /var/lib/mysql/.

In the snapshot console, you'll find various options beyond basic backup functionality. These include deleting snapshots, creating volumes from existing snapshots (which we've already covered), and managing fast snapshot restores for large volumes, albeit with additional charges.

Additionally, you can leverage snapshots to create images, known as Amazon Machine Images (AMIs). If the snapshot originates from a root volume, you can directly create an image from it.

Another useful feature is the ability to copy snapshots. By clicking the "copy" button, you can duplicate snapshots to different regions, which is particularly handy for multisite setups. Moreover, during the copying process, you have the option to enable encryption for added security.

Furthermore, the snapshot interface offers a "modify permissions" button, allowing you to adjust access settings. This enables you to make snapshots public or share them with specific AWS accounts by providing their account IDs.

In summary, snapshots offer a simple yet powerful toolset with various benefits, making them an indispensable asset for managing AWS resources effectively.

Elastic

-
- Load Balancer Load balancing

is a
c
rucial aspect of managing clusters of servers efficiently. It involves distributing incoming network traffic across multiple servers to ensure optimal resource utilization, reliability, and scalability.

- In AWS, Elastic Load Balancer (ELB) provides a scalable and reliable solution for load balancing, allowing users to distribute traffic across multiple targets seamlessly.

- Key Components of ELB:
  - Frontend Port and Backend Port:
    - The frontend port is the port on which the load balancer listens for incoming traffic. For example, for HTTPS connections, the frontend port would typically be 443.
    - The backend port is the port on which the load balancer forwards traffic to the backend servers. It corresponds to the port where the application/service is running on the backend servers. For instance, if a Tomcat server is running on port 8080 on backend instances, the backend port would be set to 8080.

  - Types of ELB:
    - Classic Load Balancer: Operates at the network layer (Layer 4) of the OSI model. It provides basic load balancing capabilities and distributes incoming traffic to backend instances based on configured rules.
    - Application Load Balancer (ALB): Operates at the application layer (Layer 7) of the OSI model. ALB supports advanced routing based on URL paths, hostnames, and other application-level attributes, making it ideal for HTTP and HTTPS traffic.
  - Network Load Balancer (NLB): Operates at the transport layer (Layer 4) of the OSI model. NLB is designed to handle extremely high volumes of traffic and offers ultralow latency, making it suitable for TCP, UDP, and TLS traffic.

  - Load Balancer Targets:
    - ELB distributes traffic to backend targets, which can include EC2 instances, containers, IP addresses, or Lambda functions.

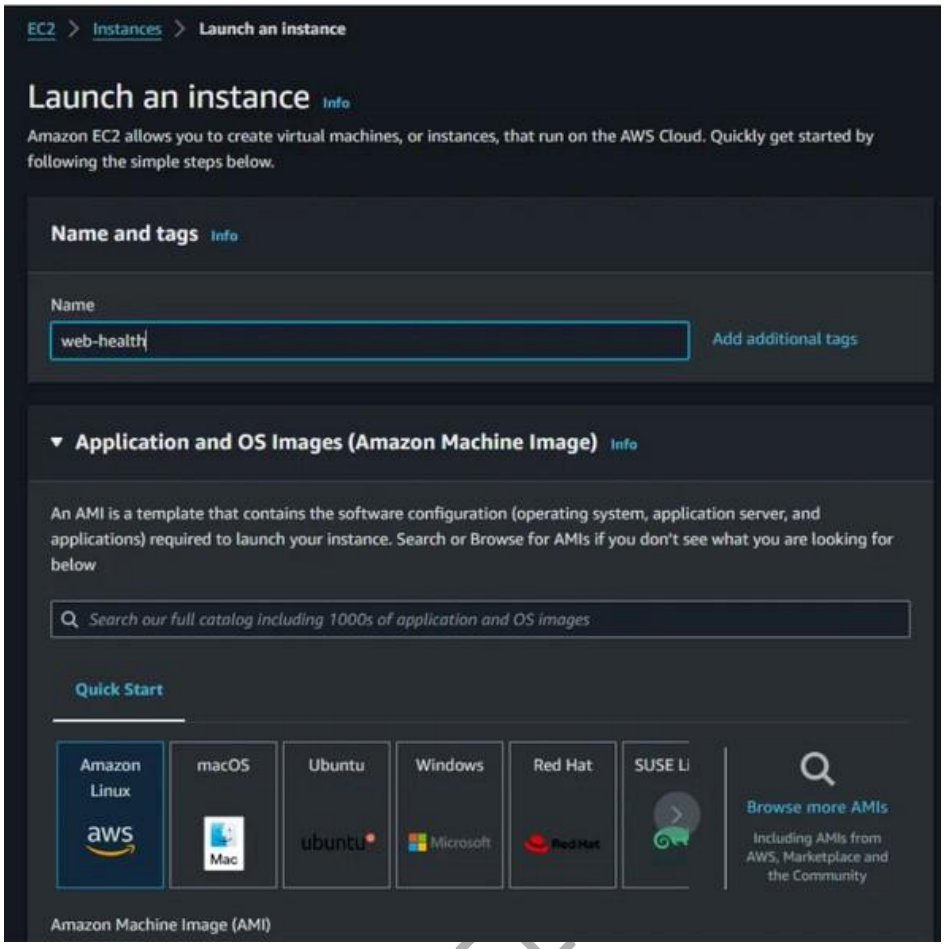
- Targets are distributed across multiple availability zones to improve fault tolerance and high availability.

Benefits of ELB:

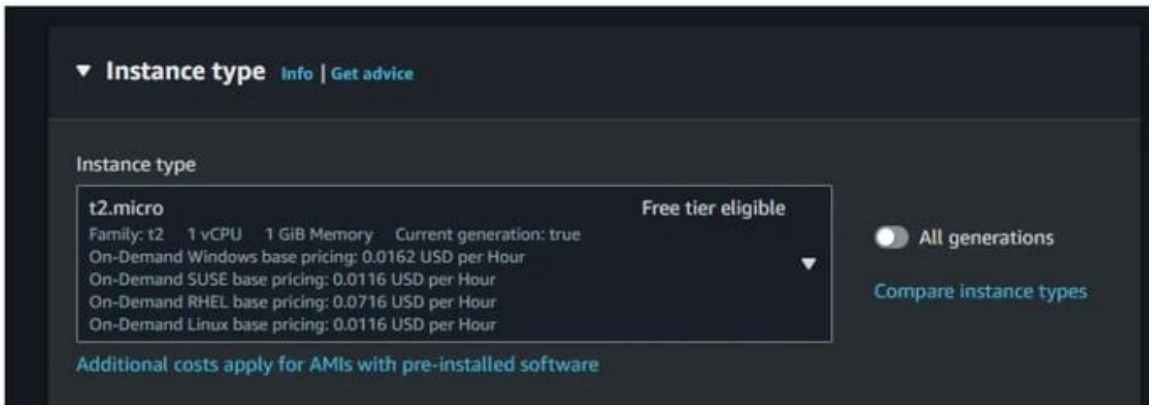
- Scalability: ELB automatically scales in response to incoming traffic, ensuring that applications remain responsive even during traffic spikes.
- Fault Tolerance: By distributing traffic across multiple targets and availability zones, ELB enhances fault tolerance and resiliency.
- Simplified Management: ELB abstracts away the complexities of managing load balancers, allowing users to focus on building and scaling their applications.

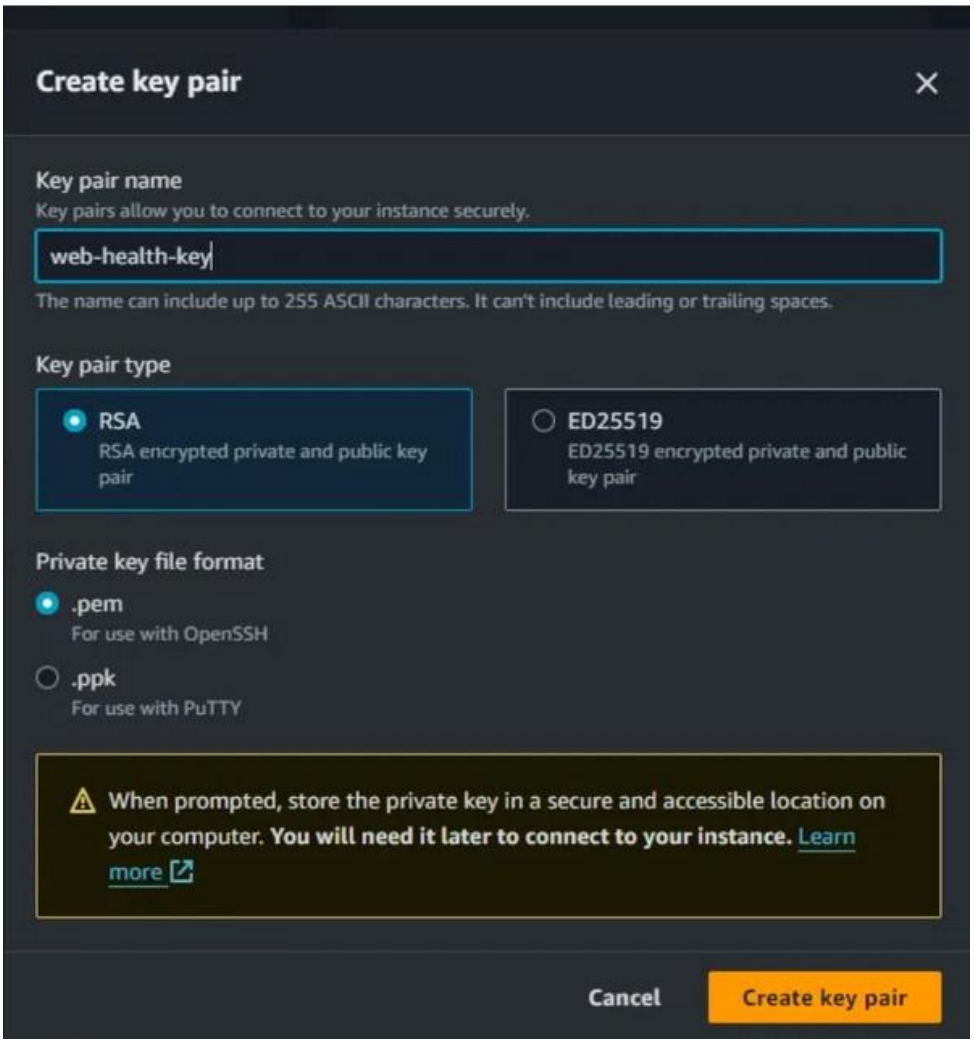
Let's proceed with launching an instance and naming it as "web-health".

You have the flexibility to choose between Amazon Linux, Ubuntu, or CentOS. The script provided in the resource section is compatible with both RPM-based and Debian-based operating systems. Since Amazon Linux is RPM-based, we'll stick with it for this instance.

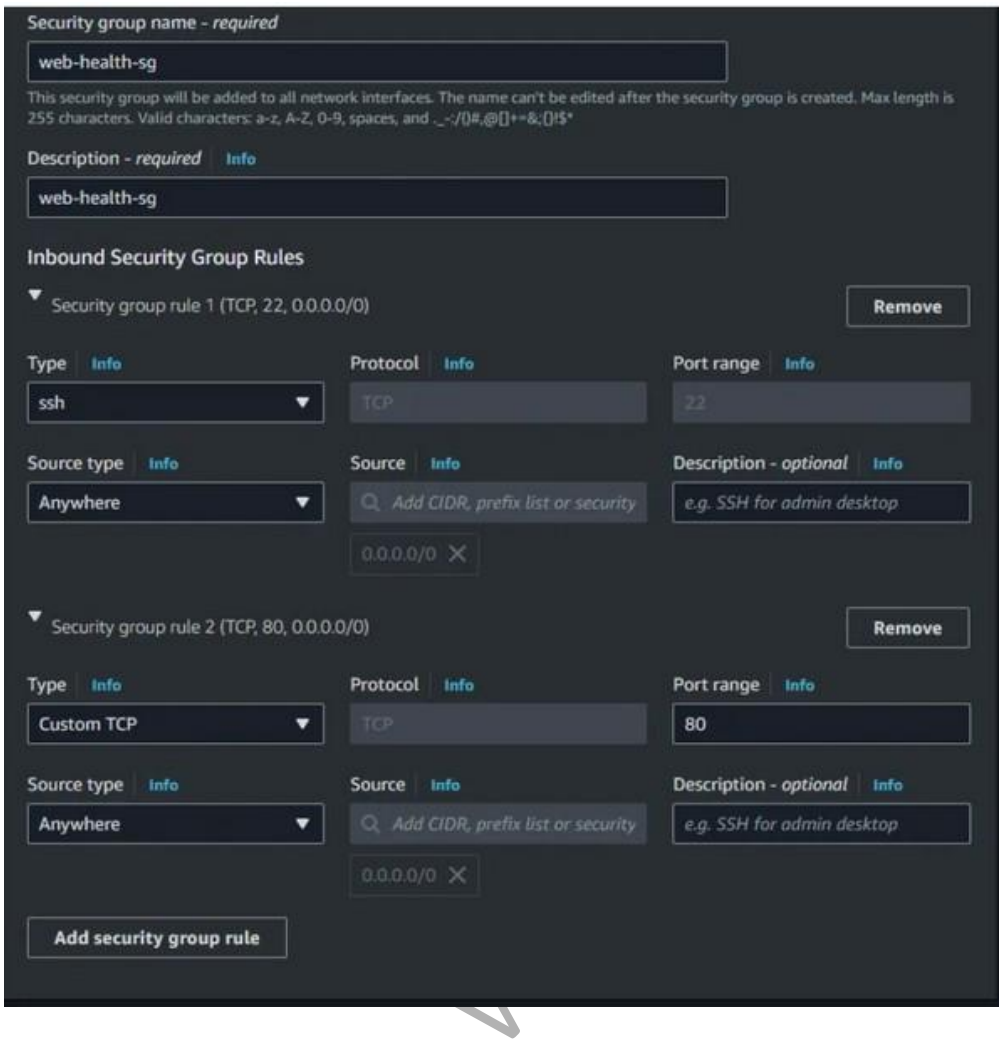


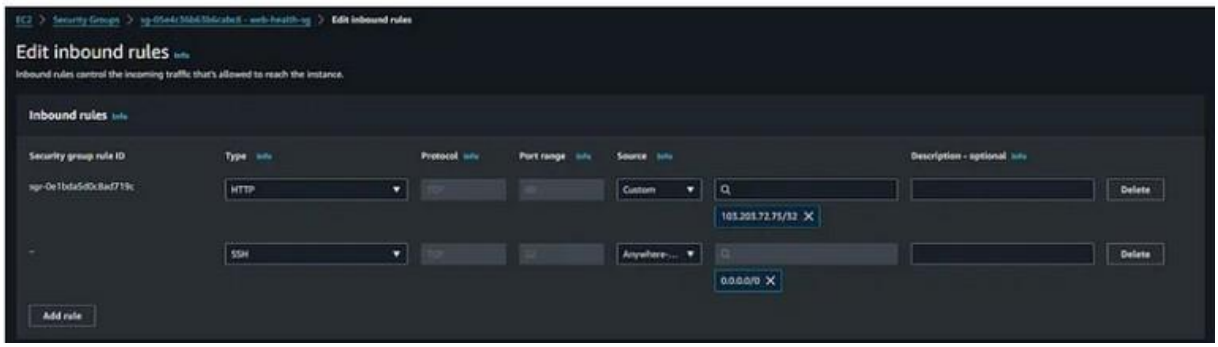
Let's opt for a t2.micro instance type and create a new key pair named "web-healthkey" for secure access.





Now, for the security group, let's create a new one named "web-health-sg", allowing inbound traffic on port 22 for SSH from your IP, and an additional rule for port 80, allowing traffic only from your IP.





Moving forward, let's proceed with the default storage settings and navigate to advanced details. Here, we'll utilize the script in the "User data" section under "Advanced details". Simply copy and paste it as instructed.

Script:

```
#!/bin/bash

# Variable Declaration
#PACKAGE="httpd wget unzip"
#SVC="httpd"
URL='https://www.tooplate.com/zip-templates/2098_health.zip'
ART_NAM
E='2098_
health'
TEMPDIR=
"/tmp/we
bfiles"
yum --help
&>
/dev/null

if [ $? -eq 0 ]
then
# Set Variables for CentOS
PACKAGE="httpd wget
unzip"  SVC="httpd"

echo "Running Setup on CentOS"
# Installing Dependencies
echo "########################################"
echo "Installing packages."
echo "########################################"
sudo yum install
$PACKAGE -y > /dev/null
echo

# Start & Enable Service
```

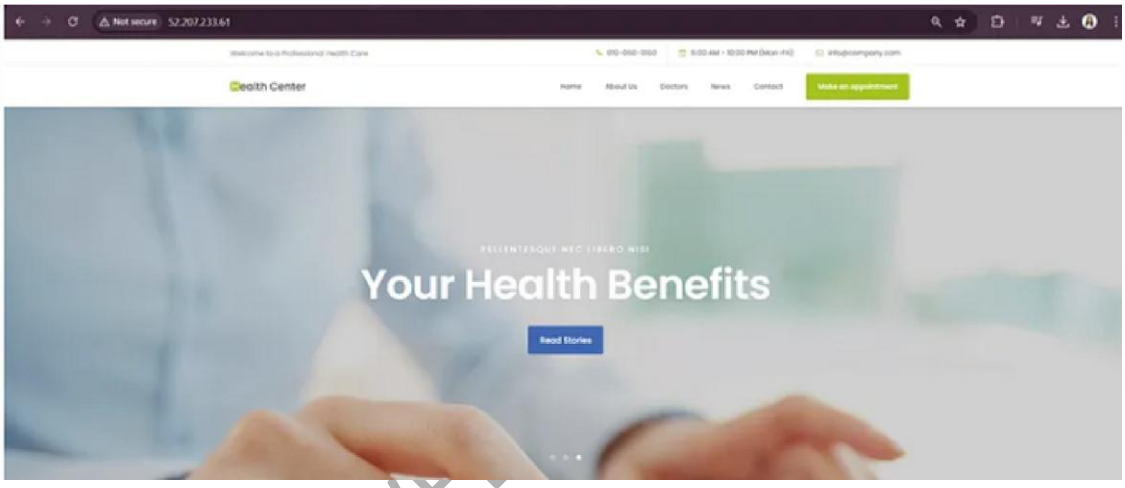
```
echo
"###################################
#######"  echo "Start & Enable HTTPD
Service"
echo
"###################################
##########"  sudo systemctl start $SVC
sudo systemctl enable $SVC  echo

# Creating Temp Directory
echo "#########################################"
echo "Starting Artifact Deployment"
echo
"###################################
##########"  mkdir -p $TEMPDIR  cd
$TEMPDIR  echo

fi
  wget $URL >
  /dev/null  unzip
  $ART_NAME.zip >
  /dev/null  sudo cp -r
  $ART_NAME/*
  /var/www/html/
  echo

  # Bounce Service
  echo
  "###################################
  #######"  echo "Restarting HTTPD service"
  echo "#########################################"
  systemc
  tl
  restart
  $SVC
  echo
```

```
# Clean Up
echo
"###################################
#######"  echo "Removing Temporary Files"
echo
"###################################
##########"  rm -rf $TEMPDIR  echo

sudo
systemctl
status
$SVC  ls
/var/ww
w/html/

else
# Set Variables for Ubuntu
PACKAGE="apache2 wget
unzip"  SVC="apache2"

echo "Running Setup on CentOS"
# Installing Dependencies
echo "#########################################"
echo "Installing packages."
echo
"###################################
#######"  sudo apt update
sudo apt install $PACKAGE
-y > /dev/null  echo

# Start & Enable Service
echo
"###################################
#######"  echo "Start & Enable HTTPD
Service"
echo
"###################################
```

```
  ##########"  sudo systemctl start $SVC
  sudo systemctl enable $SVC  echo

 # Creating Temp Directory
 echo "########################################"
 echo "Starting Artifact Deployment"
 echo
 "##############################
 ##########"  mkdir -p $TEMPDIR  cd
 $TEMPDIR  echo

 wget $URL >
 /dev/null  unzip
 $ART_NAME.zip >
 /dev/null  sudo cp -r
 $ART_NAME/*
 /var/www/html/
 echo

 # Bounce Service
 echo
 "##############################
 #######"  echo "Restarting HTTPD service"
 echo
 "##############################
 ##########"  systemctl restart $SVC
 echo

 # Clean Up
 echo
 "##############################
 #######"  echo "Removing Temporary Files"
 echo
 "##############################
 ##########"  rm -rf $TEMPDIR  echo
```

sudo
systemctl
status
$SVC ls
/var/ww
w/html/

After launching the instance, it typically takes about 5 minutes to initialize. Once it's up, we can check the public IP address and access the website via a browser.

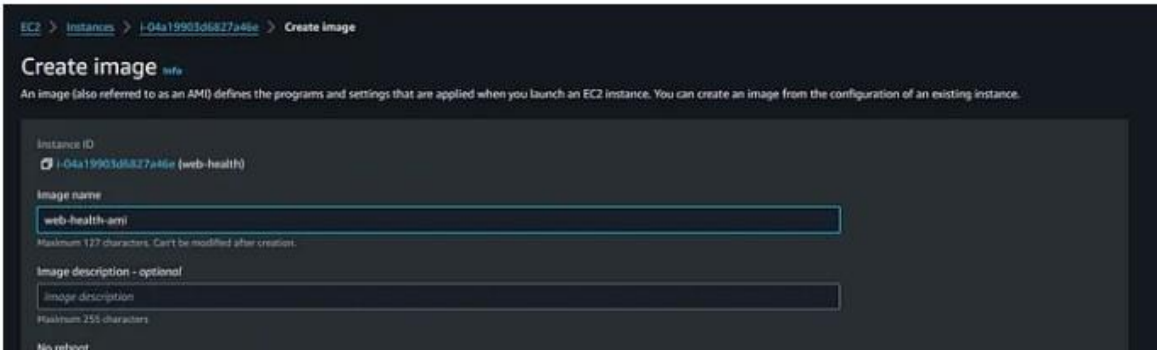


In case you encounter any issues accessing the website, ensure that the security group allows traffic from your IP and that the HTTP service is running on the instance.

Following the successful setup of our website on the EC2 instance, we'll proceed with creating an AMI and a launch template for future deployments.

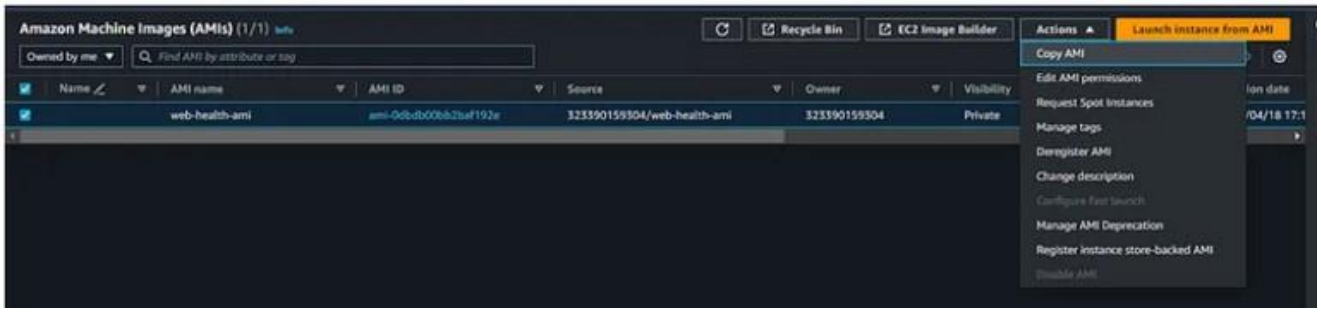
Creating an AMI involves selecting the instance, navigating to Actions > Image and templates > Create image, and specifying a name for the AMI.

Provide a name for the AMI — Here I gave — “web-health-ami”.



Once the AMI is created, we can leverage it to launch instances with identical configurations, ensuring consistency across our cluster.

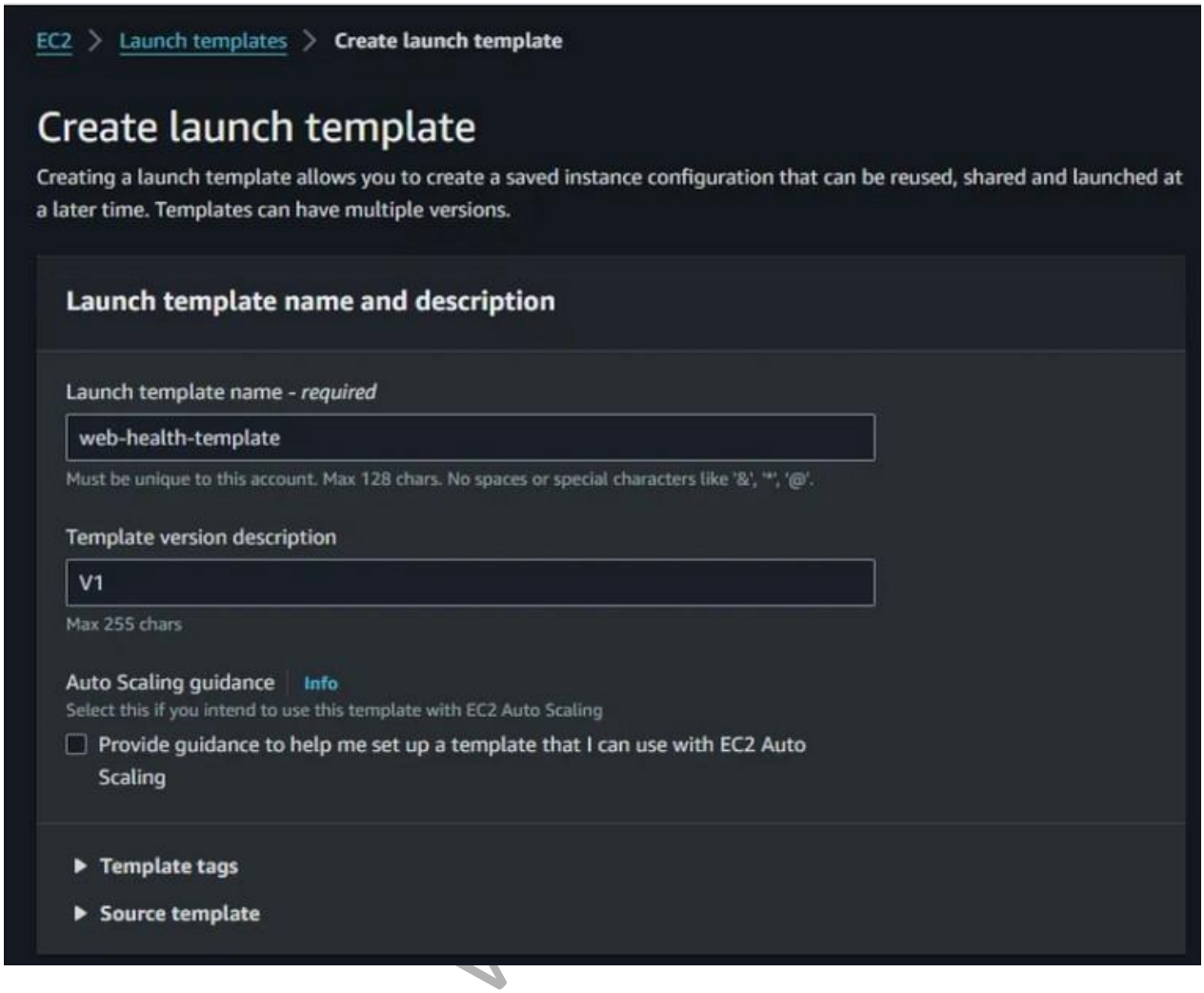
Additionally, we can explore options like copying the AMI to other regions or sharing it with other AWS accounts for collaboration.



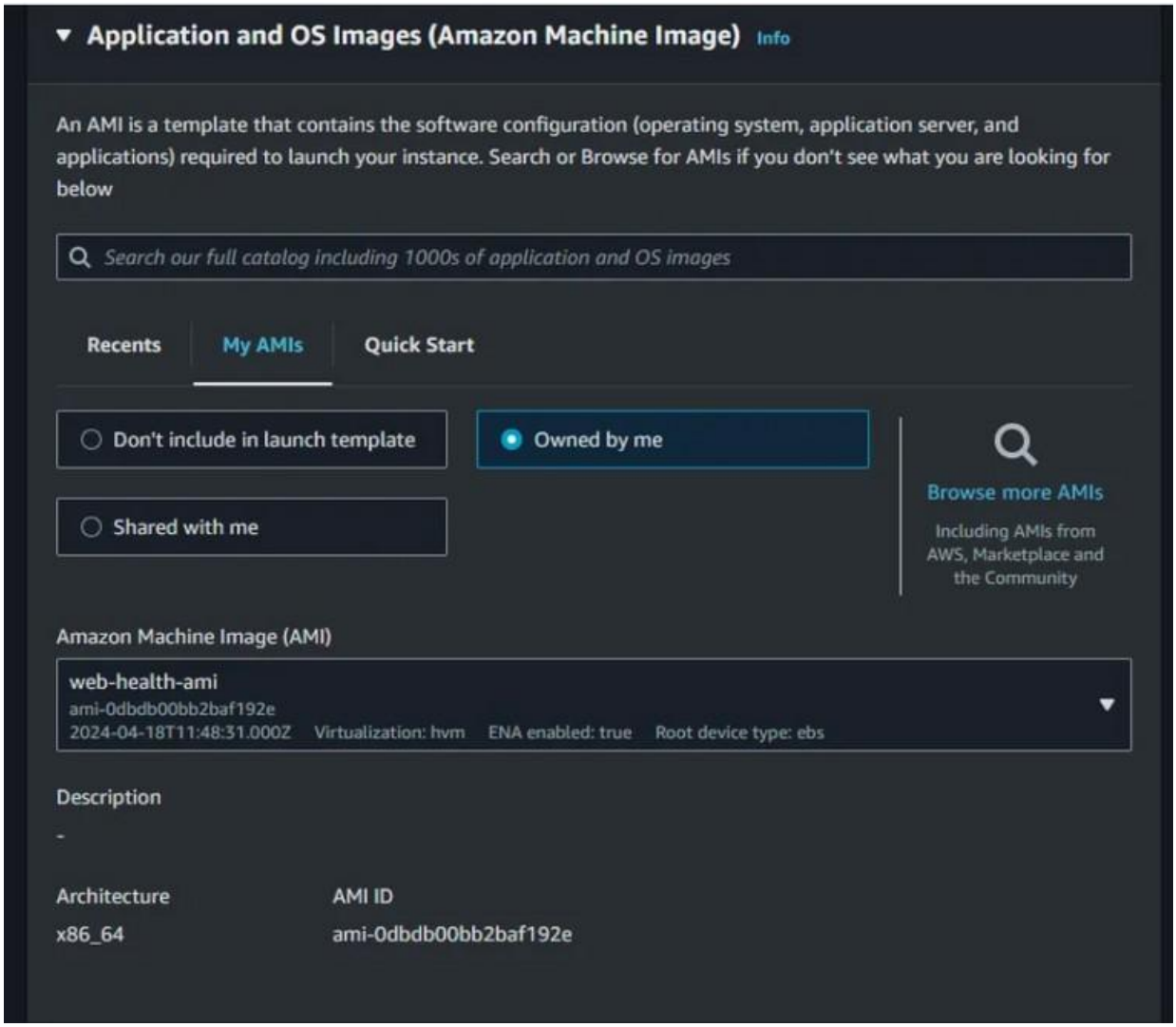
Also, AWS offers a service known as EC2 Image Builder, allowing you to establish automated pipelines for image creation.

The image pipeline in Image Builder defines all aspects of the process to customize images. It consists of the image recipe, infrastructure configuration, distribution, and test settings.

With the AMI in place, we can streamline instance launches by creating a launch template. Given name as “web-health-template” and version as “V1”

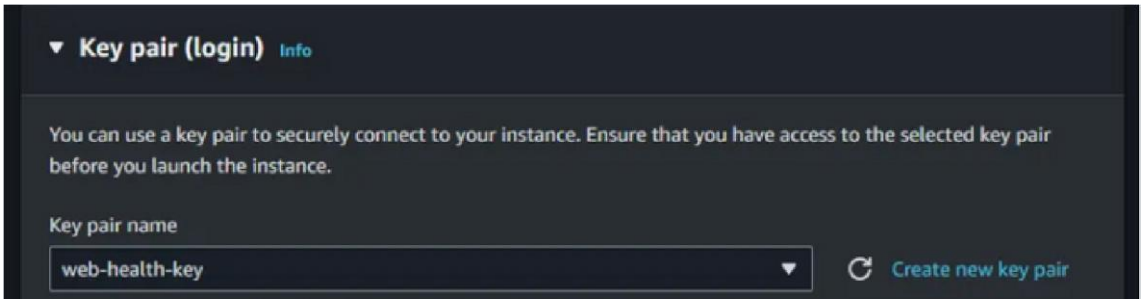


Then choose the “My AMIs” and select the AMI we created just before “web-healthami”

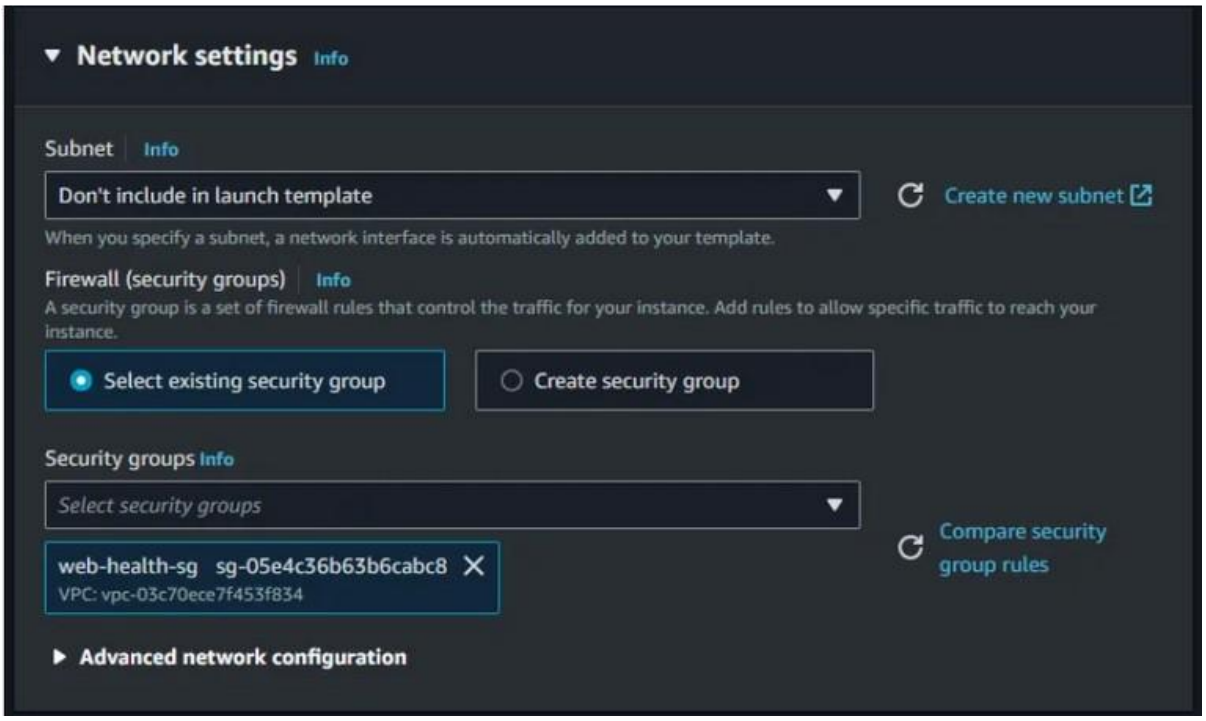


Select the instance type as t2.micro

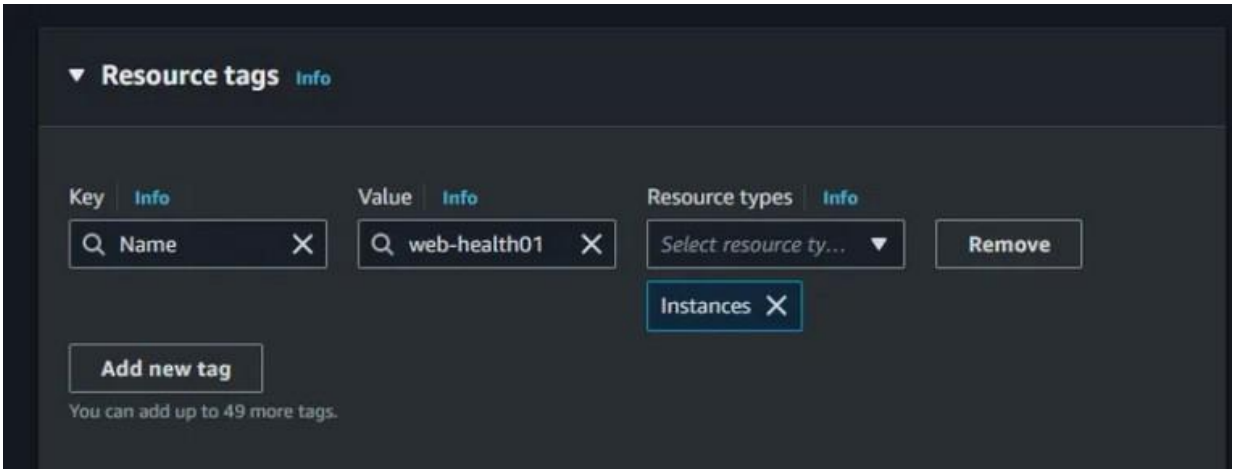
Select our key pair we created for the instance “web-health”

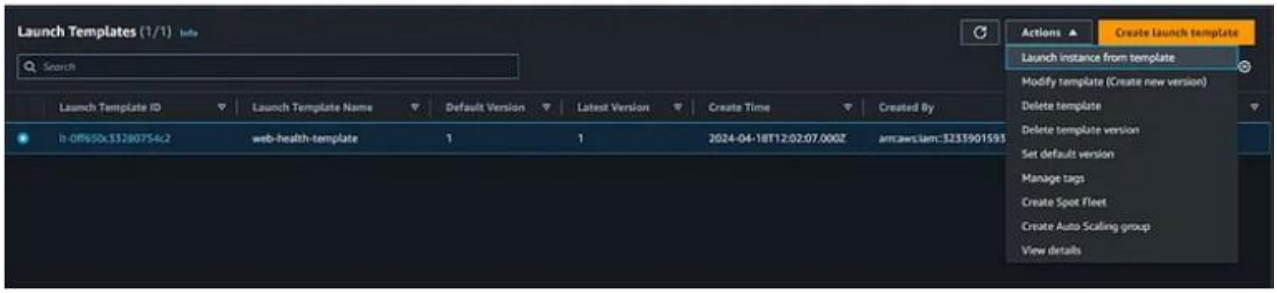


Select the security group as “web-health-sg” we created for our instance “web-health”



Just give as below: When launching an instance from this template, I’ll modify the name to follow a format such as 2,3,4 etc.



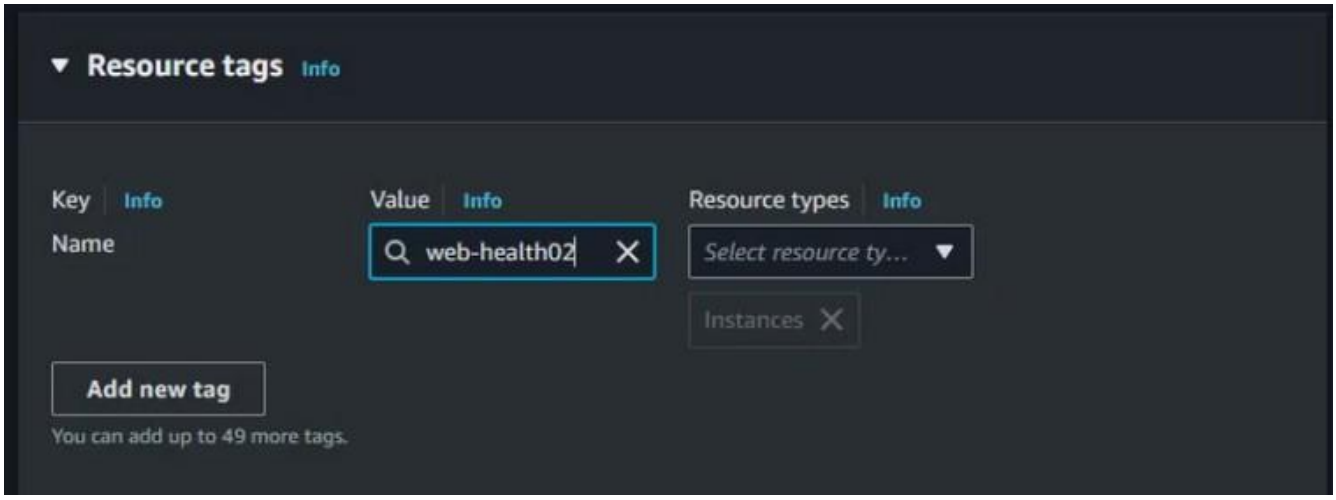


With the AMI in place, we can streamline instance launches by creating a launch template. This template encapsulates all the configuration details, allowing for quick and consistent instance deployments.

By utilizing launch templates, we can expedite the process of launching instances and maintain standardized configurations across our infrastructure.

After configuring our instances, AMI, and launch template, we'll proceed to set up a load balancer to distribute traffic efficiently across our web servers.

Simply navigate to the template section, locate your desired template, and select it. Then, choose the action "launch instance from template." During the launch process, you'll have the opportunity to make adjustments as needed. For example, you can modify the tag to "web-health01 to web-health02 two" before proceeding to launch the instance. That's all there is to it.



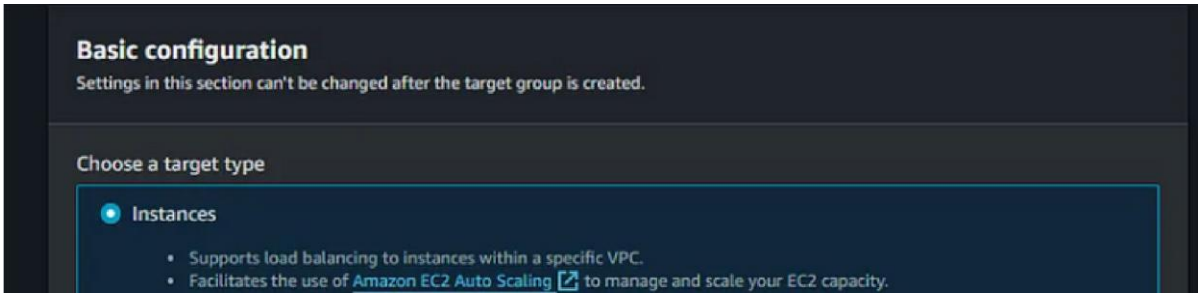
Now we have 2 instances which we created from templates and we can create 'n' number of instances upon our business needs.

We have successfully created an AMI and established a launch template utilizing this custom AMI.

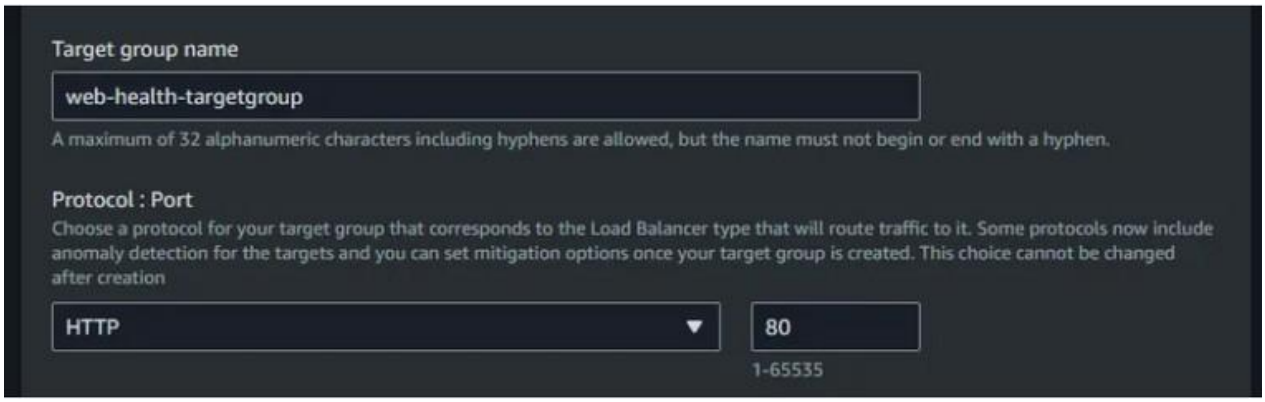
Presently, we have two web servers operating here. These servers are not individually accessible by users. Instead, users must access them through a unified endpoint, which directs requests to either of these instances. This single endpoint is facilitated by a load balancer.

In the Load Balancer section, we will first navigate to target groups. Target groups function as collections of instances, each subject to health checks. Essentially, they are groups comprising instances.
Let's proceed to create a target group.
Starting with creating a target group, we'll define health checks and specify the instances to include.

Choose target type as "Instances"



Provide a name as below and Port number 80, protocol HTTP.

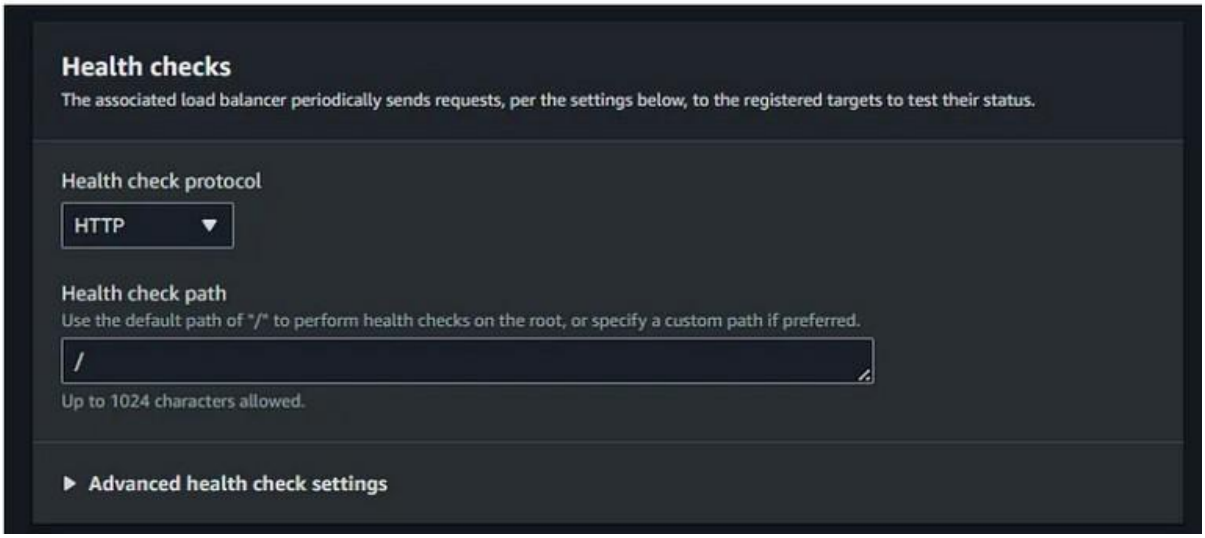


In our setup, we've implemented health checks within the target group. These health checks allow the target group to assess the status of each instance. If an instance is deemed unhealthy during these checks, the target group will refrain from routing requests to that specific instance.

Now, you might wonder, how does the target group determine if an instance is healthy or not? It's a similar process to how we, as humans, assess the health of a website. We simply open the website in a browser and check if it's functioning properly. If everything looks good, we consider the website to be healthy.
Similarly, the target group conducts health checks by sending HTTP requests to the instances. When configuring these health checks, we specify the protocol (HTTP) and the health check path (/).

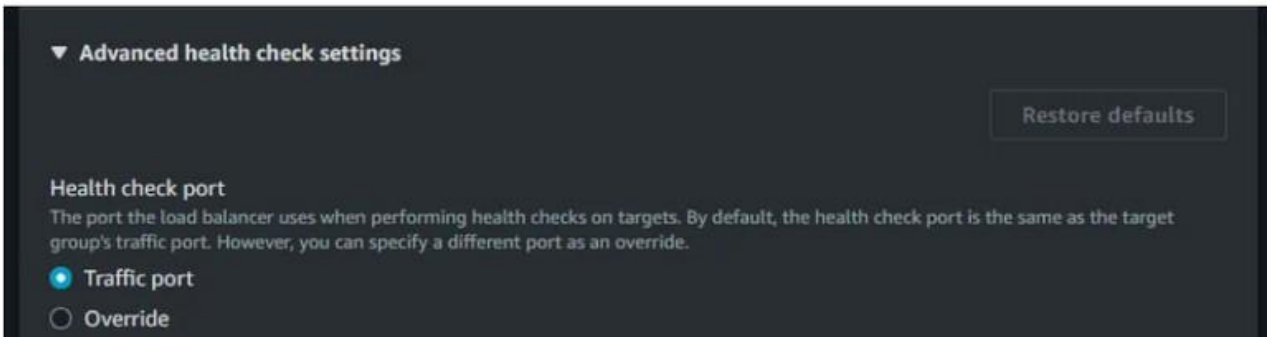
Let's break down what the health check path (/) signifies. Imagine your website's URL is http://IP:80. The slash at the end of this URL represents the root directory of the website. It's where the homepage resides.

Now, if your website has subfolders, such as "videos" or "images," the health check path would be adjusted accordingly (e.g., /videos or /images).
In essence, by specifying the health check path as "/", we're instructing the target group to check the root directory of our website. When the target group receives a successful response (HTTP 200 OK) from this path, it considers the instance to be healthy.



In our setup, traffic is expected to flow through the default port, which is port 80. However, if your website operates on a different port, you would need to specify it here.

To do so, you would select the “override” option and provide the appropriate port number, such as 8090. However, since we know that our website operates on port 80, we will leave it as is.



The “Healthy threshold” parameter determines the number of consecutive successful health checks required before declaring the instance as healthy. By default, it is set to five, but you can adjust it to two or ten as per your preference. With a threshold of five, the system checks the instance’s health five times, and if all checks pass successfully, the instance is marked as healthy.

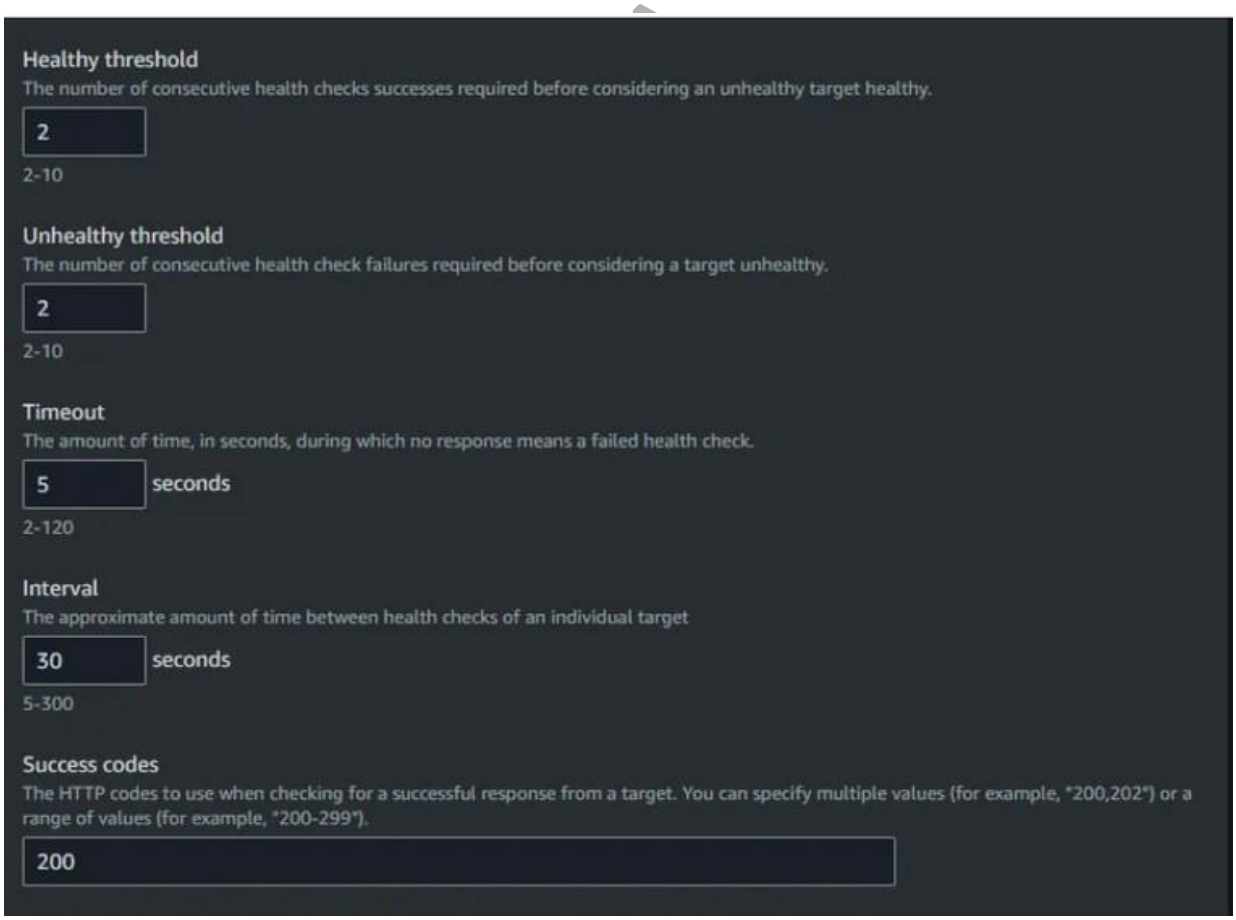
In our case, I will reduce it to two. This means that the instance will be considered healthy if it passes two consecutive health checks. Similarly, the “Unhealthy threshold” specifies the number of consecutive failed health checks required to declare the instance

as unhealthy. If an instance returns an unhealthy status twice in a row, it will be marked as unhealthy.

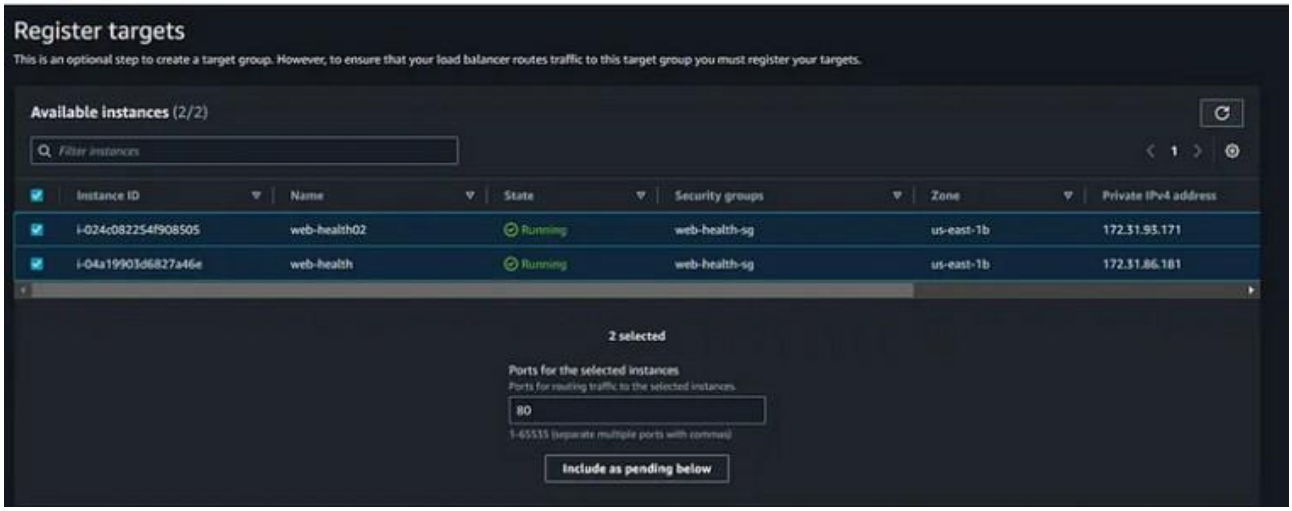
The “Timeout” parameter determines how long the system waits for a response from the website during each health check. If the website does not respond within the specified timeout period (default is 5 seconds), the system moves on to the next check.

Health checks, whether for determining the instance’s health or unhealthiness, occur at regular intervals, typically every 30 seconds. This interval can be adjusted within a range of 5 to 300 seconds.

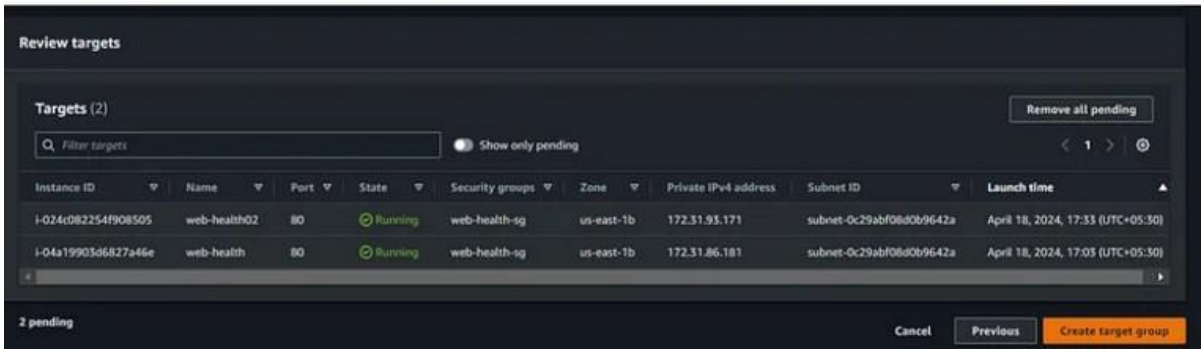
When performing health checks, the system looks for a success exit code, typically HTTP 200, indicating that the website is functioning properly. While we can visually confirm this by checking the website in a browser, the target group relies on these exit codes to determine the instance’s health status.



So the below are the instances running and click on the “Include as pending below option”.

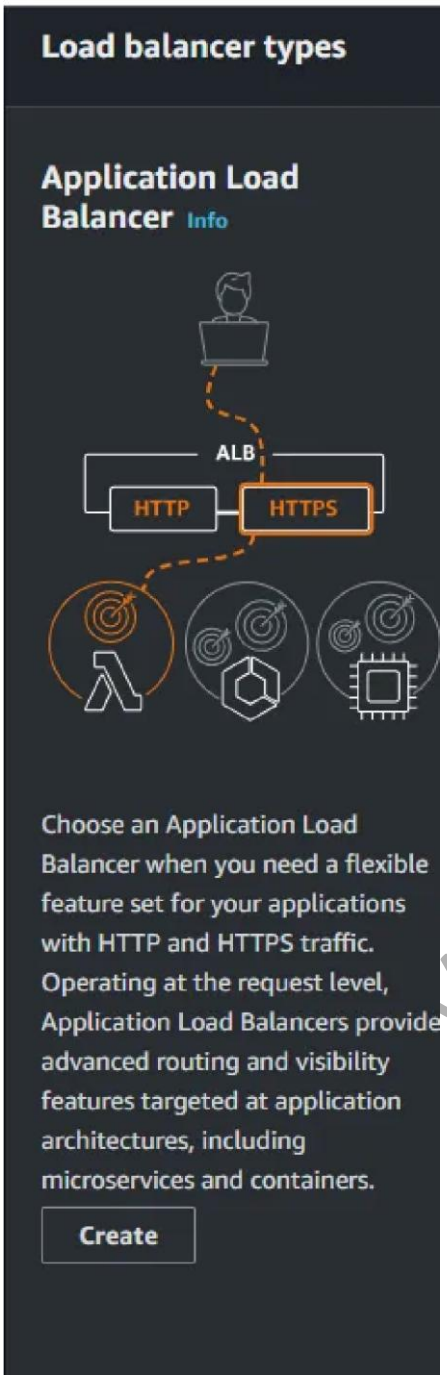


And the targets are in running state.

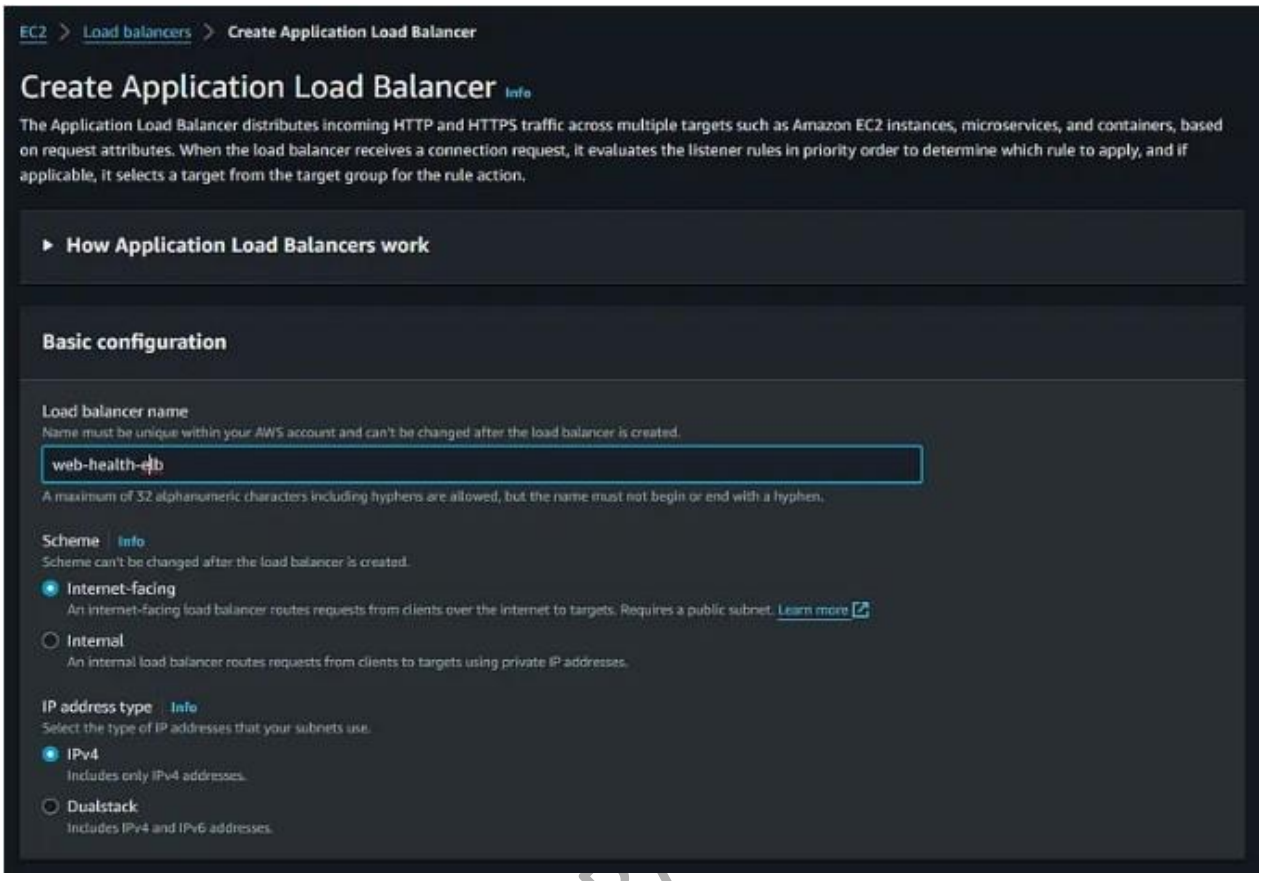


Once the target group is set up, we’ll proceed to create an application load balancer (ALB), specifying its name, internet-facing nature, and availability zones.

We will go with application load balancer for http, https traffic.



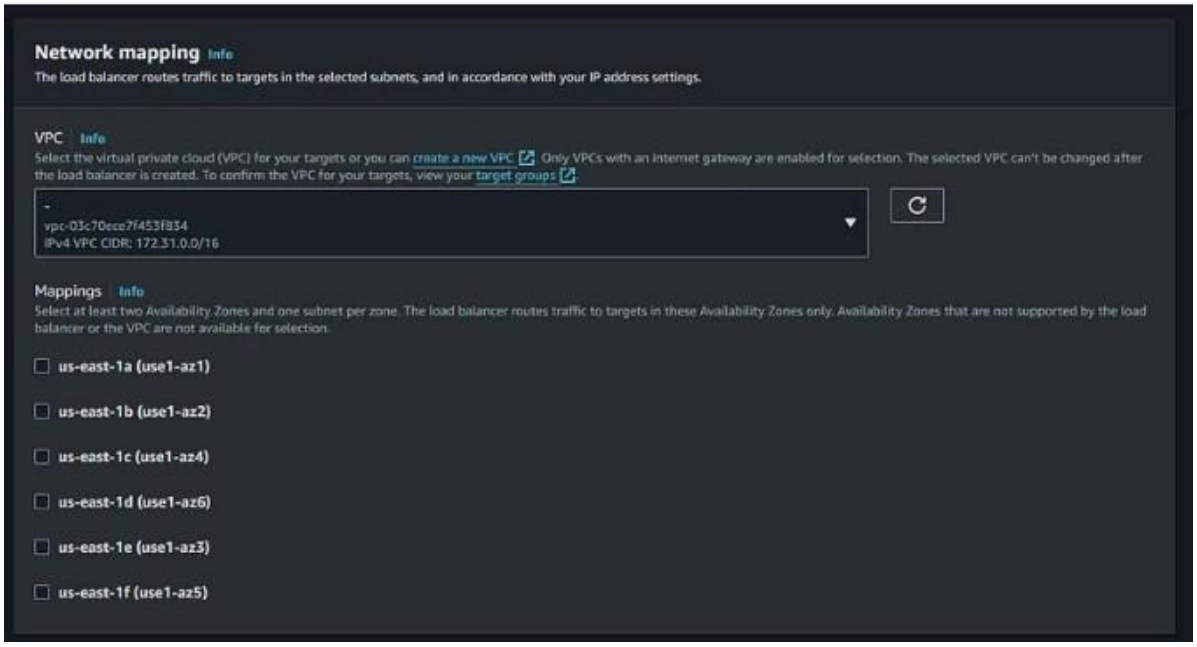
Give a load balancer name “web-health-elb” and We want it to be available on the Internet, so we’ll keep it Internet facing.



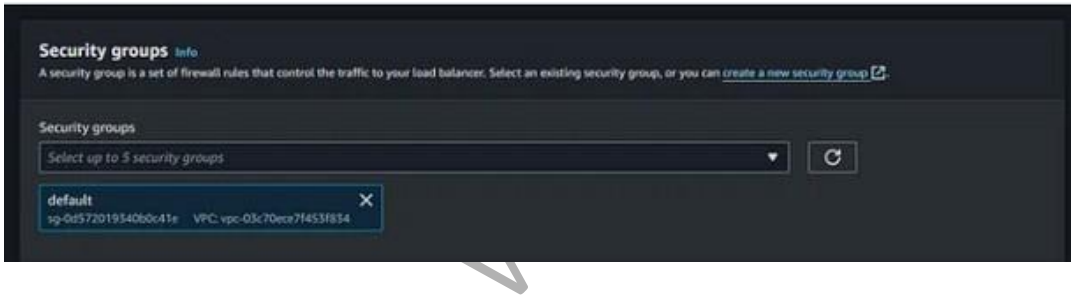
When setting up the load balancer, it's necessary to specify the availability zones where the load balancer will operate. The system requires a minimum of two zones to ensure redundancy and high availability.

For our configuration, I will choose to select all available zones. This ensures that the load balancer operates across multiple zones, maximizing its availability and fault tolerance. This approach provides a high level of availability at the load balancer level, which is crucial for maintaining continuous service availability.

By selecting all zones, we distribute the load balancing functionality across multiple data centers or regions, reducing the risk of downtime due to failures in any single zone. This enhances the resilience and reliability of our application infrastructure, allowing it to withstand potential disruptions in individual zones.

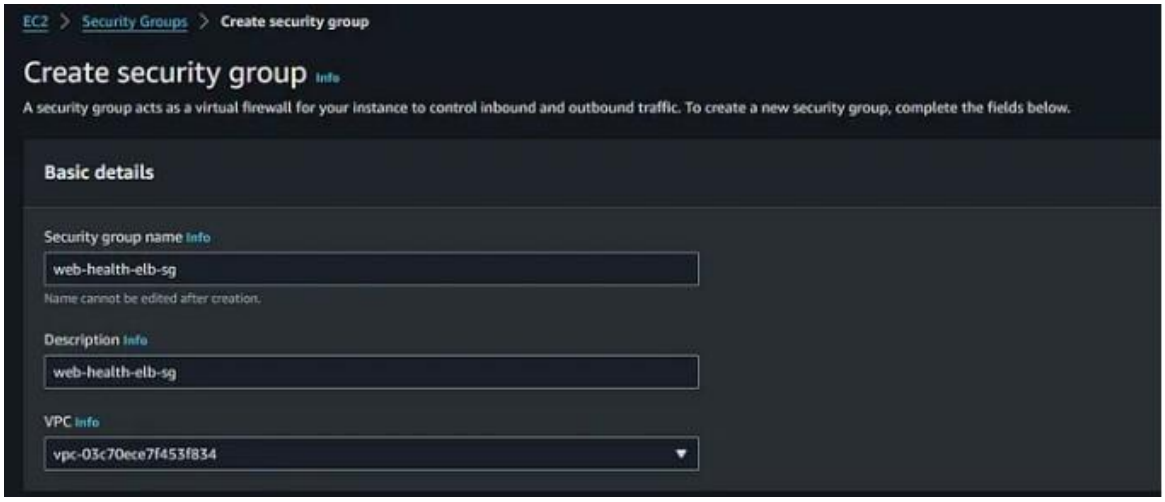


Click on the “Create a new security group”



For the security configuration of our load balancer, we’ll create a new security group to define its access permissions. It’s important to name this security group appropriately for easy identification and management.

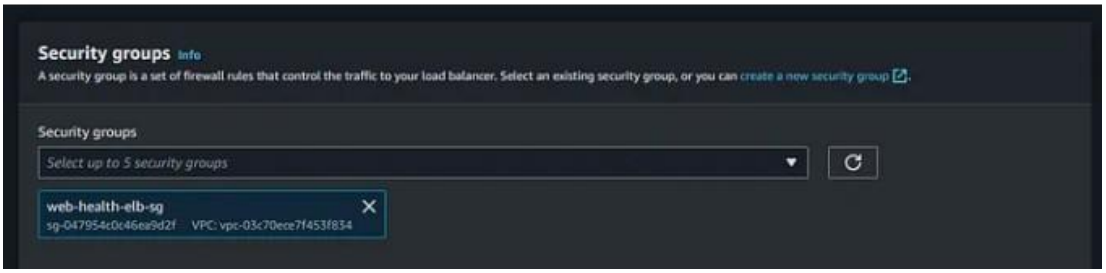
We’ll name it “web-health-elb-sg” to indicate that it’s the security group for our health project’s load balancer. Using descriptive names helps maintain clarity and organization within our AWS environment.



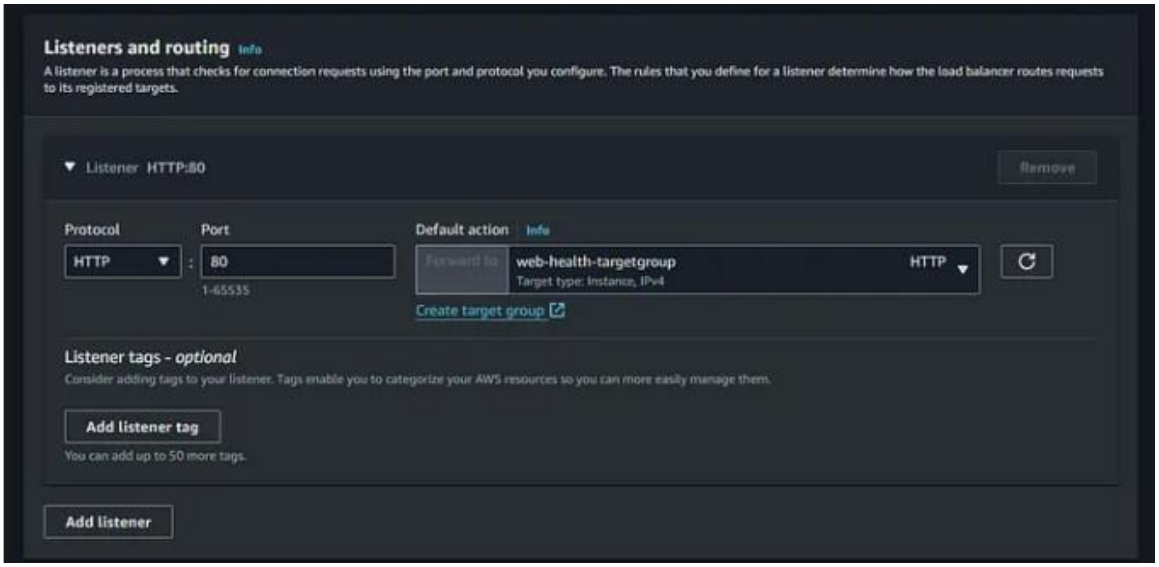
As for the inbound rule, we want to allow access on port 80 from anywhere, making the load balancer accessible to the public. We'll specify the protocol as IPv6 to accommodate modern internet standards, especially considering the increasing adoption of IPv6 by internet service providers, including mobile networks.

Ensuring broad accessibility while maintaining security is crucial for our load balancer's functionality and usability. Making these considerations and adopting best practices, such as proper naming conventions and access control, contributes to the overall robustness of our infrastructure.

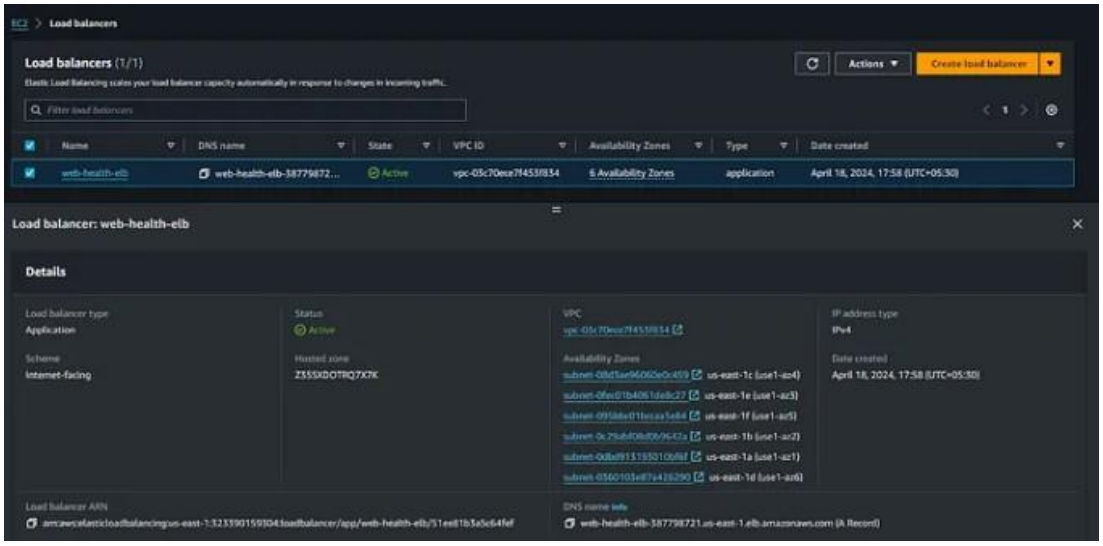
Now select the security group we created just now:



Ensuring that the ALB's security group allows inbound traffic on port 80 from any source, we'll configure listeners to route traffic to the target group on port 80.



Now let's create a load balancer.



Upon viewing the load balancer, you'll notice that it initially enters the provisioning state, transitioning to the active state once fully provisioned. However, despite reaching the active state, attempting to access the website directly from the load balancer may not yield the expected results.

When encountering this issue, accessing the website via the load balancer's DNS name might result in a 504 Gateway Timeout error, indicating a failure to establish a connection between the load balancer and the instances. Despite verifying that the instances are healthy, the presence of this error suggests a potential obstacle preventing traffic from passing between the load balancer and the instances.

To address this challenge, we need to identify and resolve the underlying cause obstructing traffic flow between the load balancer and the instances. This discrepancy could be attributed to misconfigured security groups, routing issues, or other networkrelated factors.

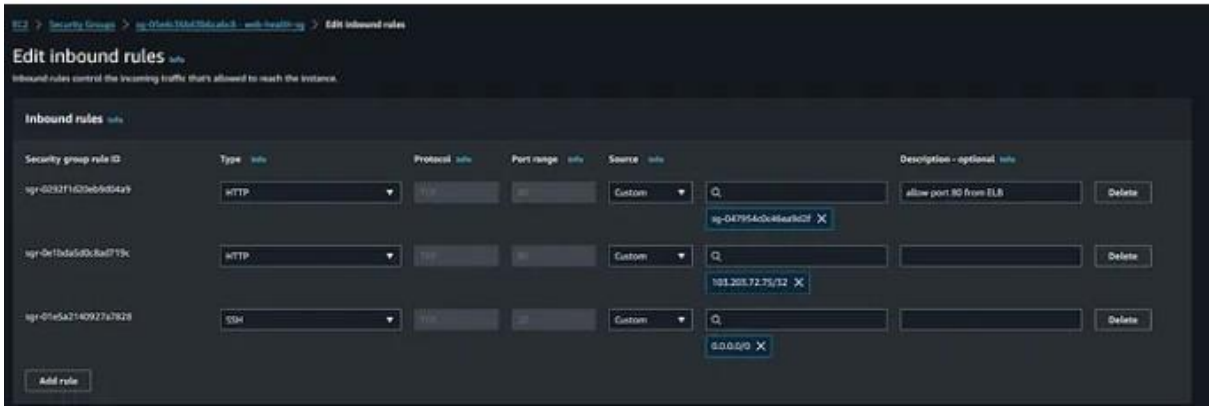
By diagnosing and rectifying this barrier, we can ensure seamless connectivity between the load balancer and the instances, thereby resolving the 504 Gateway Timeout error and enabling users to access the website reliably through the load balancer.

Each instance is shielded by a security group, a vital layer of defense governing inbound and outbound traffic. Upon inspection, we observed that both instances share the same security group. However, upon reviewing its inbound rules, we discovered that port 80 was only permitted from our IP address, rather than from the load balancer.
To rectify this, we need to adjust the security group settings to allow traffic from the load balancer. But before proceeding, it's prudent to verify the status of the target group. The target group conducts health checks to determine an instance's well-being. If an instance is deemed unhealthy, the target group will cease routing traffic to it. In this scenario, instances were marked as unhealthy due to the inability to access port 80.

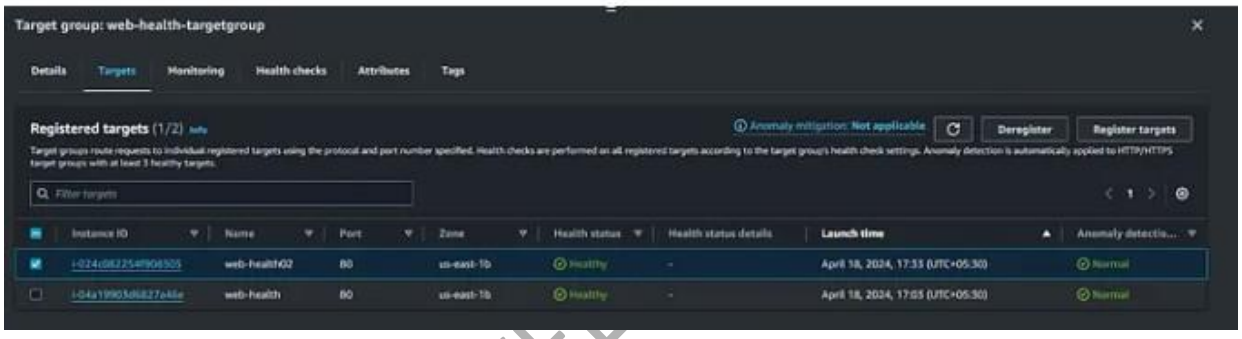
Returning to the security group settings, we locate the security group associated with the instances and modify its inbound rules. Specifically, we add a new rule permitting port 80 access from the security group of the load balancer. By specifying the source as the load balancer's security group, we ensure that any member of the load balancer's group can communicate with the instances over port 80.

Additionally, it's beneficial to include a descriptive label for this rule, such as "allow port 80 from ELB," to enhance clarity and documentation. Descriptive labels streamline management and aid in understanding the purpose of each rule, promoting efficient administration and troubleshooting.

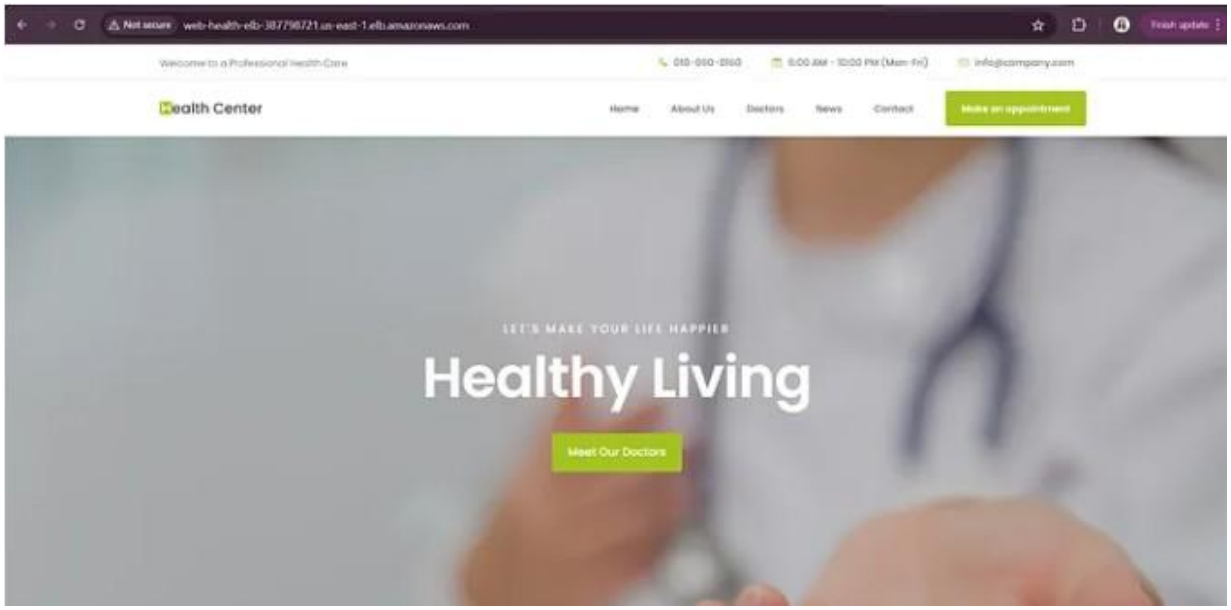
Make the security groups inbound rules as below of “web-health”.



Save the rule.
Make sure that the targets are healthy.



Now check it on the browser:



Managing instances within target groups is a critical aspect of optimizing the performance and availability of applications deployed on AWS. Target

groups serve as collections of instances that are subject to health checks, ensuring that traffic is routed only to healthy instances. Here are some key points to consider when managing instances within target groups:

1. Health Monitoring: Target groups continuously monitor the health of instances by conducting health checks based on configured parameters. These health checks assess the instance's ability to respond to requests and determine its overall health status.
2. Registration and Deregistration: Instances can be registered or deregistered from target groups dynamically based on operational requirements. This flexibility allows for seamless scaling and maintenance operations without disrupting application availability.
3. Maintenance and Scaling: Registering or deregistering instances from target groups is a common practice during maintenance activities or when scaling the application. By removing unhealthy or underperforming instances from the target group, traffic can be effectively redirected to healthy instances, ensuring consistent application performance.
4. Load Balancing: Target groups play a crucial role in load balancing by distributing incoming traffic across multiple instances based on predefined rules and health check results. This ensures optimal resource utilization and enhances application reliability and scalability.
5. Automation: Managing instances within target groups can be automated using AWS services such as Auto Scaling, which automatically adjusts the number of instances in response to changing demand. This automation helps optimize resource usage and ensures that the application can handle fluctuations in traffic effectively.

Finally, we'll familiarize ourselves with managing instances within the target group, including registering and deregistering instances as needed for maintenance or scaling purposes.

By grasping these concepts and experimenting with the setup, we can gain a comprehensive understanding of deploying and managing web applications on AWS.

AWS CloudWatch

Introduction to CloudWatch:

- AWS CloudWatch is a comprehensive monitoring service offered by Amazon Web Services.

- Originally focused on monitoring, CloudWatch has expanded its capabilities to include logging, events, and more.
- It serves as a central hub for monitoring the performance and health of AWS environments.

Key Features of CloudWatch:

- Monitoring Service:
  - CloudWatch serves as a monitoring solution for AWS resources, tracking performance metrics such as CPU utilization, disk I/O, and network traffic.
  - It automatically generates standard metrics for various AWS services used in a region.
- Logging Solution:
  - In addition to monitoring, CloudWatch functions as a logging solution, allowing users to collect, store, and analyze log data generated by AWS resources and applications.
  - Logs from services like EC2 instances can be streamed to CloudWatch for centralized log management.
- Event Monitoring:
  - CloudWatch captures real-time events within the AWS environment, such as instance launches, terminations, or volume creations.
  - Users can set triggers and notifications based on these events, often integrated with AWS Lambda functions.
- Standard and Custom Metrics:
  - CloudWatch provides both standard and custom metrics for monitoring AWS resources.
  - Standard metrics cover common performance indicators like CPU utilization, network traffic, and disk operations.
- Users can define custom metrics tailored to their specific monitoring needs.
- Alarms and Notifications:
  - Users can set alarms on CloudWatch metrics to trigger notifications when predefined thresholds are breached.
  - Notifications can be sent via email or integrated with Amazon SNS for broader alerting capabilities.
- Integration with AWS Services:

- CloudWatch seamlessly integrates with various AWS services, including EC2 instances and EBS volumes.
- Metrics and logs from these services are collected and monitored by CloudWatch, providing insights into resource performance.

Practical Use Cases:

- CloudWatch simplifies monitoring by automatically collecting metrics for AWS resources.
- Users can customize monitoring settings and set up alarms to receive timely notifications of any performance anomalies.
- Practical examples include setting alarms for CPU utilization exceeding a certain threshold, which triggers email notifications via SNS.

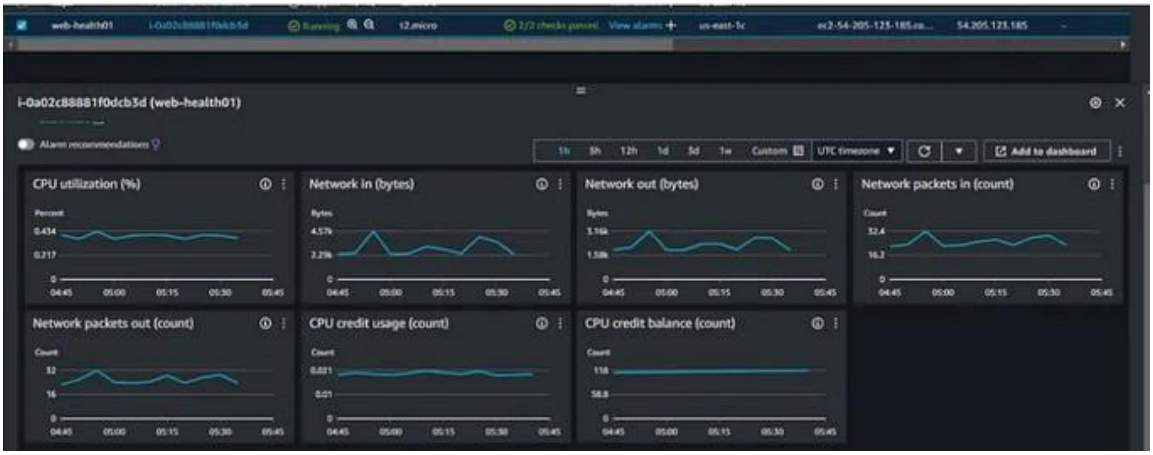
If you've already reviewed the previous blog post, we've provided a template for launching EC2 instances. Kindly review it, or proceed to create an EC2 instance. Select the EC2 instance and click on monitoring:

Before proceeding further, take note of the various metric names available, such as CPU utilization, which indicates the percentage of CPU being used over time. Additionally, observe other metrics like status checks, network in and out (in bytes), network packets in and out (count), and disk read operations.

These metrics are automatically generated by CloudWatch when you launch an EC2 instance.

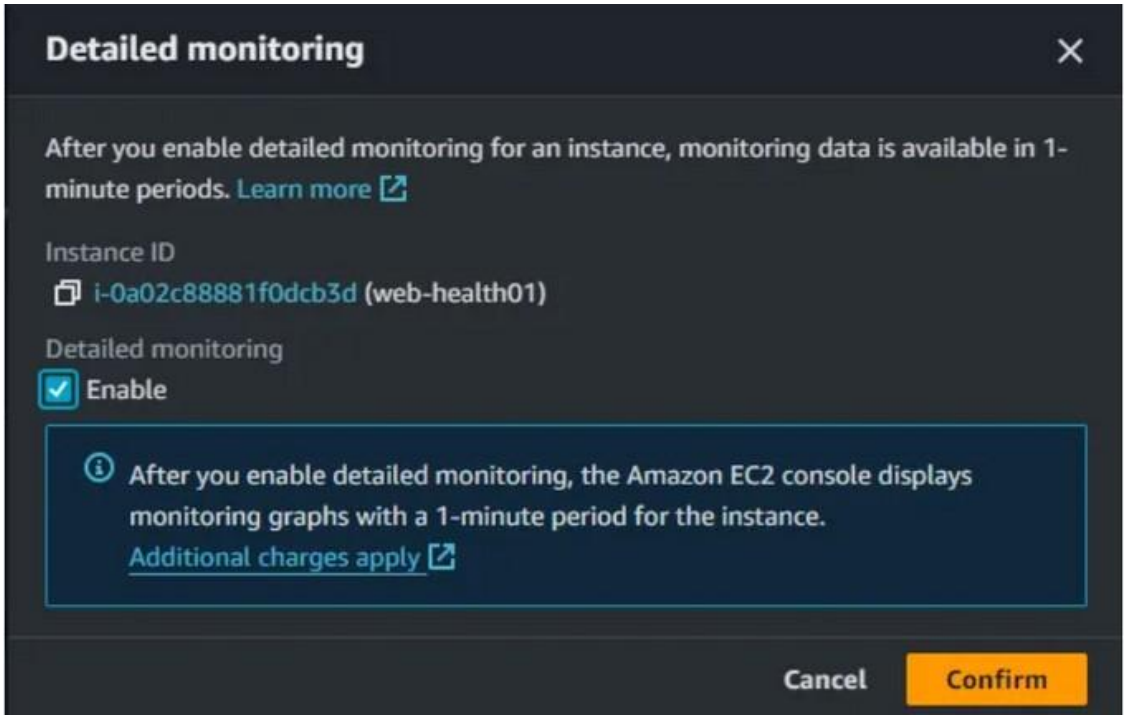
If you require additional metrics, such as RAM or disk utilization, you'll need to create custom metrics.However, for the purpose of this session, we'll focus solely on CPU utilization, as it's one of the most critical metrics to monitor.

By default, CloudWatch checks and updates these metrics every 5 minutes, populating the corresponding graphs.If you prefer more frequent updates, you can enable detailed monitoring, though it's important to note that this option isn't free.



For this hands-on, enabling detailed monitoring is optional.

1. Select the option to “Enable” detailed monitoring, indicating that CloudWatch should monitor metrics every minute.
2. Keep in mind that detailed monitoring incurs additional charges compared to the default 5-minute monitoring interval.
3. Despite the additional cost, enabling detailed monitoring offers more granular insights into your AWS environment’s performance.



Log in to the terminal of the EC2 instance and switch as root user.

Then install the stress tool along with its dependencies on your instance.

Stress is a versatile tool designed to test the limits of your Linux operating system, particularly focusing on CPU performance and other system metrics. Simply running the stress command allows you to stress the CPU. You can specify parameters such as the number of CPUs to stress and even utilize functions like the square root function. Additionally, stress can be used to stress other components like IO and RAM, but for our purpose, we're concentrating on CPU stress testing.

yum install
stress -y

```
[root@ip-172-31-29-118 ~]# nohup stress -c 3 -t 100 &
[2] 66003
[root@ip-172-31-29-118 ~]# nohup: ignoring input and appending output to 'nohup.out'
```

nohup
stress -c 3 -
t 100 &

This command executes the stress utility in the background using the nohup command, which allows the process to continue running even after the terminal session is terminated. Let's break down the command:

- nohup: Prevents the following command from being terminated when the terminal session ends.
- stress: The stress testing utility.
- -c 3: Specifies that 3 worker threads will be used to stress the
- CPU cores.
- -t 100: Specifies that the stress test will

run for 100 seconds. &: Runs the command in the background.

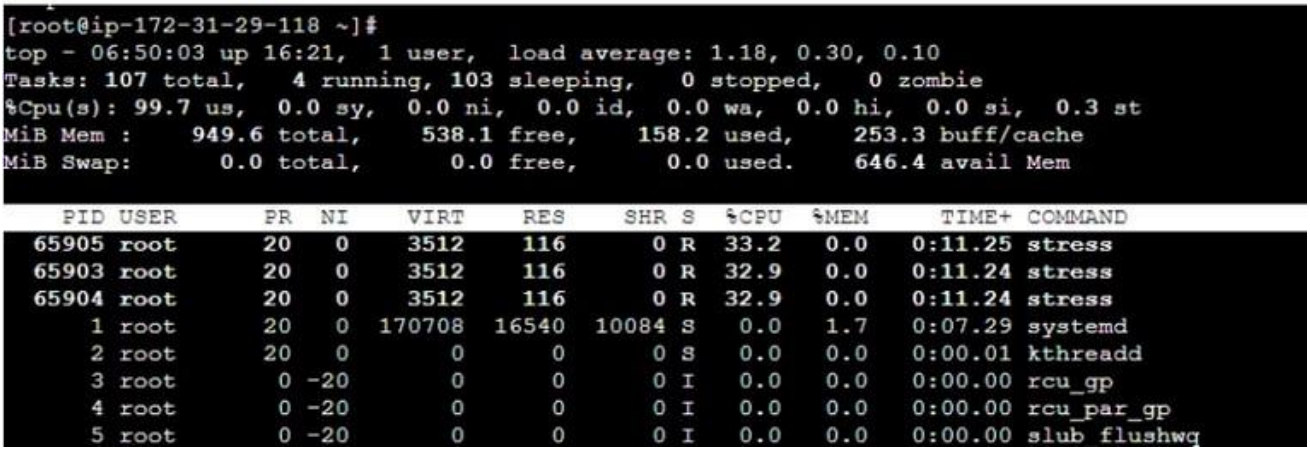
So, this command will run the stress test with 3 worker threads for 100 seconds, and the process will continue running even if the terminal session is closed. Execute the top command.

top

You'll notice four stress processes running, indicating the CPU utilization at 100 or close to it, reflected by the load average increment.

Continuously stressing the instance like this will be monitored by CloudWatch, updating the graph every minute.

Repeat this process a few times, running for intervals like 100 seconds and 200 seconds. After a few minutes, you'll observe a pattern forming on the graph.



Let's create a script for this process.

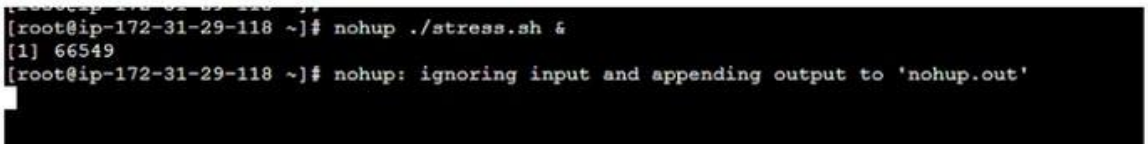
vim stress.sh
sleep 60 && stress -c 4 -t 60 && sleep 60 && stress -c 4 -t 60 && sleep 60 && stress -c 4 -t 30 && sleep 60 && stress -c 4 -t 100 && sleep 30 && stress -c 4 -t 200 :wq!

This sequence of commands progressively varies the duration of the stress tests, interspersed with periods of rest. It's a way to simulate different levels of CPU load on the system over time. designed to stress test the system's CPU using the stress utility. Let's break down each part:

■sleep 60: This command pauses execution for 60 seconds before proceeding to the next command. It introduces a delay in the sequence.
■stress -c 4 -t 60: This command runs the stress utility with the following options:
■-c 4: Specifies that 4 worker threads will be used to stress the CPU cores.
■-t 60: Specifies that the stress test will run for 60 seconds.
■sleep 60: Another pause of 60 seconds.
■stress -c 4 -t 60: Similar to the second command, this runs a stress test for another 60 seconds.
■sleep 60: Another pause of 60 seconds.
■stress -c 4 -t 30: This command runs a shorter stress test for 30 seconds.
■sleep 60: Another pause of 60 seconds.
■stress -c 4 -t 100: This command runs a longer stress test for 100 seconds.
■sleep 30: A shorter pause of 30 seconds.
■stress -c 4 -t 200: This command runs an even longer stress test for 200 seconds.

Execute this script:

./stress.sh



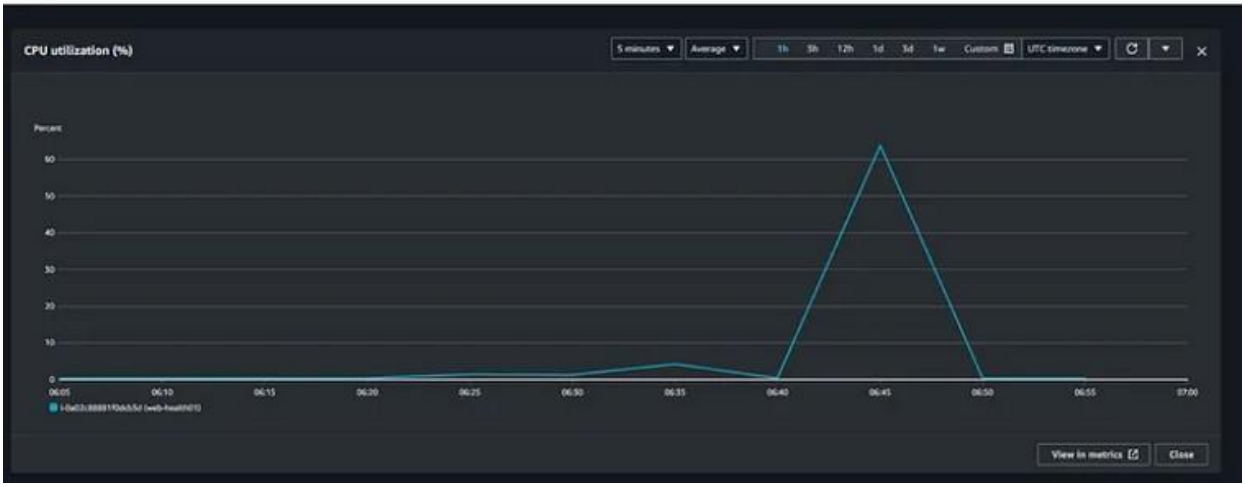
We generate a graph by running the command. I simply executed it using nohup stress.sh &.

Ensure your script

is executable.

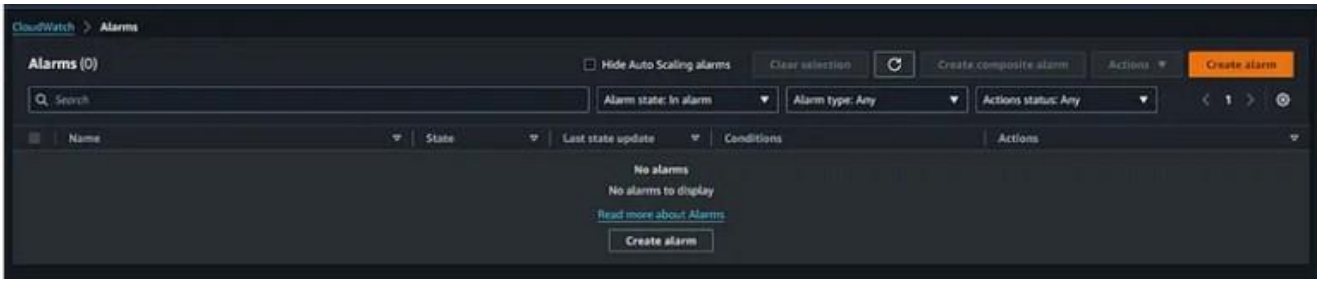
chmod +x stress.sh

Now, when observing the top output, you may intermittently notice the stress command based on the ongoing operations within your script.



Now, let's navigate to the CloudWatch service to configure an alarm for CPU utilization on this instance.

First, head over to the "All alarms" section. Next, click on the "Create Alarm" button.

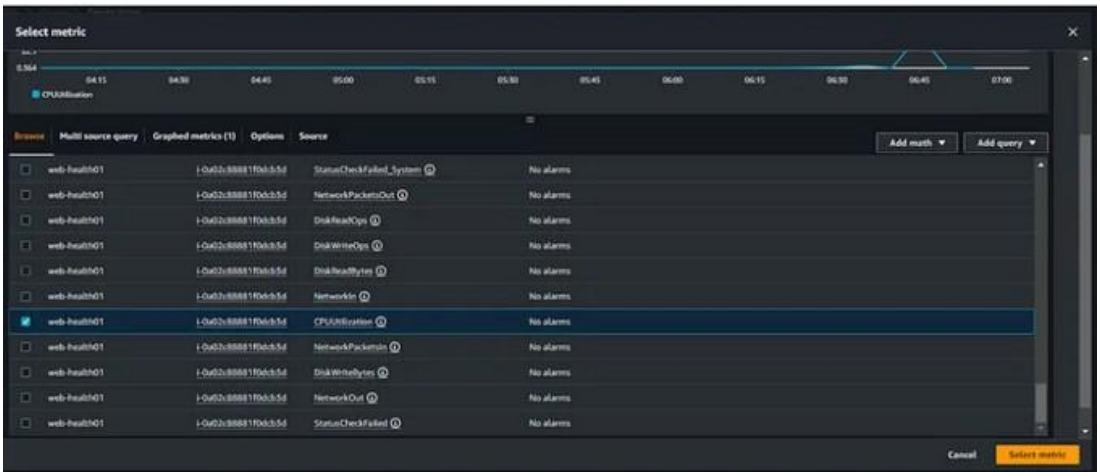


To select the CPU utilization metric, start by navigating to the EC2 service.

Then, locate the "Per-instance metrics" section and find your instance. If you don't immediately see your instance, wait for a few moments for the information to load.

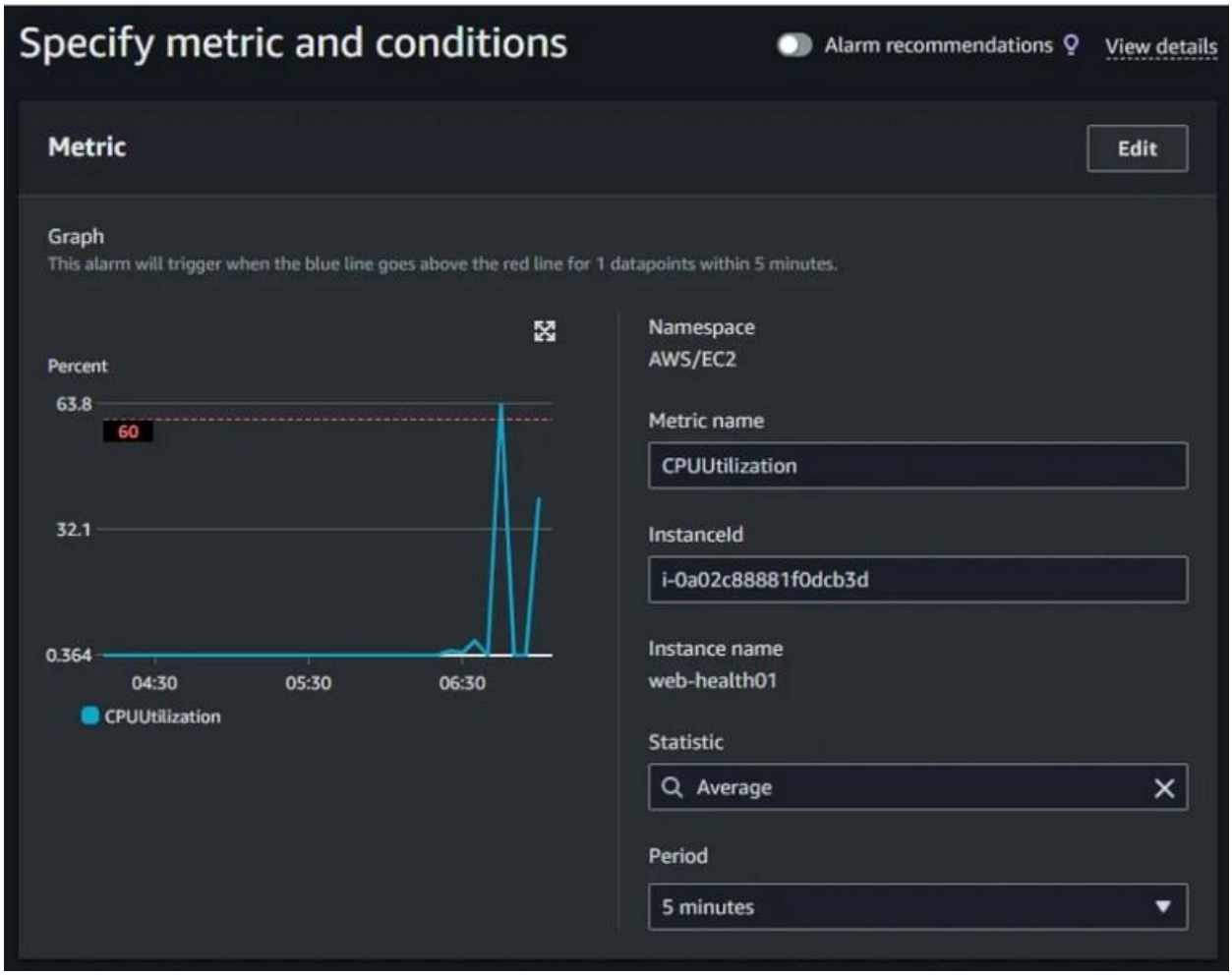
Once you've found your instance, look for the CPU utilization metric.

Select the CPU utilization metric by clicking on it.



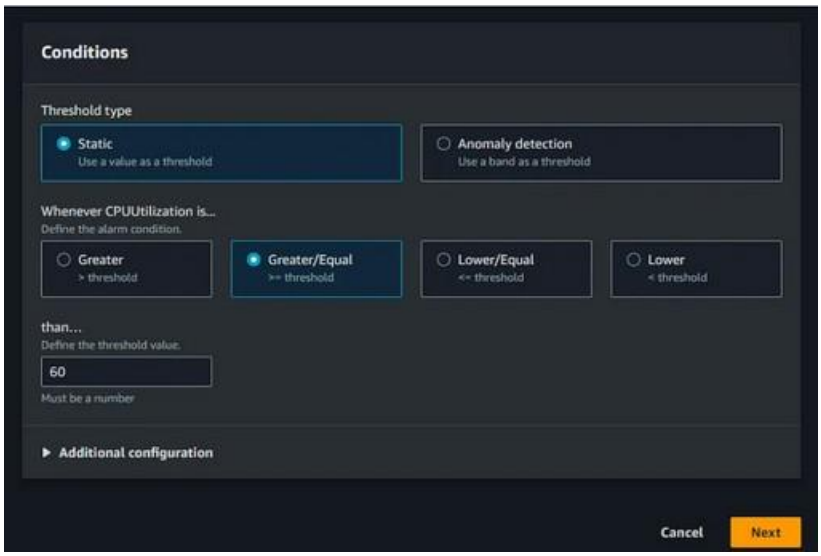
In this step, we can choose the period for which the alarm will be evaluated. For this demonstration, I'll keep the period set to 5 minutes.

Next, we specify the condition for triggering the alarm based on CPU utilization.



For example, we can set the condition to trigger the alarm if the CPU utilization is greater than or equal to 60 for a period of 5 minutes.

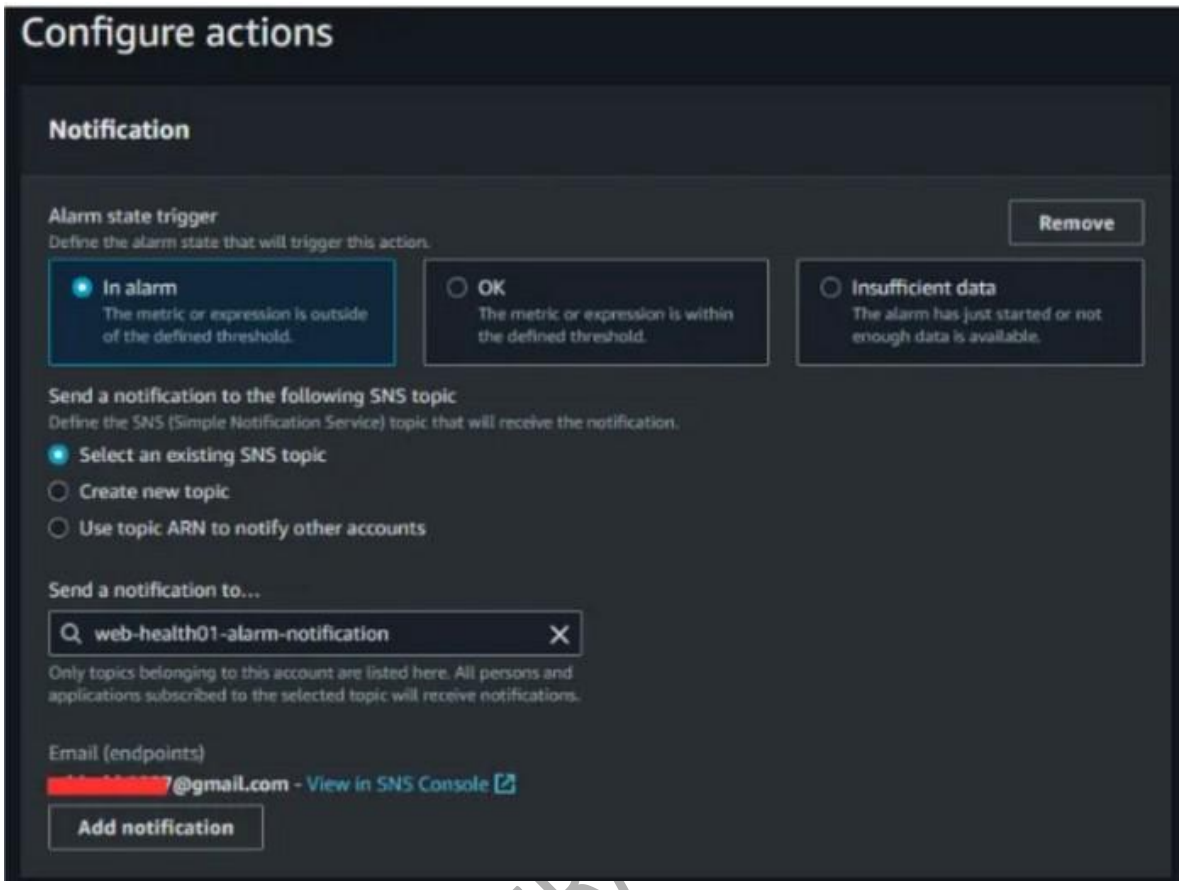
Once we've defined the condition, we proceed to the next step.



If you don't find your desired topic listed, you can click on "Create Topic." Provide a name for the topic, enter your email address, and click on "Create Topic."

However, since we already created a topic for billing alarms earlier, I'll select the same topic, which contains my email address.

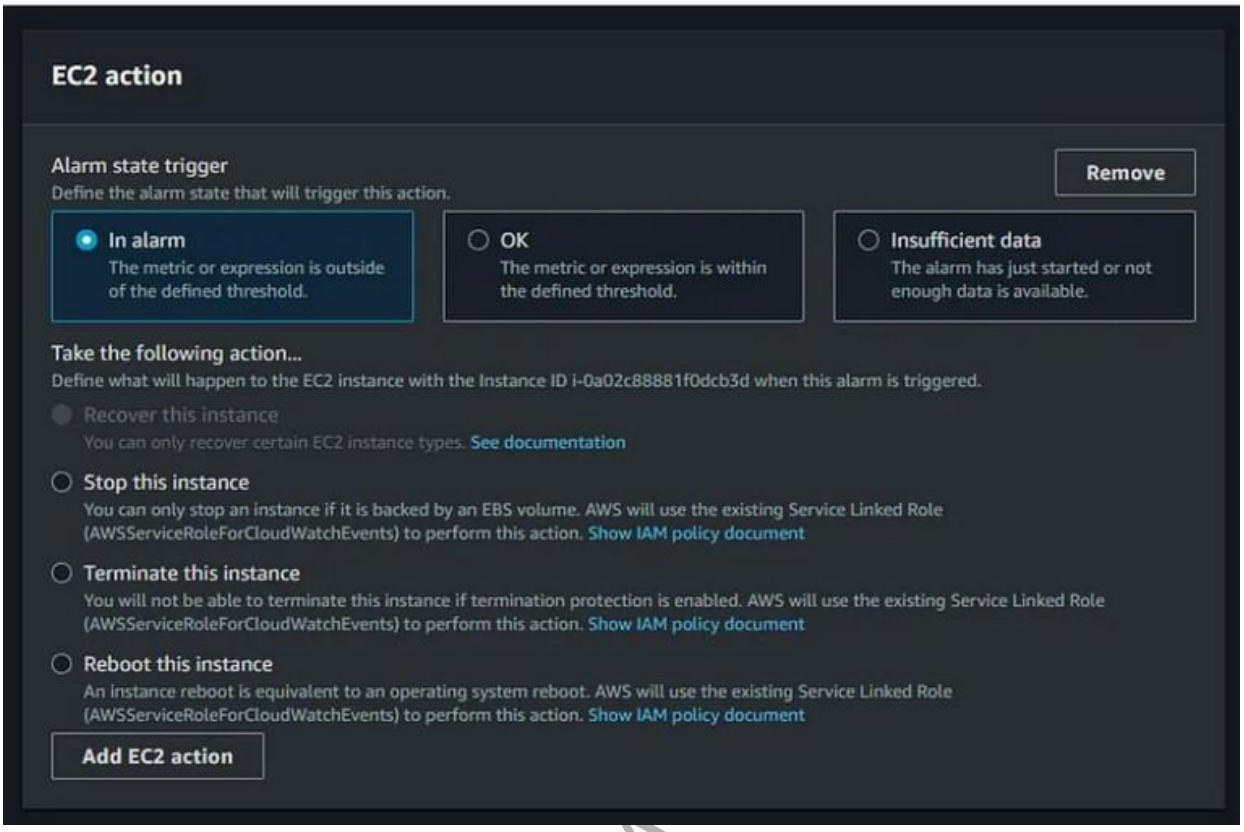
This topic is designated for notifications. So, when the alarm state is triggered, it will send a notification to this topic, ultimately resulting in an email notification being sent to me.



There are several other actions you can take, such as EC2 actions. For instance, you might want to stop, terminate, or reboot the instance if it triggers an alarm.

In some cases, a high CPU utilization might prevent you from logging into the instance via SSH. Rebooting the instance could be a temporary solution to address this issue.

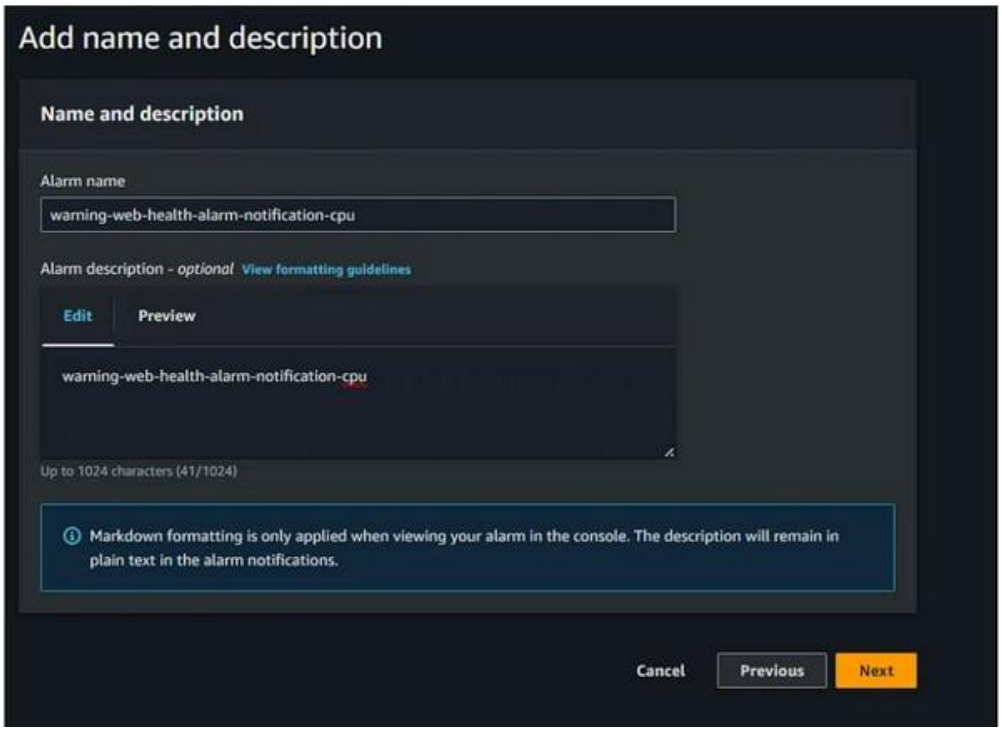
However, for the purpose of this demonstration, we will skip these additional actions.
We will stick to email notifications only and proceed to the next step.



Let's give the alarm a descriptive name.

The naming convention will be: "warning-web-health-alarm-notification-cpu" for the specific instance.

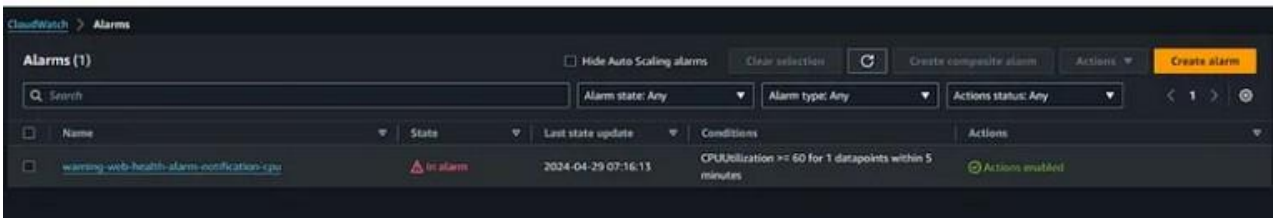
In our organization, a warning is triggered when the utilization exceeds 60%. We could set up additional alarms for critical situations, such as when it surpasses 80%.
However, for now, we'll proceed with just one alarm.
Now, let's move on to the next step.



After a while, CloudWatch will collect the data and display whether the instance is in an alarm state or if it's okay.

Now, this process aligns with the basic principles of any monitoring tool. You have metrics or checks, alarms triggered by those metrics or checks, and actions associated with those alarms, such as sending email notifications.

Whether it's Prometheus, Nagios, Icinga, Zenos, or others, the concept remains similar. However, with CloudWatch, monitoring is already configured, and you only need to set up alarms. In contrast, with other tools, such as those you set up yourself, the entire monitoring system needs configuration, typically by the monitoring or administration team.

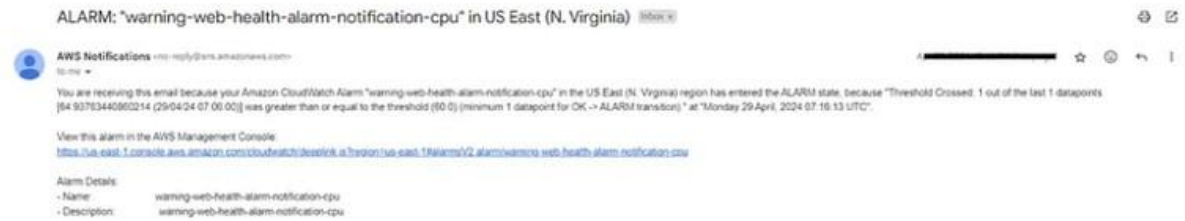


Currently, the graph fluctuates because I've executed the stress command multiple times. Remember, you need to run the stress

command for at least 5 minutes or longer for it to exceed the threshold and trigger the email notification.
After running the command for 5 minutes, wait for the notification to arrive in your inbox.

Moreover, you can create a reverse alarm to monitor the instance's OK state. For instance, if the CPU utilization is below 40, it's considered OK. You can set up alarms for different thresholds and states, and you have the flexibility to create up to ten alarms under the free tier.

Email notification:



I encourage you to experiment with these alarms to familiarize yourself with their functionalities. Set alarms for various data points, both in alarm and OK states, using different conditions like greater than or less than. Once you're comfortable, remember to terminate the instance and delete the alarms before moving on to the next section, particularly the auto-scaling group section.

Auto Scaling Groups

Auto Scaling Groups (ASG) seamlessly integrates with CloudWatch to monitor specific metrics, such as CPU utilization. If a metric, like CPU utilization, exceeds a predefined threshold, ASG adjusts capacity by either adding or removing instances in the group. This adjustment is based on CloudWatch alarms, which trigger actions to maintain performance or control costs.

When creating an auto scaling group, you'll configure launch configurations or launch templates, similar to those used in EC2 instances for ELB exercises. These configurations provide the necessary information for launching instances within the ASG.

To dynamically adjust capacity based on metrics, ASG employs scaling policies. You can define these policies, such as step scaling, to add or remove instances based on workload fluctuations. For instance, you could specify that if CPU usage exceeds 60%, launch two additional instances, or if it exceeds 80%, launch four instances.

When setting up an auto scaling group, you specify parameters like minimum, desired, and maximum capacity. The minimum size ensures a baseline level of instances that cannot be terminated, while desired capacity represents the typical number of running instances. The maximum size limits the total number of instances the group can scale up to, helping to control costs.

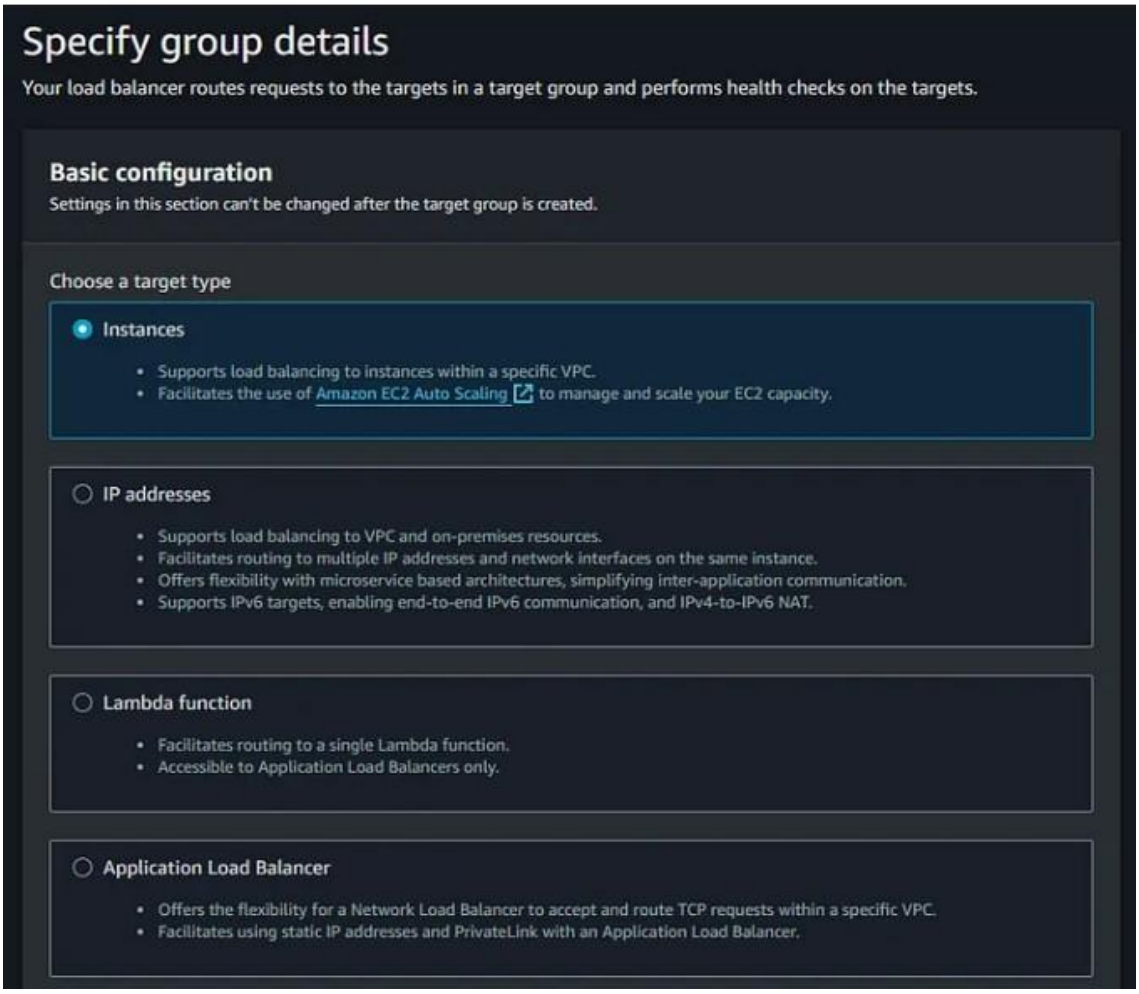
In practice, ASG operates by creating a group powered by launch configurations. When a metric breach triggers a CloudWatch alarm, scaling policies kick in to launch or terminate instances accordingly, ensuring optimal performance and resource utilization.
Now, let's dive into the practical implementation of these concepts.

Before diving into the auto scaling group setup, let's ensure we have all the necessary components in place.

Firstly, we'll need the launch template, which should already be created as per the instructions provided in the load balancer section. If the launch template has been removed or needs to be recreated,

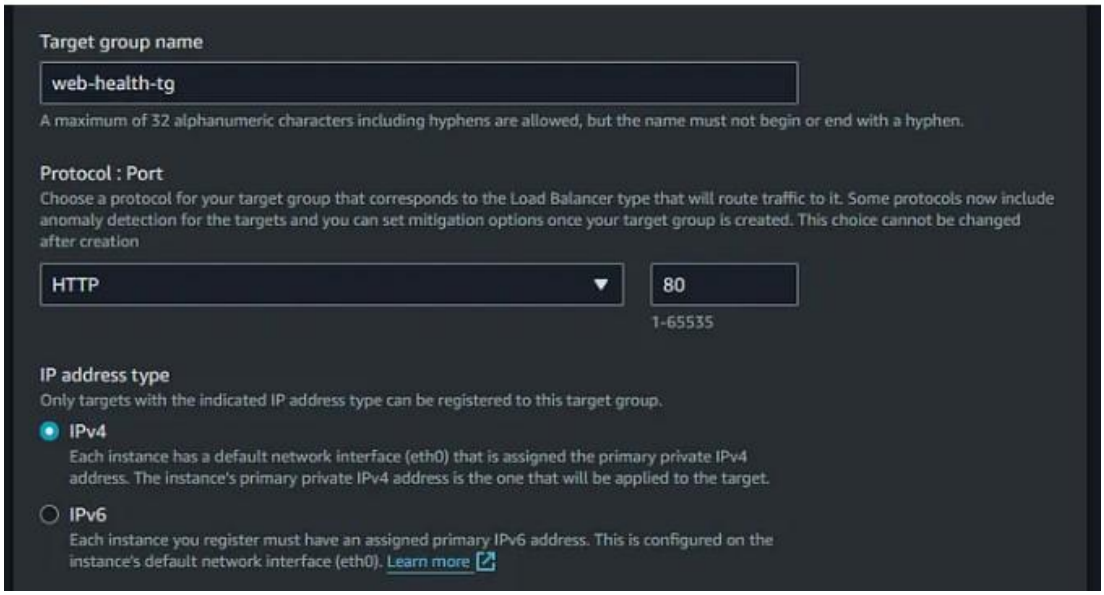
refer back to the load balancer section for detailed steps on how to create it.

Now, let's proceed to the Target Group setup.
■Navigate to the Target Group section.
■Click on the "Create target group" button.



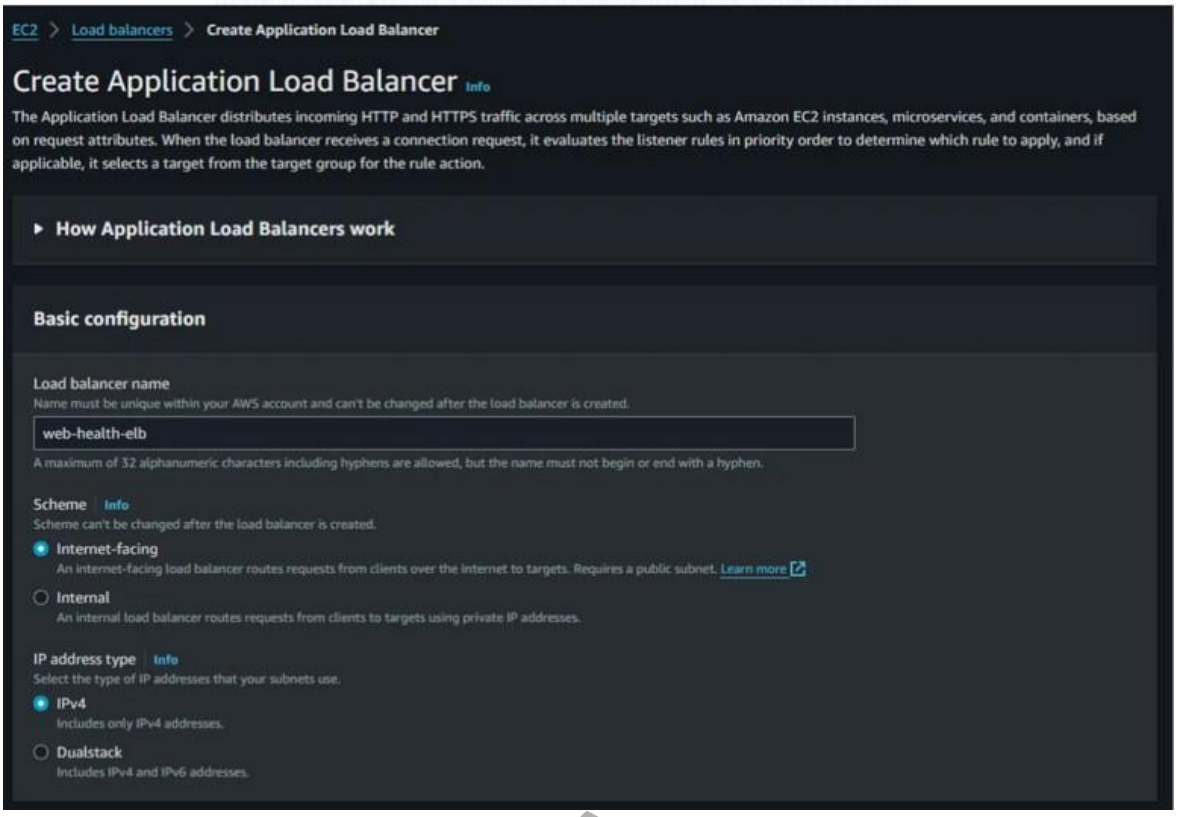
Configure the target group settings, such as name. For example, let's name it "webhealth-tg".

■Keep the other configurations as default and proceed by clicking "Next".
■Since we don't have any instances yet, select the option to create an empty target group.
■Complete the target group creation process by following the prompts.

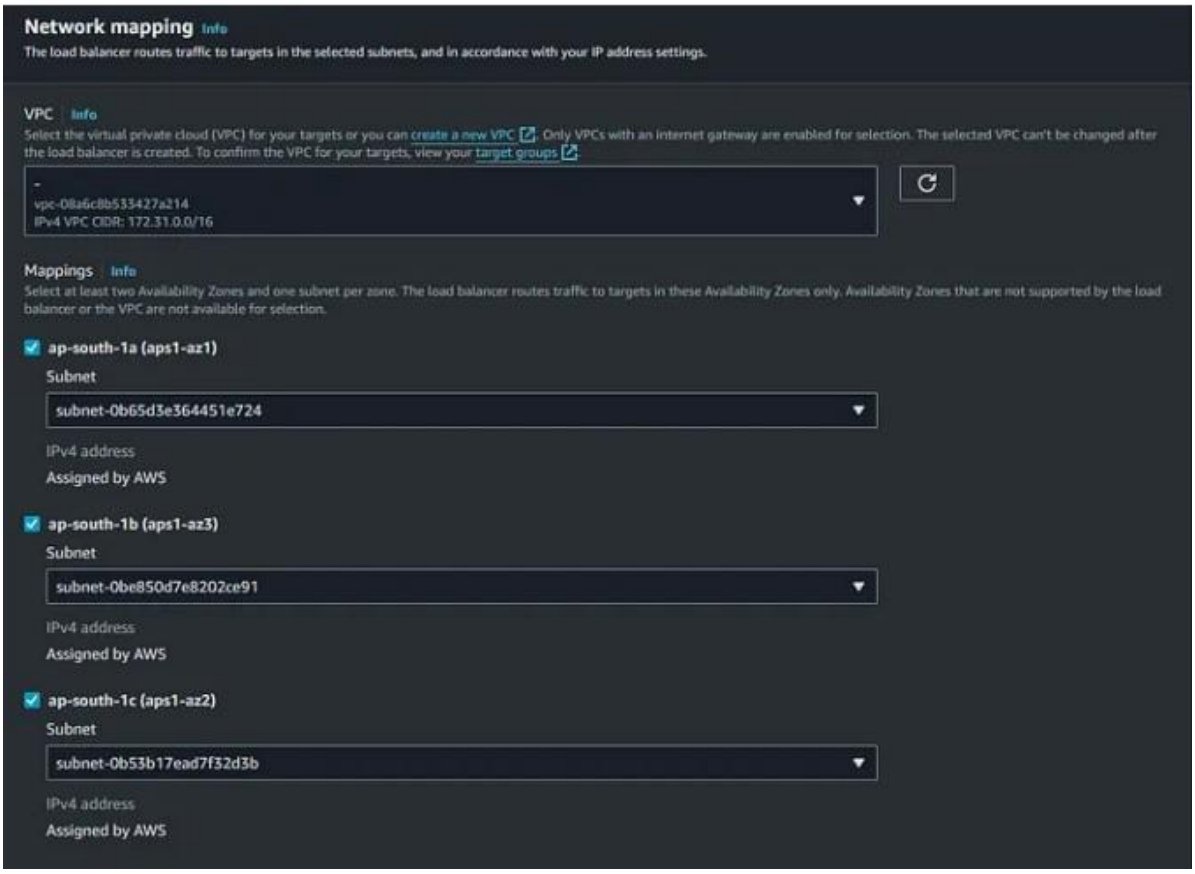


Indeed, the Auto Scaling group will automatically add instances to this target group as needed based on your scaling policies and the metrics you've configured, ensuring seamless scalability and efficient resource management.

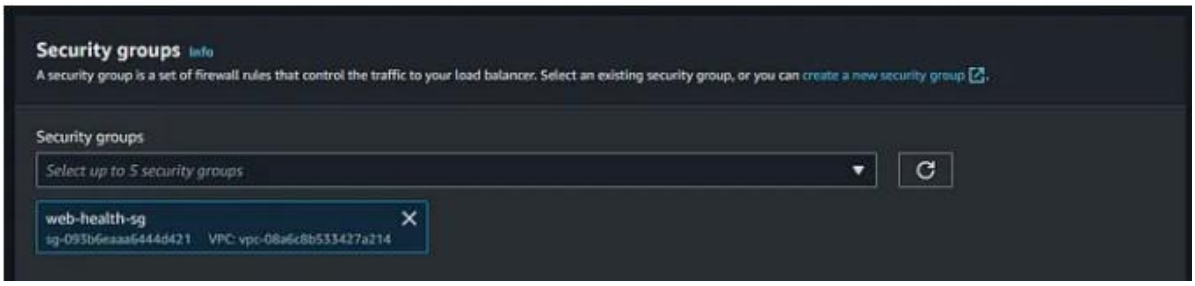
We'll proceed by creating a load balancer for this target group. Let's select "Create load balancer" and opt for the application load balancer, maintaining consistency with our previous setup.

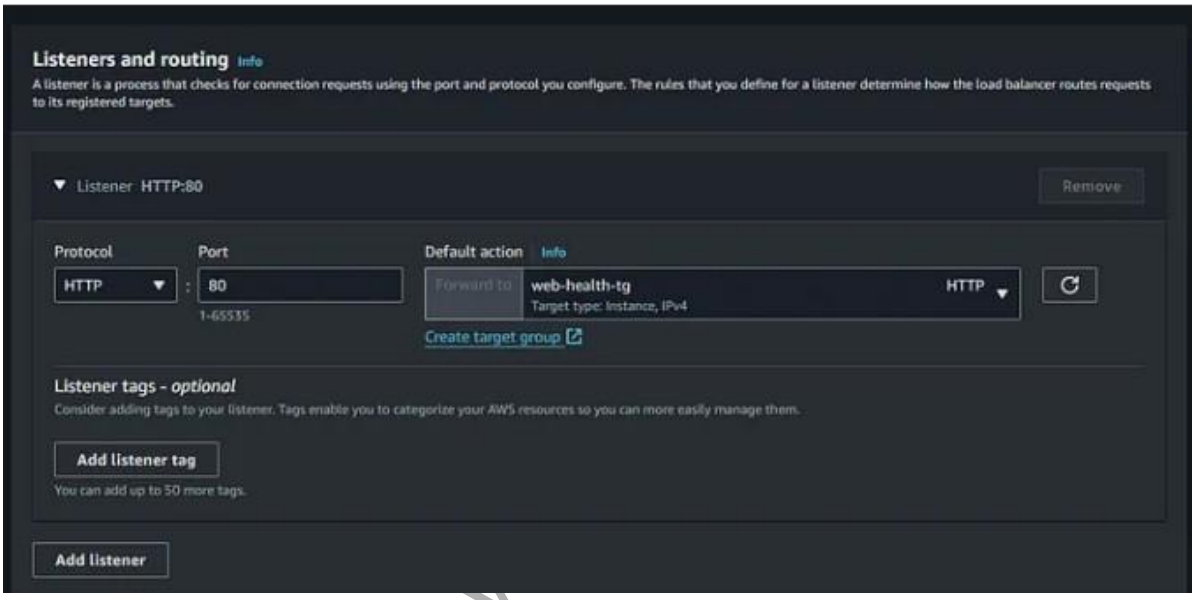


We'll name it "health-elb" and choose the same availability zones and security groups used previously.



Remember to remove the Default Security Group. Next, we'll configure it to forward traffic to the target group we just created.



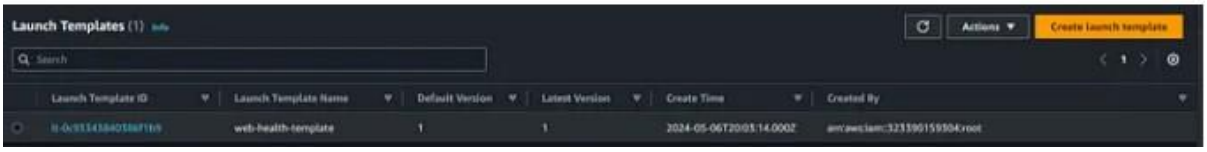


If you already have these resources set up, you can reuse them; otherwise, go ahead and create them now. Here's our load balancer.

Let's revisit the launch template. This template is essential as it defines the configuration for the instances that will be launched by the auto scaling group.

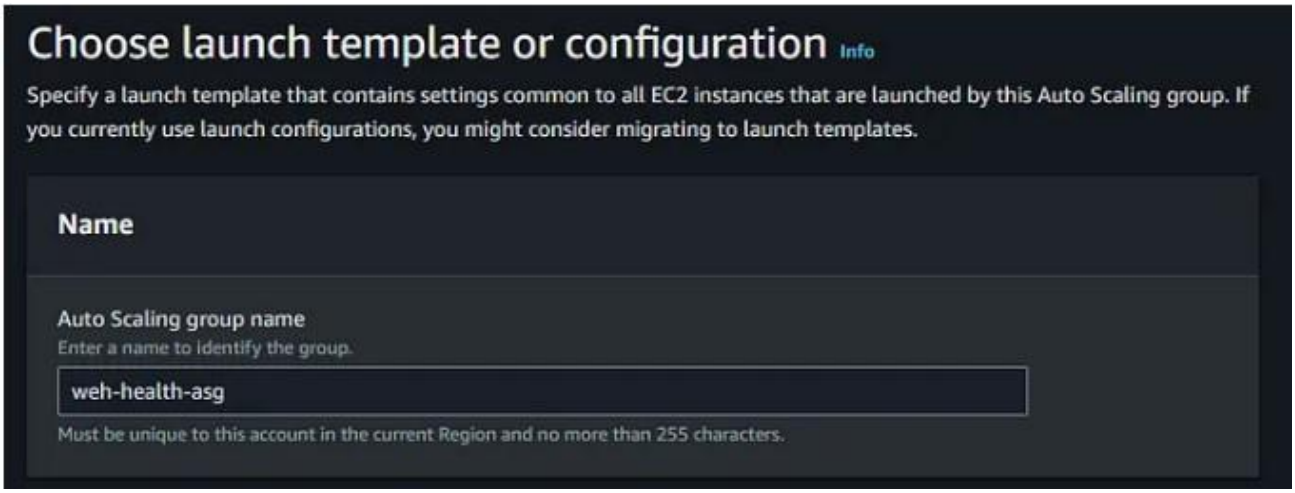
It's important to verify that the required AMI is available and accessible. If you've already prepared the AMI, it should be readily available for selection within the launch template settings.

The launch template specifies details such as the Amazon Machine Image (AMI) to be used for the instances.



Now, let's navigate to the auto scaling groups section.

Let's initiate the creation process by assigning a name to our auto scaling group. For simplicity, let's name it "web-health-asg".

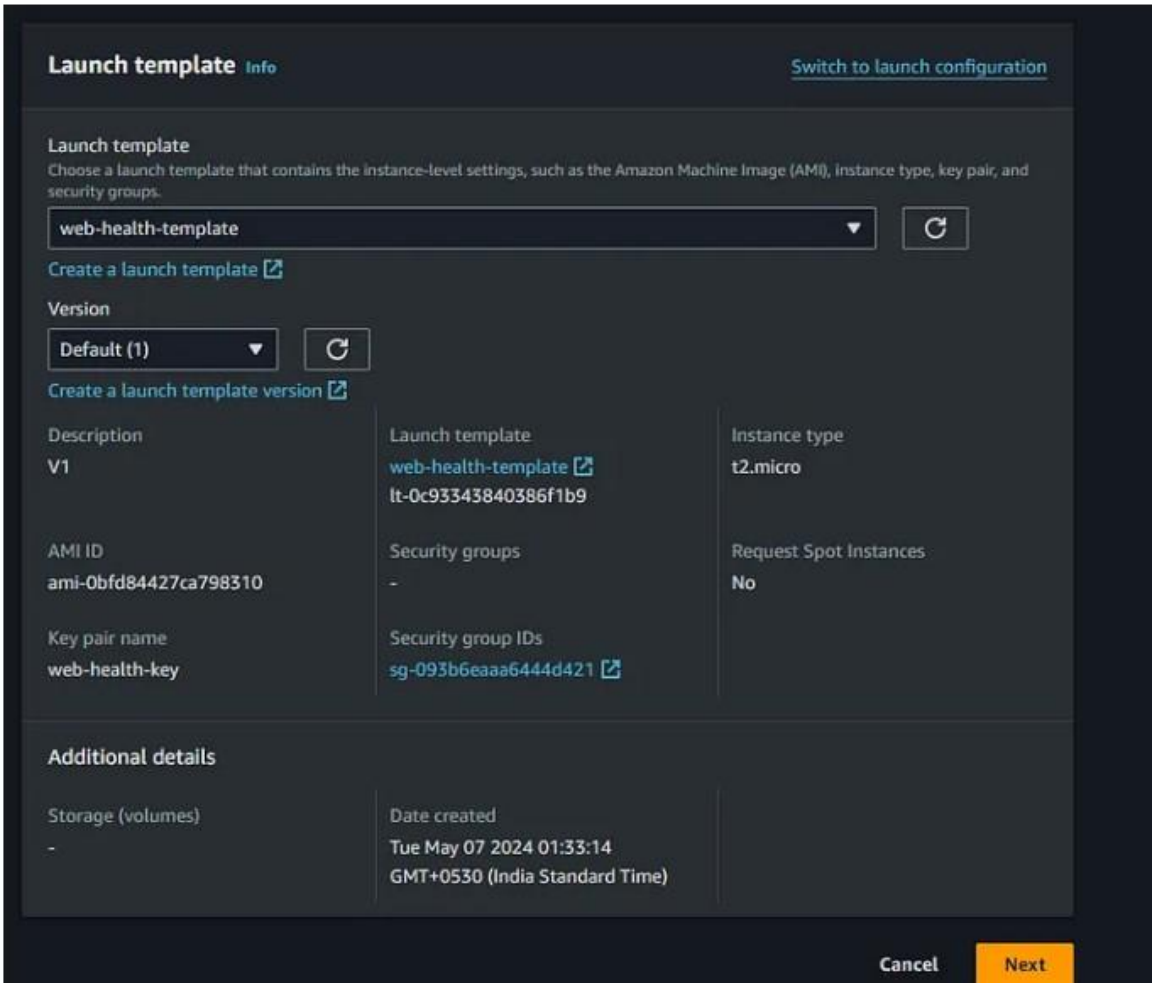


Next, we'll select our launch template for the auto scaling group.

You might also notice the option to use a launch configuration, which serves a similar purpose to a launch template. However, AWS now recommends using launch templates due to their added flexibility.

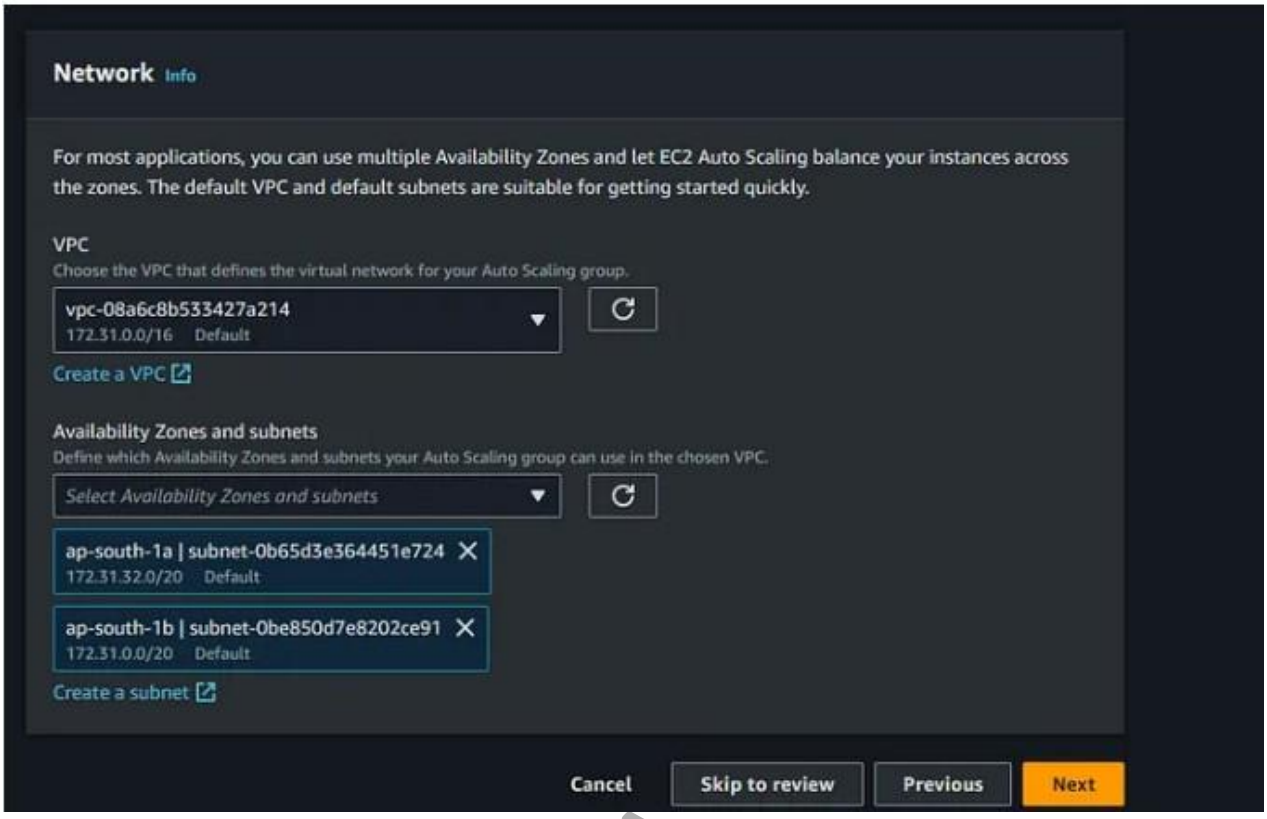
One notable advantage of launch templates is their editability. Unlike launch configurations, which cannot be modified once created, launch templates allow for easy edits and the creation of multiple versions. This versatility enables you to tailor your configurations to different environments or instance types seamlessly.

As we proceed, we'll opt for the launch template, leveraging its benefits for our auto scaling group. You'll notice the option to select different versions of the launch template, providing flexibility in configuration management.



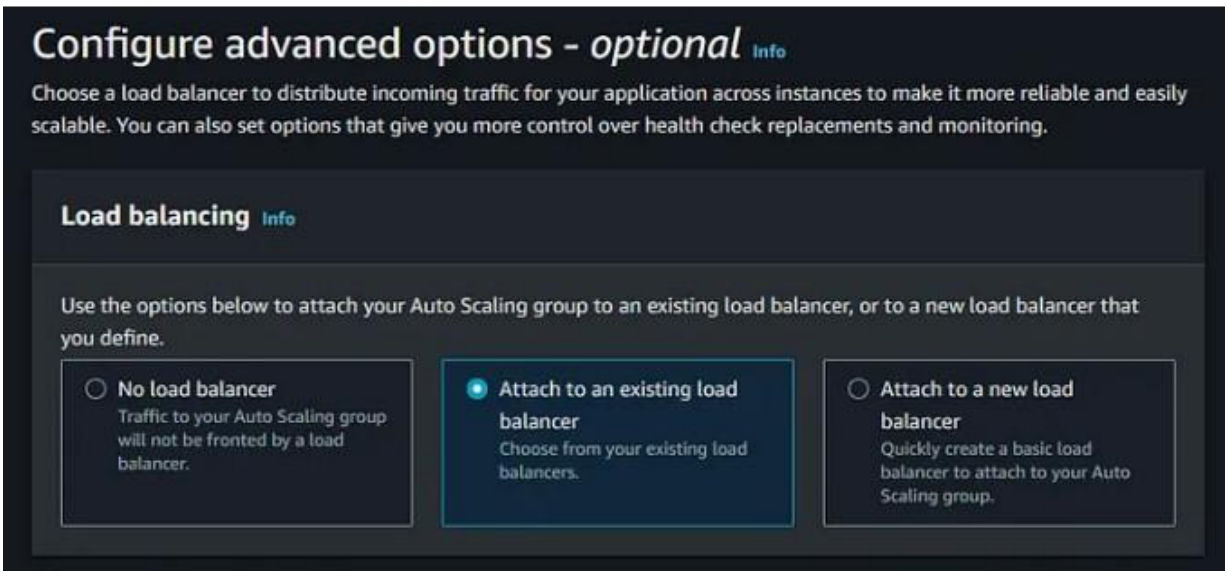
Choose instance launch options:

In this step, we need to specify the availability zones where we want our instances to be launched.
You have the option to select specific zones or allow AWS to choose based on availability. It's generally recommended to choose at least two zones for redundancy and fault tolerance.
Let's proceed by selecting all zones and moving to the next step.

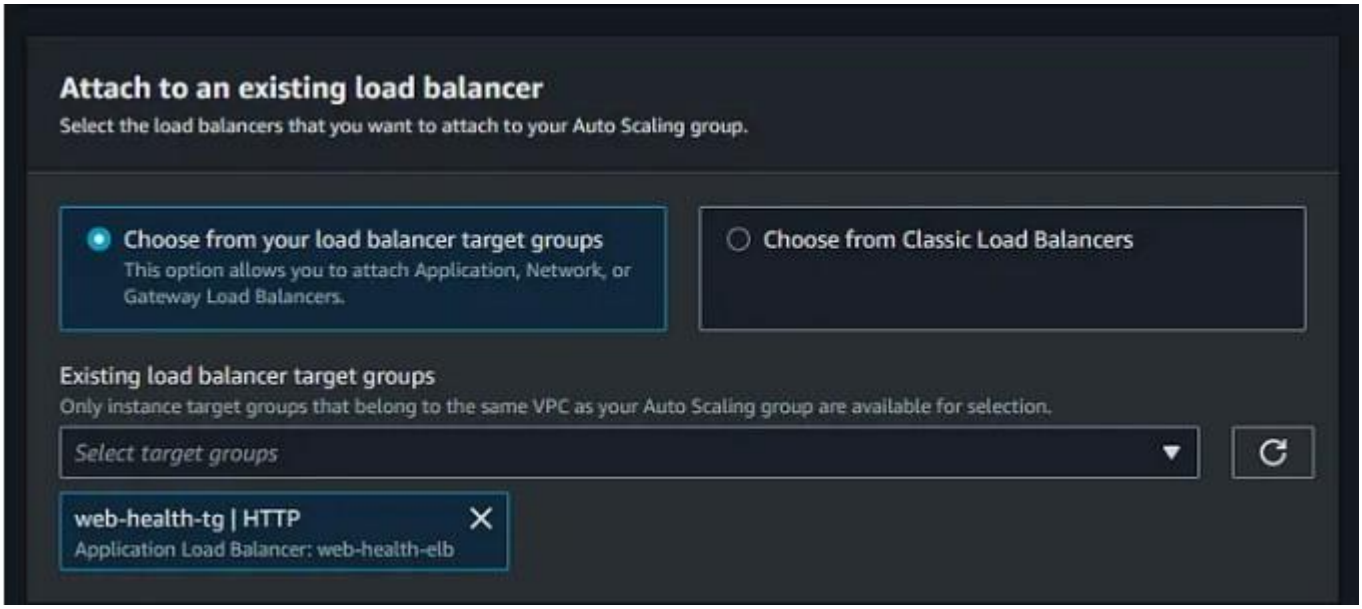


Configure advanced options

In this step, we have the option to attach our auto scaling group to an existing load balancer.
Since our website will benefit from a load balancer, especially with multiple instances, we'll choose to attach it to an existing load balancer.



We'll then select the target group that we previously created. This ensures that the instances launched by our auto scaling group are registered with the specified target group, allowing the load balancer to distribute traffic effectively across these instances.



In this step, we configure the health check settings for our instances within the auto scaling group.

The primary responsibility of the auto scaling group is to maintain the desired number of healthy instances, ensuring optimal performance and reliability.

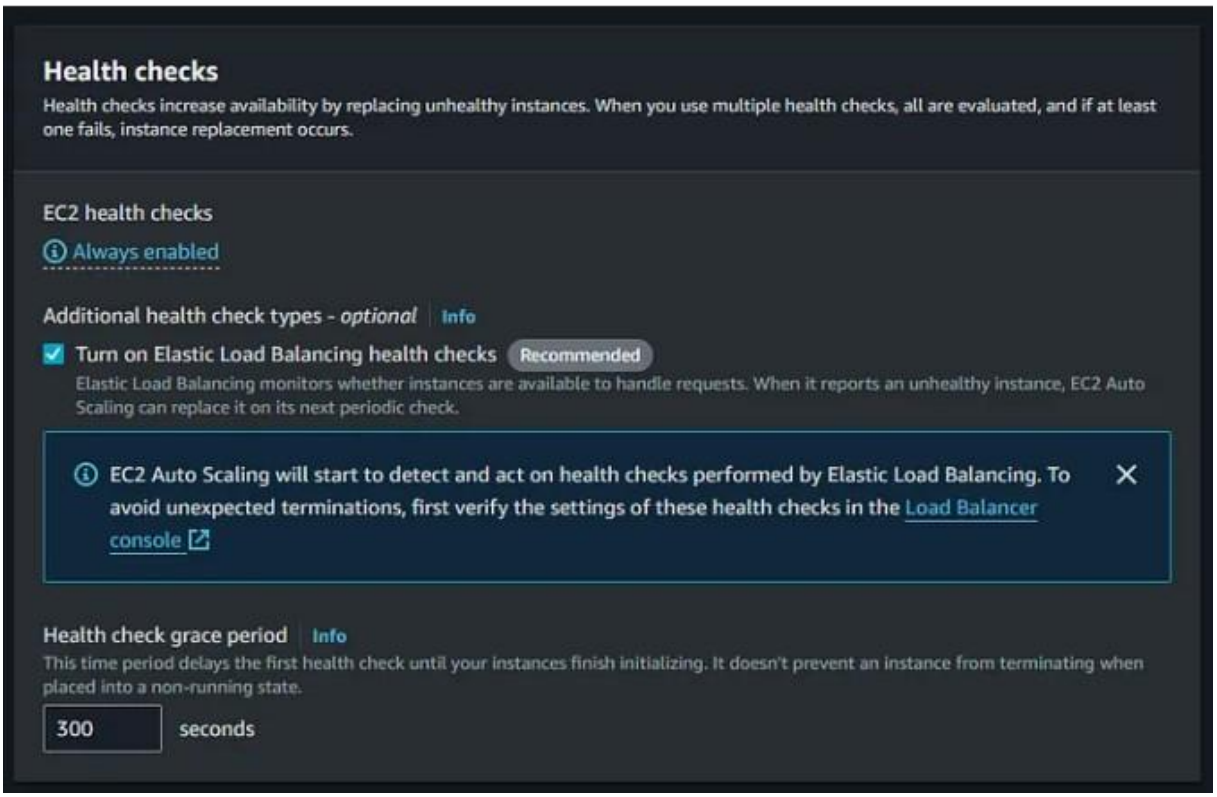
By default, the auto scaling group performs basic EC2 health checks, which include hardware and virtual machine checks. However, these checks only verify the instance's basic functionality and do not guarantee the health of your application or website.

To enable more comprehensive health checks, we opt for ELB (Elastic Load Balancer) health checks. This leverages the target group health check functionality discussed in the load balancer section.

The target group conducts health checks every 300 seconds, verifying the health of instances based on specified criteria such as port availability and process responsiveness. If an instance fails this

health check, indicating a potential issue with the hosted application or service, the target group declares it as unhealthy.

Upon detecting an unhealthy instance, the auto scaling group takes corrective action by terminating the problematic instance and launching a replacement. This ensures that the overall system remains resilient and capable of handling fluctuations in workload and application health effectively.



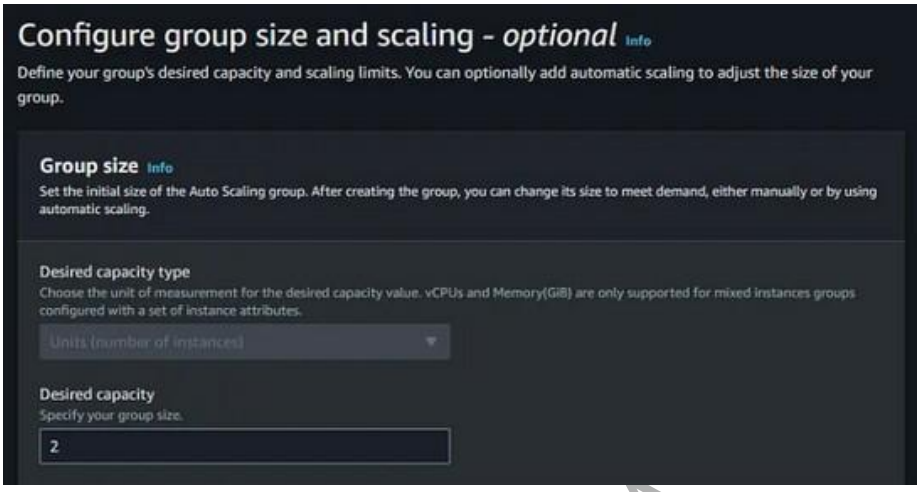
Configure group size and scaling

Now, let's proceed to the next step.
Here, we specify the capacity settings for our auto scaling group. We define the desired, minimum, and maximum number of instances that the group should maintain.

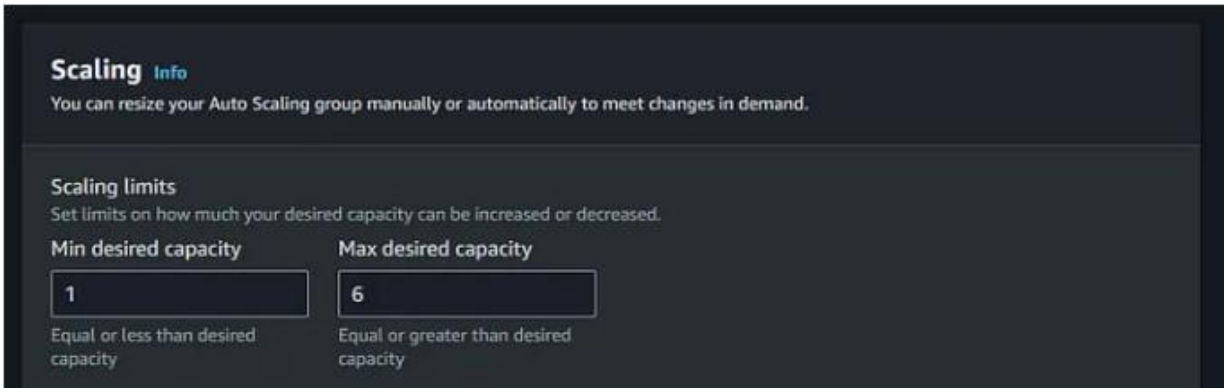
- Desired capacity: This indicates the number of instances we want to have in our auto scaling group. For example, setting it to 2 means we aim to have two instances running at all times.
- Minimum capacity: This represents the minimum number of instances that must be running in the auto scaling group,

regardless of the workload or scaling policies. In our case, we set it to 1 to ensure there is always at least one instance available.

- Maximum capacity: This defines the upper limit on the number of instances that can be launched by the auto scaling group. Even if the workload demands more instances, the group will not exceed this limit. Here, we set it to 8 to cap the maximum number of instances at eight.



Understanding the distinction between desired, minimum, and maximum capacities is crucial. While desired capacity determines the ideal number of instances, minimum capacity ensures a baseline level of availability, and maximum capacity prevents excessive scaling beyond a predetermined threshold.



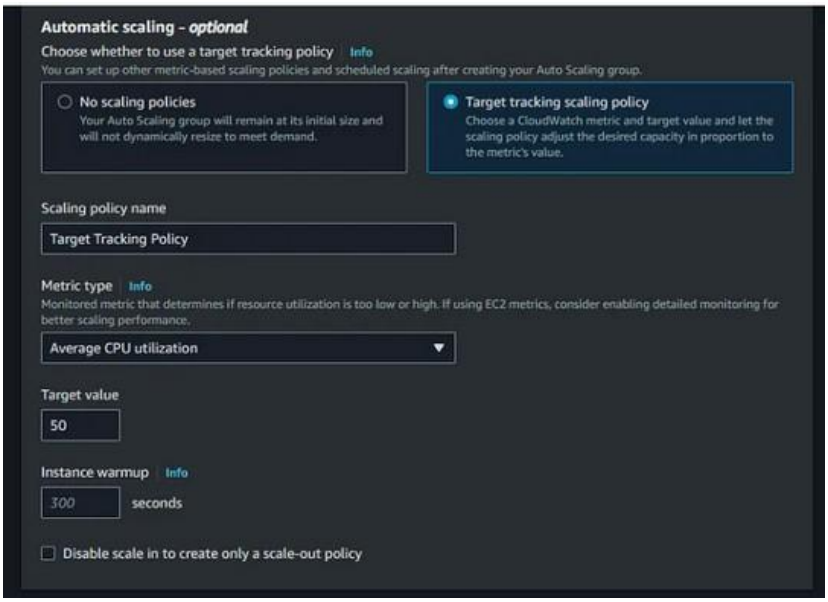
Next, we delve into scaling policies, which dictate how the auto scaling group responds to changes in workload or system metrics.

- Scaling policies: These policies govern when and how the auto scaling group should scale out (add instances) or scale in (remove instances) based on predefined conditions. We can choose from

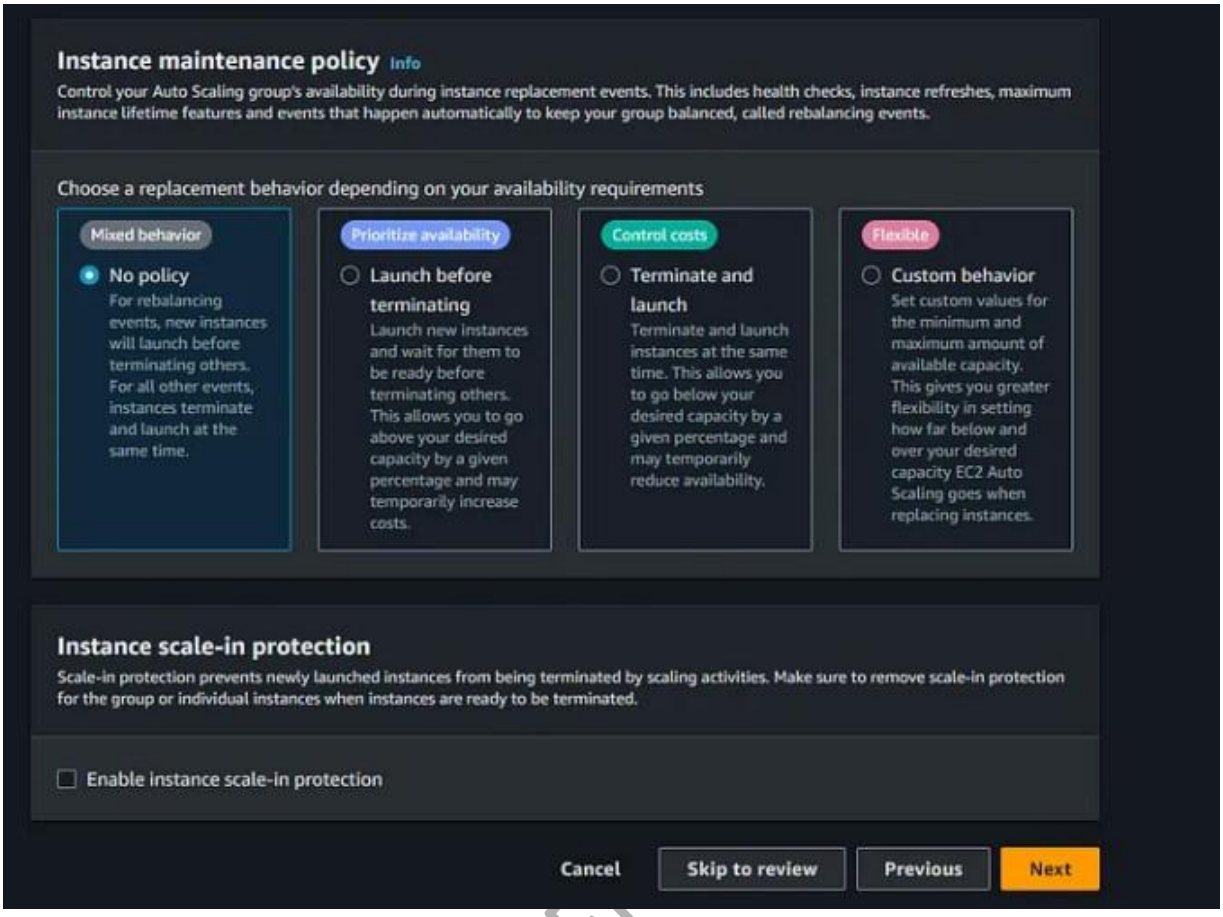
various scaling policy types, such as target tracking scaling policies or simple scaling policies.

For instance, we opt for a target tracking scaling policy based on CPU utilization. If the combined CPU utilization across instances exceeds 50%, the auto scaling group will add instances to handle the increased workload. Conversely, if CPU utilization drops below 50%, instances may be removed to optimize resource usage.

Additionally, we have the option to enable scaling protection, which safeguards newly launched instances from immediate termination. This feature can be beneficial in scenarios where instances require time to initialize or synchronize data.



By configuring these capacity and scaling policy settings, we ensure that our auto scaling group operates efficiently, maintaining optimal performance while dynamically adjusting to changing demand.



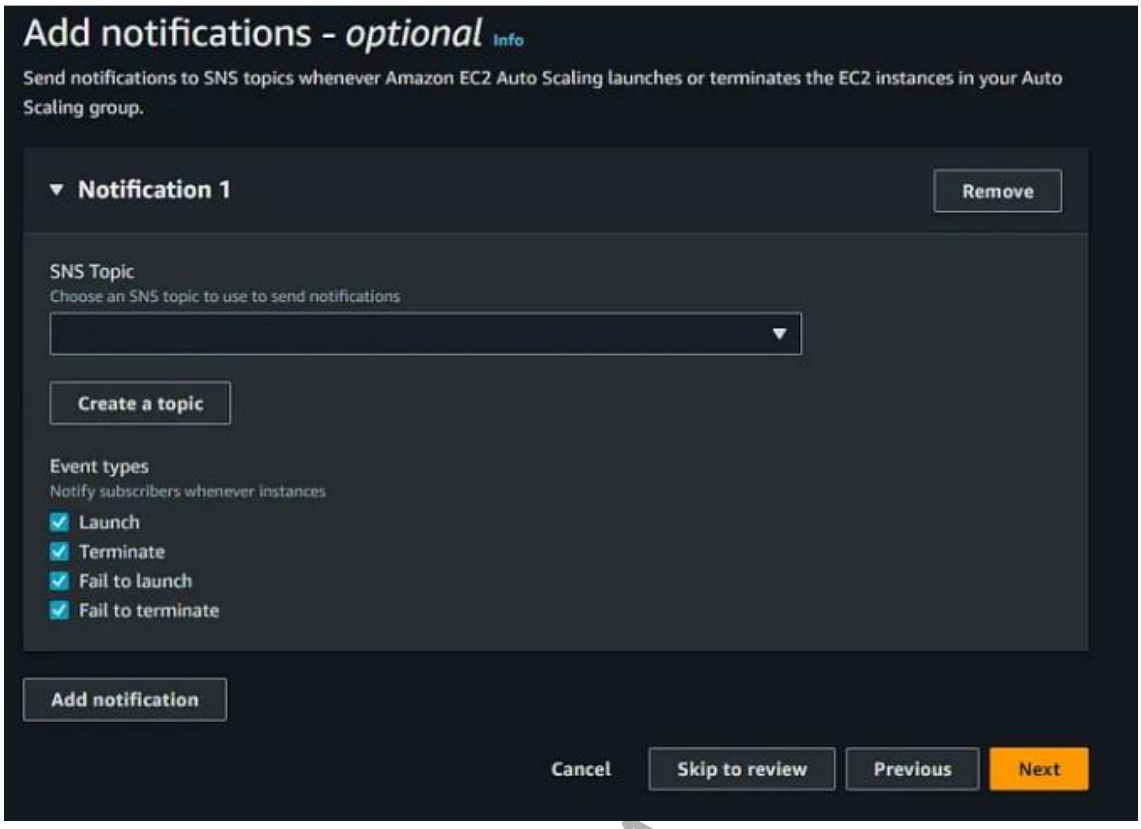
Add notifications

Now, let's proceed to the next step.

Here, we have the option to configure event notifications for various actions performed by the auto scaling group.

- Notification settings: We can specify a notification topic to receive notifications for specific events related to the auto scaling group's lifecycle. These events include instance launches, terminations, failures to launch, and failures to terminate.

By defining a notification topic, we ensure that relevant stakeholders or automated systems are promptly informed of any changes or issues affecting the auto scaling group's operation. This facilitates proactive monitoring and troubleshooting, allowing us to maintain the reliability and availability of our infrastructure.
After configuring the notification settings as needed, we can proceed to the next stage of the auto scaling group setup process.

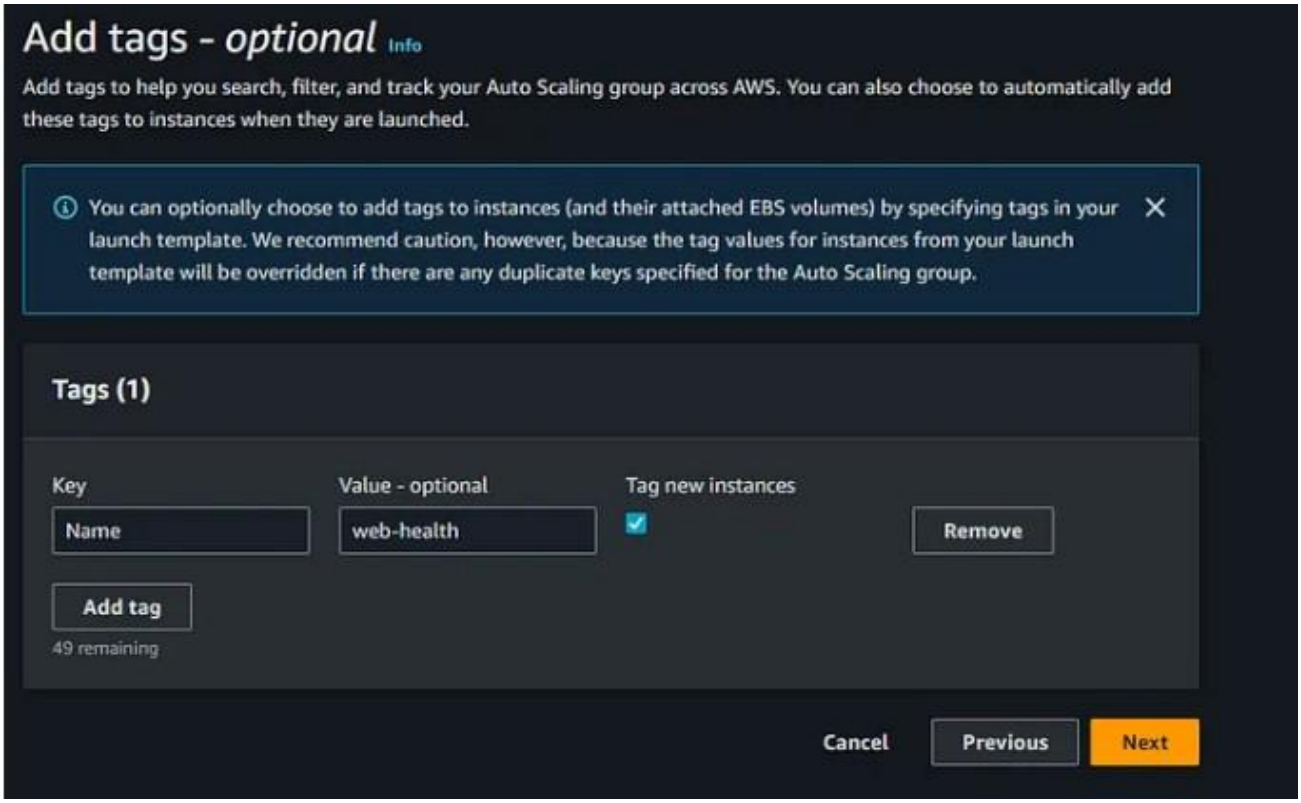


Add tags

Let’s add tags for identification purposes.

- Tag Name: “web-health”

We’ll keep the naming simple and generic since the number of instances can vary dynamically based on scaling policies.

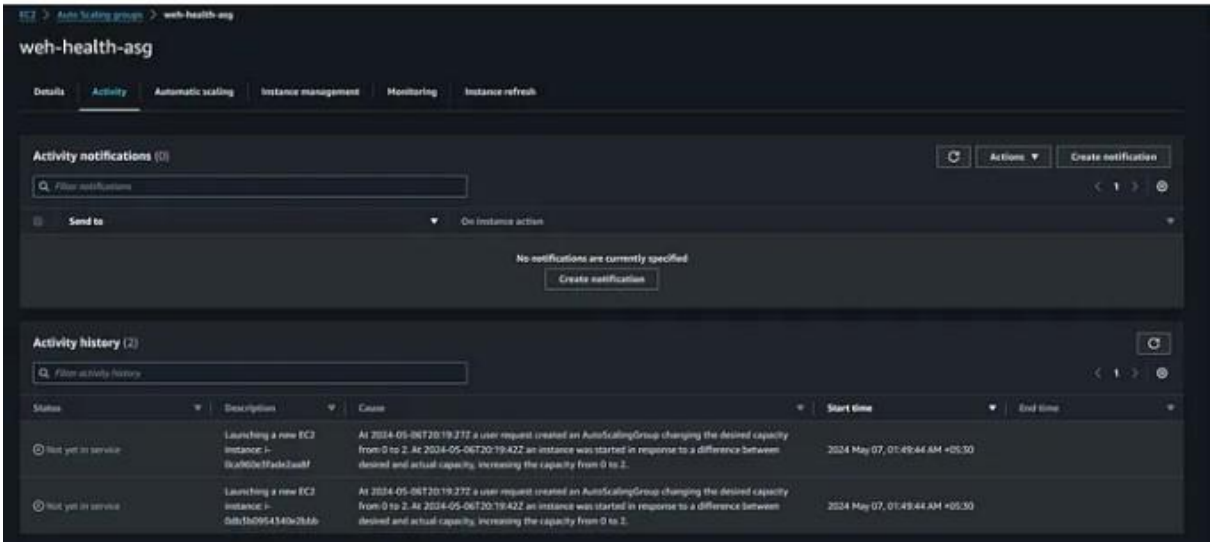


Now, let's proceed by clicking on "Next" and review all the provided information before finalizing the creation of the auto scaling group. Once verified, we can proceed with the creation process.

Currently, we don't have any instances running. As per our configuration, the desired capacity is set to two instances. So, the auto scaling group will start creating instances to meet this desired capacity.

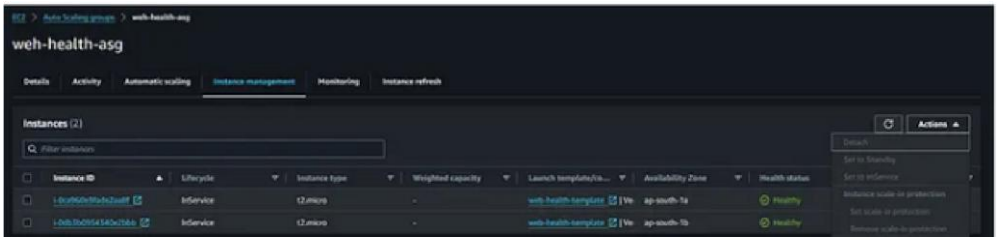
You can observe that two instances are being created to fulfill the desired capacity. Additionally, there's a scaling policy in place. If the CPU utilization drops below 50%, it will terminate instances accordingly. However, it will always maintain a minimum of one instance.

In the instance management section, you'll find all the instances within the auto scaling group. This section is particularly useful for troubleshooting purposes. Here, you can perform various actions such as detaching an instance from the auto scaling group, setting it to standby mode, or bringing it back into service.



You have the option to set scale-in protection for instances. This means that these instances won't be terminated by the auto-scaling process. This feature is particularly useful during troubleshooting or maintenance activities when you don't want instances to be terminated unexpectedly.

We'll wait for a few minutes to ensure that all instances are in a healthy state. Once they are marked as healthy, we'll explore some additional options. Currently, both instances are running and healthy. You should also see them updated in the target group.

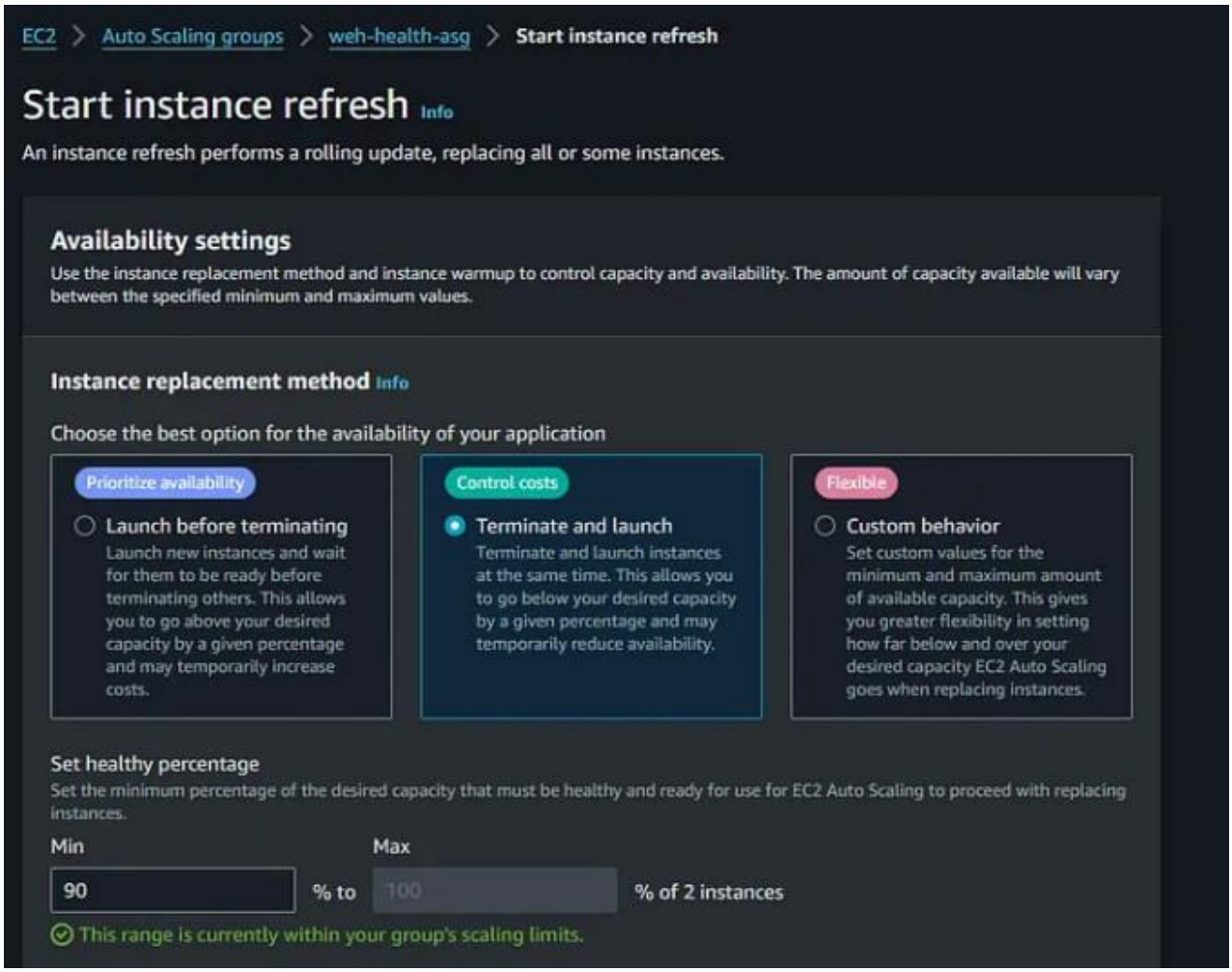


When managing instances within an auto-scaling group, it's crucial to remember that these instances are dynamic. They are automatically created and deleted by the autoscaling group based on the defined policies and metrics. Therefore, manual changes to instances should be avoided to prevent unexpected behavior.

Any changes to instances should be made through the launch template. This includes changes such as instance type, security group, application upgrades, or software installations. If you need to

make changes, create a new launch template or a new version of the existing launch template with the desired configurations.

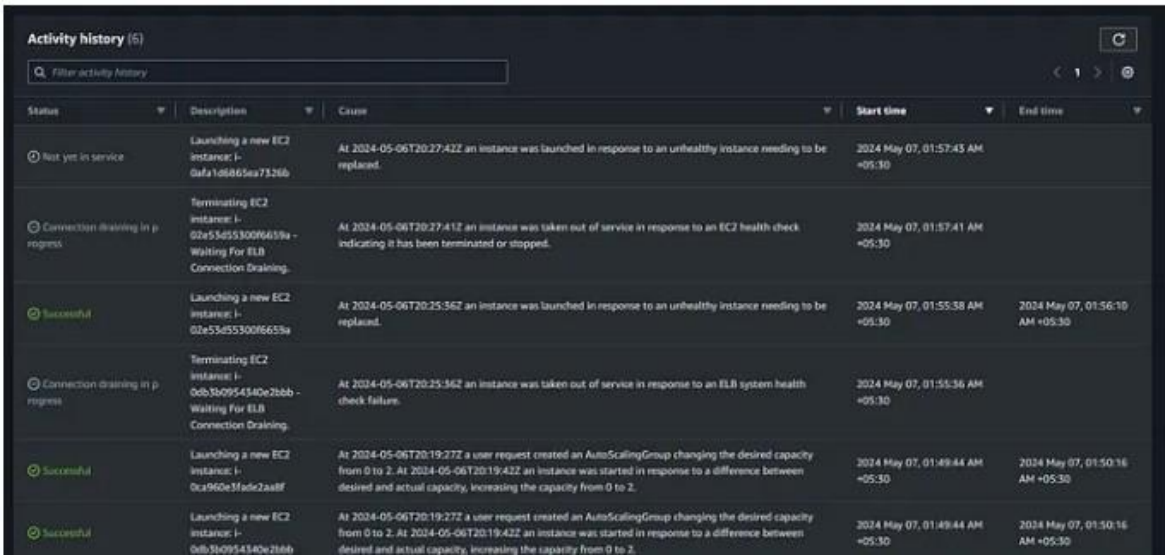
To apply changes to instances, you can edit the auto-scaling group and select the new launch template. Once the changes are saved, you must initiate an instance refresh to apply the changes to existing instances. During instance refresh, instances are terminated and recreated based on the updated launch template.



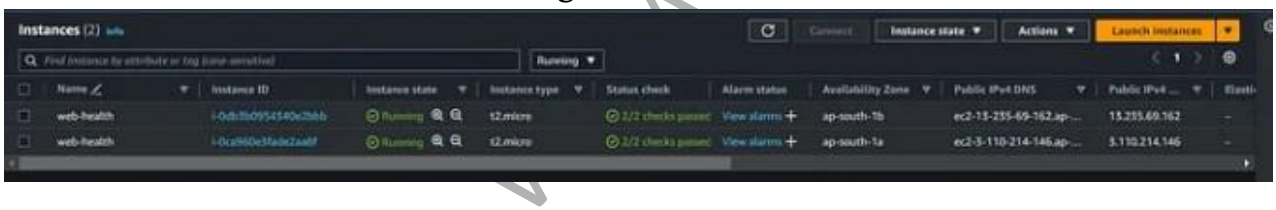
When initiating an instance refresh, ensure that you specify a minimum healthy percentage. This ensures that a certain percentage of instances remain healthy during the refresh process, preventing disruptions to your application. It's recommended to set a higher healthy percentage to maintain application stability.
Once changes are applied and instance refresh is initiated, monitor the auto-scaling group activity to ensure that new instances are launched successfully and the application remains available. Avoid

manual termination of instances unless necessary for troubleshooting or maintenance purposes.

Please note — instance: i-02e53d55300f6659a I stopped purposefully.



We can see 2 instances are running:



By following these best practices, you can effectively manage instances within your auto-scaling group while ensuring the reliability and scalability of your application.

In conclusion, auto-scaling groups in AWS offer a powerful solution for managing the scalability and availability of your application infrastructure. By dynamically adjusting the number of EC2 instances based on predefined policies and metrics, auto-scaling groups enable you to efficiently handle fluctuating traffic loads while maintaining optimal performance and cost-effectiveness.
Key points to remember include:

1. Dynamic Nature: Auto-scaling groups automatically create and terminate EC2 instances based on configured scaling policies,

ensuring that your application can handle varying levels of demand without manual intervention.

2. Launch Templates: Changes to instance configurations should be made through launch templates, allowing for easy management of instance settings such as instance type, security groups, and software configurations.
3. Instance Refresh: When updating configurations, initiate an instance refresh to apply changes to existing instances. Specify a minimum healthy percentage to ensure application stability during the refresh process.
4. Avoid Manual Changes: Manual changes to instances within auto-scaling groups should be avoided to prevent conflicts with scaling policies and unexpected behavior. Instead, utilize launch templates for any necessary updates.
5. Monitoring and Maintenance: Regularly monitor the auto-scaling group activity to ensure that instances are being created and terminated as expected. Conduct troubleshooting and maintenance activities carefully to avoid disruptions to your application.

Amazon S3

Amazon S3 (Simple Storage Service):

Amazon S3 is a scalable object storage service provided by AWS. It allows users to store and retrive any amount of data from anywhere on the internet. S3 is widely used for its durability, scalability, and ease of intergration with other AWS services.

How S3 Works:

1.Object-based Storage: S3 stores data as objects, which include the data itself, metadata, and a unique identifier. These objects are stored in "buckets."

2. Bucket Structure: A bucket is a container for objects. Users can upload objects to buckets, and each object is identified by a unique key.

3. Global Access: S3 allows data access over the internet using API calls, SDKs, or the AWS Management Console.

4. High Durability & Availability: S3 is designed for 99.999999999% (11 nines) durability, with data automatically replicated across multiple availability zones.

Use Cases of S3:

1.Data Storage & Backup: Regular backups for applications, systerns, and databases.

2. Big Data & Analytics: Storing large datasets for analysis and processing.

3. Website Hosting: Hosting static websites (HTML, CSS, JavaScript).

4. Content Distribution: Storing media files for distribution (e.g., videos, software).

5. Disaster Recovery: Storing critical data for quick recovery

Difference between EBS and S3:

EBS (Elastic Block Store):

- Type: Block storage for EC2 instances.
- Use Case: Low-latency storage for databases, file • •

- Attachment: Only attached to one EC2 instance at a time.
- Type: Object storage for unstructured data (images, backups).
- Use Case: Highly scalable and accessible from anywhere.
- Attachment: Not tied to any single instance, can be accessed globally

Cold Data vs. Hot Data:

- Hot Data: Frequently accessed data requiring lowlatenc storage (e.g., S3 Standard or EBS).
- Cold Data: Rarely accessed data stored in lower-cost solutions (e.g., S3 Glacier) for long-term retention.

Object Storage Class vs. Block Storage Class:

- Object Storage (S3): Stores data as objects with unique identifiers, suited for unstructured data (documents, media).
- Block Storage (EBS): Stores data in blocks, similar to a hard drive, designed for structured data (databases, file systems).

S3 Storage Classes:

1.S3 Standard: General-purpose storage for frequently accessed data. All Bookmarks

Try Premium for 10

2.S3 Intelligent-Tiering: Automatically moves data between frequent and infrequent access tiers to save costs.

3.S3 Standard-IA (Infrequent Access): Lower cost for infrequently accessed data.

4.S3 One Zone-IA: Similar to Standard-IA but stores data in a single availability zone.

5.S3 Glacier: Low-cost archival storage for data rarely accessed.

6.S3 Glacier Deep Archive: Lowest-cost storage for long-term, rarely accessed data.

Advantages of S3:

- Scalability: Automatically scales to accommodate any amount of data.
- Durability: 99.999999999% durability with automatic replication across multiple locations.
- Security: Supports encryption, access control policies, and integrates with AWS Identity and Access Management (IAM).
- Cost-Effective: Pay for what you use, with multiple storage classes to optimize cost based on data access frequency.

Disadvantages of S3:

- Access Latency: Not suitable for low-latency, high-

Performance applications like databases (use EBS for that).

- Complexity in Management: Managing large

Datasets or setting up lifecycle policies may become complex for certain use cases.

- Data Retrieval Costs: Higher retrieval costs in certain storage classes like S3 Glacier or Glacier Deep Archive.

Features of S3:

1. Versioning: Tracks changes to objects, allowing you Of revert to previous versions.
2. Lifecycle Management: Automates the movement of Data to lower-cost storage classes.

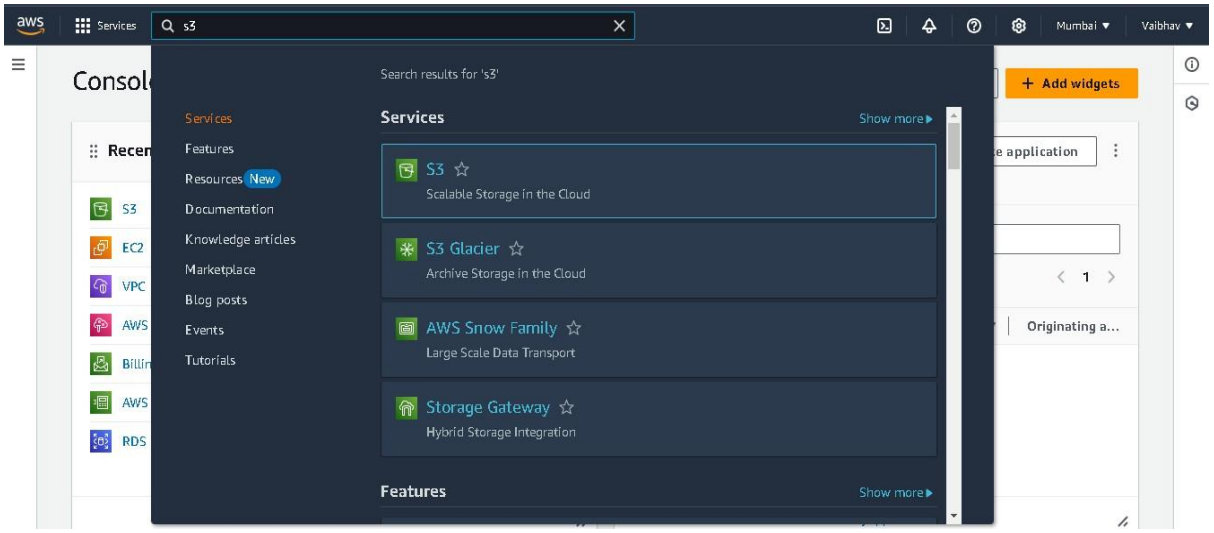
3.Cross-Region Replication: Automatically replicates Data across AWS regions for better durability.

4. Encryption: Supports server-side and client-side Encryption for securing data.
5. Access Control: Fine-grained control over who can Access or modify data using IAM policies, bucket Policies, and access control lists (ACLs).

Step-by-Step Guide for Uploading a File to Amazon S3:

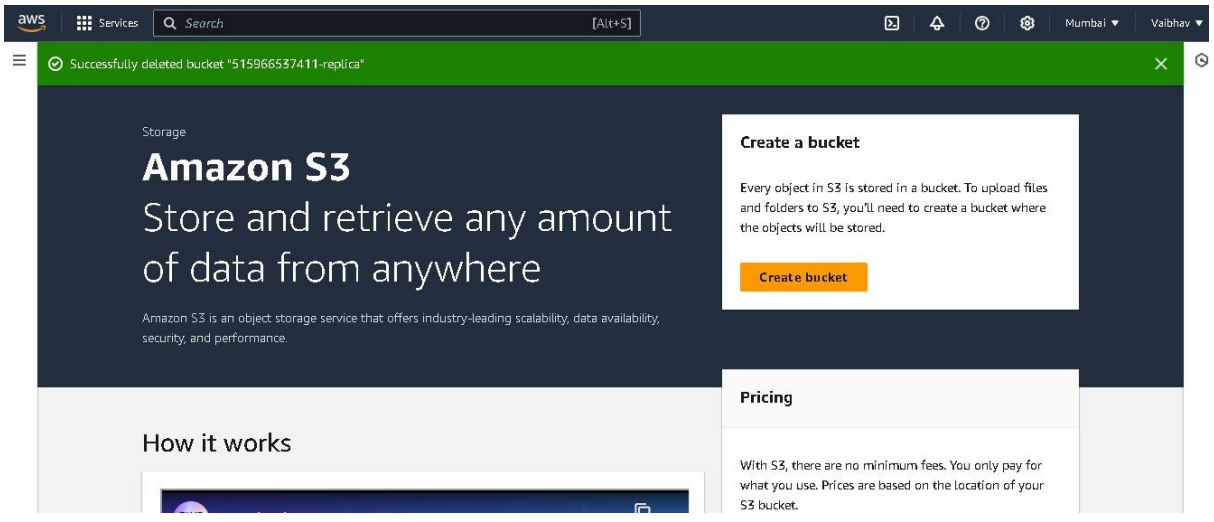
Step 1: Log in to the AWS Management console

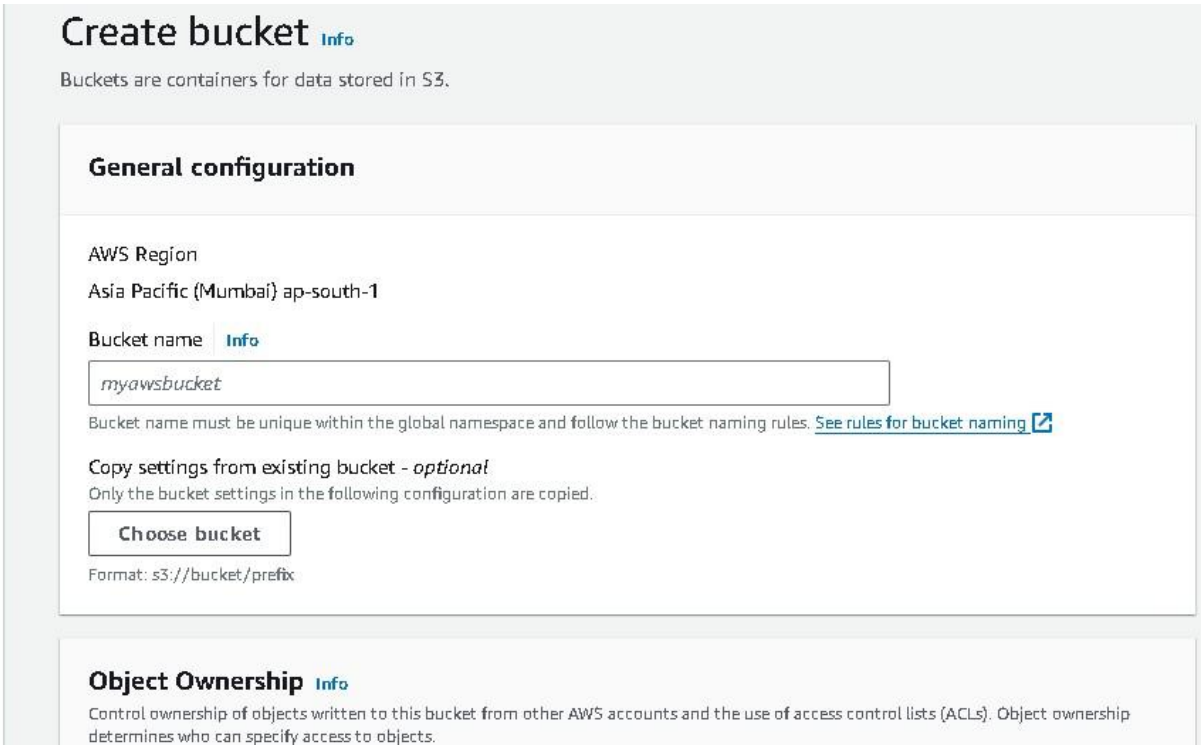
1. Go to the AWS Management Console and sing in with your credentials.
2. In the search bar, type S3 and click on the Amazon S3 service.



Step 2: Create a New S3 Bucket

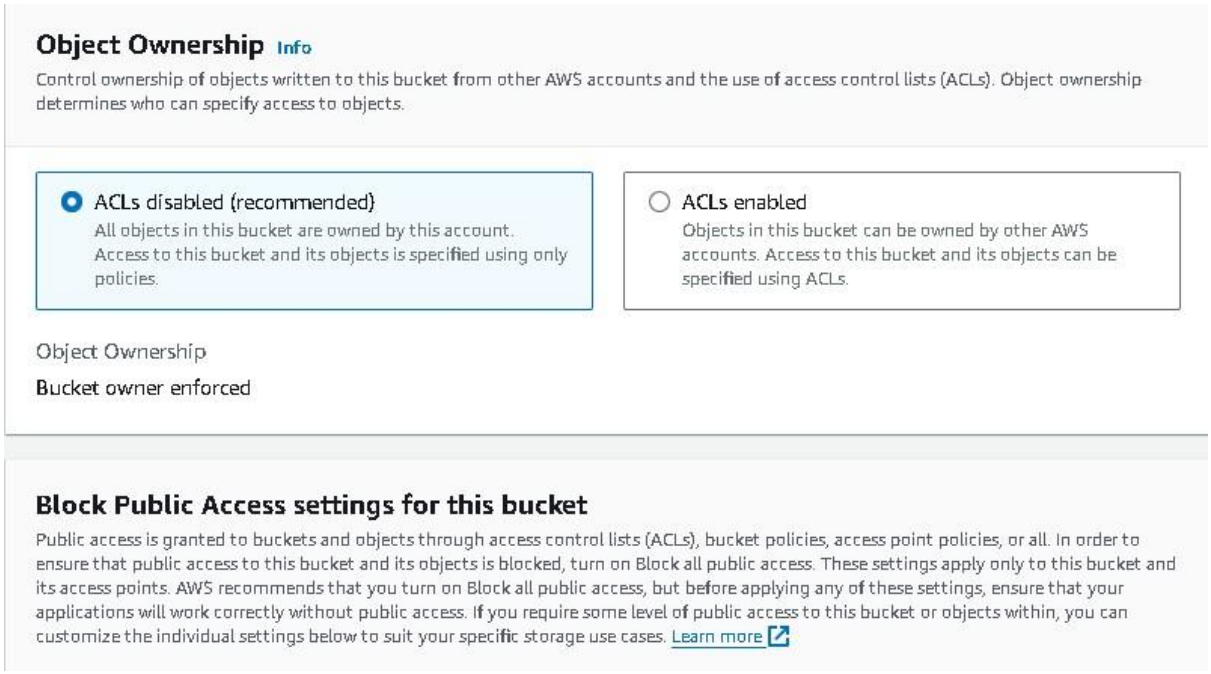
1. Once in the S3 dashboard, click Create Bucket.
2. Enter a Bucket Name (it must be unique globally across all AWS users)

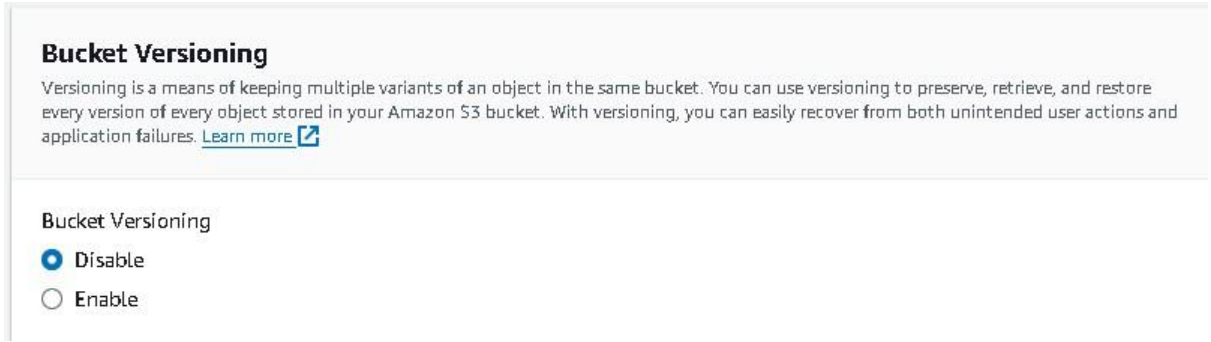


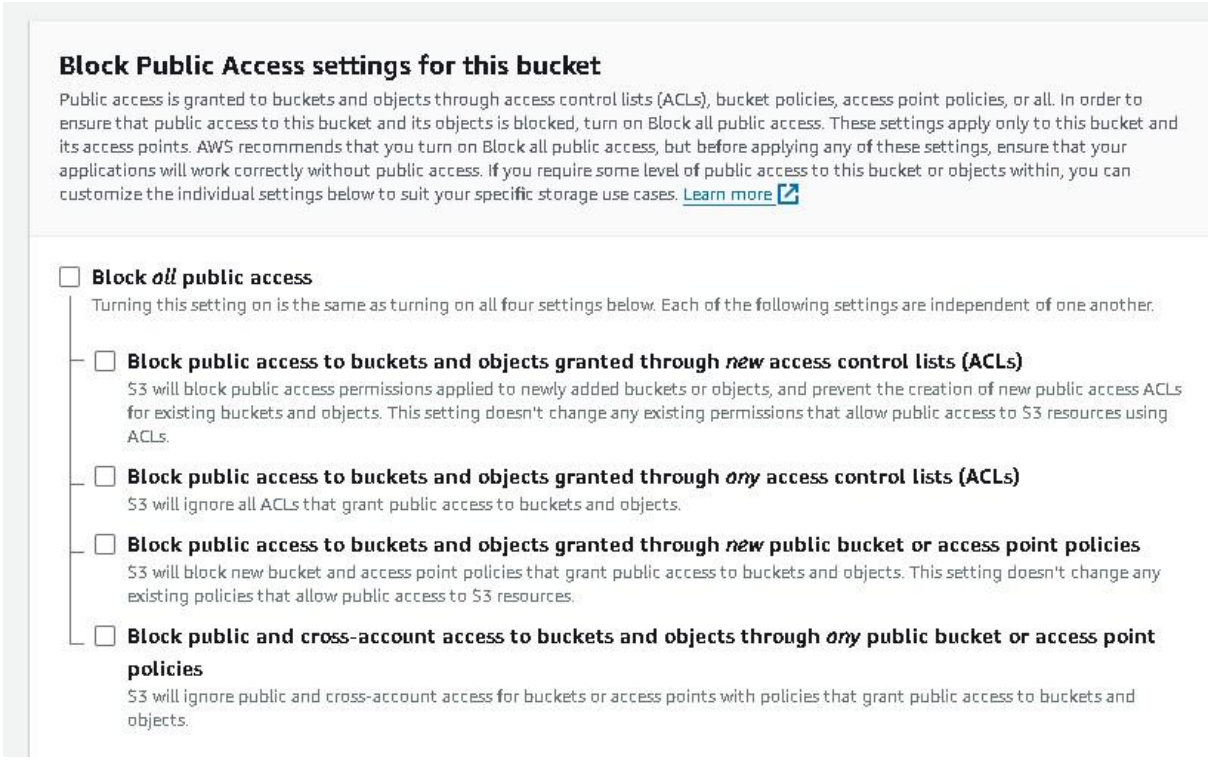


3. Select the AWS Region where you want to store your data (choose a region closest to your location for better performance.)
4. Configure bucket settings.

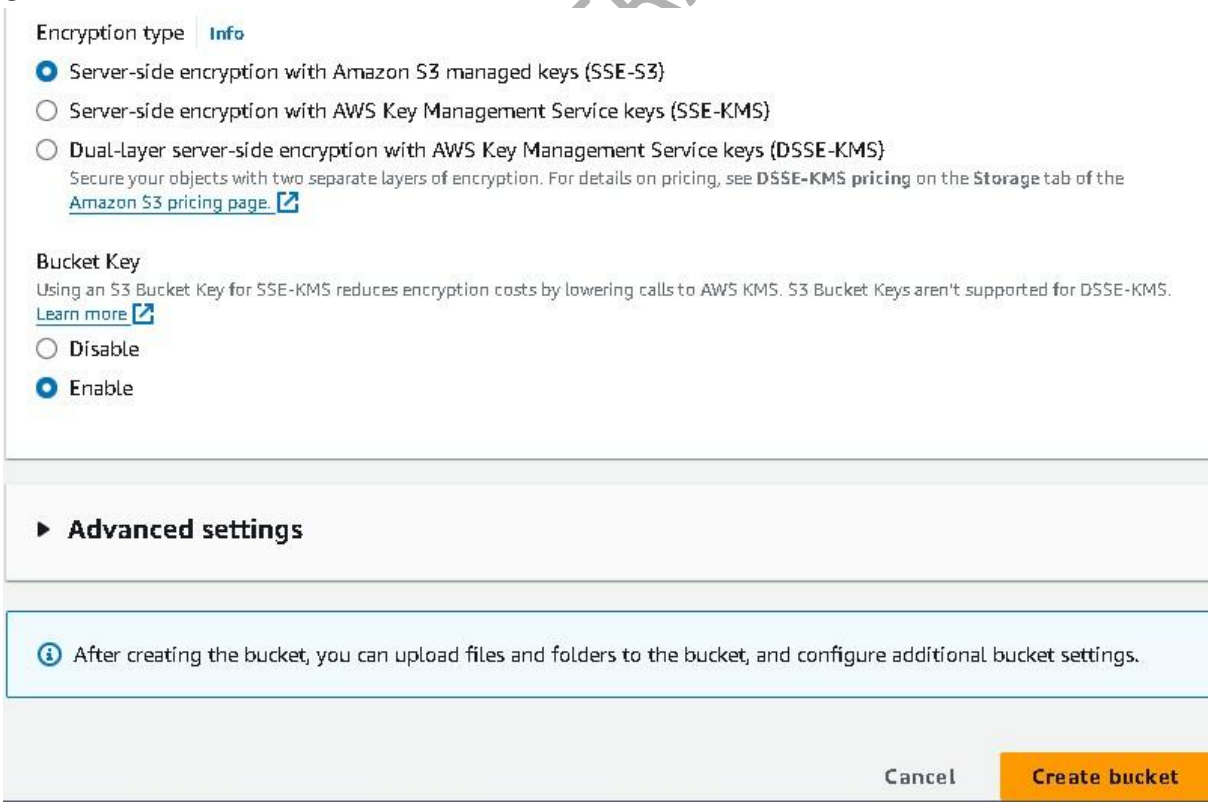
- Block Public Access: You can keep public access blocked unless you want to make your files accessible to the public.
- Bucket Versioning: You can enable or disable versioning. This feature keeps multiple version of an object.


Object Ownership Info
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.
ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.
Object Ownership
Bucket owner enforced
Block Public Access settings for this bucket
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more


Bucket Versioning
Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more
Bucket Versioning
Disable
Enable

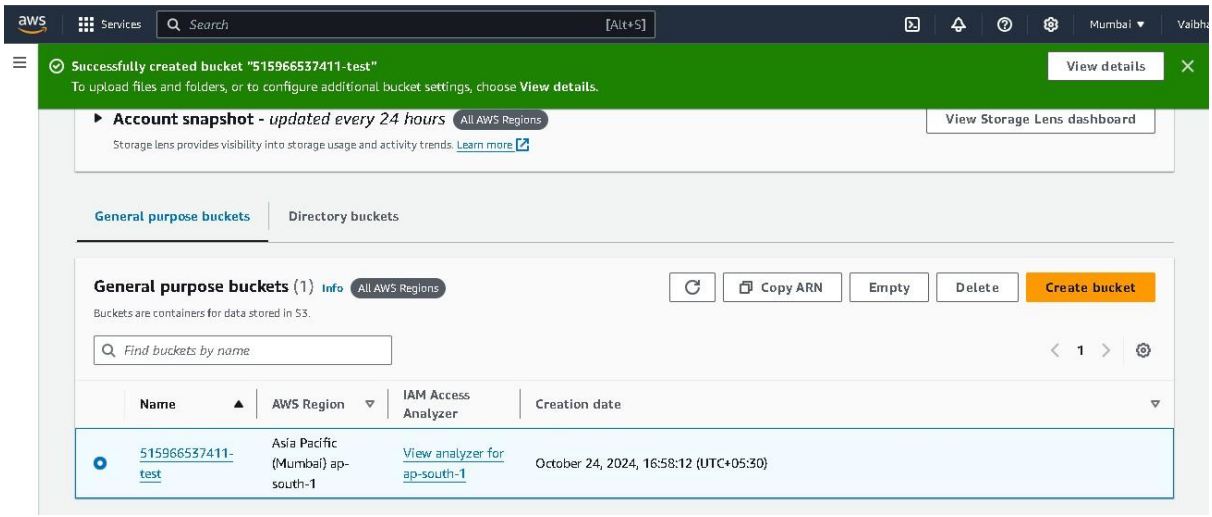


5. Scroll down and click Create Bucket.

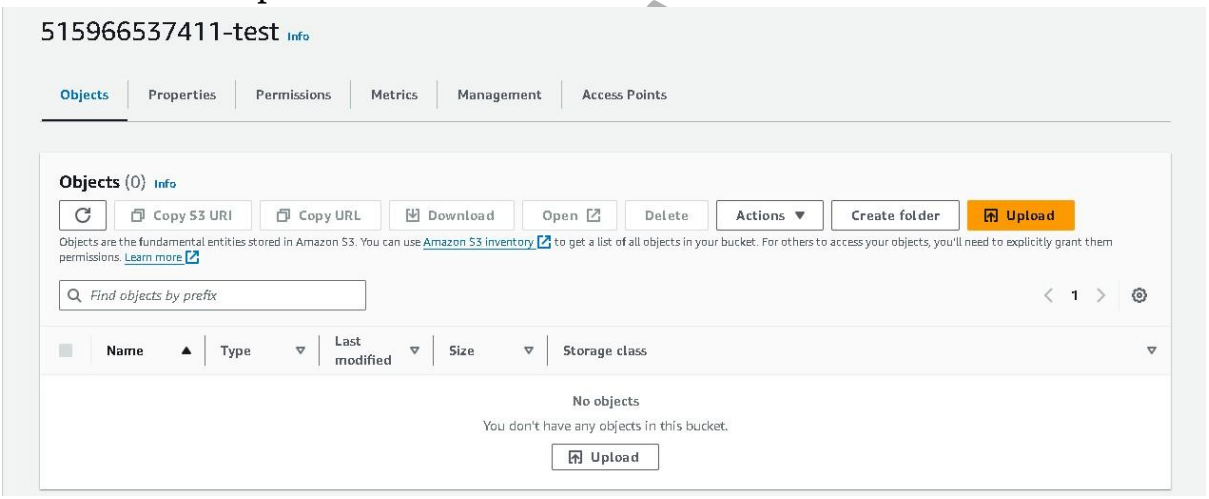


Step 3: Upload a File to the S3 bucket

1. Once the bucket is created, click on your bucket's name to open it.



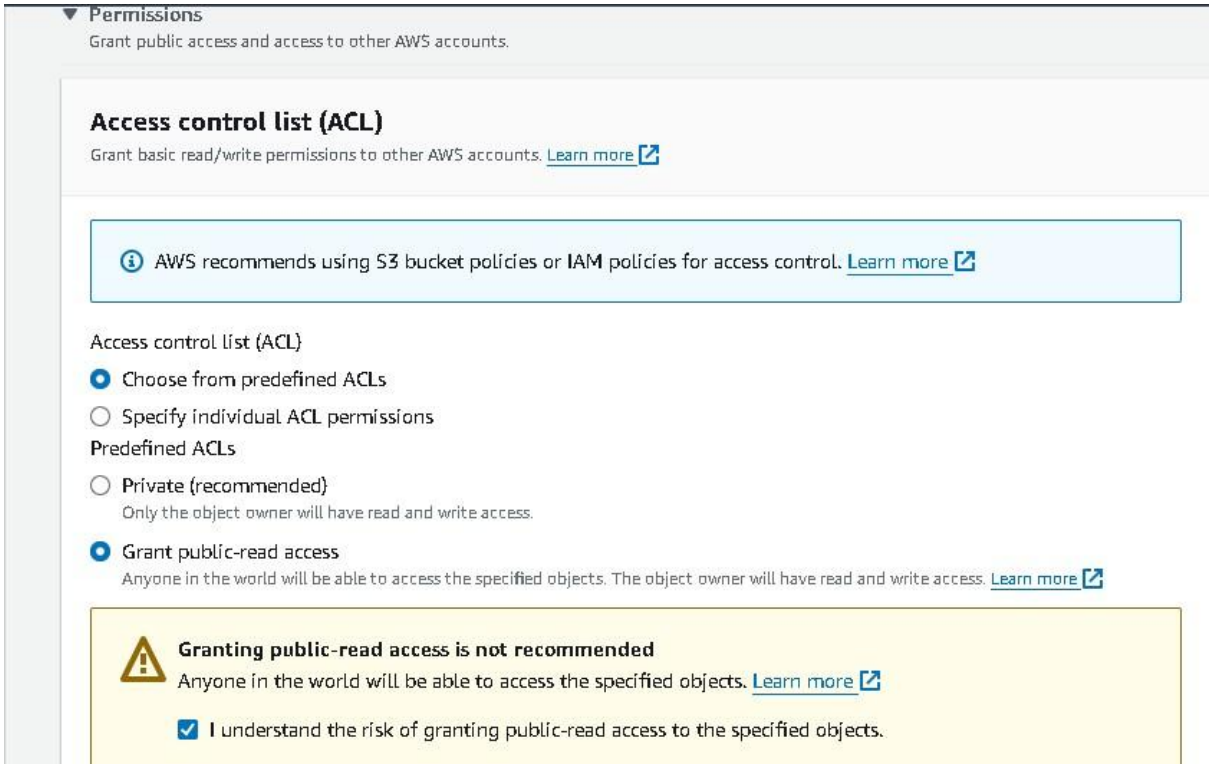
2. Click upload on the bucket's dashboard.

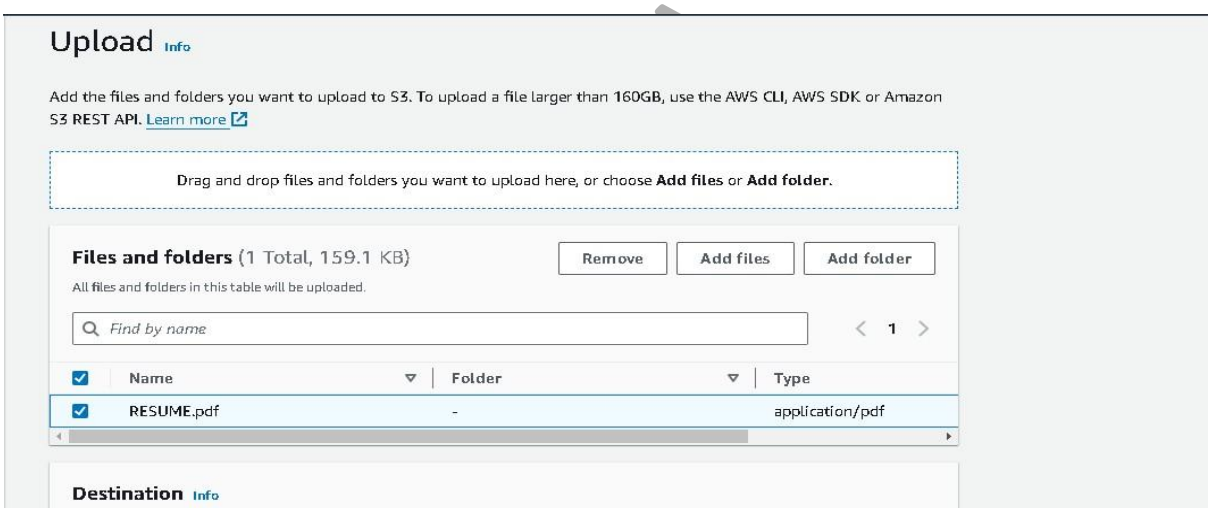


3. In the Upload window, either drag and drop a file from your local machine or click add files to browse for the file you want to upload.

4. After adding the file, you can configure permission:

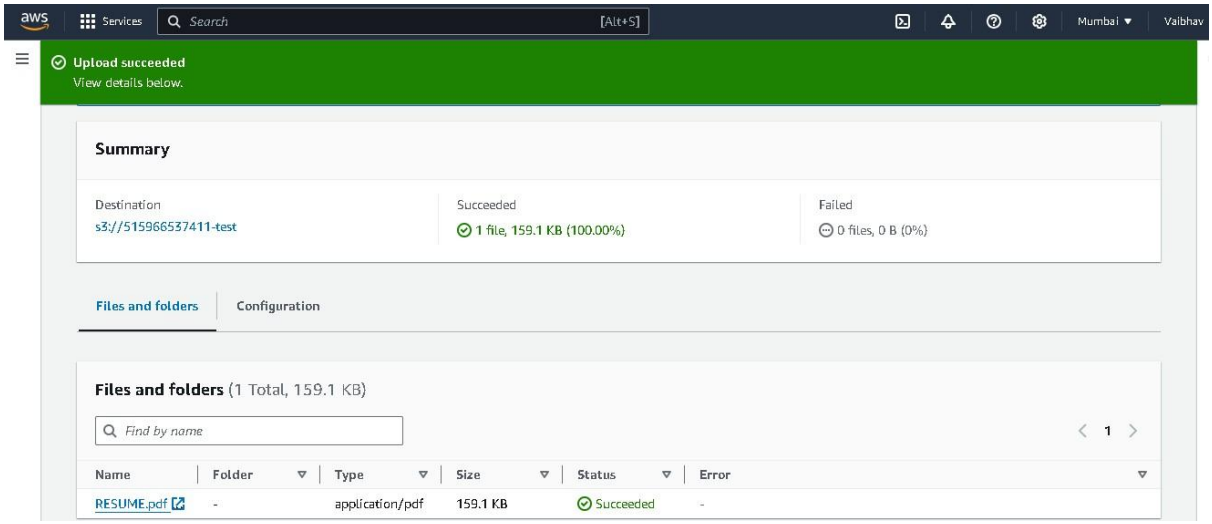
• Manage public permissions: Decide if the file Should be public or private.





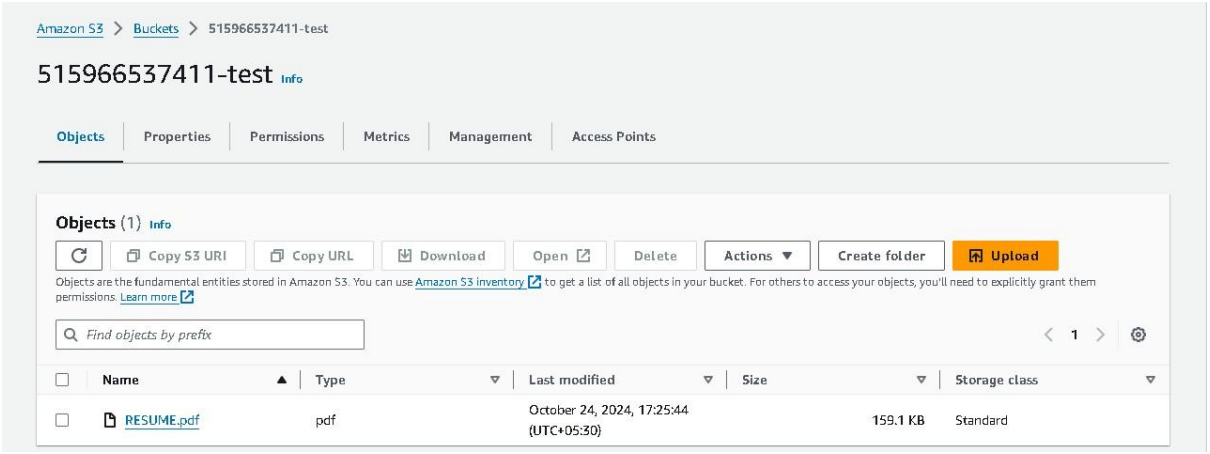
5. You can leave storage class settings as default (S3 Standard), or choose other storage classes like Glacier for archival.

6. Once you've set the file and permissions, click Upload.



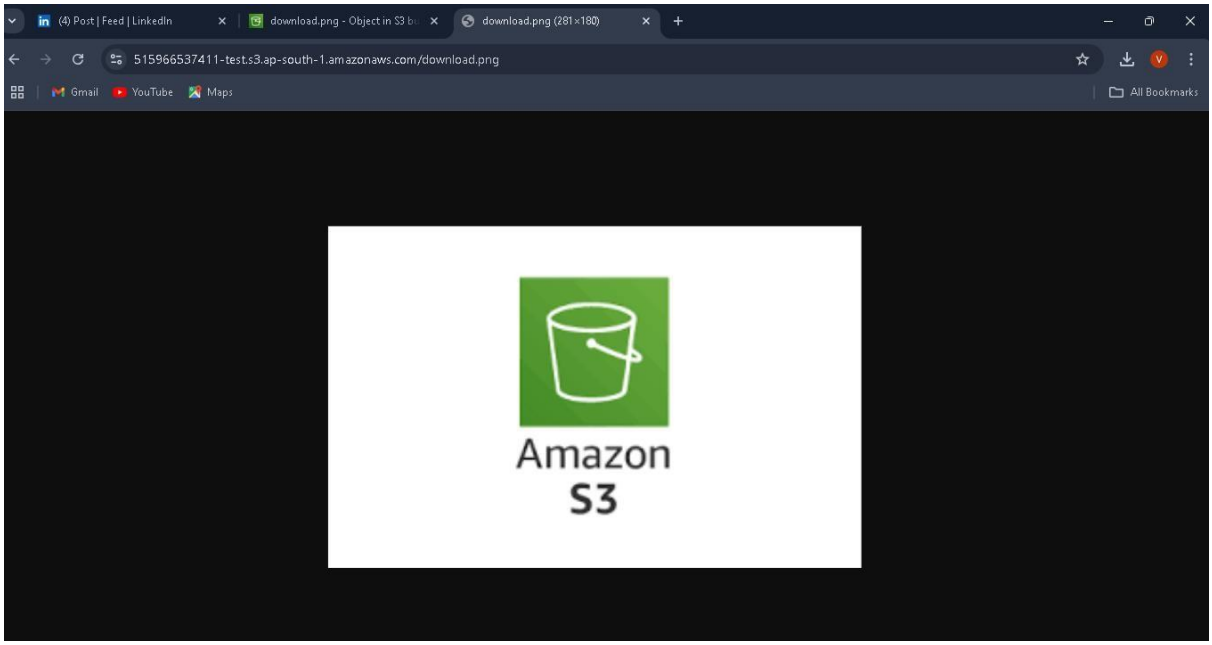
Step 4: Verify the Upload

1. Once the file is uploaded, you will see it in the bucket. You can click on the file name to view its properties.



2. The file has a URL that you can use to access it (based on permissions). The URL format will look like this:

https://<bucket-name>.s3.<region>.amazo naws.com/<file-name>



Step 5: Access and Manage Your File

- After uploading, you can manage your file's permissions, move it to different storage classes (like Glacier), or delete it.
- If you want to download the file from the bucket, You can either use the AWS CLI or the S3 console.

Additional Poinsta:

- Public Access: If you want the file to be publicly accessible, make sure to set the object's permissions accordingly.
- Storage Classes: By default, objects are uploaded to the S3 Standard storage class. You can later change it to Glacier or Intelligent-Tiering, depending on your use case.

Reminder: Delete S3 Bucket After Completing Your Work

Once you've completed your tasks and no longer need the S3 bucket, make sure to delete it to avoid unnecessary storage costs. Deleting the bucket will permanently remove all data, so ensure you have saved any important files. Always clean up unused resources after finishing your work.

With these steps, you can successfully upload files from your local machine to Amazon S3!

AWS IAM (Identity and Access Management Service)

AWS IAM is used to securely control individual and group access to AWS resources.
IAM makes it easy to provide multiple users secure access to AWS resources.
IAM can be used to manage:

- Users.
- Groups.
- Access policies.
- Roles.
- User credentials.
- User password policies.
- Multi-factor authentication (MFA).
- API keys for programmatic access (CLI).

Provides centralized control of your AWS account.
Enables shared access to your AWS account.
By default, new users are created with NO access to any AWS services – they can only login to the AWS console.
Permission must be explicitly granted to allow a user to access an AWS service.
IAM users are individuals who have been granted access to an AWS account.
Each IAM user has three main components:

- A username.
- A password.
- Permissions to access various resources.

You can apply granular permissions with IAM.
You can assign users individual security credentials such as access keys, passwords, and multi-factor authentication devices.
IAM is not used for application-level authentication.
Identity Federation (including AD, Facebook etc). can be configured allowing secure access to resources in an AWS account without creating an IAM user account.
Multi-factor authentication (MFA) can be enabled/enforced for the AWS account and for individual users under the account.
MFA uses an authentication device that continually generates random, six-digit, single-use authentication codes.
You can authenticate using an MFA device in the following three ways:

- Through the AWS Management Console – the user is prompted for a user name, password, and authentication code.
- Using the AWS API – restrictions are added to IAM policies and developers can request temporary security credentials and pass MFA parameters in their AWS STS API requests.
- Using the AWS CLI by obtaining temporary security credentials from STS (aws sts getsession-token).

Want to see how to setup MFA? In the brief AWS Hands-on Labs video tutorial below, you'll learn how to activate a virtual Multi-factor Authentication (MFA) for your AWS Root Account. In under 5 minutes, we cover: Deleting the Root Account Access Key and Activating MultiFactor Authentication.
It is a best practice to use MFA for all users and to use U2F or hardware MFA devices for all privileged users.
IAM is universal (global) and does not apply to regions.
IAM is eventually consistent.
IAM replicates data across multiple data centres around the world.
The "root account" is the account created when you setup the AWS account. It has complete Admin access and is the only account that has this access by default.
It is a best practice to not use the root account for anything other than billing.
Power user access allows all permissions except the management of groups and users in IAM.

Temporary security credentials consist of the AWS access key ID, secret access key, and security token.
IAM can assign temporary security credentials to provide users with temporary access to services/resources.
To sign-in you must provide your account ID or account alias in addition to a user name and password.
The sign-in URL includes the account ID or account alias, e.g.:

https://My_AWS_Account_ID.signin.aws.amazon.com/console/.

Alternatively, you can sign-in at the following URL and enter your account ID or alias manually:

https://console.aws.amazon.com/.

IAM integrates with many different AWS services.
IAM supports PCI DSS compliance.
AWS recommend that you use the AWS SDKs to make programmatic API calls to IAM.
However, you can also use the IAM Query API to make direct calls to the IAM web service.

IAM Elements

Principals:

- An entity that can take an action on an AWS resource.
- Your administrative IAM user is your first principal.
- You can allow users and services to assume a role.
- IAM supports federated users.
- IAM supports programmatic access to allow an application to access your AWS account.
- IAM users, roles, federated users, and applications are all AWS principals.

Requests:

- Principals send requests via the Console, CLI, SDKs, or APIs.
- Requests are:

- Actions (or operations) that the principal wants to perform.
- Resources upon which the actions are performed.
- Principal information including the environment from which the request was made.

- Request context – AWS gathers the request information:
  - Principal (requester).
  - Aggregate permissions associated with the principal.
  - Environment data, such as IP address, user agent, SSL status

    etc.
  - Resource data, or data that is related to the resource being requested.

Authentication:

- A principal sending a request must be authenticated to send a request to AWS.
- To authenticate from the console, you must sign in with your user name and password.
- To authenticate from the API or CLI, you must provide your access key and secret key.

Authorization:

- IAM uses values from the request context to check for matching policies and determines whether to allow or deny the request.
- IAM policies are stored in IAM as JSON documents and specify the permissions that are allowed or denied.
- IAM policies can be:
  - User (identity) based policies.
  - Resource-based policies.
- IAM checks each policy that matches the context of your request.
- If a single policy has a deny action IAM denies the request and stops evaluating (explicit deny).

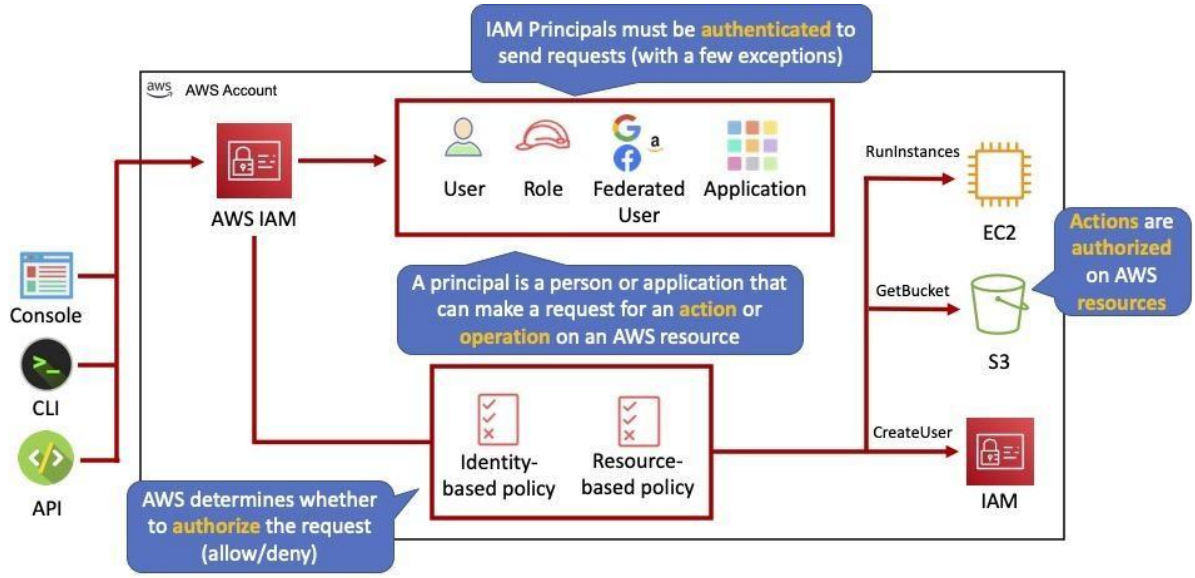
- Evaluation logic:
    - By default all requests are denied (implicit deny).
    - An explicit allow overrides the implicit deny.
    - An explicit deny overrides any explicit allows.
- Only the root user has access to all resources in the account by default.

Actions:

- Actions are defined by a service.
- Actions are the things you can do to a resource such as viewing, creating, editing, deleting.
- Any actions on resources that are not explicitly allowed are denied.
- To allow a principal to perform an action you must include the necessary actions in a policy that applies to the principal or the affected resource.

Resources:

- A resource is an entity that exists within a service.
- E.g. EC2 instances, S3 buckets, IAM users, and DynamoDB tables.
- Each AWS service defines a set of actions that can be performed on the resource.
- After AWS approves the actions in your request, those actions can be performed on the related resources within your account.



Authentication Methods

Console password:

- A password that the user can enter to sign into interactive sessions such as the AWS Management Console.
- You can allow users to change their own passwords.
- You can allow selected IAM users to change their passwords by disabling the option for all users and using an IAM policy to grant permissions for the selected users.

Access Keys:

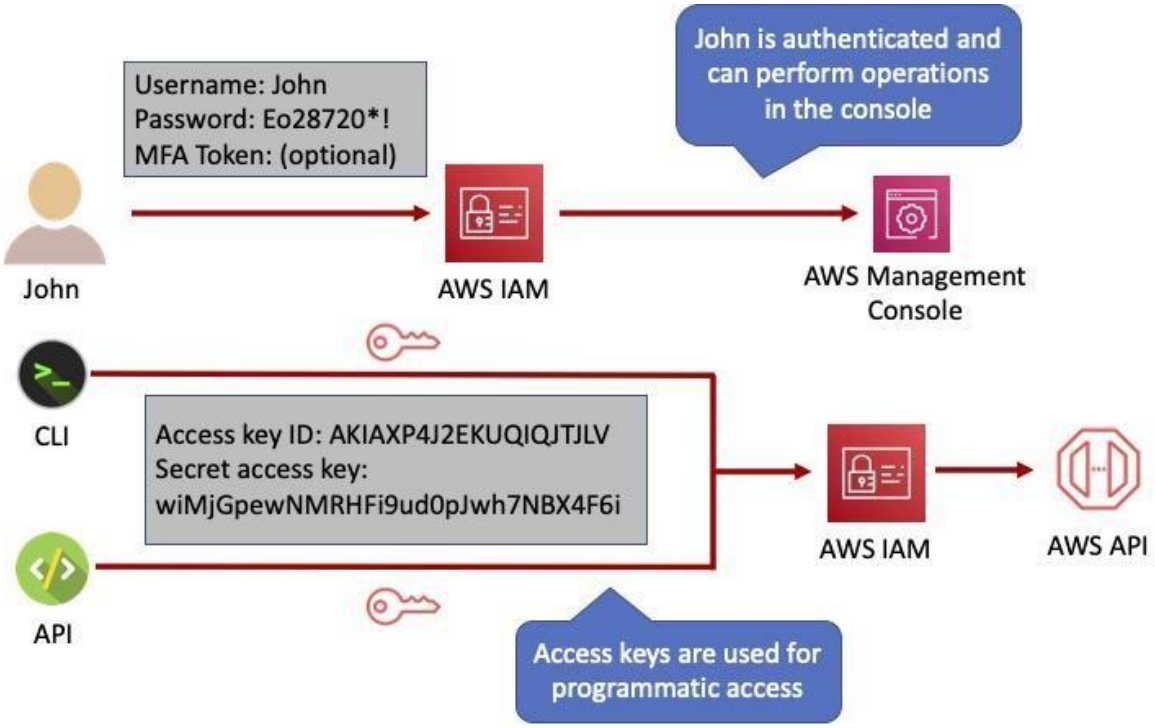
- A combination of an access key ID and a secret access key.
- You can assign two active access keys to a user at a time.
- These can be used to make programmatic calls to AWS when using the API in program code or at a command prompt when using the AWS CLI or the AWS PowerShell tools.
- You can create, modify, view, or rotate access keys.
- When created IAM returns the access key ID and secret access key.
- The secret access is returned only at creation time and if lost a new key must be created.
- Ensure access keys and secret access keys are stored securely.
- Users can be given access to change their own keys through IAM policy (not from the console).

- You can disable a user's access key which prevents it from being used for API calls.

Server certificates:

- SSL/TLS certificates that you can use to authenticate with some AWS services.
- AWS recommends that you use the AWS Certificate Manager (ACM) to provision, manage and deploy your server certificates.
- Use IAM only when you must support HTTPS connections in a region that is not supported by ACM.

The following diagram shows the different methods of authentication available with IAM:



IAM Users

An IAM user is an entity that represents a person or service.

Can be assigned:

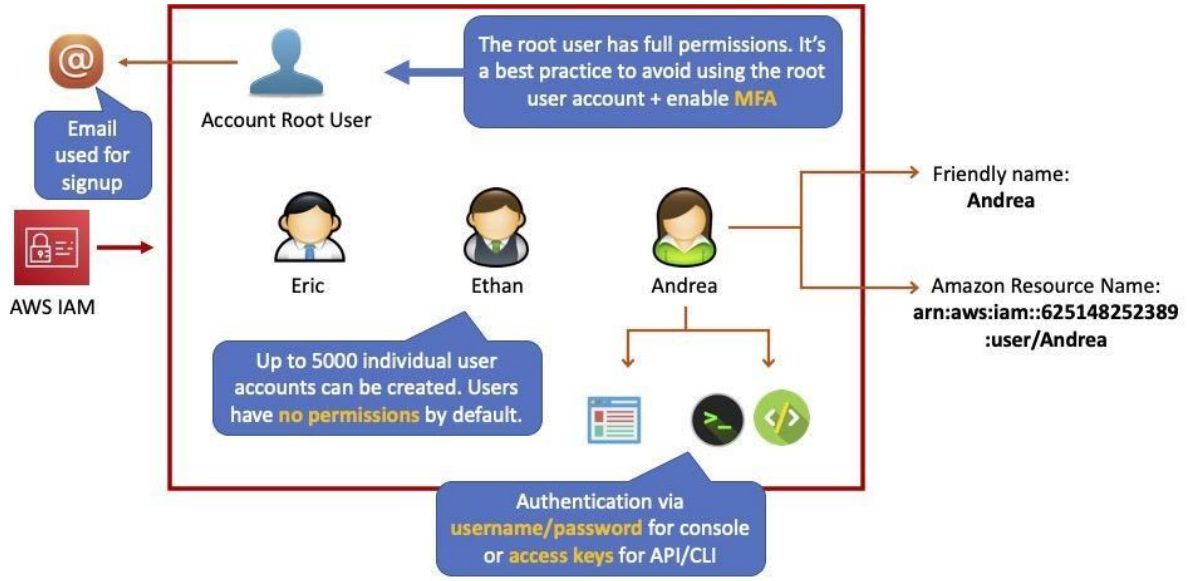
- An access key ID and secret access key for programmatic access to the AWS API, CLI, SDK, and other development tools.
- A password for access to the management console.

By default users cannot access anything in your account.

The account root user credentials are the email address used to create the account and a password.
The root account has full administrative permissions, and these cannot be restricted.
Best practice for root accounts:

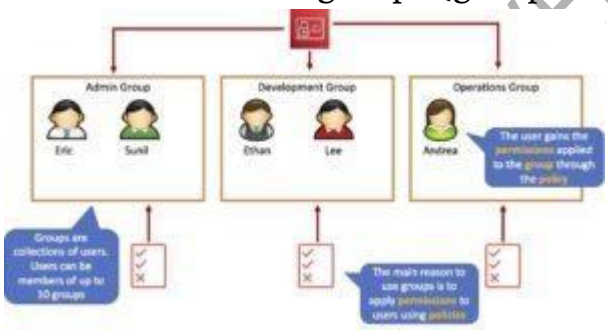
- Don’t use the root user credentials.
- Don’t share the root user credentials.
- Create an IAM user and assign administrative permissions as required.
- Enable MFA.

IAM users can be created to represent applications, and these are known as “service accounts”.
You can have up to 5000 users per AWS account.
Each user account has a friendly name and an ARN which uniquely identifies the user across AWS.
A unique ID is also created which is returned only when you create the user using the API, Tools for Windows PowerShell, or the AWS CLI.
You should create individual IAM accounts for users (best practice not to share accounts).
The Access Key ID and Secret Access Key are not the same as a password and cannot be used to login to the AWS console.
The Access Key ID and Secret Access Key can only be generated once and must be regenerated if lost.
A password policy can be defined for enforcing password length, complexity etc. (applies to all users).
You can allow or disallow the ability to change passwords using an IAM policy.
Access keys and passwords should be changed regularly.



Groups

Groups are collections of users and have policies attached to them.
A group is not an identity and cannot be identified as a principal in an IAM policy.
Use groups to assign permissions to users.
Use the principal of least privilege when assigning permissions.
You cannot nest groups (groups within groups).



Roles

Roles are created and then “assumed” by trusted entities and define a set of permissions for making AWS service requests.
With IAM Roles you can delegate permissions to resources for users and services without using permanent credentials (e.g. user name and password).
IAM users or AWS services can assume a role to obtain temporary security credentials that can be used to make AWS API calls.
You can delegate using roles.

There are no credentials associated with a role (password or access keys).
IAM users can temporarily assume a role to take on permissions for a specific task.
A role can be assigned to a federated user who signs in using an

external identity provider. Temporary credentials are primarily used

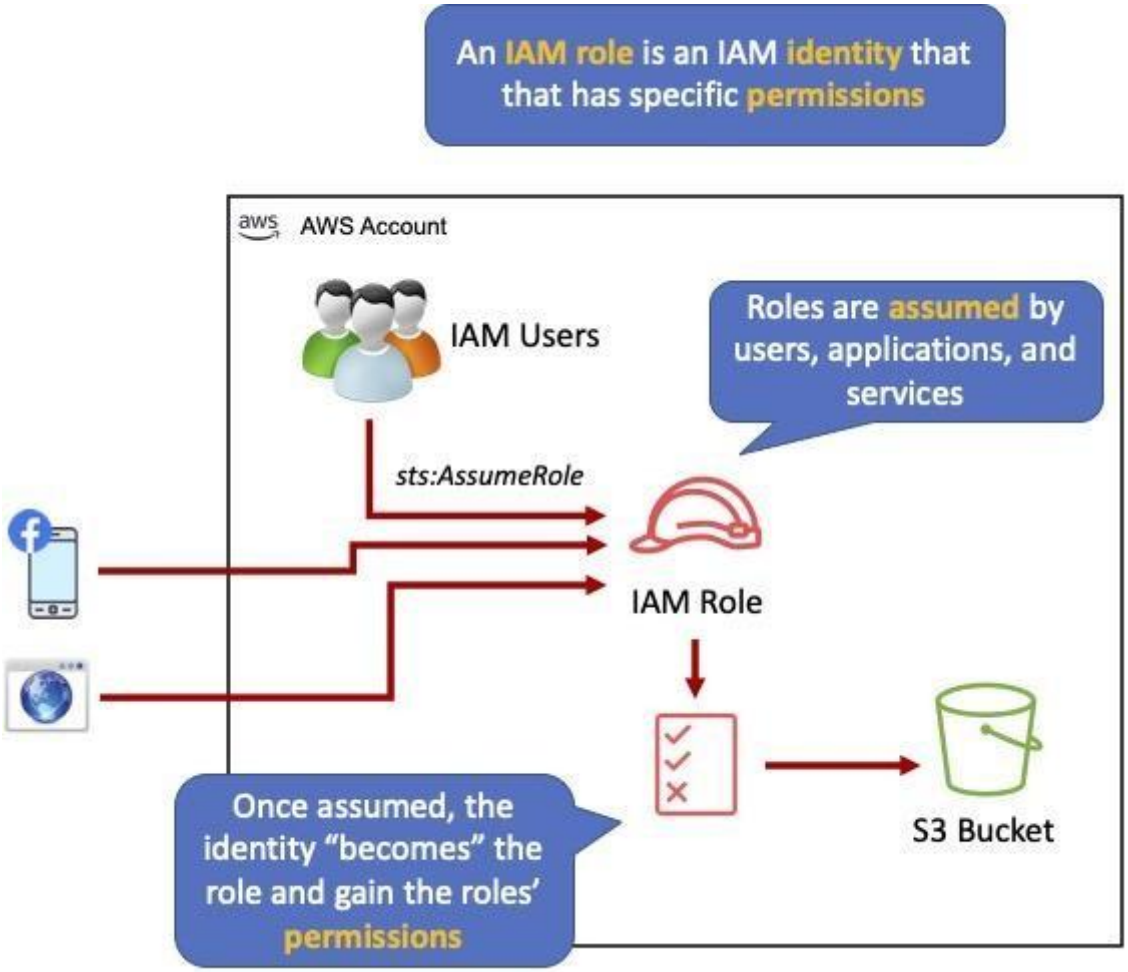
with IAM roles and automatically expire.

Roles can be assumed temporarily through the console or programmatically with the AWS CLI, Tools for Windows PowerShell, or API.
IAM roles with EC2 instances:

- IAM roles can be used for granting applications running on EC2 instances permissions to AWS API requests using instance profiles.
- Only one role can be assigned to an EC2 instance at a time.
- A role can be assigned at the EC2 instance creation time or at any time afterwards.
- When using the AWS CLI or API instance profiles must be created manually (it's automatic and transparent through the console).
- Applications retrieve temporary security credentials from the instance metadata.

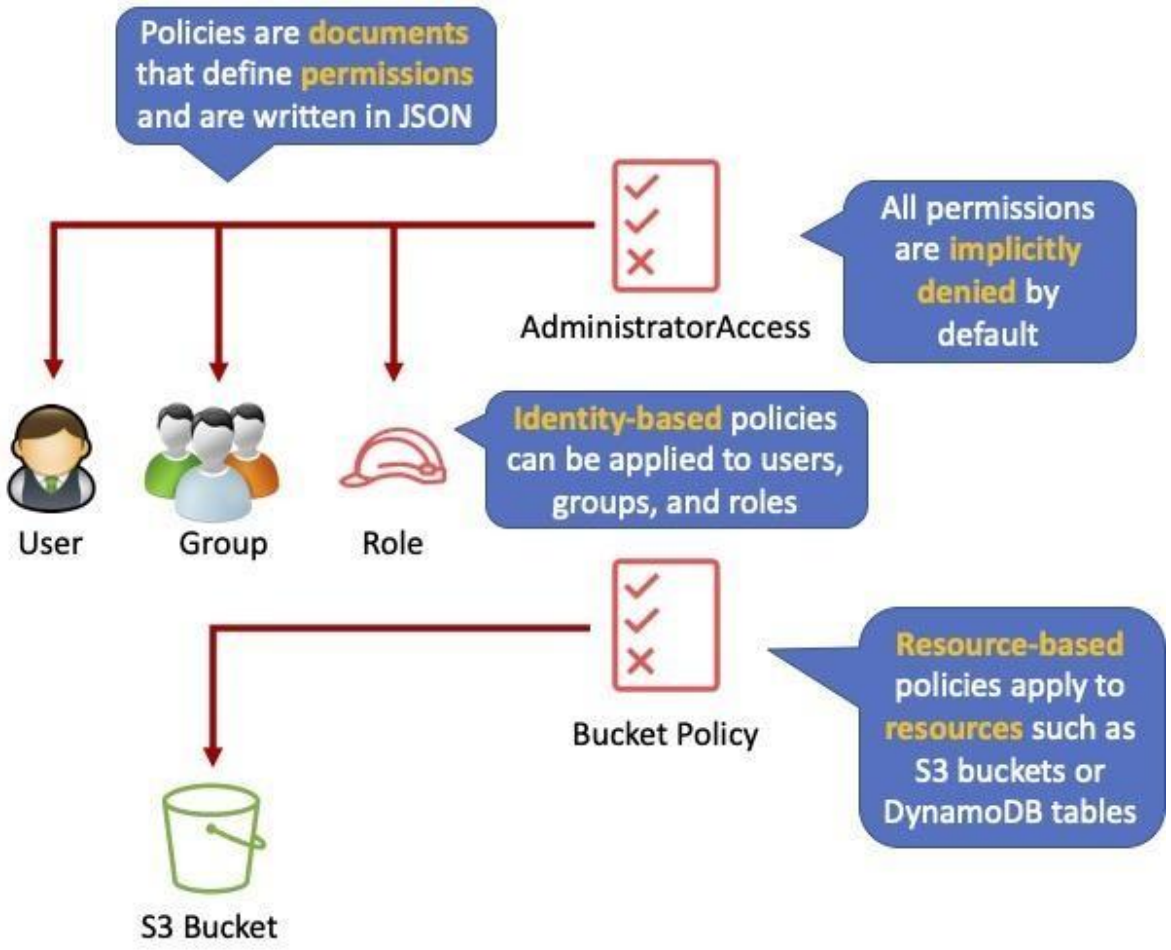
Role Delegation:

- Create an IAM role with two policies:
    - Permissions policy – grants the user of the role the required permissions on a resource.
    - Trust policy – specifies the trusted accounts that are allowed to assume the role.
- Wildcards (*) cannot be specified as a principal.
- A permissions policy must also be attached to the user in the trusted account.



Policies

Policies are documents that define permissions and can be applied to users, groups, and roles.
Policy documents are written in JSON (key value pair that consists of an attribute and a value).
All permissions are implicitly denied by default.
The most restrictive policy is applied.
The IAM policy simulator is a tool to help you understand, test, and validate the effects of access control policies.
The Condition element can be used to apply further conditional logic.



Inline Policies vs Managed Policies

There are 3 types of policies:

- Managed policies.
- Customer managed policies.
- Inline policies.

Managed Policy:

- Created and administered by AWS.
- Used for common use cases based on job function.
- Save you having to create policies yourself.
- Can be attached to multiple users, groups, or roles within and across AWS accounts.
- Cannot change the permissions assigned.

Customer Managed Policy:

- Standalone policy that you create and administer in your own AWS account.
- Can be attached to multiple users, groups, and roles – but only within your own account.
- Can be created by copying an existing managed policy and then customizing it.
- Recommended for use cases where the existing AWS Managed Policies don't meet the needs of your environment.

Inline Policy:

- Inline policies are embedded within the user, group, or role to which it is applied.
- Strict 1:1 relationship between the entity and the policy.
- When you delete the user, group, or role in which the inline policy is embedded, the policy will also be deleted.
- In most cases, AWS recommends using Managed Policies instead of inline policies.
- Inline policies are useful when you want to be sure that the permissions in a policy are not inadvertently assigned to any other user, group, or role.

AWS Managed and Customer Managed Policies

An AWS managed policy is a standalone policy that is created and administered by AWS.
Standalone policy means that the policy has its own Amazon Resource Name (ARN) that includes the policy name.
AWS managed policies are designed to provide permissions for many common use cases.
You cannot change the permissions defined in AWS managed policies.
Some AWS managed policies are designed for specific job functions.
The job-specific AWS managed policies include:

- Administrator.
- Billing.

- Database Administrator.
- Data Scientist.
- Developer Power User.

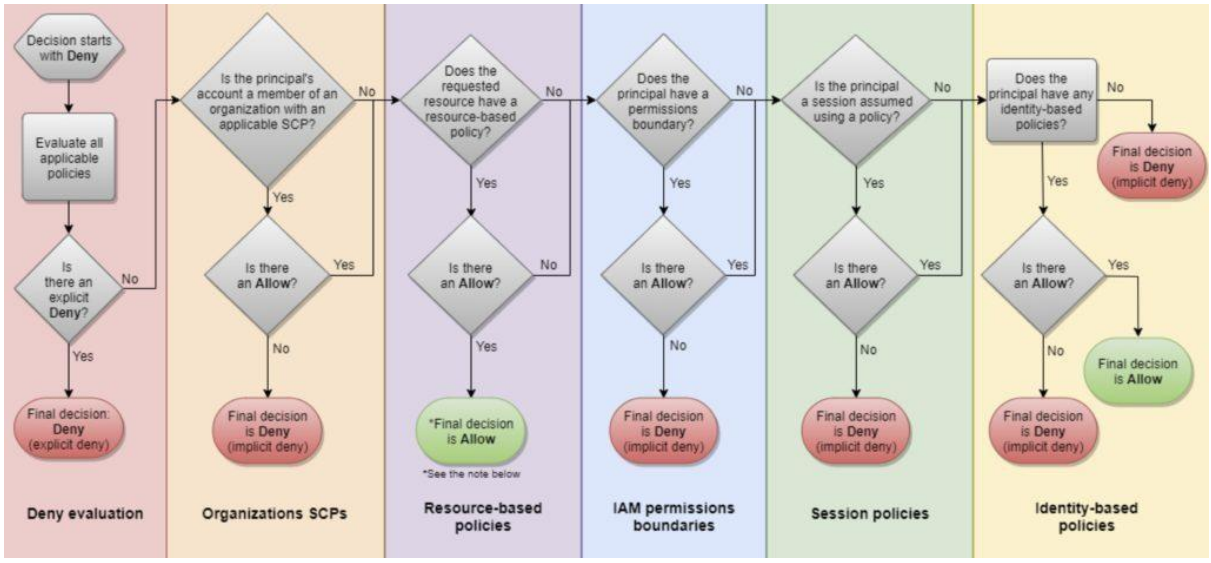
Network Administrator.

- Security Auditor.
- Support User.
- System Administrator.
- View-Only User.

You can create standalone policies that you administer in your own AWS account, which we refer to as customer managed policies.
You can then attach the policies to multiple principal entities in your AWS account.
When you attach a policy to a principal entity, you give the entity the permissions that are defined in the policy.
IAM Policy Evaluation Logic
By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access).
An explicit allow in an identity-based or resource-based policy overrides this default.
If a permissions boundary, Organizations SCP, or session policy is present, it might override the allow with an implicit deny.
An explicit deny in any policy overrides any allows.
A few concepts should be known to understand the logic:

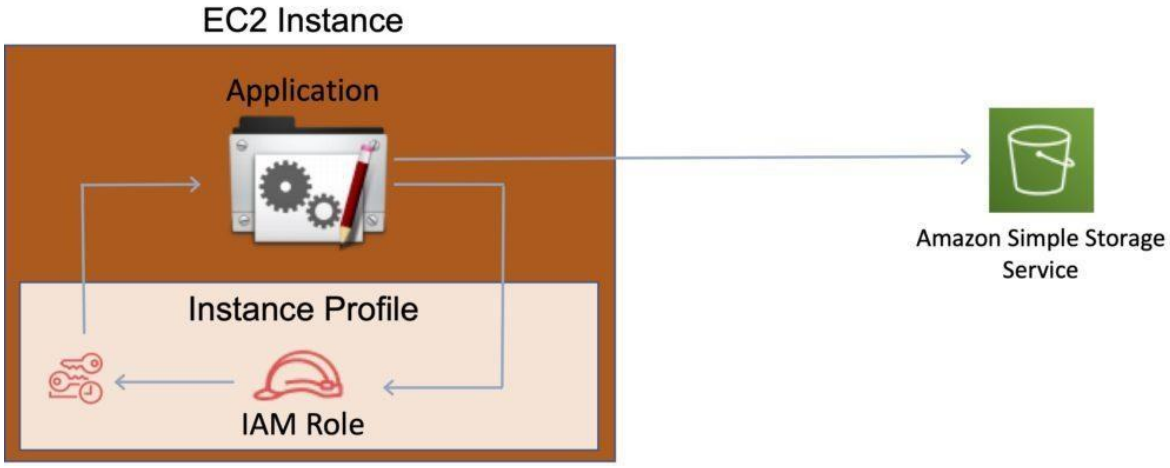
- Identity-based policies – Identity-based policies are attached to an IAM identity (user, group of users, or role) and grant permissions to IAM entities (users and roles).
- Resource-based policies – Resource-based policies grant permissions to the principal (account, user, role, or federated user) specified as the principal.
- IAM permissions boundaries – Permissions boundaries are an advanced feature that sets the maximum permissions that an identity-based policy can grant to an IAM entity (user or role).
- AWS Organizations service control policies (SCPs) – Organizations SCPs specify the maximum permissions for an organization or organizational unit (OU). Session policies – Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session for a role or federated user.

The following flowchart details the IAM policy evaluation logic:



IAM Instance Profiles

An instance profile is a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts.
An instance profile can contain only one IAM role, although a role can be included in multiple instance profiles.



You can use the following AWS CLI commands to work with instance profiles in an AWS account:

- Create an instance profile: aws iam create-instance-profile
- Add a role to an instance profile: aws iam add-role-to-instance-profile
- List instance profiles: aws iam list-instance-profiles, aws iam list-instance-profiles-forrole
- Get information about an instance profile: aws iam get-instance-profile
- Remove a role from an instance profile: aws iam remove-role-from-instance-profile
  Delete an instance profile: aws iam delete-instance-profile

AWS Security Token Service

The AWS Security Token Service (STS) is a web service that enables you to request temporary, limited-privilege credentials for IAM users or for users that you authenticate (federated users).
By default, AWS STS is available as a global service, and all AWS STS requests go to a single endpoint at https://sts.amazonaws.com
You can optionally send your AWS STS requests to endpoints in any region (can reduce latency).
Credentials will always work globally.
STS supports AWS CloudTrail, which records AWS calls for your AWS account and delivers log files to an S3 bucket.
Temporary security credentials work almost identically to long-term access key credentials that IAM users can use, with the following differences:

- Temporary security credentials are short-term.
- They can be configured to last anywhere from a few minutes to several hours.
- After the credentials expire, AWS no longer recognizes them or allows any kind of access to API requests made with them.
- Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested.
- When (or even before) the temporary security credentials expire, the user can request new credentials, if the user requesting them still has permission to do so.

- You do not have to distribute or embed long-term AWS security credentials with an application.
- You can provide access to your AWS resources to users without having to define an AWS identity for them (temporary security credentials are the basis for IAM Roles and ID Federation).
- The temporary security credentials have a limited lifetime, so you do not have to rotate them or explicitly revoke them when they’re no longer needed.
- After temporary security credentials expire, they cannot be reused (you can specify how long the credentials are valid for, up to a maximum limit).

The AWS STS API action returns temporary security credentials that consist of:

- An access key which consists of an access key ID and a secret ID.
- A session token.
- Expiration or duration of validity.

- Users (or an application that the user runs) can use these credentials to access your resources.

With STS you can request a session token using one of the following APIs:

- AssumeRole – can only be used by IAM users (can be used for MFA).
- AssumeRoleWithSAML – can be used by any user who passes a SAML authentication response that indicates authentication from a known (trusted) identity provider.
- AssumeRoleWithWebIdentity – can be used by an user who passes a web identity token that indicates authentication from a known (trusted) identity provider.
- GetSessionToken – can be used by an IAM user or AWS account root user (can be used for MFA).
- GetFederationToken – can be used by an IAM user or AWS account root user.

AWS recommends using Cognito for identity federation with Internet identity providers.
Users can come from three sources.
Federation (typically AD):

- Uses SAML 2.0.
- Grants temporary access based on the users AD credentials.
- Does not need to be a user in IAM.
- Single sign-on allows users to login to the AWS console without assigning IAM credentials.

Federation with Mobile Apps:

- Use Facebook/Amazon/Google or other OpenID providers to login.

Cross Account Access:

- Lets users from one AWS account access resources in another.
- To make a request in a different account the resource in that account must have an attached resource-based policy with the permissions you need.

  Or you must assume a role (identity-based policy) within that account with the permissions you need.

There are a couple of ways STS can be used.
Scenario 1:

1. Develop an Identity Broker to communicate with LDAP and AWS STS.
2. Identity Broker always authenticates with LDAP first, then with AWS STS.

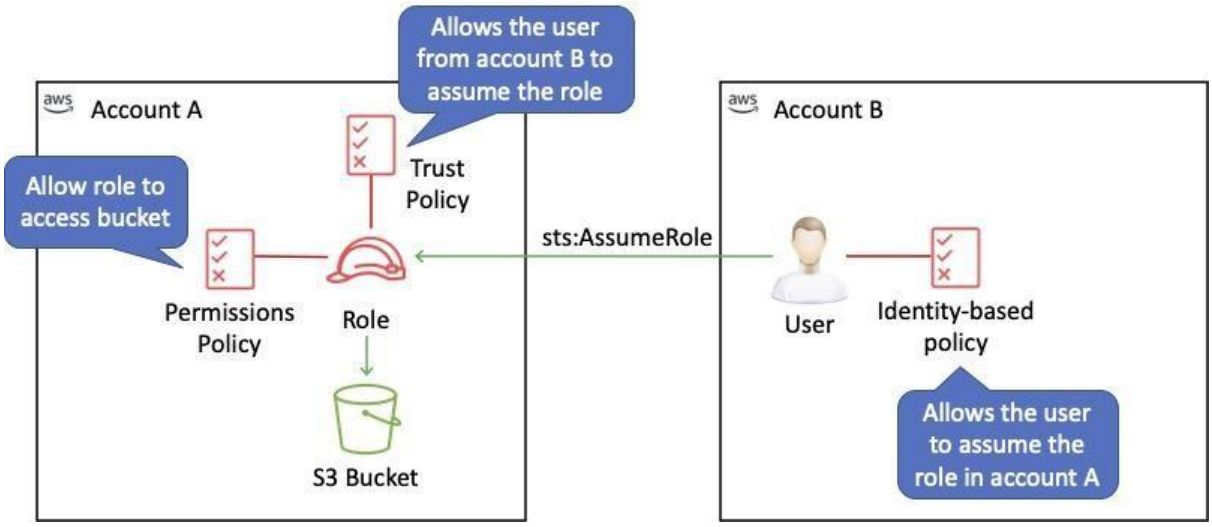
3. Application then gets temporary access to AWS resources.

Scenario 2:

1. Develop an Identity Broker to communicate with LDAP and AWS STS.
2. Identity Broker authenticates with LDAP first, then gets an IAM role associated with the user.
3. Application then authenticates with STS and assumes that IAM role.
4. Application uses that IAM role to interact with the service.

Cross Account Access

Useful for situations where an AWS customer has separate AWS account – for example for development and production resources.
Cross Account Access makes is easier to work productively within a multi-account (or multirole) AWS environment by making is easy to switch roles within the AWS Management Console.
Can sign-in to the console using your IAM username and then switch the console to manage another account without having to enter another user name and password.
Let's users from one AWS account access resources in another.
To make a request in a different account the resource in that account must have an attached resource-based policy with the permissions you need.
Or you must assume a role (identity-based policy) within that account with the permissions you need.



IAM Best Practices

To secure AWS resources it is recommended that you follow these best practices:

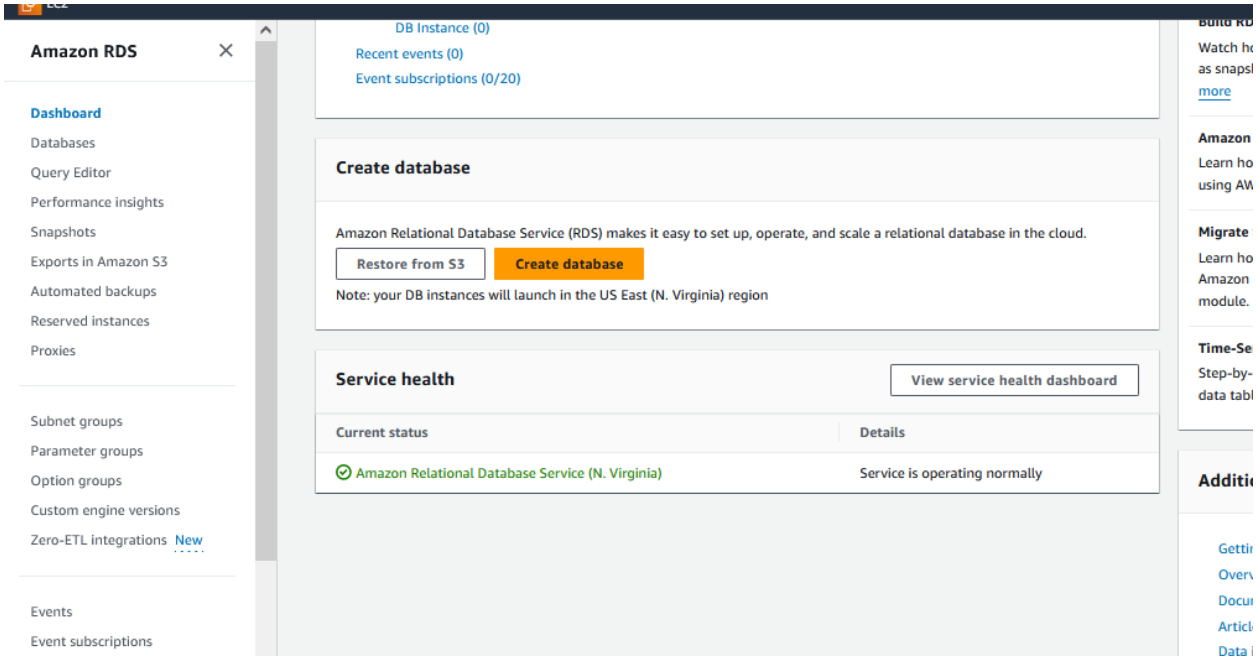
- Lock away your AWS account root user access keys.

- Use roles to delegate permissions.
- Grant least privilege.
- Get started using permissions with AWS managed policies.
- Validate your policies.
- Use customer managed policies instead of inline policies.
- Use access levels to review IAM permissions.
- Configure a strong password policy for your users.
- Enable MFA.
- Use roles for applications that run on Amazon EC2 instances.
- Do not share access keys.
- Rotate credentials regularly.
- Remove unnecessary credentials.
- Use policy conditions for extra security.
- Monitor activity in your AWS account.

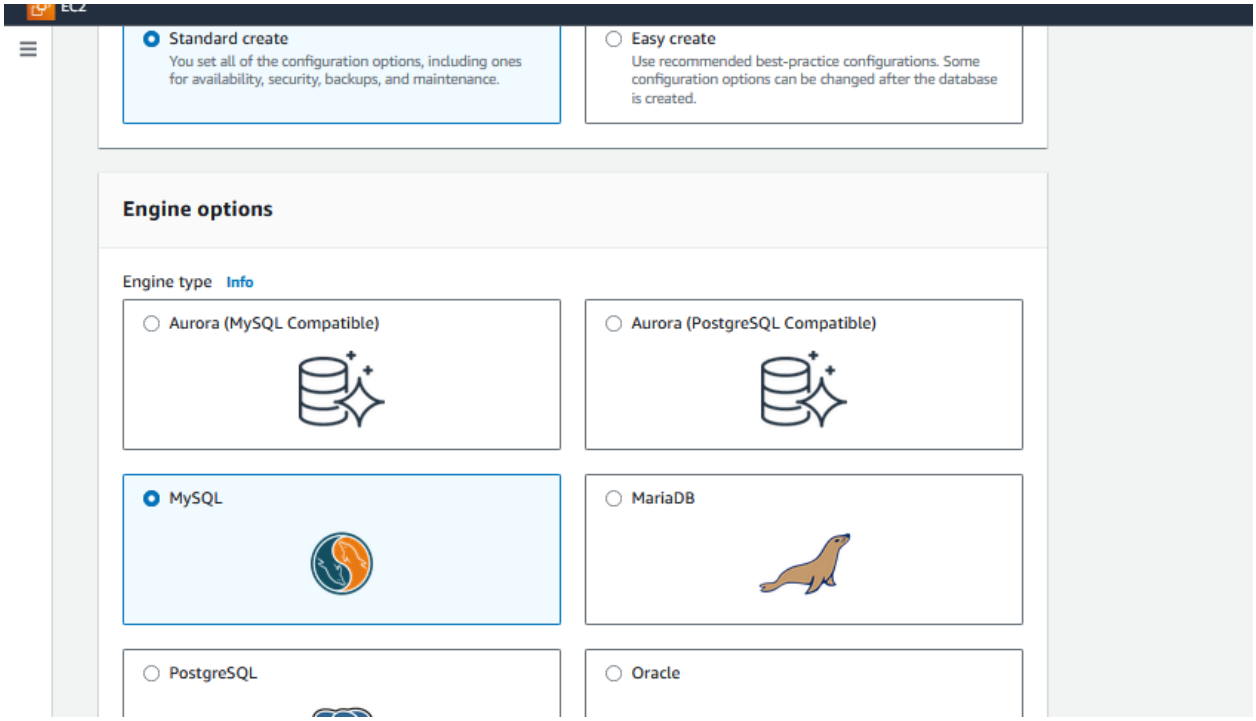
RDS

Create RDS : Amazon Relational Database Service

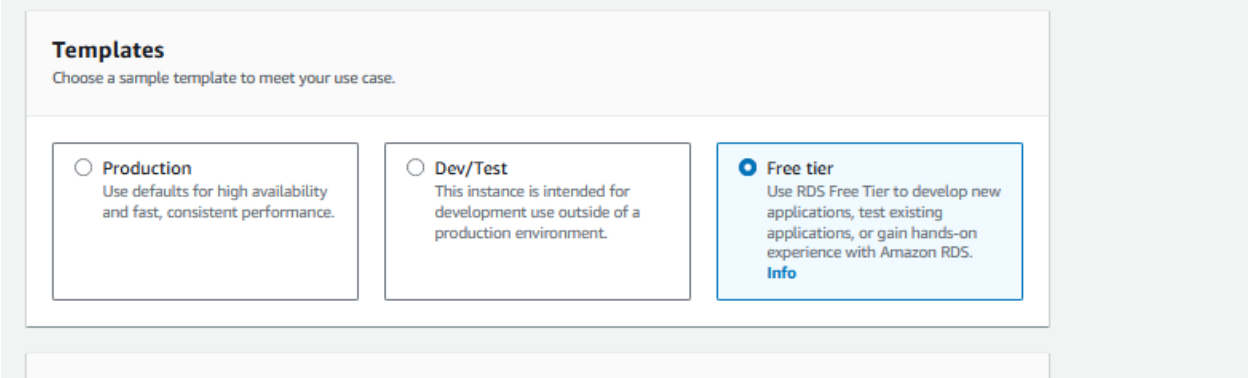
Select create database



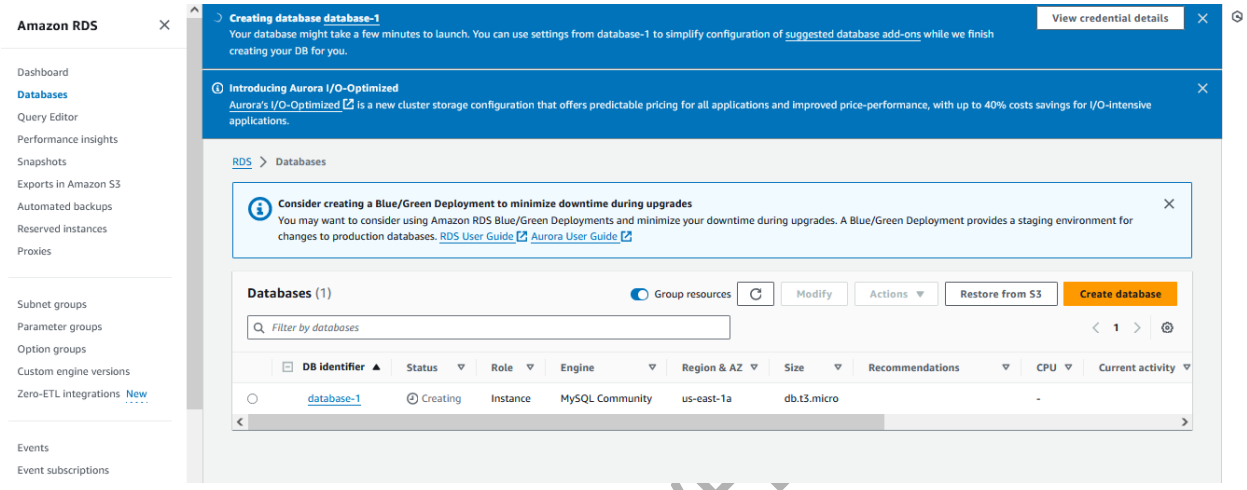
Select standard create and choose MyQL


Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.
Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.
Engine options
Engine type Info
Aurora (MySQL Compatible)
Aurora (PostgreSQL Compatible)
MySQL
MariaDB
PostgreSQL
Oracle

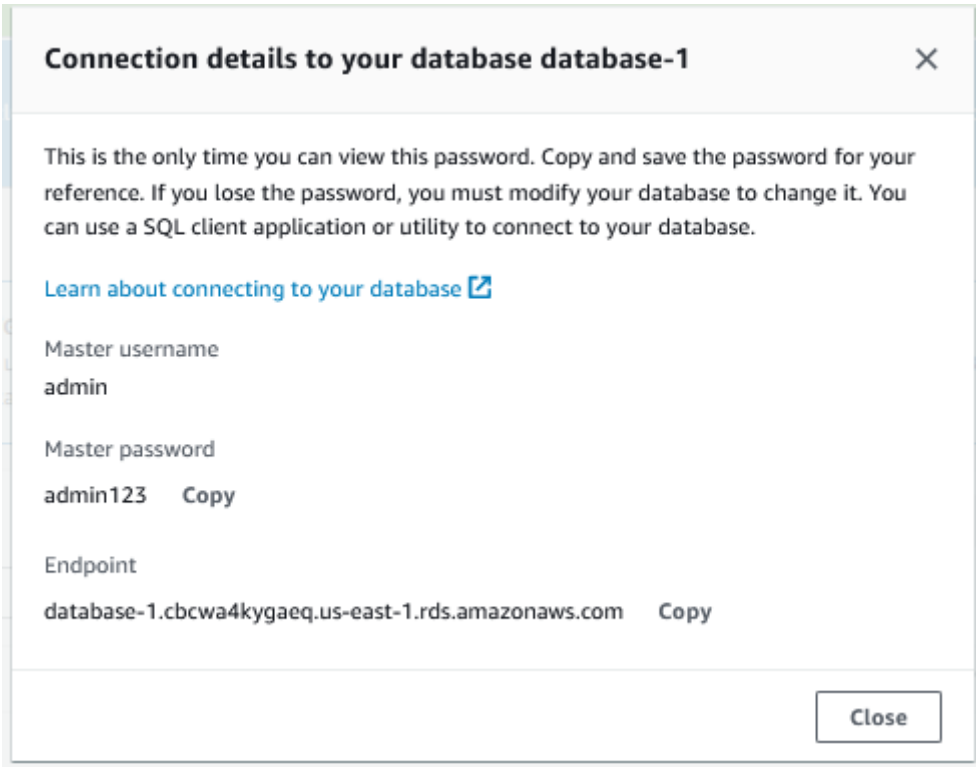
Choose free tier model

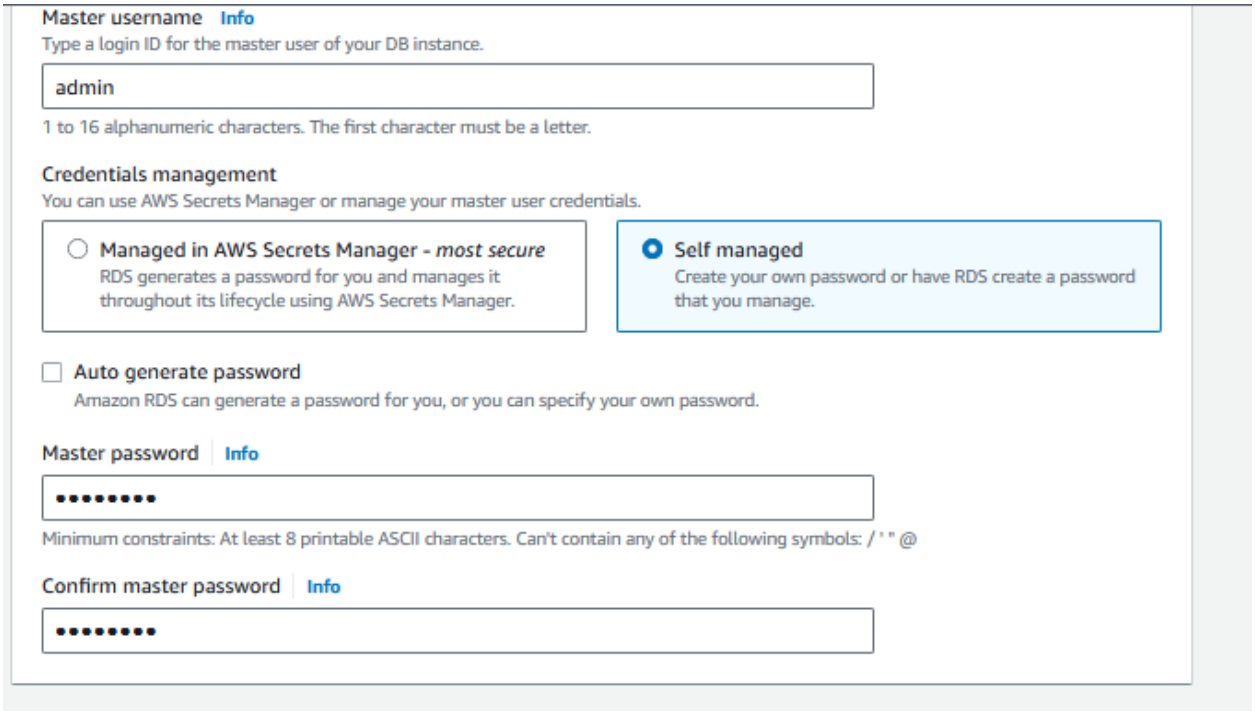


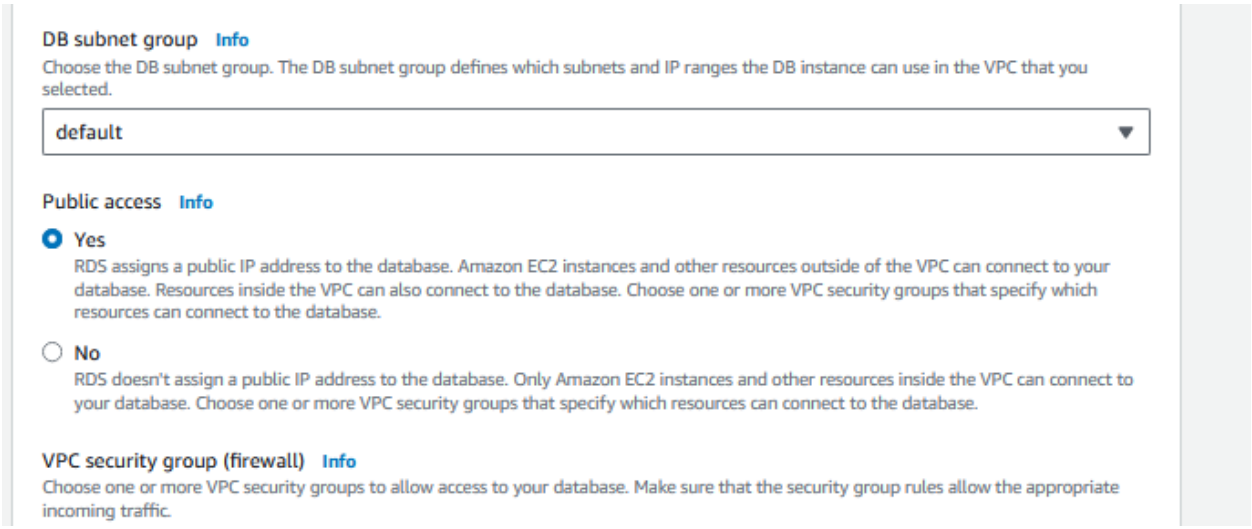
Provide password



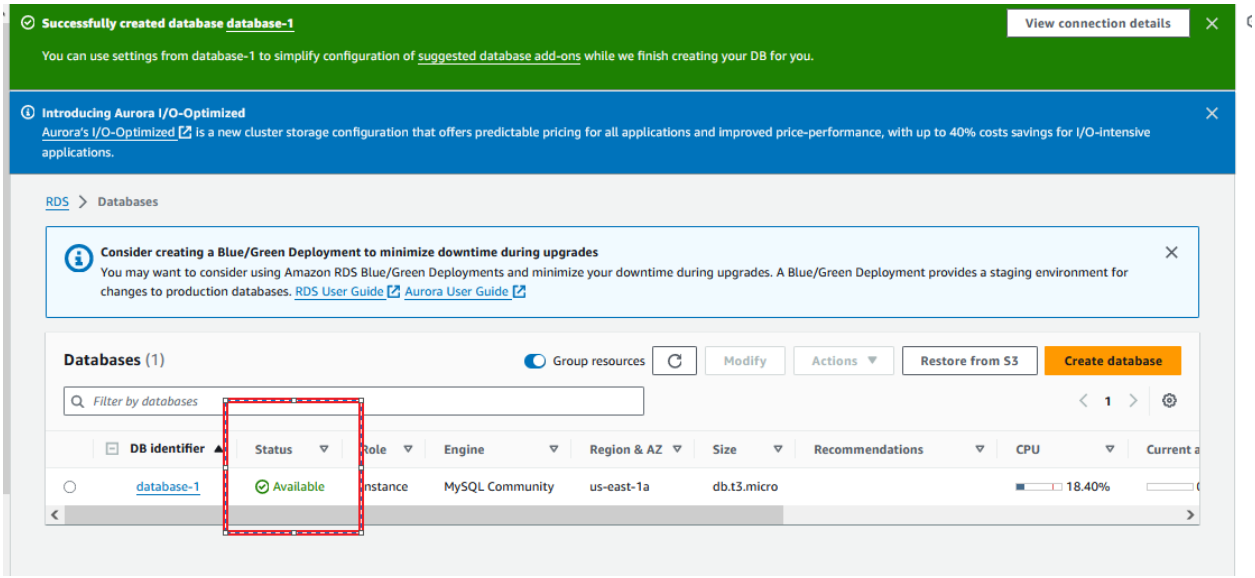
if you want to see your credentials, you can click on view credentials details


Connection details to your database database-1
This is the only time you can view this password. Copy and save the password for your reference. If you lose the password, you must modify your database to change it. You can use a SQL client application or utility to connect to your database.
Learn about connecting to your database
Master username
admin
Master password
admin123 Copy
Endpoint
database-1.cbcwa4kygaeq.us-east-1.rds.amazonaws.com Copy
Close


Master username Info
Type a login ID for the master user of your DB instance.
admin
1 to 16 alphanumeric characters. The first character must be a letter.
Credentials management
You can use AWS Secrets Manager or manage your master user credentials.
Managed in AWS Secrets Manager - most secure
RDS generates a password for you and manages it throughout its lifecycle using AWS Secrets Manager.
Self managed
Create your own password or have RDS create a password that you manage.
Auto generate password
Amazon RDS can generate a password for you, or you can specify your own password.
Master password Info
Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / ' " @
Confirm master password Info

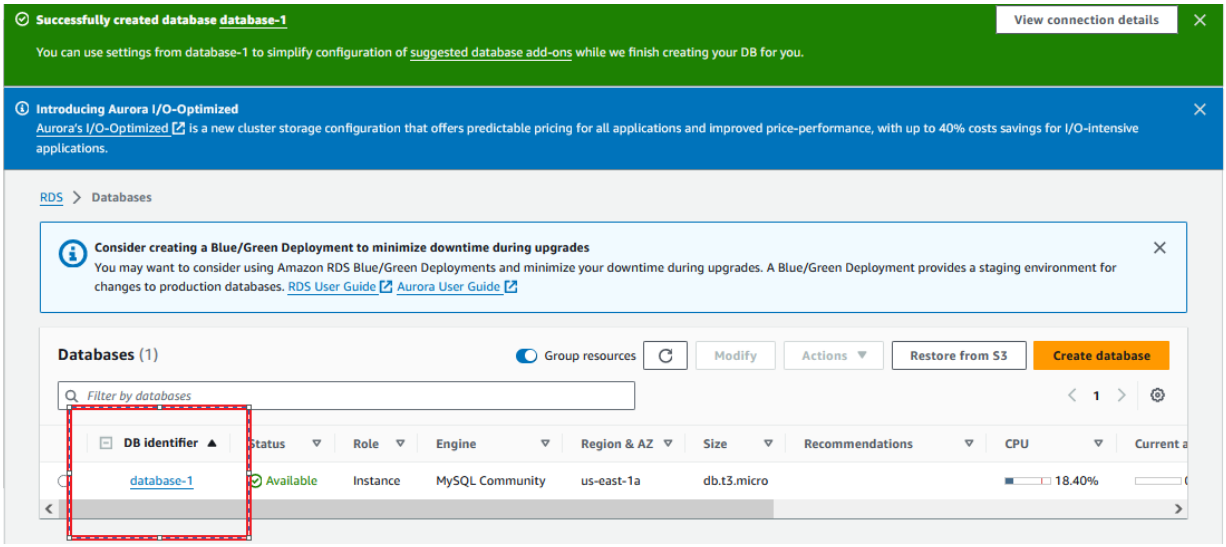


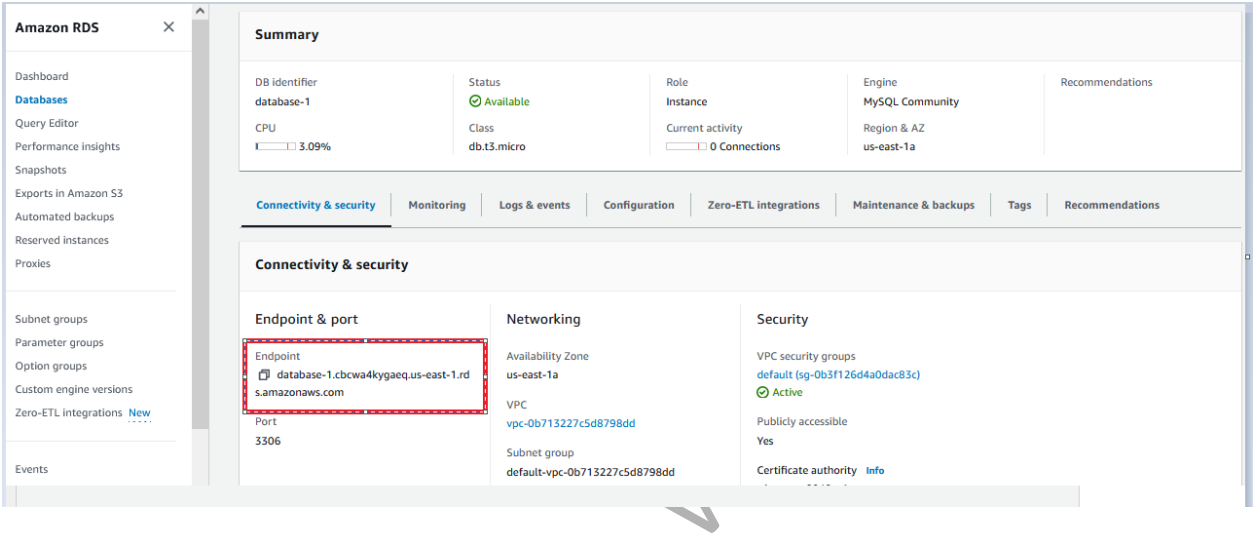
Click on create database button



Database would take few mins to up and running.

Click on the Database-1 to capture the endpoint.





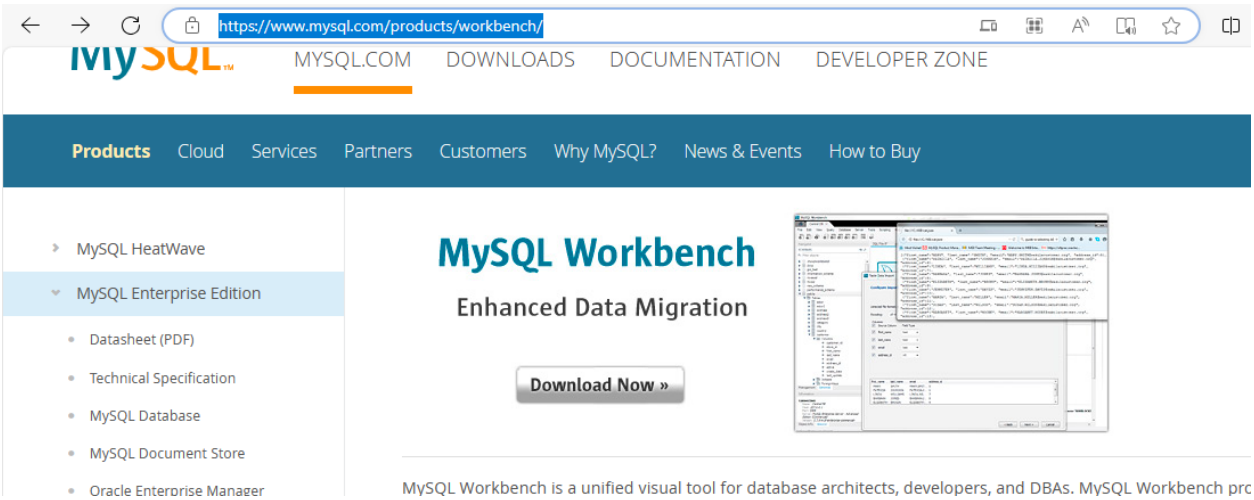
Copy the Endpoint

Endpoint: database-1.cbcwa4kygaeq.us-east-1.rds.amazonaws.com

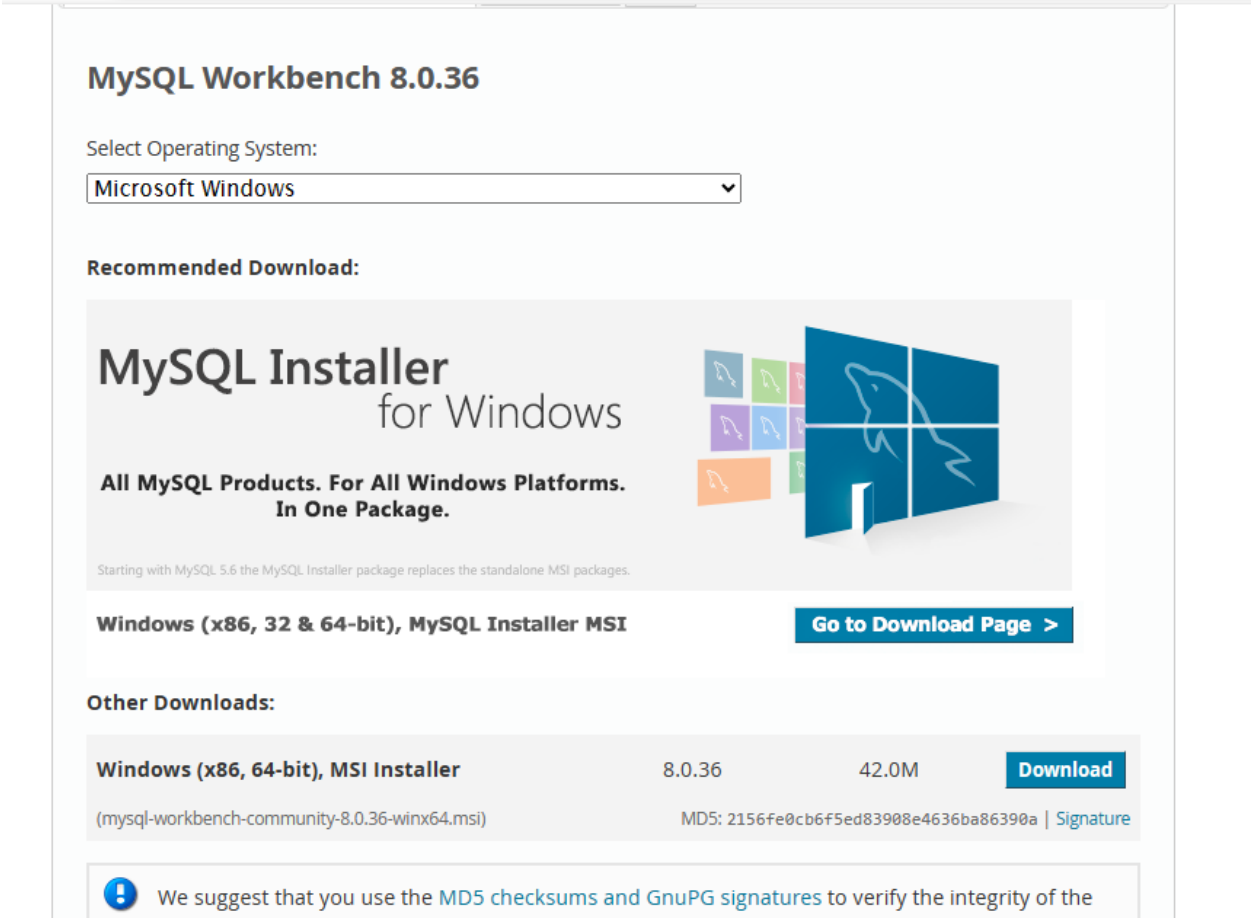
This endpoint we are using it to connect from MySQL client.

[MySQL :: MySQL Workbench](https://www.mysql.com/products/workbench/) download

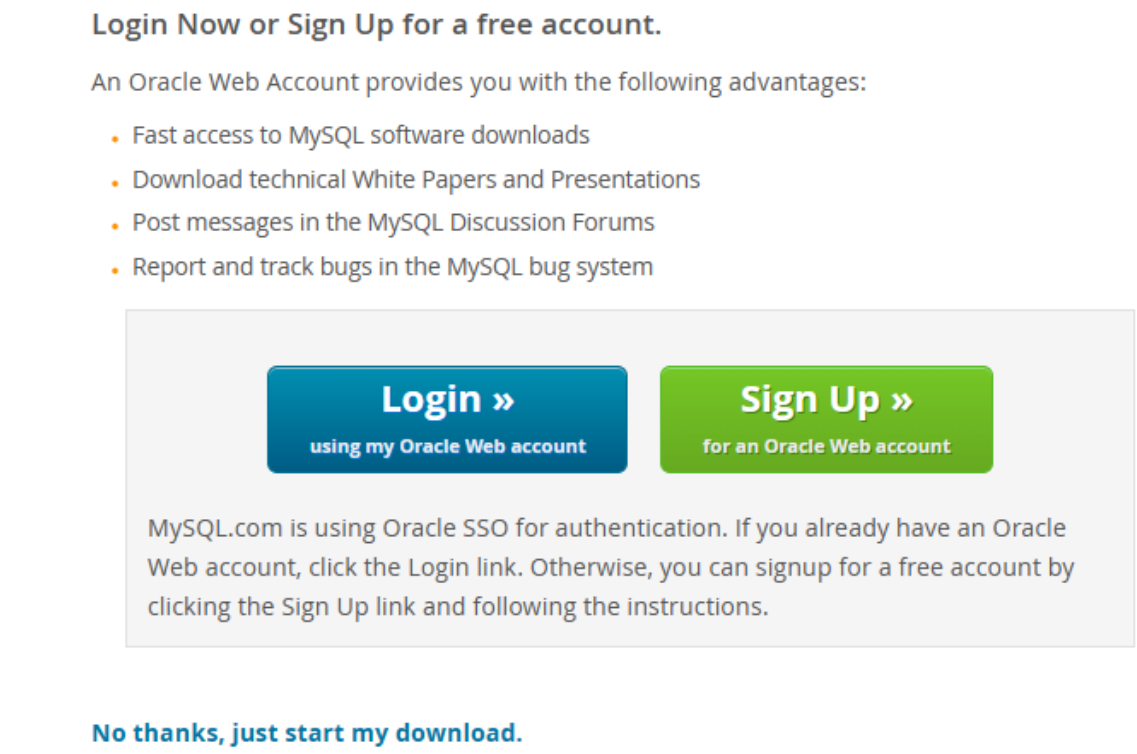
https://www.mysql.com/products/workbench/



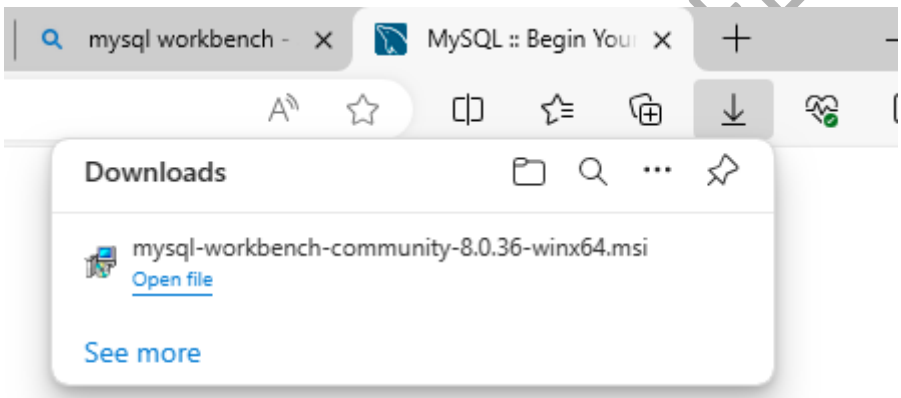
Click on Download Now



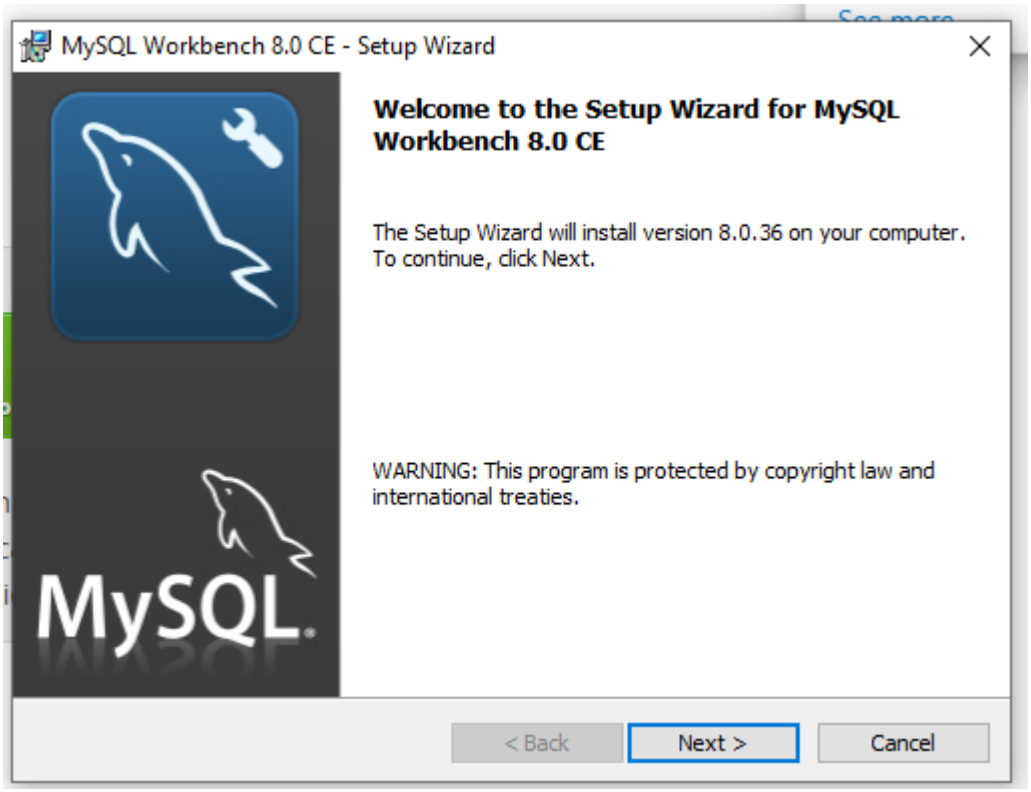
Click on Download button



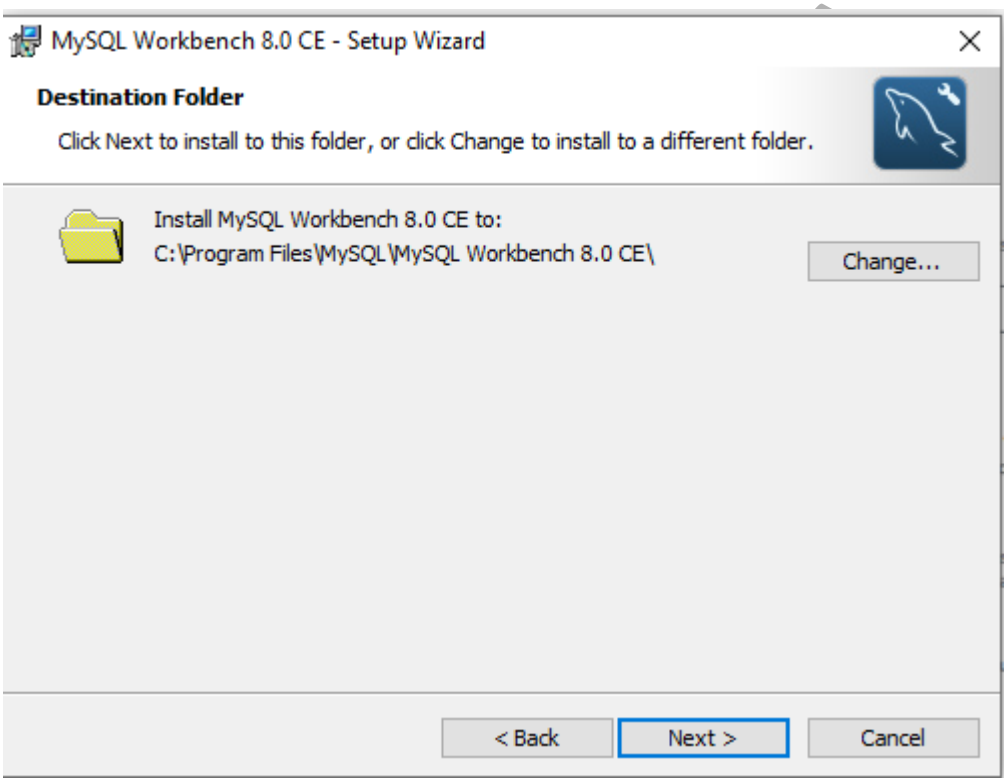
Click here to download with login : [No thanks, just start my download.]



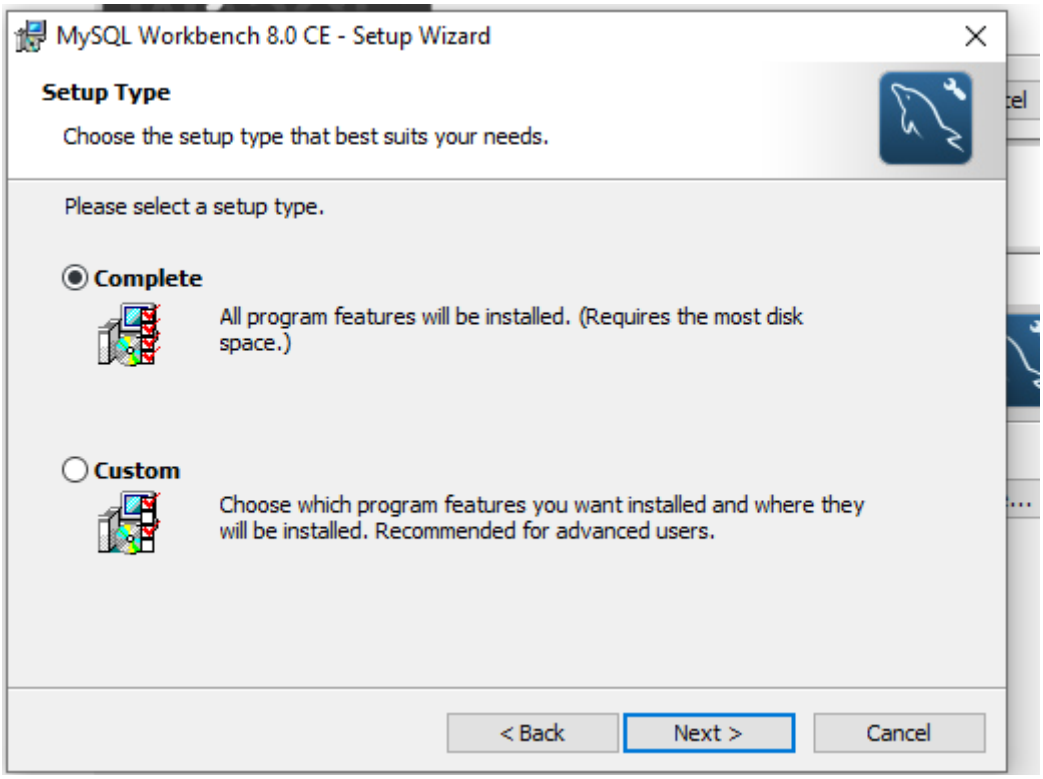
The download mysql client will be available in downloads, then double click on .msi file



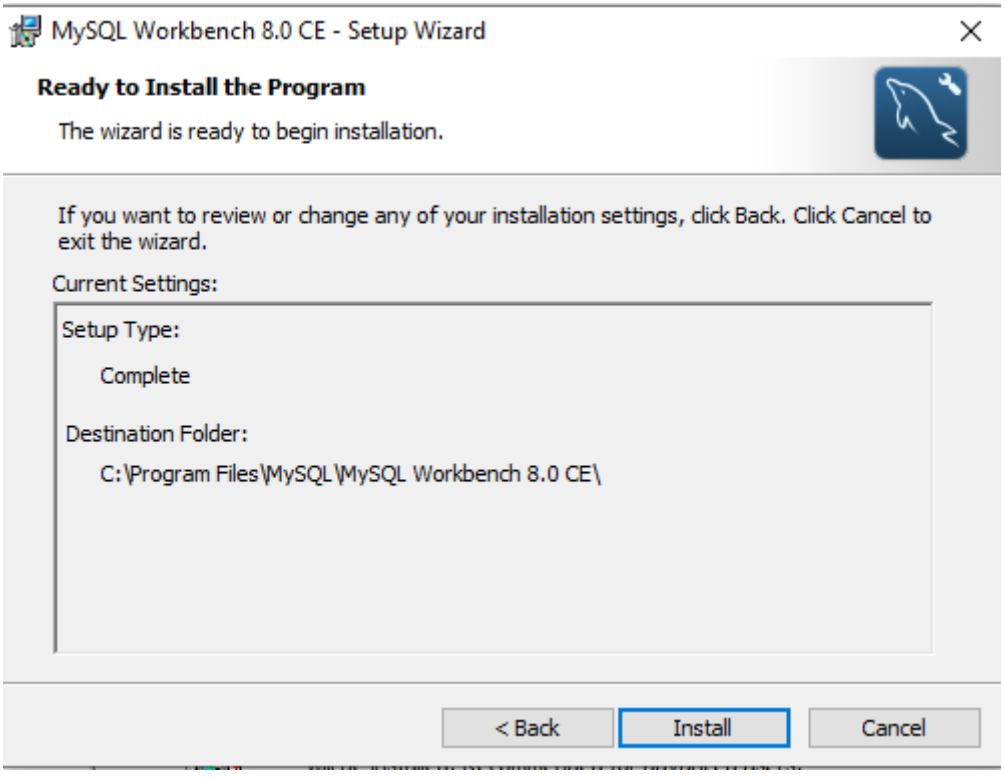
Click on Next



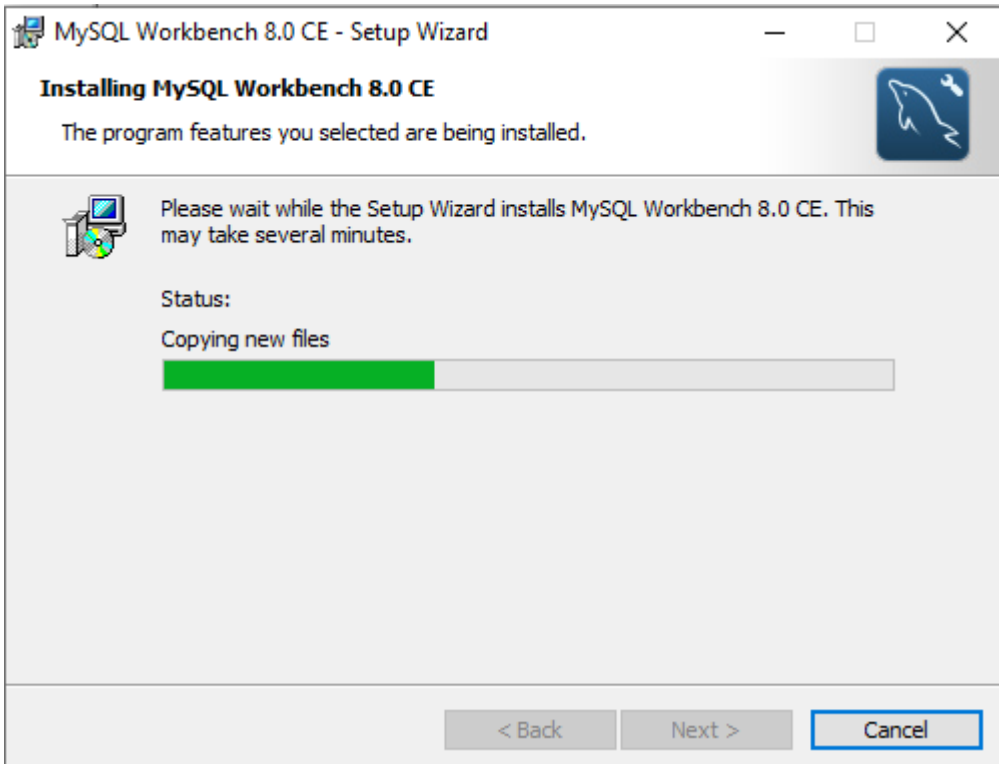
Click on Next



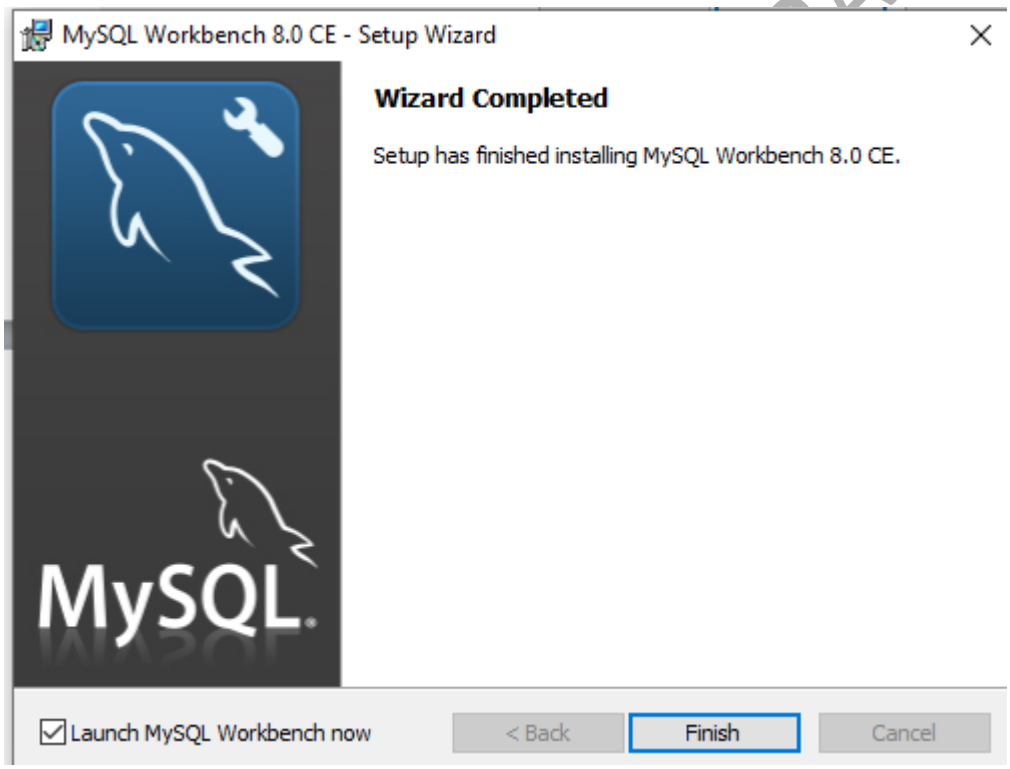
Click on Next



Click on Install

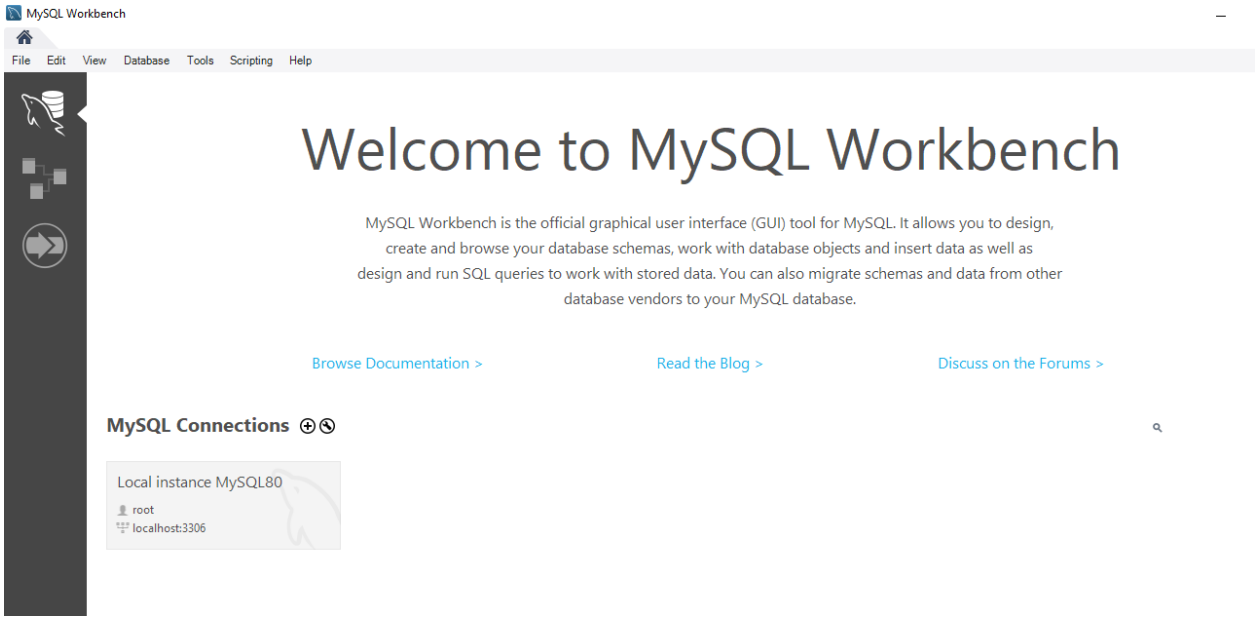


Wait till Finish button shows

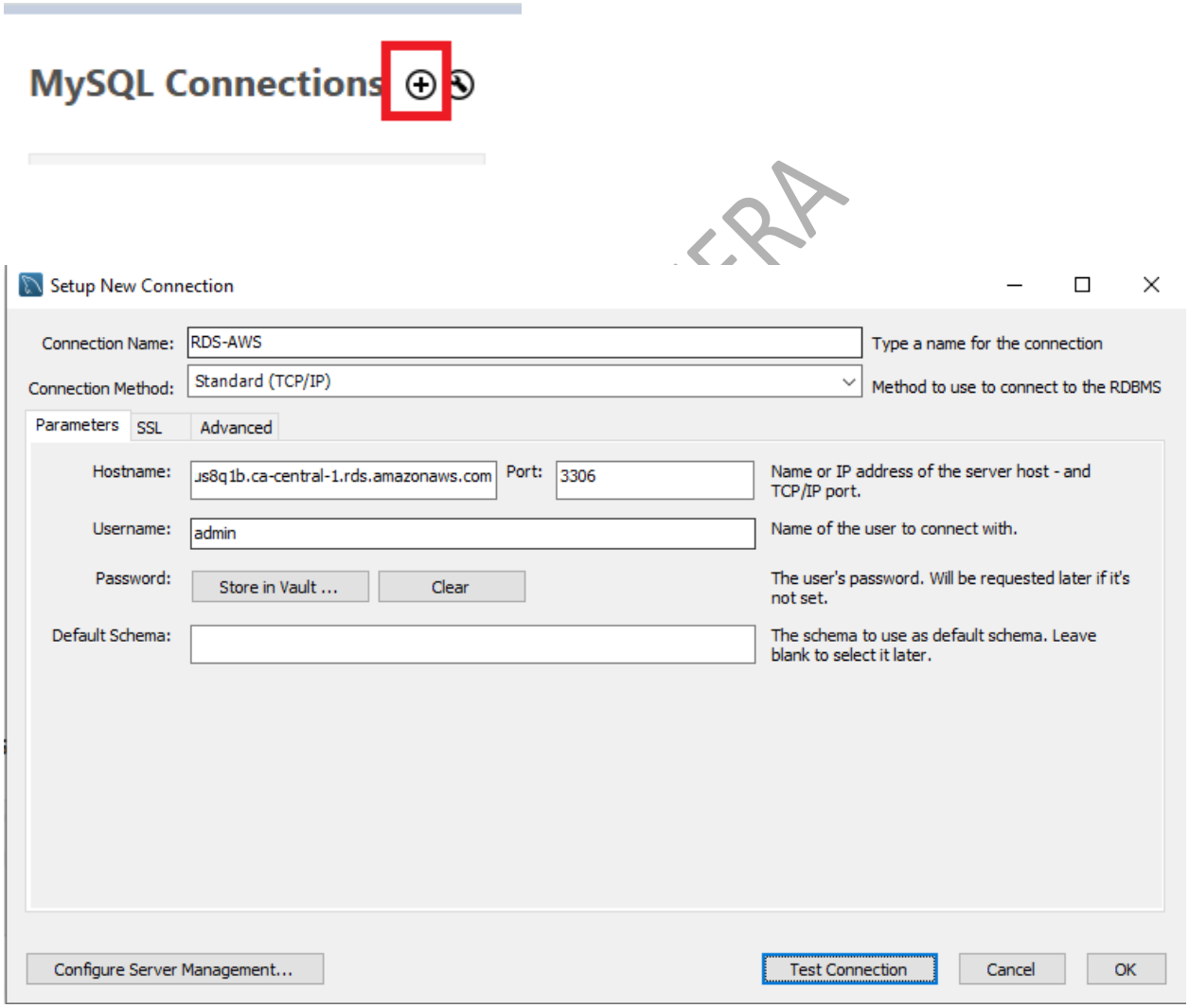


Click on Finish

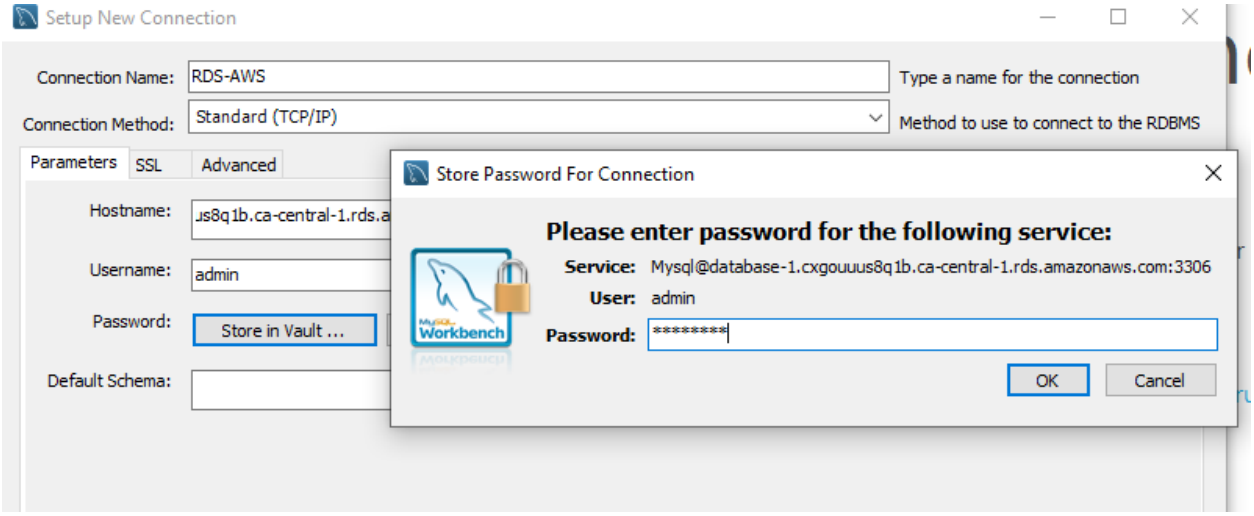
Now launch the mysql workbench to connect AWS RDS



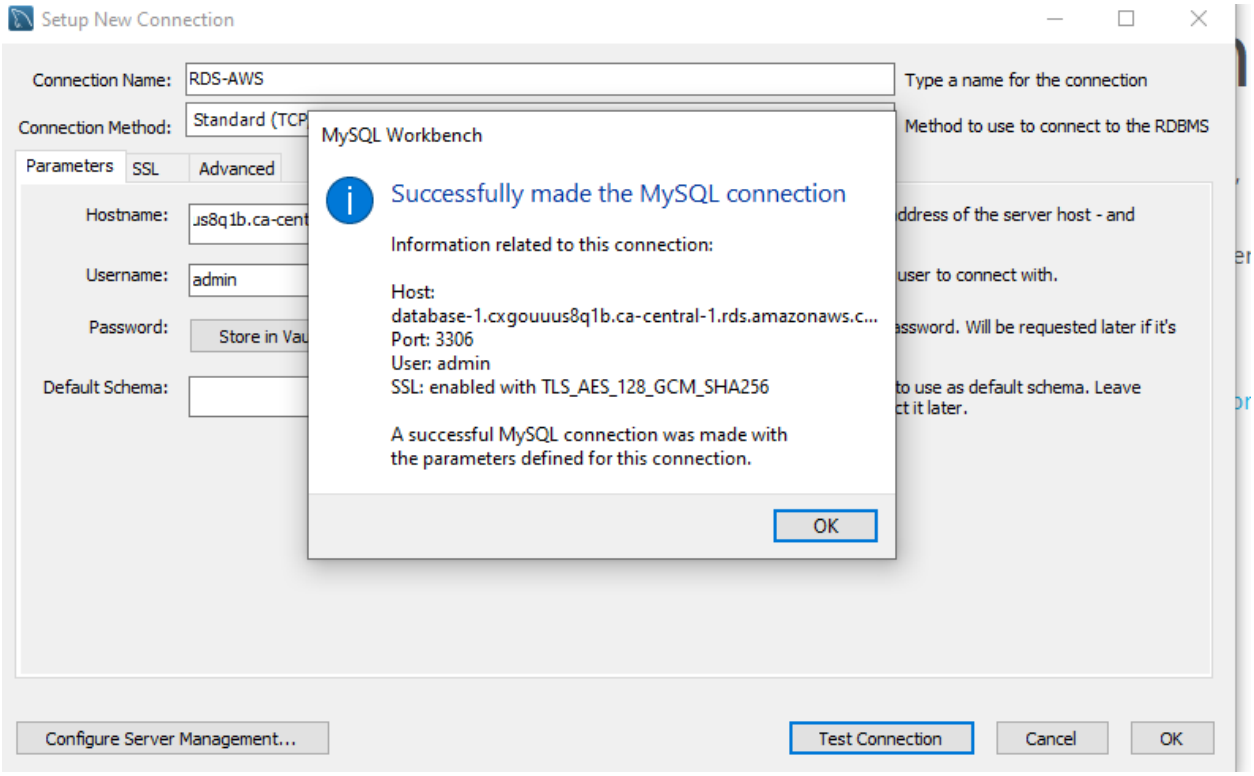
Click on MySQL connections + symbol to configure our RDS mysql db



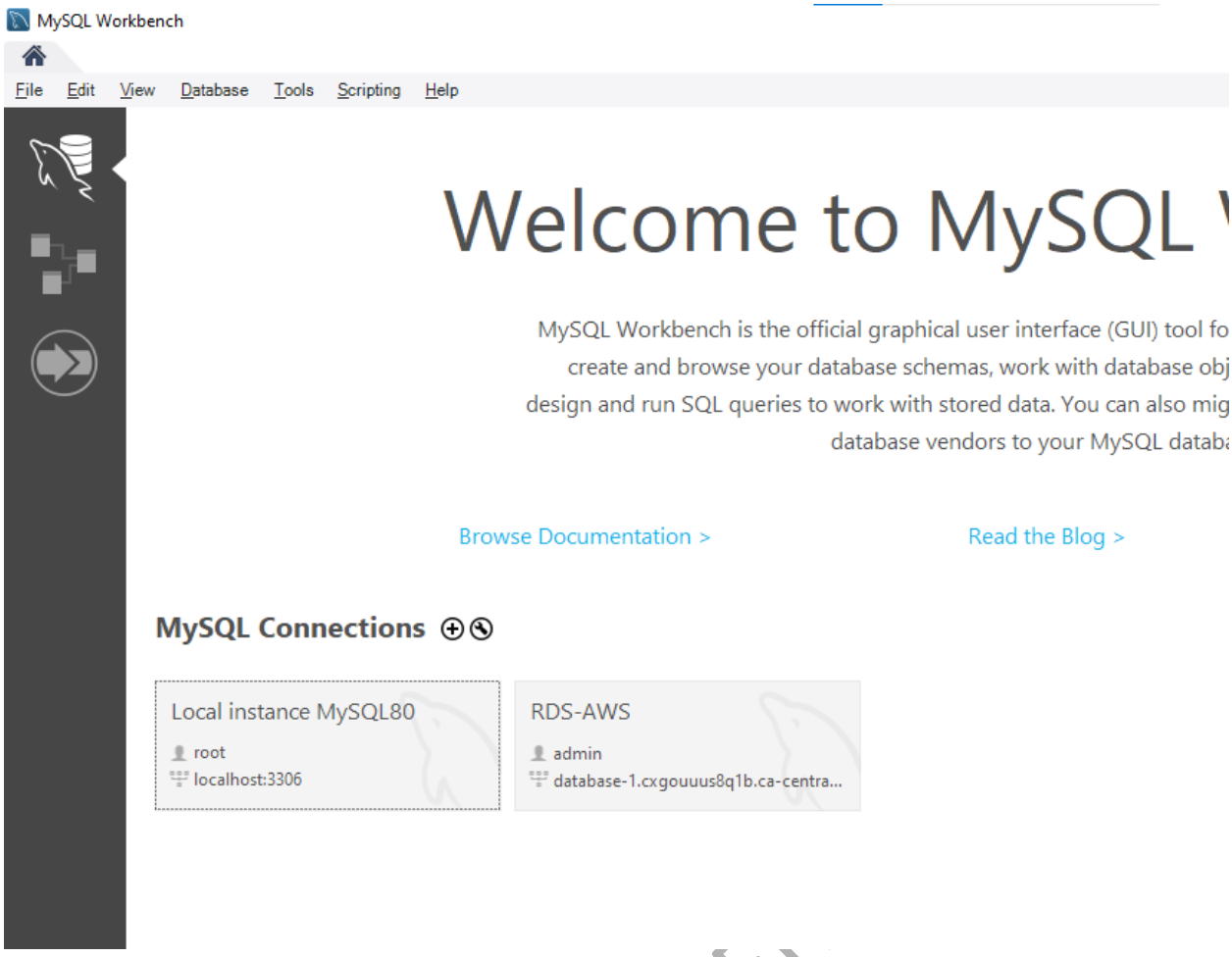
Click on the Store in Vault to enter the password



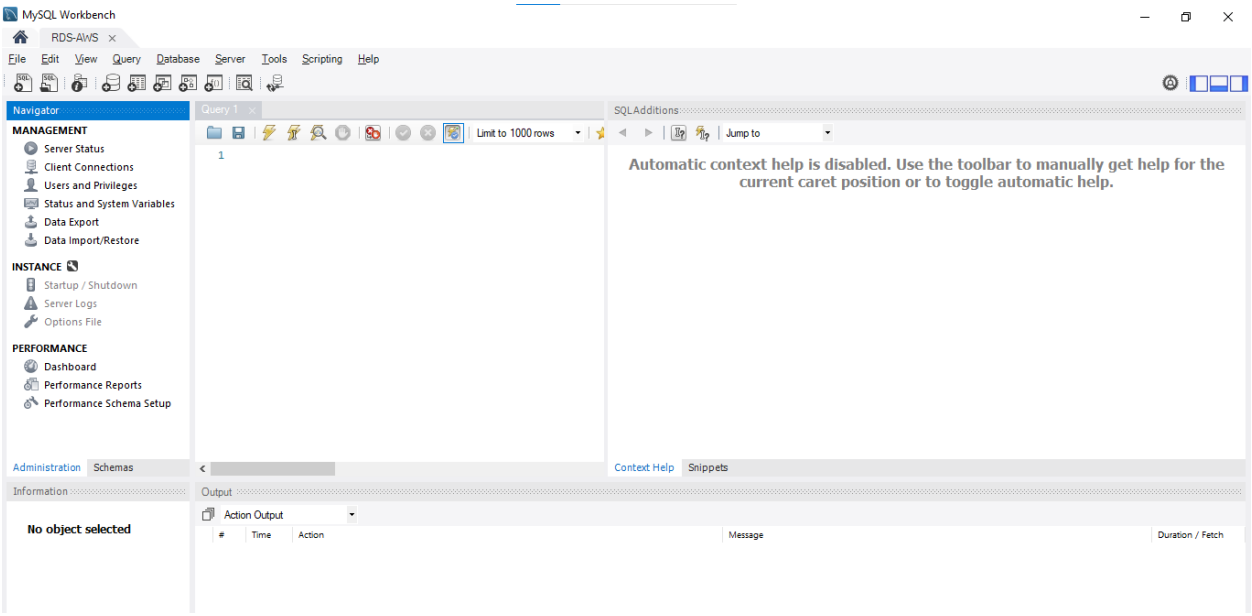
Click on OK and then click on Test connection.



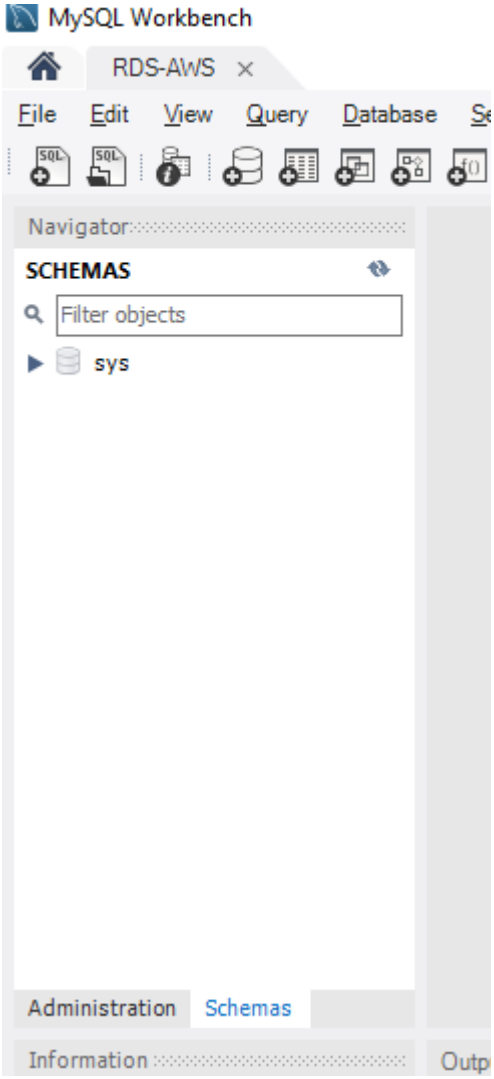
Connection got successful



Click on our RDS-AWS db to create database and tables.

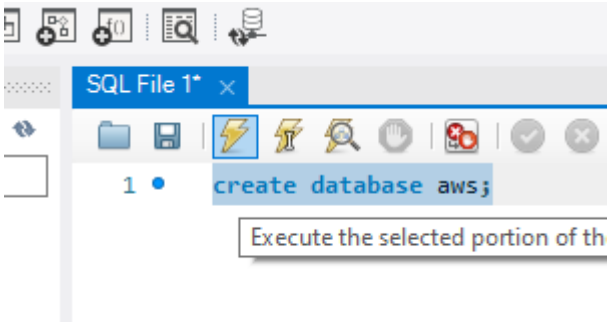


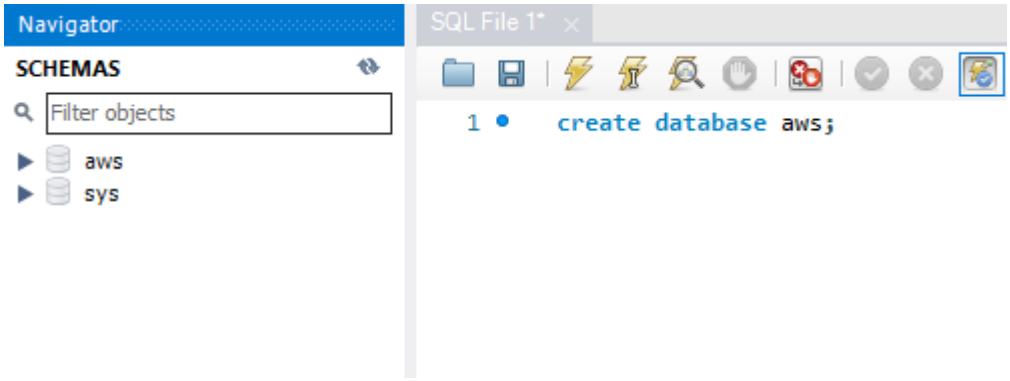
Click on Schemas to see the databases.



Create Database, Table and rows

Click on the execute query button

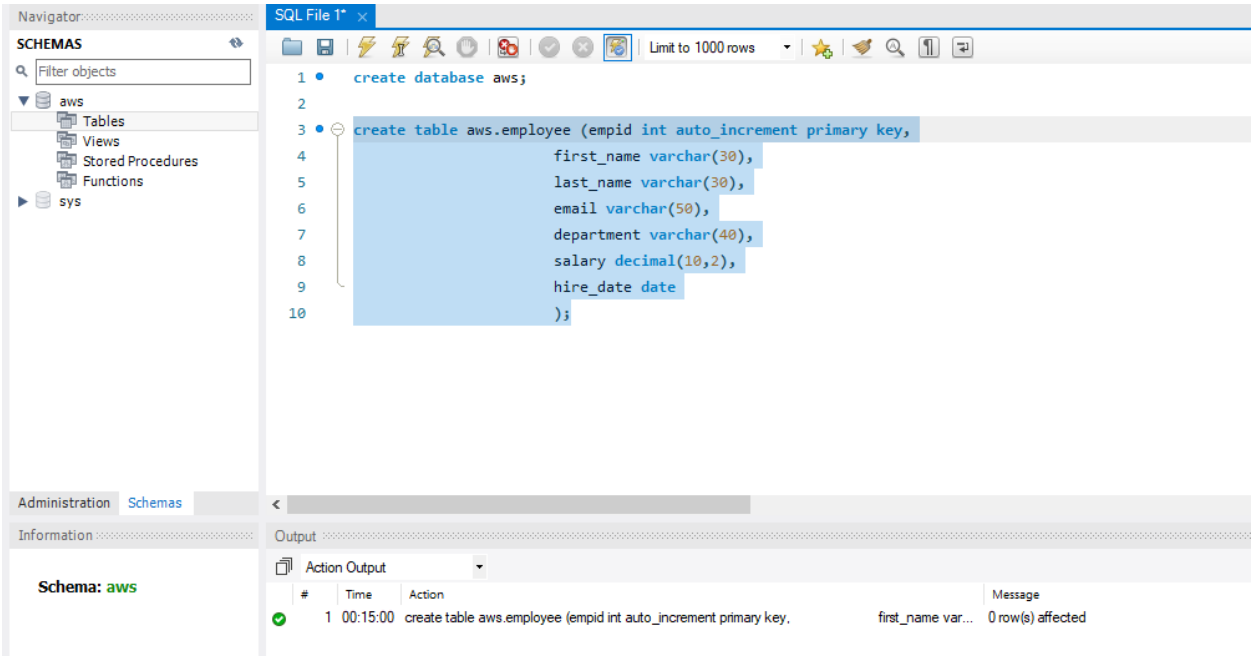


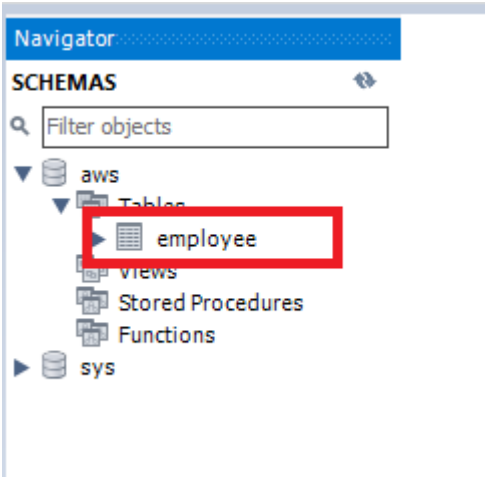


Aws db created

Run the below query to create a table in aws db


```
create table aws.employee (empid int auto_increment primary key,
first_name varchar(30),
last_name varchar(30),
email varchar(50),
department varchar(40),
salary decimal(10,2),
hire_date date
);
```





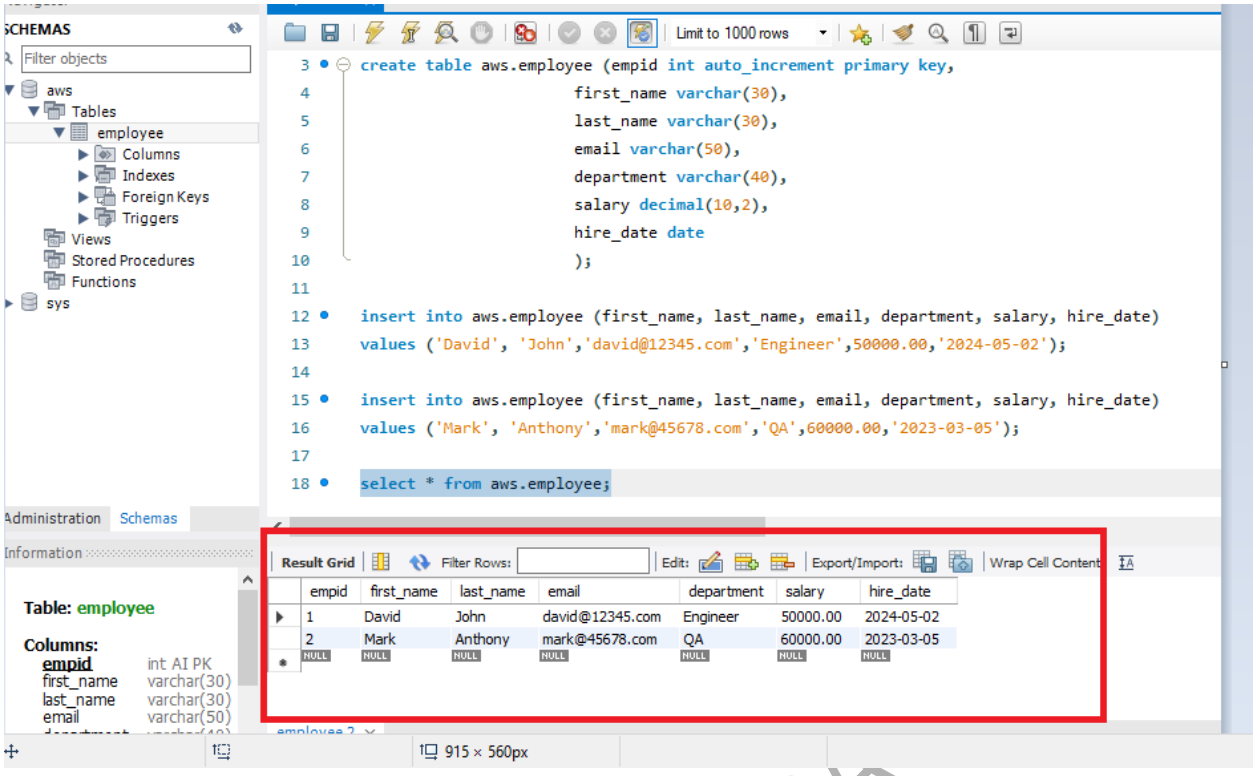
Employee table created

Run the below 2 query to insert the data and

```
insert into aws.employee (first_name, last_name, email, department, salary, hire_date)

values ('David', 'John','david@12345.com','Engineer',50000.00,'2024-05-02');


insert into aws.employee (first_name, last_name, email, department, salary, hire_date)

values ('Mark', 'Anthony','mark@45678.com','QA',60000.00,'2023-03-05');
```

Run the below query to view the data

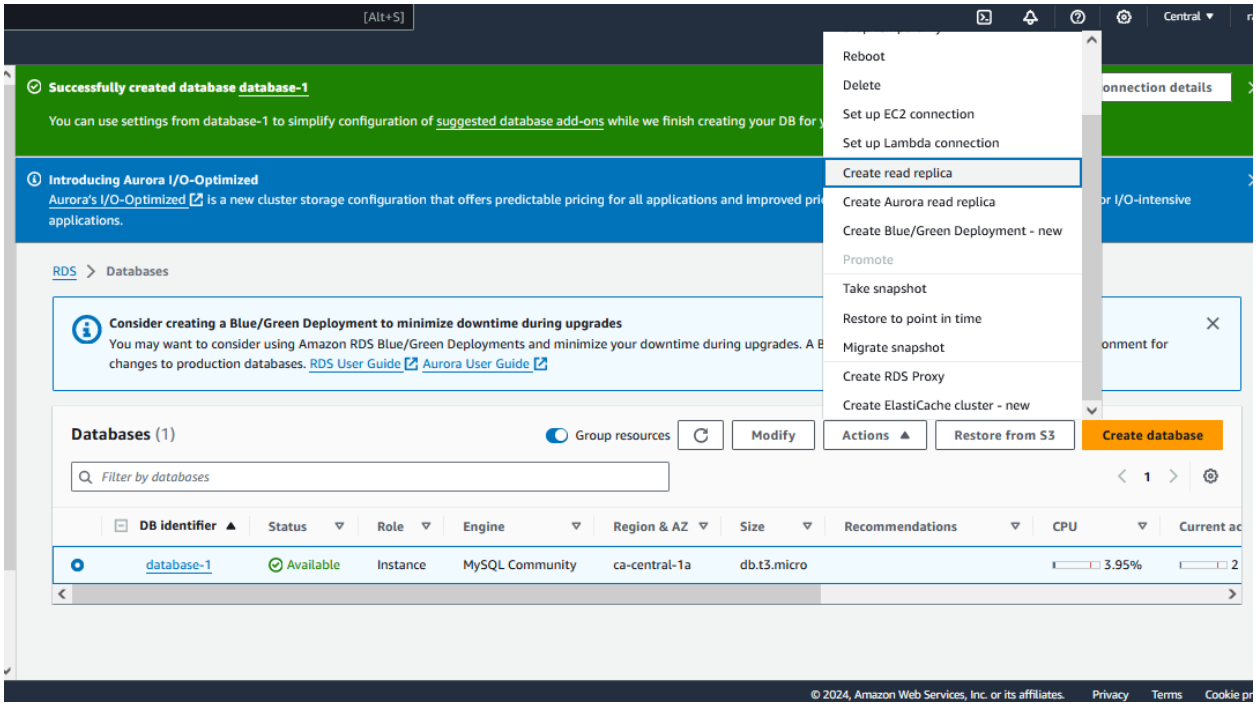
```
select * from aws.employee;
```

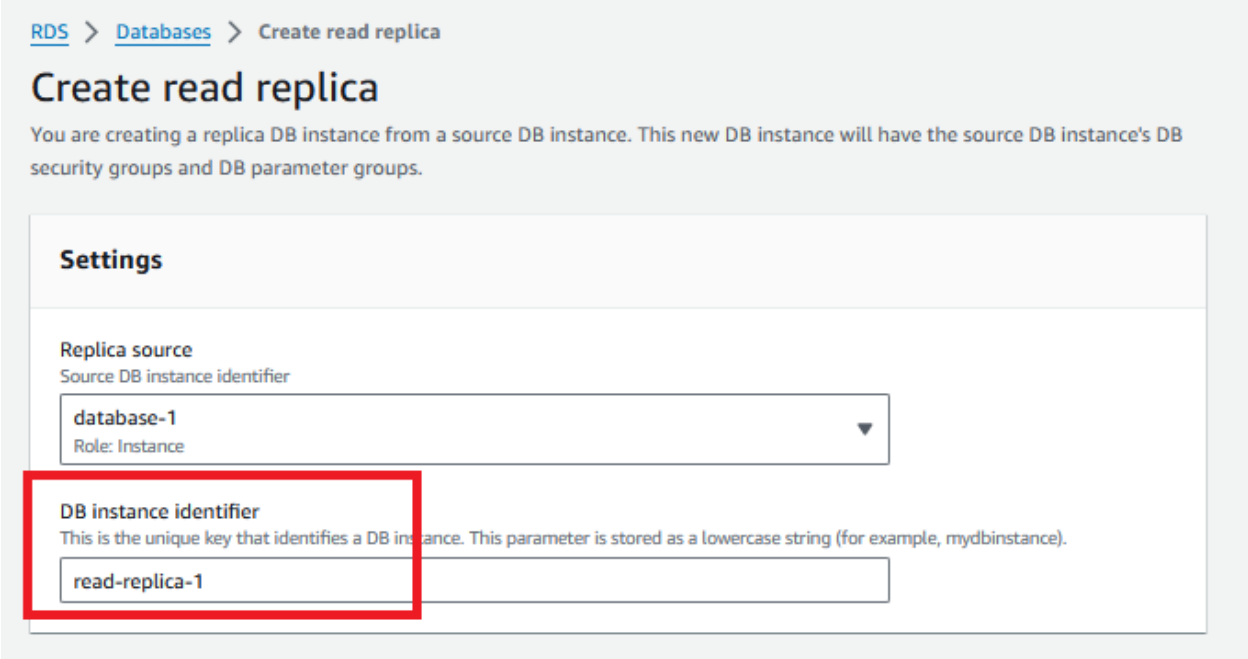


Now we able to see the data in employee table

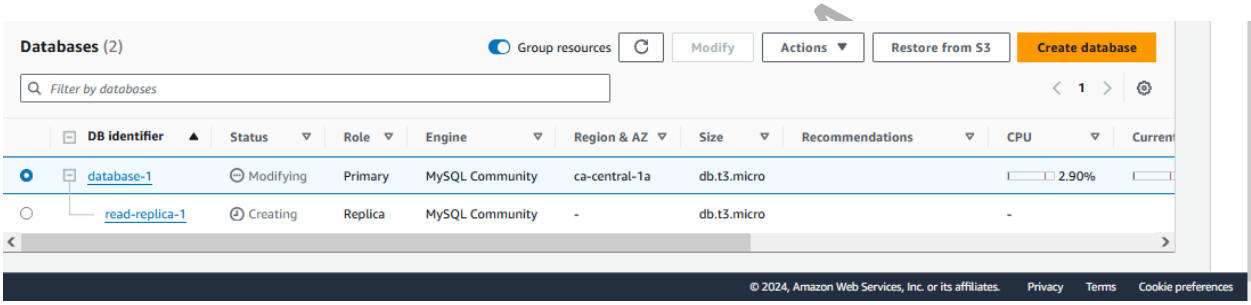
Create Read Replica from Master Data Base:

Select database and click on read replica

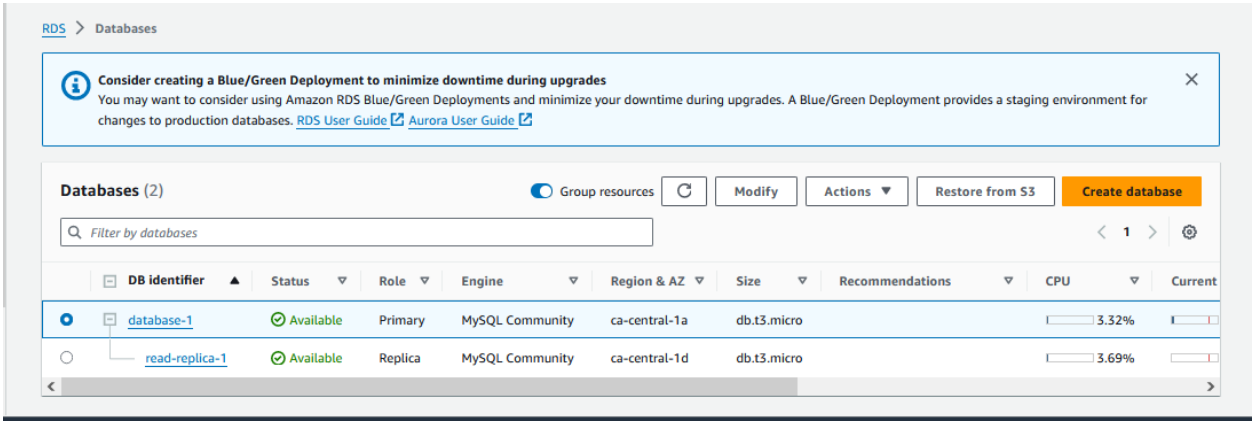




Provide read-replica-1 and the click on create read replica button

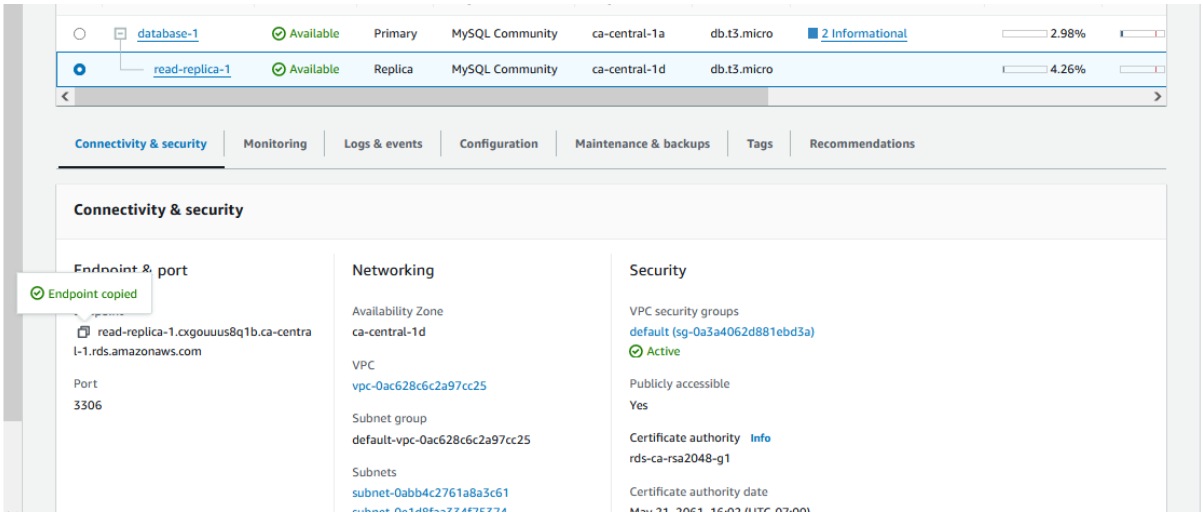


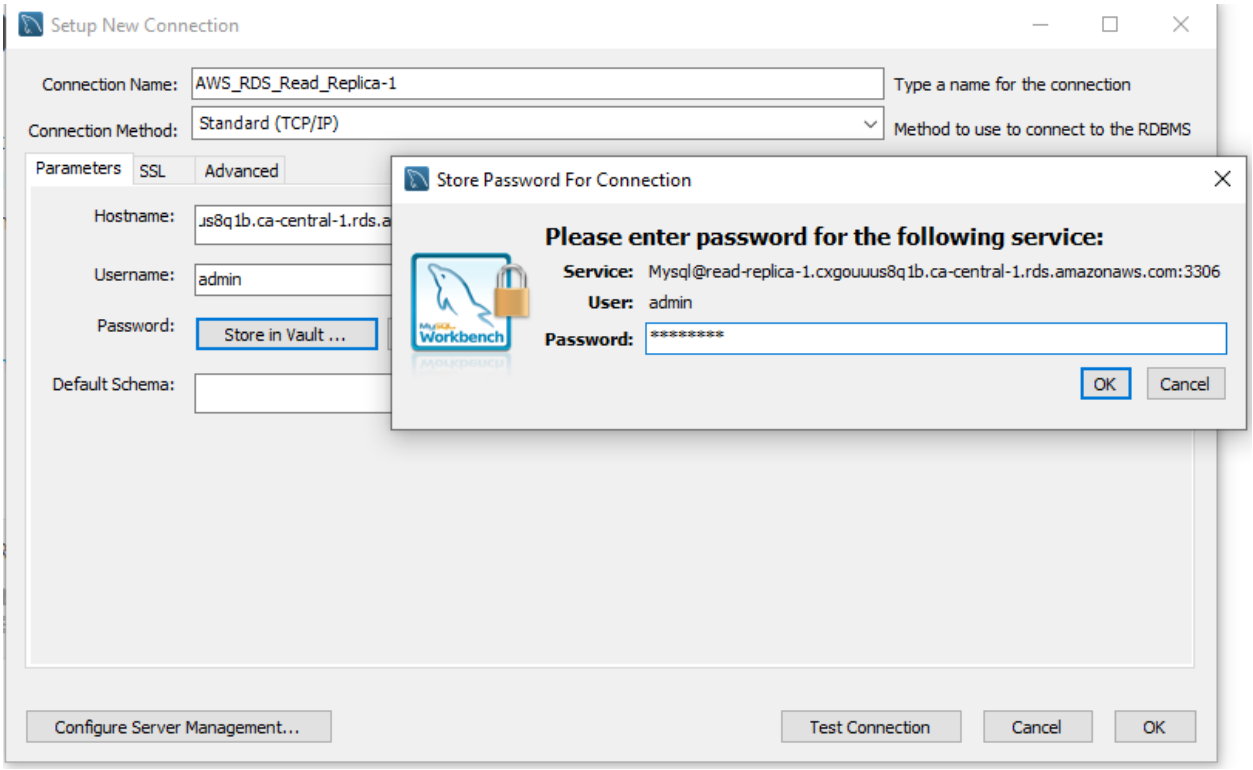
Lets wait for up and running

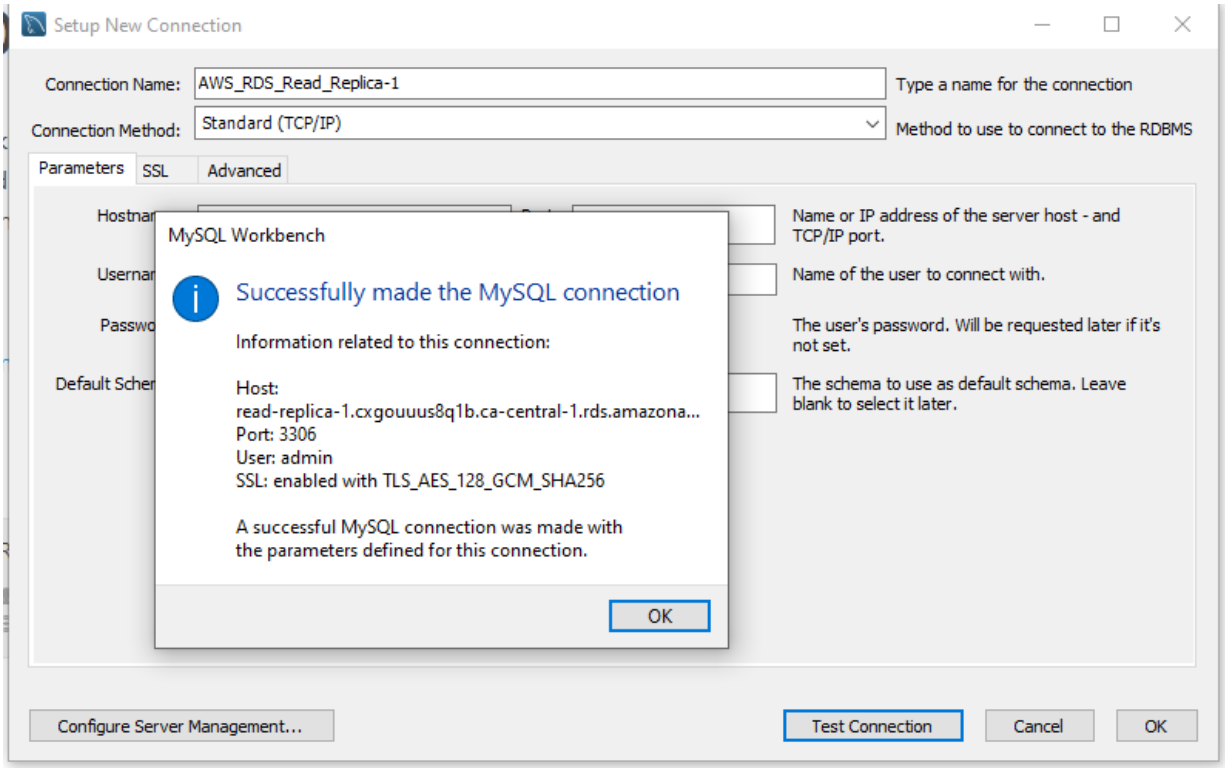


Both replica and Primary db available

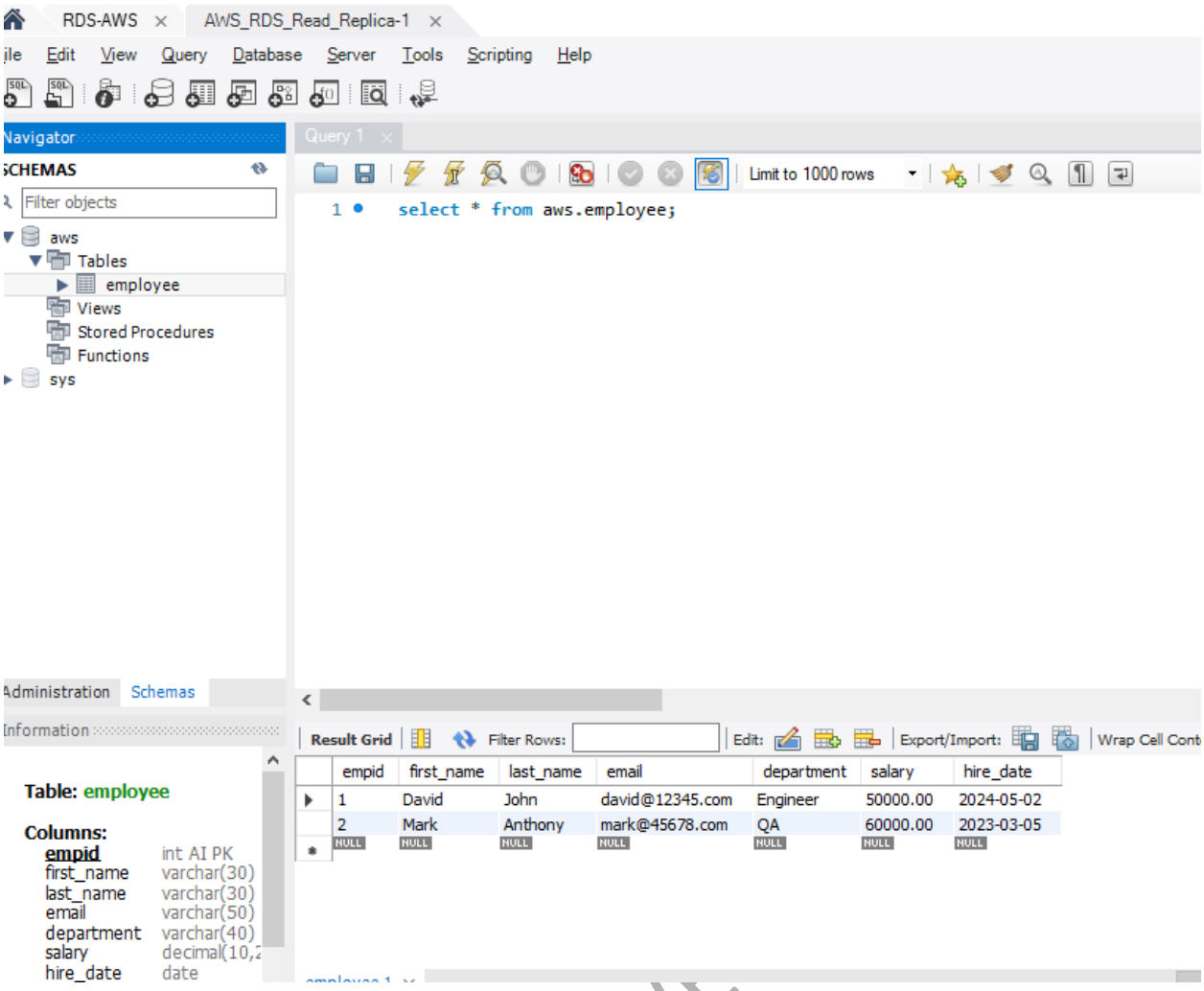
Open read-replica-1 to copy the endpoint to connect through mysql client.


database-1
Available
Primary
MySQL Community
ca-central-1a
db.t3.micro
2 Informational
2.98%
read-replica-1
Available
Replica
MySQL Community
ca-central-1d
db.t3.micro
4.26%
Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance & backups
Tags
Recommendations
Connectivity & security
Endpoint & port
Endpoint copied
read-replica-1.cxgouuus8q1b.ca-central-1.rds.amazonaws.com
Port
3306
Networking
Availability Zone
ca-central-1d
VPC
vpc-0ac628c6c2a97cc25
Subnet group
default-vpc-0ac628c6c2a97cc25
Subnets
subnet-0abb4c2761a8a3c61
Security
VPC security groups
default (sg-0a3a4062d881ebd3a)
Active
Publicly accessible
Yes
Certificate authority Info
rds-ca-rsa2048-g1
Certificate authority date


Setup New Connection
Connection Name:
AWS_RDS_Read_Replica-1
Type a name for the connection
Connection Method:
Standard (TCP/IP)
Method to use to connect to the RDBMS
Parameters
SSL
Advanced
Hostname:
Username:
admin
Password:
Store in Vault ...
Default Schema:
Configure Server Management...
Test Connection
Cancel
OK
Store Password For Connection
Please enter password for the following service:
Service:
Mysql@read-replica-1.cxgouuus8q1b.ca-central-1.rds.amazonaws.com:3306
User:
admin
Password:
****
OK
Cancel



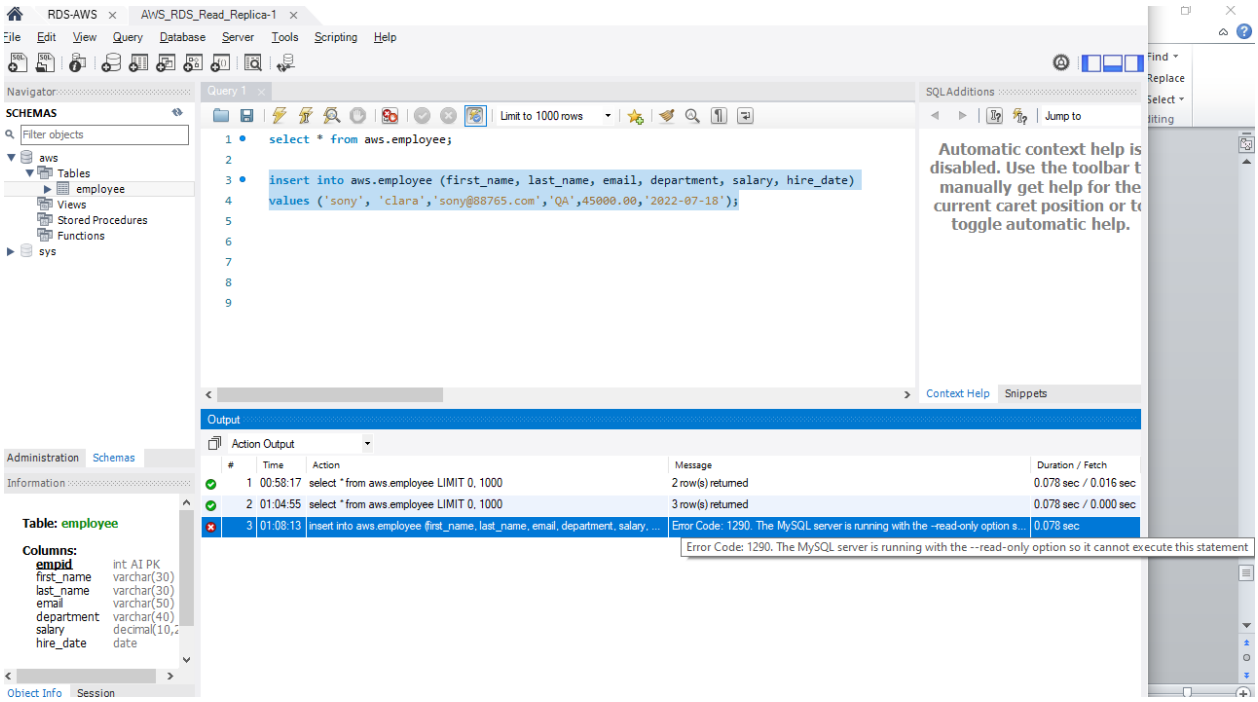
All table info available in read-replica-1 without creating any data.



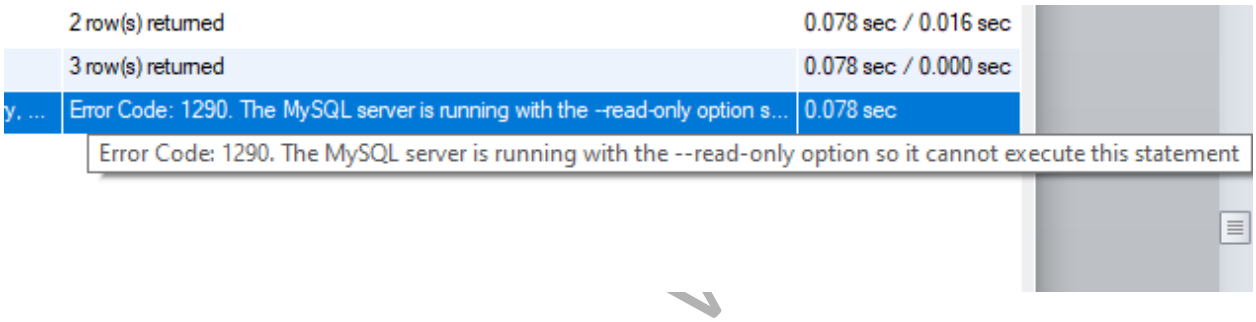
Create 3rd record in Master db

```
insert into aws.employee (first_name, last_name, email, department, salary, hire_date)

values ('Honey', 'Mony','honey@88765.com','QA',66000.00,'2022-08-22');
```

Now try to create 4th record in read-replica-1, lets see..

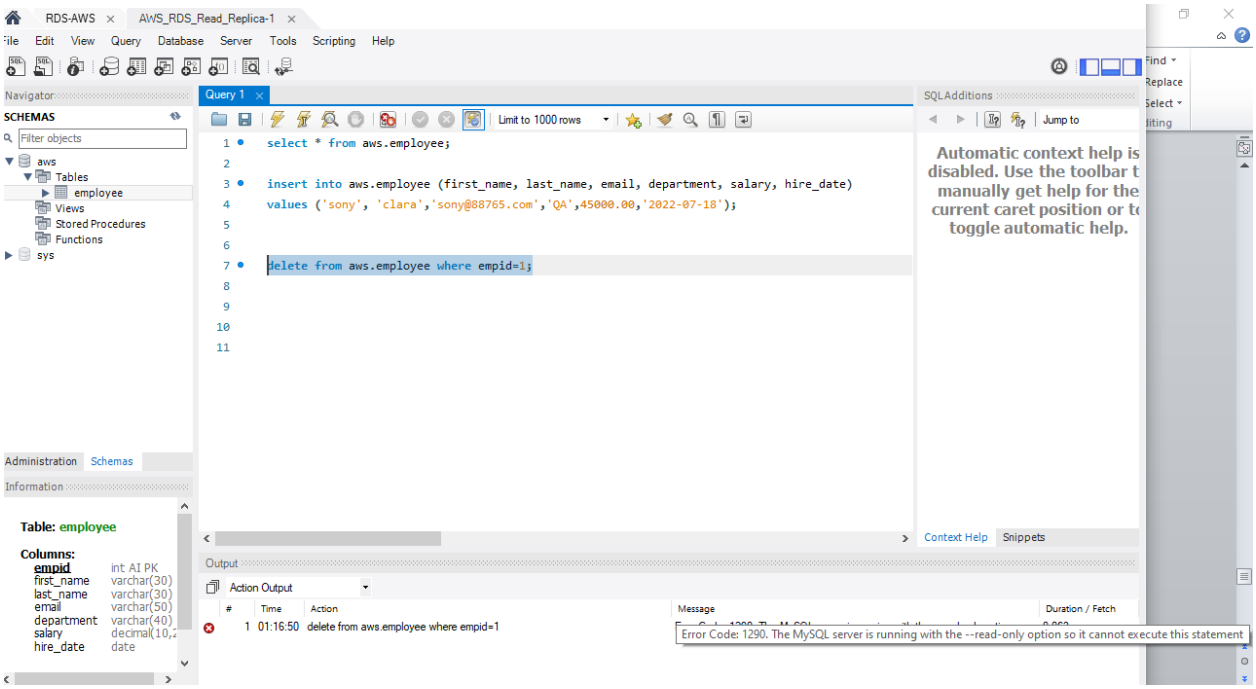


Error:

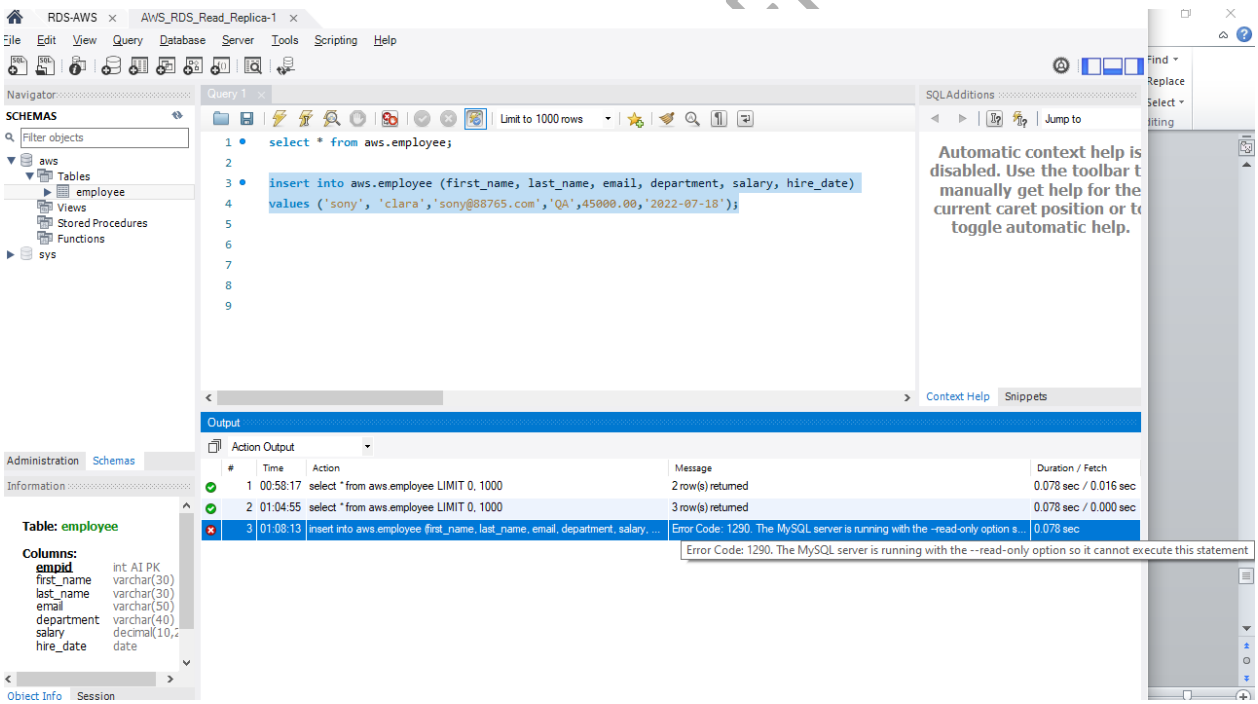


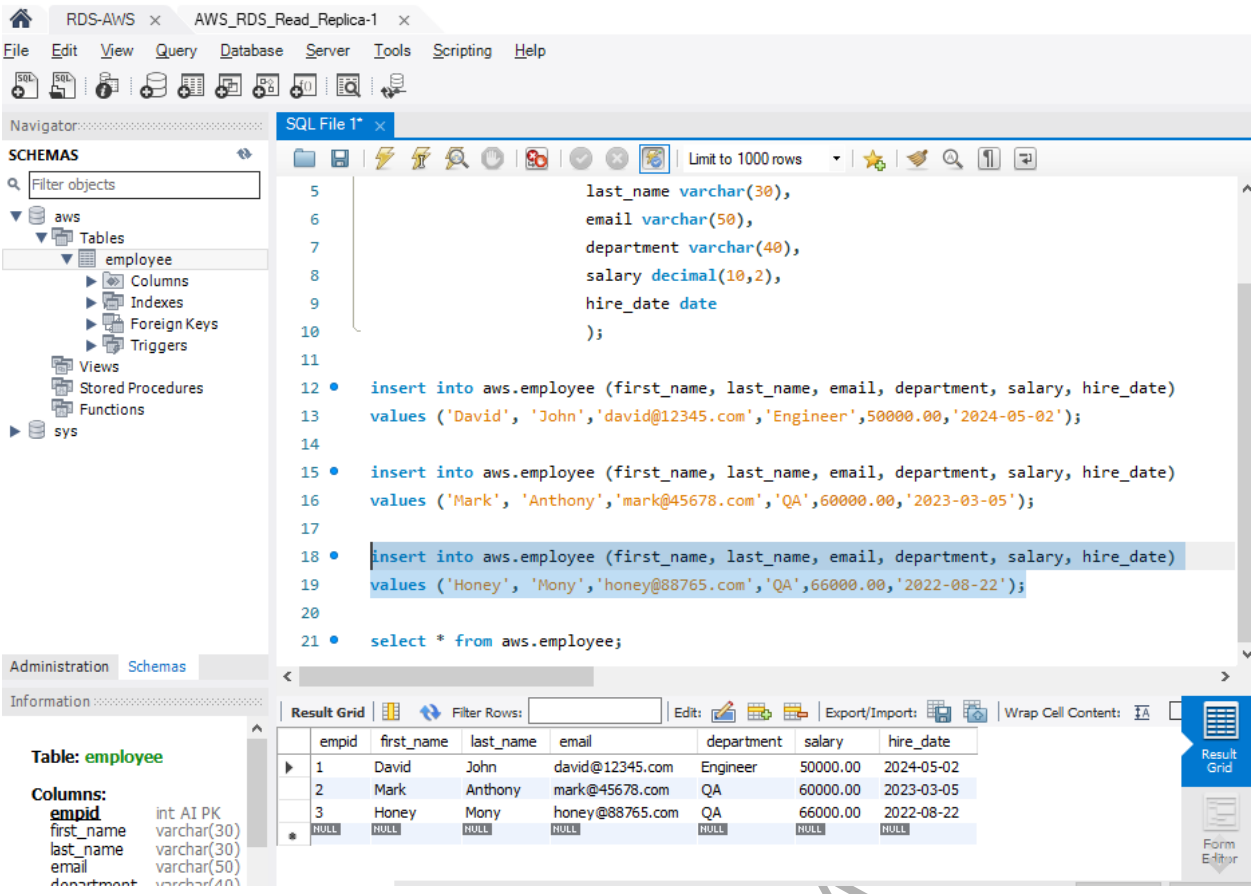
By seeing this error we have no access to execute insert, delete queries in read-replica-1

Try to delete a row from read-replica-1, lets see

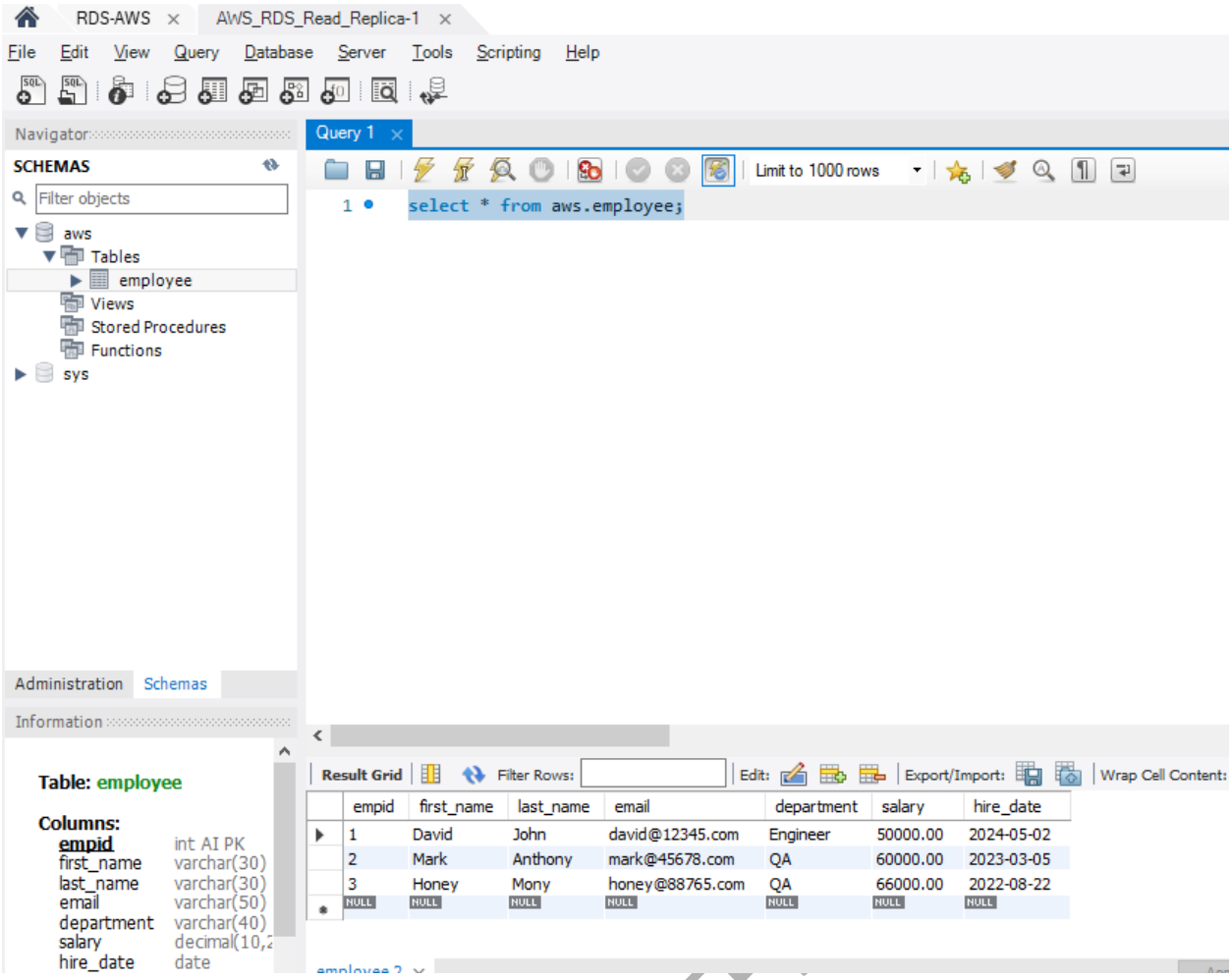


Delete query also not having the permission in read-replica-1 db.





The master data immediately reflecting into read-replica-1 db



verview of Amazon RDS (Relational Database Service)

Amazon RDS (Relational Database Service) is a fully managed relational database service provided by AWS. It allows you to run a relational database in the cloud without needing to worry about infrastructure management tasks such as hardware provisioning, database setup, patching, and backups. RDS makes it easy to set up, operate, and scale relational databases for web applications and other workloads.

Key Features of Amazon RDS:

1. Managed Service: Amazon handles the administrative tasks such as backups, patch management, and failover, reducing the operational overhead.
2. Multiple Database Engines: Supports popular relational database engines, including:
    - MySQL
    - PostgreSQL
    - MariaDB
    - Oracle
    - SQL Server

3. Scalability:
    - You can easily scale the compute (CPU, memory) and storage (storage size and throughput) resources for your database as your workload grows.
4. High Availability and Durability:
    - Multi-AZ Deployments: Ensures high availability by replicating data synchronously to a secondary instance in another Availability Zone.
    - Automated Backups: Amazon RDS supports point-in-time backups, allowing you to restore your database to any specific time.
5. Security:
    - Integration with AWS Identity and Access Management (IAM) for fine-grained access control.
    - Encryption at rest and in transit for data protection.
6. Automated Backups: RDS automatically backs up your database and retains backups for up to 35 days.
7. Monitoring: RDS integrates with Amazon CloudWatch to provide metrics on database performance (e.g., CPU utilization, disk I/O, query performance).

Amazon RDS Use Cases:

- Web Applications: Running the backend relational databases for web and mobile applications.
- Analytics and Data Warehousing: Storing structured data and running complex queries.
- Business Applications: Running enterprise applications that rely on relational databases, such as ERP and CRM systems.

Overview of RDS Read Replicas

An RDS Read Replica is a read-only copy of an RDS primary database instance that asynchronously replicates data from the primary database. It is designed to offload read-heavy database traffic from the primary instance, improving read scalability and reducing latency for read requests.

Key Features of RDS Read Replicas:

1. Asynchronous Replication:
    - Data is asynchronously copied from the primary database to the read replica, meaning there may be a slight delay (lag) between the primary and the replica.

- This lag is usually small (seconds to a few minutes) but may vary depending on traffic and database size.

2. Improved Read Performance:
   - Offload read queries to the replica instead of the primary database, improving overall application performance.
   - Multiple read replicas can be created to handle large numbers of read queries in parallel.
3. Geographic Distribution:
   - Cross-Region Replication: You can create read replicas in a different AWS region to serve global users with lower latency.
   - This is useful for improving application performance for users far away from the primary database's region.
4. Scalability:
   - Adding more read replicas can horizontally scale the read capacity of your database, allowing you to handle higher read traffic without affecting the primary instance.
5. Failover and Disaster Recovery:
   - Although read replicas are typically used for scaling read traffic, in the case of a failure in the primary instance, you can promote a read replica to become the new primary database instance.
   - This promotes fault tolerance and enhances availability, especially when used in conjunction with Multi-AZ deployments.
6. Backup and Maintenance:
   - Unlike the primary instance, read replicas don't have automatic backups or maintenance windows, but you can manually back them up if needed.
7. Write Restrictions:
   - Read replicas can only serve read queries. You cannot write data directly to the replica. If you need to update data, it must be done on the primary instance, which will asynchronously replicate the changes to the replica.

How RDS Read Replicas Work

1. Replication Process:

- Asynchronous: The replication process is asynchronous, meaning the primary instance and the read replica are not always perfectly in sync. This ensures low latency for write operations on the primary instance.
- Replication Mechanism: For engines like MySQL and PostgreSQL, Amazon RDS uses database-level replication to synchronize data between the primary database and the replica. Changes made to the primary instance are propagated to the read replica.

2. Creating Read Replicas:
   - You create read replicas from an existing RDS primary instance. AWS offers the option to create a read replica through the RDS management console, CLI, or API.
   - When creating a replica, you can choose different instance sizes and even place them in a different availability zone or region.
3. Read Traffic Distribution:
   - Applications can direct read queries to the read replica(s) by using the endpoint provided for the replica.
   - The primary database instance continues to handle write operations, while the read replicas handle read queries.
4. Promotion of Read Replicas:
   - If needed, you can promote a read replica to become the primary database instance. This is often done in the case of disaster recovery or if you want to change the primary instance for performance reasons.
5. Replication Lag:
   - There is a small delay between when a change occurs on the primary instance and when it is reflected on the replica (known as replication lag).
   - Typically, this lag is very small, but it can vary based on the database load and network conditions.

Benefits of Using RDS Read Replicas

- Offload Read Traffic: Read replicas can handle read-heavy workloads, which reduces the load on the primary instance and helps scale applications.
- Improved Performance: By distributing read queries across multiple replicas, you can significantly improve the performance of your application, particularly in read-intensive applications like reporting, analytics, or caching.

- High Availability and Disaster Recovery: Read replicas can serve as a backup in case of primary instance failure. In an emergency, you can promote a read replica to become the new primary instance, which reduces downtime.
- Global Reach: Read replicas can be placed in different AWS regions, providing faster read access to users globally.

Limitations of RDS Read Replicas

- Read-Only: Read replicas are read-only, which means you cannot perform write operations directly on them.
- Replication Lag: Since replication is asynchronous, there is always a slight delay between changes made in the primary instance and when they are reflected in the replica.
- Not a Full Backup: Read replicas are not intended to be used as backups for disaster recovery, though you can promote them to become a standalone database in case of failover.

Use Cases for RDS Read Replicas

1. Scalable Web Applications: When your application has a high volume of read queries (e.g., e-commerce sites, content management systems), you can use read replicas to offload read traffic and enhance performance.
2. Analytics: Use read replicas to run analytics and reporting queries without affecting the performance of the production database.
3. Disaster Recovery: Promote a read replica to the primary in case of failure or for planned migrations.
4. Global Applications: Deploy read replicas in different regions to reduce latency and improve performance for geographically distributed users.

AWS CloudWatch

AWS CloudWatch

Amazon CloudWatch is a powerful monitoring and observability service from AWS that provides real-time insights into the performance, health, and operational status of your AWS resources and applications. It helps you collect and track metrics, collect and monitor log files, set alarms, and automatically react to changes in your AWS environment.

CloudWatch is primarily used for:

- Monitoring cloud resources and applications
- Collecting and storing log files
- Creating alarms based on specific metrics
- Visualizing data through dashboards
- Automating responses to events and thresholds

Core Features of AWS CloudWatch

1. CloudWatch Metrics:
    - Metrics are the fundamental data points that CloudWatch uses to monitor the performance of AWS resources. AWS services (like EC2, S3, RDS, Lambda, etc.) automatically send metrics to CloudWatch at regular intervals (typically every minute).
    - These metrics can include information like:
        - CPU utilization, memory usage, disk I/O, network traffic for EC2 instances
        - Request count, latency, error rates for AWS services like ELB, API Gateway
        - DB connections, read/write latency, free storage space for Amazon RDS
    - You can also publish custom metrics from your own applications, giving you flexibility to monitor any data that is important to your specific use case.
2. CloudWatch Alarms:
    - CloudWatch Alarms allow you to define thresholds for specific metrics. When these thresholds are breached (e.g., CPU utilization exceeds 80%), CloudWatch can trigger automated actions such as:
        - Sending notifications via Amazon SNS (Simple Notification Service)
        - Executing AWS Lambda functions
        - Auto-scaling actions (e.g., scaling EC2 instances up/down)
    - You can configure alarms based on any metric, including custom metrics.
3. CloudWatch Logs:

- CloudWatch Logs helps you collect and monitor log data from AWS services, EC2 instances, Lambda functions, and other sources.
- Log data from your applications, system events, or security logs can be pushed to CloudWatch for centralized storage and real-time analysis.
- Features of CloudWatch Logs include:
  - Log Groups and Log Streams: Group and organize logs from different sources.
  - Log Retention: Automatically delete old logs after a specified retention period.
  - Metric Filters: Create CloudWatch metrics from specific patterns in your logs (e.g., error codes, application events).
  - Real-time Log Monitoring: Detect specific events or errors in your logs and set alarms for specific log patterns.

4. CloudWatch Dashboards:
   - CloudWatch Dashboards provide a customizable, graphical interface for monitoring and visualizing your metrics and logs. You can create dashboards to display multiple metrics from different AWS services and even custom metrics.
   - Dashboards can be shared with other team members, and you can view data in real time, track historical trends, and gain operational insights.
   - You can visualize metrics like CPU utilization, application response time, or any custom metric across multiple AWS resources.
5. CloudWatch Events (Now part of Amazon EventBridge):
   - CloudWatch Events enables you to respond to changes in your AWS environment in real time. It captures system events (like resource state changes, scheduled tasks, or application events) and routes them to targets (like Lambda functions, SNS topics, SQS queues).
   - With EventBridge, CloudWatch Events now supports both AWS service events and custom events from your own applications. It can also connect to third-party services and integrate with other event-driven systems.
6. CloudWatch Synthetics:
   - CloudWatch Synthetics allows you to monitor the availability and performance of your web applications by creating canaries, which are automated scripts that simulate user activity and test your endpoints (APIs, web applications) at regular intervals.
   - This helps ensure that your application is functioning correctly and provides insights into the health of your resources from an end-user perspective.
7. CloudWatch Contributor Insights:
   - Contributor Insights is a feature that analyzes and visualizes log data to identify and prioritize contributors to high load or performance issues.

- For example, you can use Contributor Insights to identify the top IP addresses causing high latencies or the services contributing to errors, helping you pinpoint issues in your environment quickly.

8. CloudWatch ServiceLens:
    - CloudWatch ServiceLens provides a unified view of your application's performance, helping you monitor, visualize, and analyze traces across multiple AWS services.
    - It integrates with AWS X-Ray for distributed tracing, allowing you to drill down into specific requests and track them through microservices, identifying bottlenecks and performance issues.

How AWS CloudWatch Works

1. Collecting Data:
    - CloudWatch automatically collects metrics from a wide variety of AWS services. For instance, Amazon EC2 instances send metrics like CPU utilization, memory usage, and disk I/O to CloudWatch.
    - For custom data, you can create your own metrics and publish them to CloudWatch using the AWS SDK or API. This allows you to monitor anything that's important to your application, such as request counts, business KPIs, or application health.
2. Setting Up Alarms:
    - Once CloudWatch is collecting metrics, you can set up CloudWatch Alarms to monitor specific metrics. If a metric exceeds or falls below a predefined threshold, the alarm can send a notification, trigger a Lambda function, or start an auto-scaling action.
    - For example, if the CPU utilization of an EC2 instance exceeds 80%, you can trigger an alarm to automatically scale the instance or send an alert via email to the DevOps team.
3. Storing and Analyzing Logs:
    - AWS CloudWatch Logs can be used to centralize the log data from all your AWS resources and applications. Logs are automatically stored in CloudWatch and can be analyzed for specific patterns using metric filters.
    - You can create custom metrics from logs (e.g., count occurrences of certain error messages or user actions) and use them to trigger alarms or visualize the trends on CloudWatch Dashboards.
4. Event-driven Automation:
    - CloudWatch Events (or EventBridge) can capture events from AWS services and send them to various targets, such as Lambda functions, SQS queues, or Step Functions. This allows for event-driven automation.

- For example, when an EC2 instance starts or stops, CloudWatch Events can trigger a Lambda function to perform additional tasks, such as updating a database or notifying a user.

Use Cases for AWS CloudWatch

1. Application Monitoring:
   - CloudWatch allows you to monitor application performance by tracking custom metrics (e.g., request count, error rates, response time) from your applications and microservices.
   - You can set up alarms based on these metrics to notify you when something goes wrong, such as an increase in error rate or slow response times.
2. Infrastructure Monitoring:
   - CloudWatch provides metrics for various AWS resources (EC2, RDS, ELB, Lambda, etc.) that allow you to track the health and performance of your infrastructure.
   - You can monitor resource utilization (CPU, memory, storage) and set up auto-scaling triggers to maintain performance as demand changes.
3. Security and Compliance:
   - CloudWatch can capture logs related to security events, such as login attempts, changes to IAM roles, and access to critical resources.
   - CloudWatch Logs can be integrated with services like AWS CloudTrail to help track API calls and monitor for any unauthorized access or anomalous behavior.
4. Business Metrics Monitoring:
   - CloudWatch can be used to monitor application-specific business metrics, such as transaction volume, sales data, and customer activity. This allows business teams to get real-time insights into application usage and performance.
5. Real-time Monitoring and Alerts:
   - Using CloudWatch dashboards, you can visualize your metrics in real-time and get alerts when certain thresholds are crossed. This helps in keeping an eye on the operational health of your AWS environment.

Benefits of AWS CloudWatch

1. Centralized Monitoring:
   - CloudWatch provides a single pane of glass to monitor and manage all your AWS resources, custom metrics, logs, and events in one place.
2. Real-time Insights:

- With CloudWatch, you can gain real-time insights into the health of your resources and applications, helping you quickly detect and respond to issues.

3. Automation:
   - You can automate actions in response to events or metrics, improving operational efficiency and reducing manual intervention.
4. Scalability:
   - CloudWatch scales automatically with your AWS resources. Whether you're running a few EC2 instances or thousands, CloudWatch can handle the volume of metrics and logs.
5. Cost-Effective:
   - You only pay for what you use, whether it's for storing logs, collecting metrics, or using alarms and dashboards. This pay-as-you-go pricing model helps keep costs manageable.
6. Security and Compliance:
   - CloudWatch integrates with AWS security and compliance tools, helping you monitor logs and events for regulatory and operational purposes.

Configure aws cloudwatch to ec2

Method 2

- Launch the ec2
- Attach the iam role {ec2- cloudwatch or admin acces}
- Connect to ec2
- Install cloud watch agent

yum install amazon-cloudwatch-agent -y

```
[root@ip-172-31-19-9 ec2-user]# sudo su -
Last login: Thu Oct 24 08:48:41 UTC 2024 on pts/1
[root@ip-172-31-19-9 ~]# yum install amazon-cloudwatch-agent -y
Last metadata expiration check: 1 day, 17:16:02 ago on Tue Oct 22 15:32:51 2024.
Dependencies resolved.
================================================================================
 Package                    Architecture          Version
================================================================================
Installing:
 amazon-cloudwatch-agent    x86_64                1.300044.0-1.amzn2023

Transaction Summary
================================================================================
Install  1 Package

Total download size: 135 M
Installed size: 445 M
Downloading Packages:
amazon-cloudwatch-agent-1.300044.0-1.amzn2023.x86_64.rpm
--------------------------------------------------------------------------------
Total
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :
```

- Edit this file or create file using below commnd

vi /opt/aws/amazon-cloudwatch-agent/bin/config.json

```
[root@ip-172-31-19-9 ~]# vi /opt/aws/amazon-cloudwatch-agent/bin/config.json
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
[root@ip-172-31-19-9 ~]#
```

- Add below json config detailes
- Change the highlighted details based on your requirement

```
{
    "agent": {
        "metrics_collection_interval": 60,
        "run_as_user": "root"
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/var/log/*",
                        "log_group_class": "STANDARD",
                        "log_group_name": "aws",
                        "log_stream_name": "devops",
                        "retention_in_days": 1
                    }
                ]
            }
        }
    },
    "metrics": {
        "aggregation_dimensions": [
            [
                "InstanceId"
            ]
        ],
        "append_dimensions": {
            "AutoScalingGroupName": "${aws:AutoScalingGroupName}",
            "ImageId": "${aws:ImageId}",
            "InstanceId": "${aws:InstanceId}",
            "InstanceType": "${aws:InstanceType}"
        },
        "metrics_collected": {
```

```
"cpu": {
        "measurement": [
                "cpu_usage_idle",
                "cpu_usage_iowait",
                "cpu_usage_user",
                "cpu_usage_system"
        ],
        "metrics_collection_interval": 60,
        "resources": [
                "*"
        ],
        "totalcpu": false
},
"disk": {
        "measurement": [
                "used_percent",
                "inodes_free"
        ],
        "metrics_collection_interval": 60,
        "resources": [
                "*"
        ]
},
"diskio": {
        "measurement": [
                "io_time"
        ],
        "metrics_collection_interval": 60,
        "resources": [
                "*"
        ]
},
"mem": {
        "measurement": [
                "mem_used_percent"
        ],
        "metrics_collection_interval": 60
},
"statsd": {
        "metrics_aggregation_interval": 60,
```

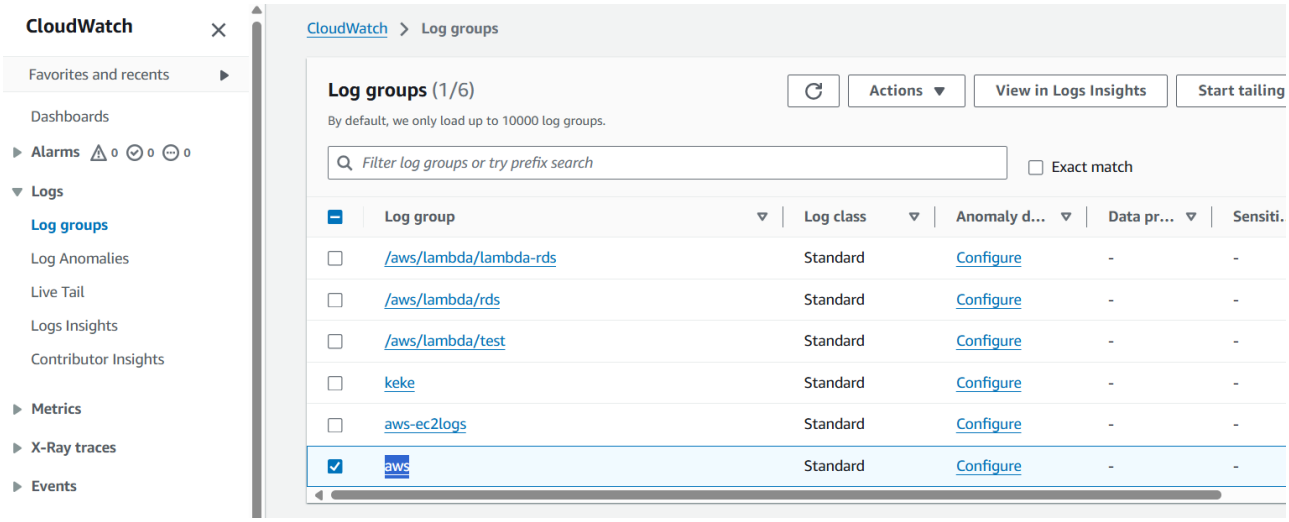
```
            "metrics_collection_interval": 10,
            "service_address": ":8125"
        },
        "swap": {
            "measurement": [
                "swap_used_percent"
            ],
            "metrics_collection_interval": 60
        }
    }
},
"traces": {
    "buffer_size_mb": 3,
    "concurrency": 8,
    "insecure": false,
    "region_override": "us-east-2",
    "traces_collected": {
        "xray": {
            "bind_address": "127.0.0.1:2000",
            "tcp_proxy": {
                "bind_address": "127.0.0.1:2000"
            }
        }
    }
}
}
```

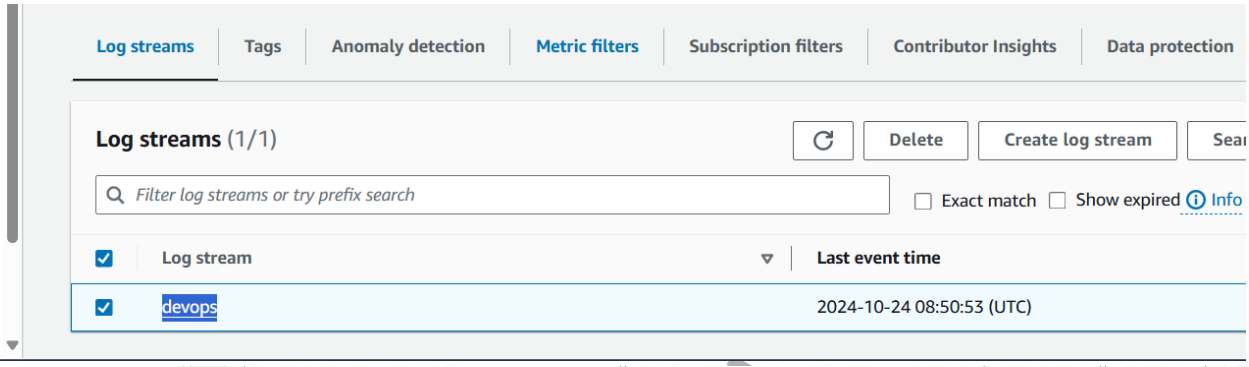

- Enter below command for strat the cloudwatch and creation of log group in cloud watch console based on config,json file

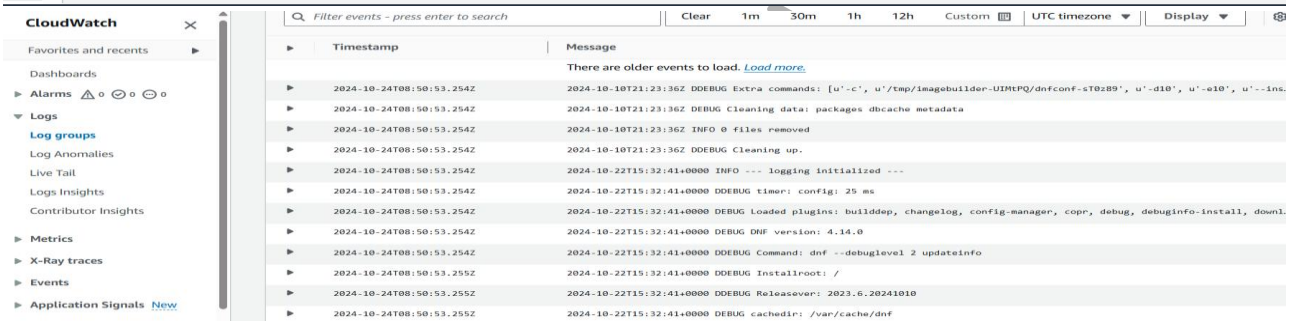
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json -s

```
[root@ip-172-31-19-9 ~]# sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/opt/aws/amazon-cloudwatch-agent
n/config.json -s
****** processing amazon-cloudwatch-agent ******
I! Trying to detect region from ec2 D! [EC2] Found active network interface I! imds retry client will retry 1 timesSuccessfully fetched the config and saved i
opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_config.json.tmp
Start configuration validation...
2024/10/24 08:50:51 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_config.json.tmp ...
2024/10/24 08:50:51 I! Valid Json input schema.
2024/10/24 08:50:51 D! ec2tagger processor required because append_dimensions is set
2024/10/24 08:50:51 D! delta processor required because metrics with diskio or net are set
2024/10/24 08:50:51 D! ec2tagger processor required because append_dimensions is set
2024/10/24 08:50:51 Configuration validation first phase succeeded
I! Detecting run_as_user...
I! Trying to detect region from ec2
D! [EC2] Found active network interface
I! imds retry client will retry 1 times
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -schematest -config /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml
Configuration validation second phase succeeded
Configuration validation succeeded
amazon-cloudwatch-agent has already been stopped
Created symlink /etc/systemd/system/multi-user.target.wants/amazon-cloudwatch-agent.service → /etc/systemd/system/amazon-cloudwatch-agent.service.
```

- Open your cloud watch and check


CloudWatch
Favorites and recents
Dashboards
Alarms
Logs
Log groups
Log Anomalies
Live Tail
Logs Insights
Contributor Insights
Metrics
X-Ray traces
Events
CloudWatch > Log groups
Log groups (1/6)
Actions
View in Logs Insights
Start tailing
By default, we only load up to 10000 log groups.
Filter log groups or try prefix search
Exact match
Log group
Log class
Anomaly d...
Data pr...
Sensiti.
/aws/lambda/lambda-rds
/aws/lambda/rds
/aws/lambda/test
keke
aws-ec2logs
aws
Standard
Configure


Log streams
Tags
Anomaly detection
Metric filters
Subscription filters
Contributor Insights
Data protection
Log streams (1/1)
Delete
Create log stream
Filter log streams or try prefix search
Exact match
Show expired
Info
Log stream
Last event time
devops
2024-10-24 08:50:53 (UTC)


CloudWatch
Favorites and recents
Dashboards
Alarms
Logs
Log groups
Log Anomalies
Live Tail
Logs Insights
Contributor Insights
Metrics
X-Ray traces
Events
Application Signals New
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display
Timestamp
Message
There are older events to load. Load more.

Certificate manager and route53

Overview of Amazon Route 53

Amazon Route 53 is a scalable and highly available Domain Name System (DNS) web service provided by AWS. It is designed to route end-user requests to various AWS resources, including EC2 instances, S3 buckets, CloudFront distributions, and other domain names, using DNS. Route 53 also integrates with AWS services, making it an ideal solution for managing DNS in the cloud.

The service is named after DNS (port 53), the protocol used for DNS queries.

Route 53 is fully managed and offers several features, including domain registration, DNS routing, health checking, and traffic management.

Key Features of Amazon Route 53

1. DNS Management
    - DNS Resolution: Route 53 translates friendly domain names (like www.example.com) into IP addresses that computers use to communicate with each other (e.g., 192.0.2.1).
    - Scalability: Route 53 can handle large volumes of DNS queries without requiring additional configuration or resources, scaling automatically based on demand.
    - Low Latency: AWS has a global network of DNS servers, so Route 53 can respond to DNS queries from users with low latency.
2. Domain Registration
    - Domain Purchase and Transfer: Route 53 allows you to register new domain names directly through AWS or transfer existing domains to AWS.
    - WHOIS Privacy: For supported TLDs (Top-Level Domains), WHOIS privacy can be enabled to protect personal information associated with domain registration.
    - Automatic Renewal: Domains can be set to renew automatically, avoiding potential disruptions in service due to expired domains.

3. Traffic Routing and Load Balancing
    - Simple Routing: Route 53 can map domain names to one or more IP addresses, making it a basic way to resolve domain names to endpoints like web servers.
    - Weighted Routing: Route 53 supports weighted routing policies that allow you to distribute traffic between different resources (e.g., EC2 instances) based on a set of weights. This is useful for gradual migrations, blue/green deployments, or testing purposes.
    - Latency-Based Routing: Route 53 can route traffic to the region with the lowest latency for the end-user, improving response time.
    - Geo-Location Routing: Route 53 can direct traffic based on the geographic location of users (e.g., route European users to a European server, and U.S. users to a U.S.-based server).
    - Failover Routing: This allows Route 53 to detect health issues with a resource and route traffic to a backup (failover) resource automatically if the primary resource becomes unavailable.
    - Multivalue Answer: Similar to simple routing, but Route 53 can return multiple IP addresses for a domain name. This allows for automatic load balancing across multiple endpoints.
4. Health Checking and Monitoring
    - Health Checks: Route 53 can monitor the health of resources (e.g., web servers, load balancers) by periodically sending HTTP(S) requests. If the resource is found to be unhealthy, Route 53 will redirect traffic to healthy endpoints based on routing policies (e.g., failover, weighted routing).
    - CloudWatch Integration: Route 53 integrates with Amazon CloudWatch to monitor DNS query activity and health check results, providing visibility into the health of your infrastructure.
5. DNS Failover
    - DNS Failover: This feature helps ensure high availability by automatically rerouting traffic to a healthy resource when a failure is detected. It is typically used in conjunction with Route 53 health checks.
6. Alias Records
    - Alias Records: Unlike traditional DNS records, Alias records allow you to map a domain name to AWS resources (e.g., S3 buckets, CloudFront distributions, Elastic Load Balancers) without needing an IP address. This is particularly useful for integrating Route 53 with AWS services.

- No Additional Cost for Alias Records: Alias records are treated as native DNS records, and AWS doesn't charge for query traffic to them (unlike standard CNAME records).

7. DNSSEC (Domain Name System Security Extensions)
    - DNSSEC Support: Route 53 supports DNSSEC to add an additional layer of security, helping to protect against DNS spoofing and cache poisoning attacks. With DNSSEC, responses to DNS queries are signed to ensure that users are directed to the correct website.
8. Traffic Flow and Global Traffic Management
    - Traffic Flow: Route 53's visual interface, Traffic Flow, allows you to create complex routing policies, including weighted, latency-based, geolocation, and failover routing, in an easy-to-use interface.
    - Geoproximity Routing: Route 53 can route traffic based on the geographic location of users and resources, which is useful for managing global traffic.
9. Integration with AWS Services
    - AWS CloudFront: Easily integrate Route 53 with Amazon CloudFront, AWS's Content Delivery Network (CDN), for faster and more secure delivery of web content globally.
    - Elastic Load Balancing (ELB): Route 53 works seamlessly with AWS's Elastic Load Balancers to route traffic to your EC2 instances, providing load balancing and scaling.
    - Amazon S3: You can use Route 53 to route traffic to Amazon S3 static websites or S3 buckets.
    - AWS Elastic Beanstalk: Route 53 can be configured to manage traffic for applications deployed using Elastic Beanstalk.

Key Benefits of Amazon Route 53

1. Highly Available and Scalable: Route 53 is designed to be highly available with a global network of DNS servers and auto-scaling to handle millions of DNS queries with low latency.
2. Ease of Use: The Route 53 console offers a simple and easy-to-navigate interface for managing DNS records and configuring routing policies.
3. Integrated with AWS: Route 53 integrates tightly with other AWS services, enabling easy management of DNS for AWS resources (like EC2, S3, CloudFront, and more).

4. Cost-Effective: Route 53 offers pay-as-you-go pricing, meaning you only pay for the DNS queries made and the resources you use. This makes it an economical choice for businesses of all sizes.
5. Security: Route 53 supports DNSSEC for domain name validation and integrates with AWS IAM for access control. Additionally, its health checks can help automatically reroute traffic during failures, ensuring continuity.
6. Global Traffic Management: With features like geo-location routing, latency-based routing, and weighted routing, Route 53 enables intelligent traffic management to improve application performance and ensure a better user experience across the globe.

Common Use Cases for Amazon Route 53

1. Website Hosting:
    - Route 53 can route users to AWS-hosted websites, such as static sites hosted on S3 or dynamic applications hosted on EC2 instances.
2. Disaster Recovery:
    - With DNS failover and health checks, Route 53 can reroute traffic to a healthy resource in the event of an outage or server failure, enabling high availability and business continuity.
3. Global Traffic Management:
    - Businesses with a global customer base can use geolocation or latency-based routing to ensure that users are directed to the nearest or fastest server, improving load times and user experience.
4. Multi-Region Applications:
    - Companies with applications deployed across multiple AWS regions can use Route 53's multi-region routing capabilities to improve redundancy and fault tolerance.
5. Content Delivery Networks (CDNs):
    - Route 53 can be used in conjunction with Amazon CloudFront to manage DNS for global content delivery, caching, and content acceleration.
6. Hybrid Cloud Environments:
    - Route 53 can be used to route traffic between on-premises systems and cloud resources, supporting hybrid cloud architectures.

Pricing for Amazon Route 53

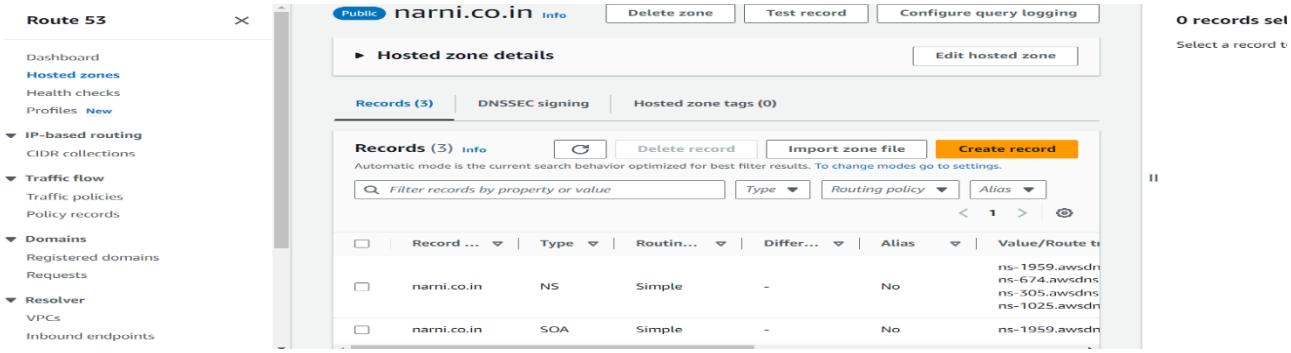
Route 53 pricing is based on several factors:

1. Hosted Zones: The number of hosted zones you create.
2. DNS Queries: The number of DNS queries processed by Route 53.
3. Domain Registration: Costs associated with registering domain names.
4. Health Checks: Fees for performing health checks on resources.
5. Traffic Flow: There is a cost for creating and managing traffic flow policies.

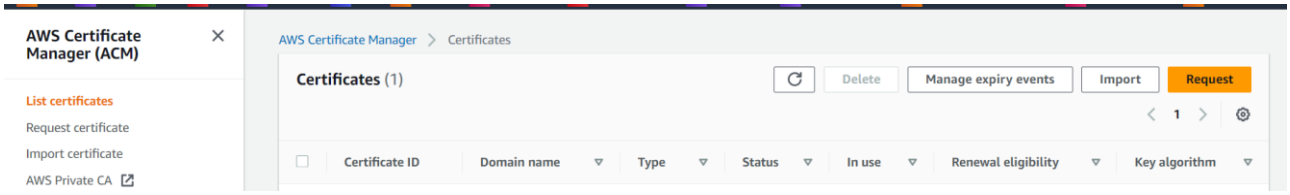
The pay-as-you-go model makes Route 53 cost-effective for small websites to large-scale applications.

Practical

Add your domain name in route 53

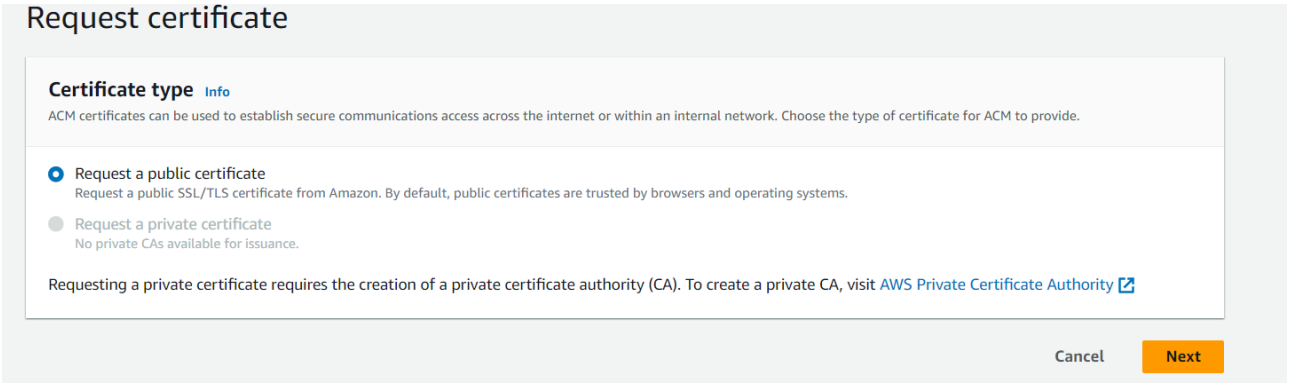


Then search for certificate manager in aws search open it

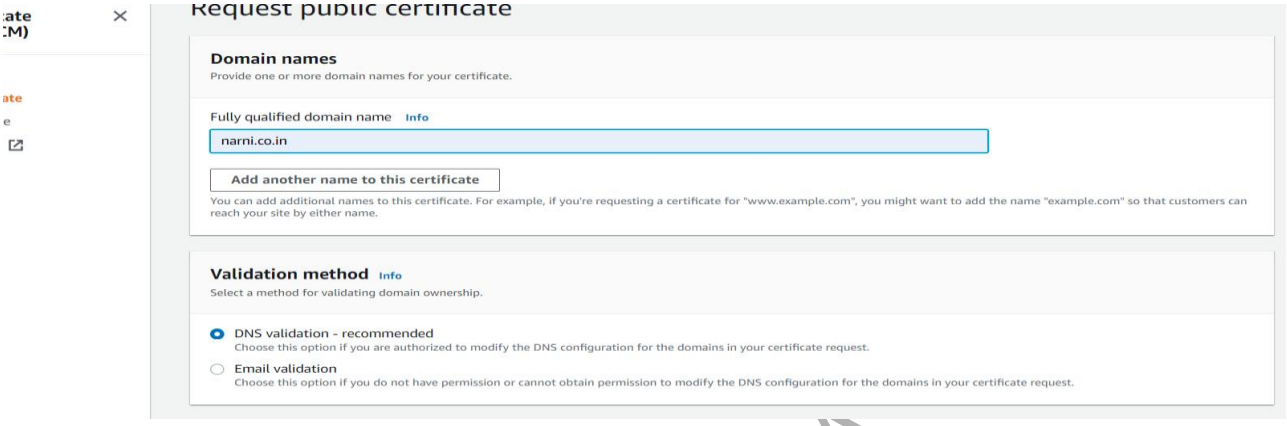


Click on request

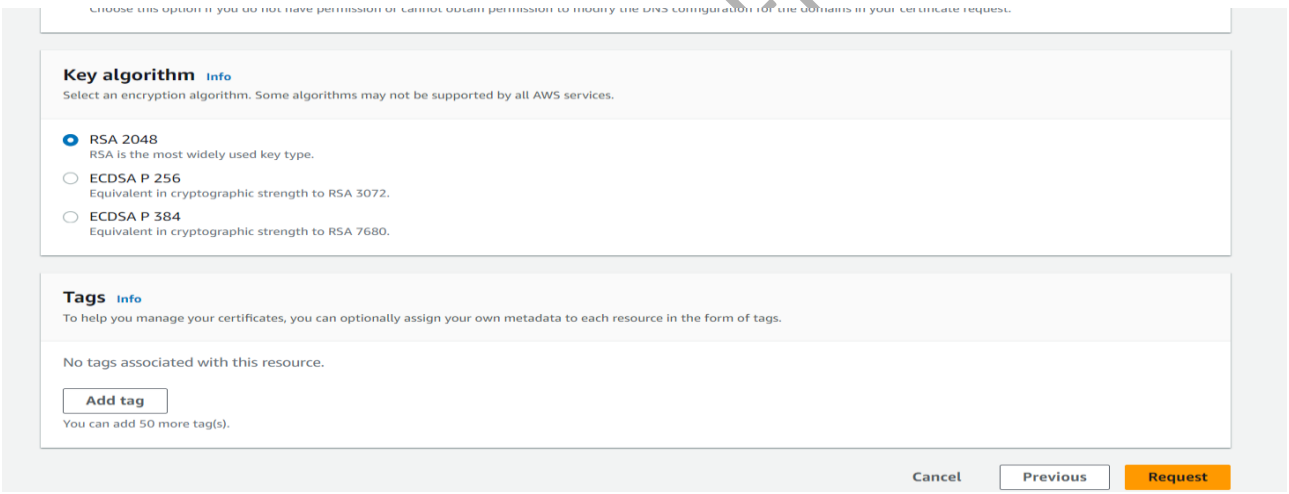
Then click on next



Enter your domain name here

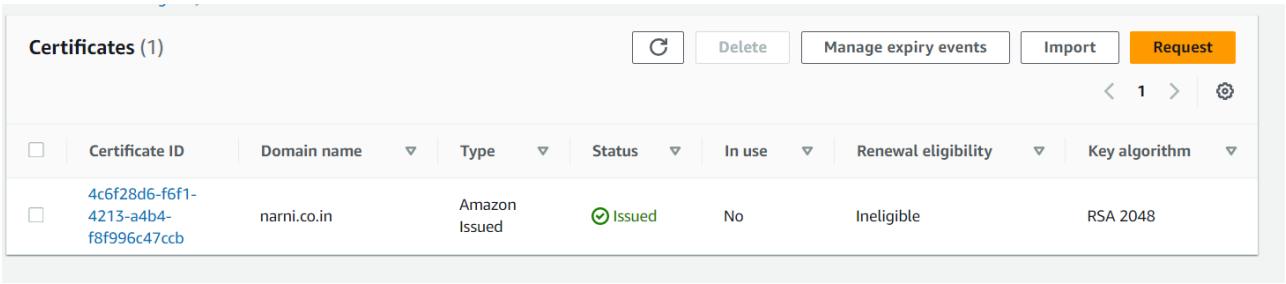


Click on request

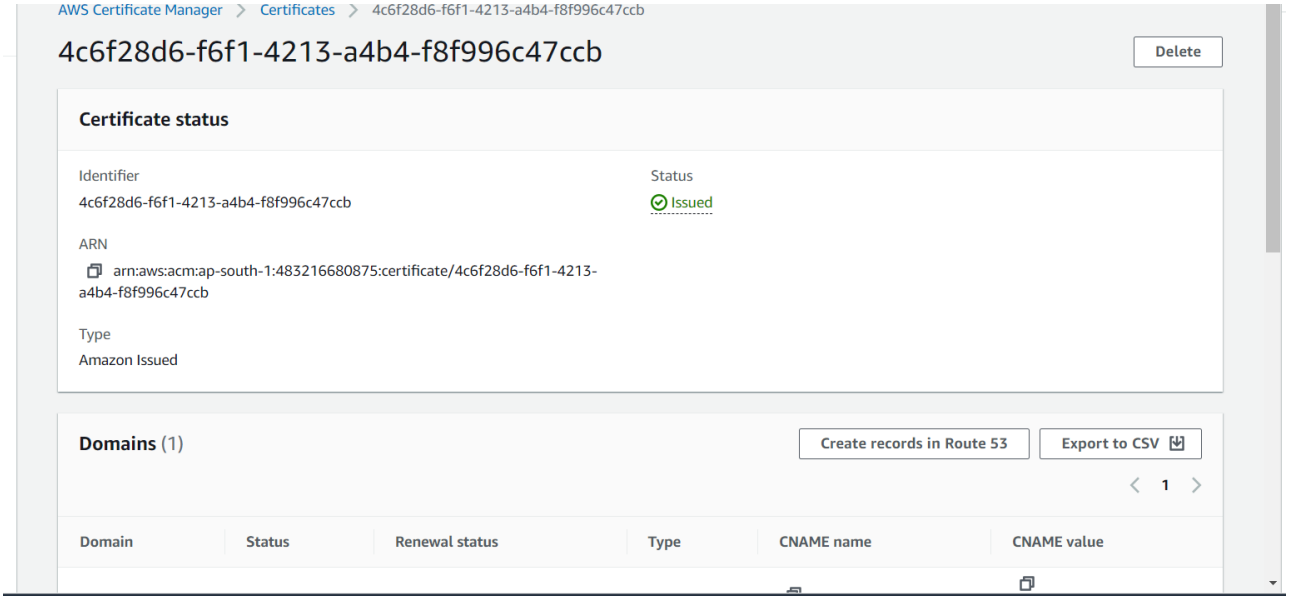


This is your certificate

Click on certificate id.

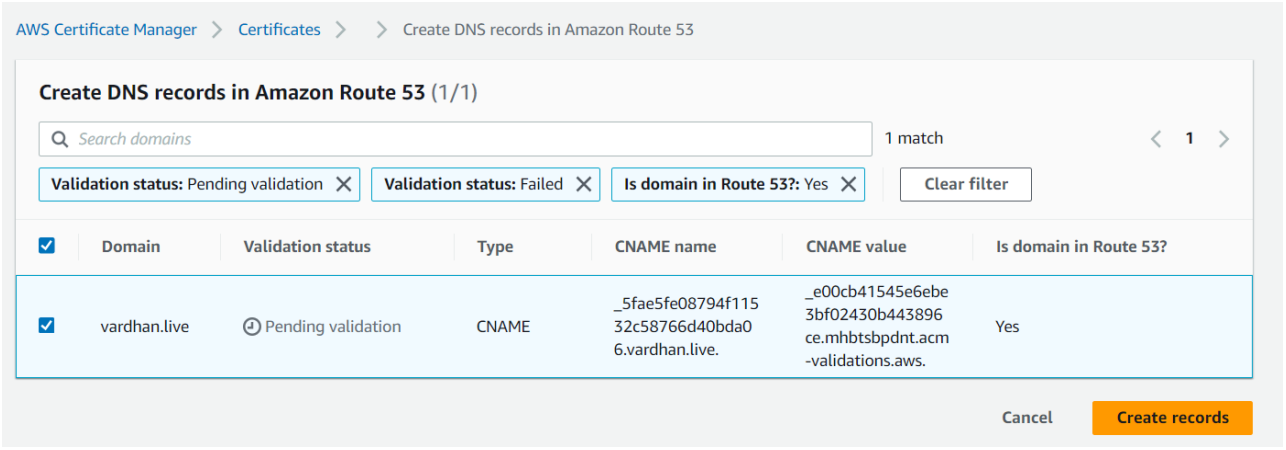


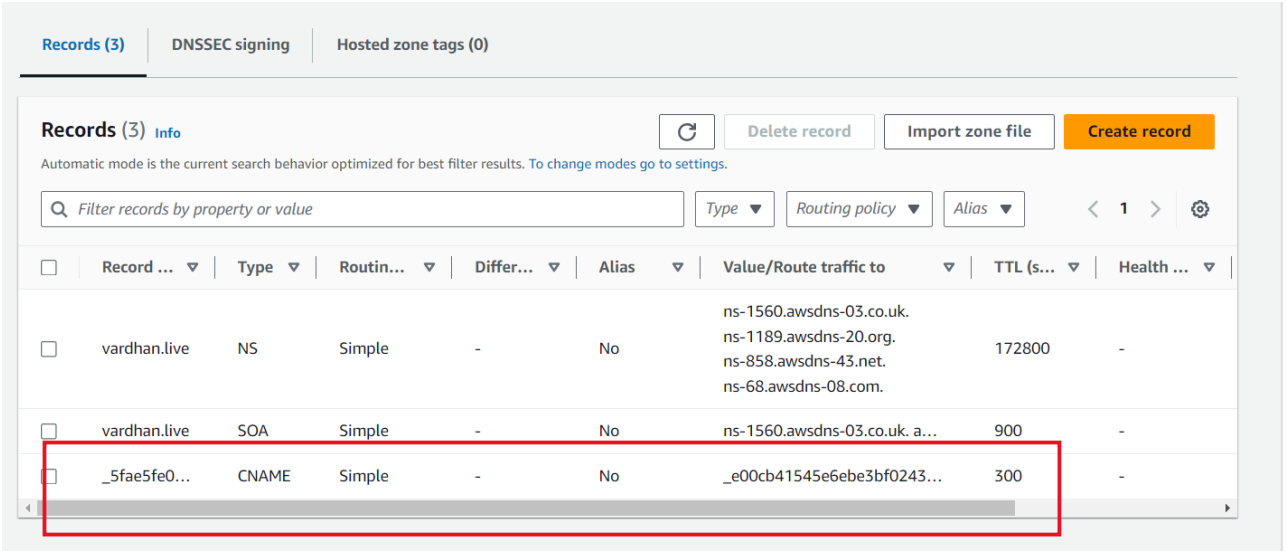
Click on create records in route 53



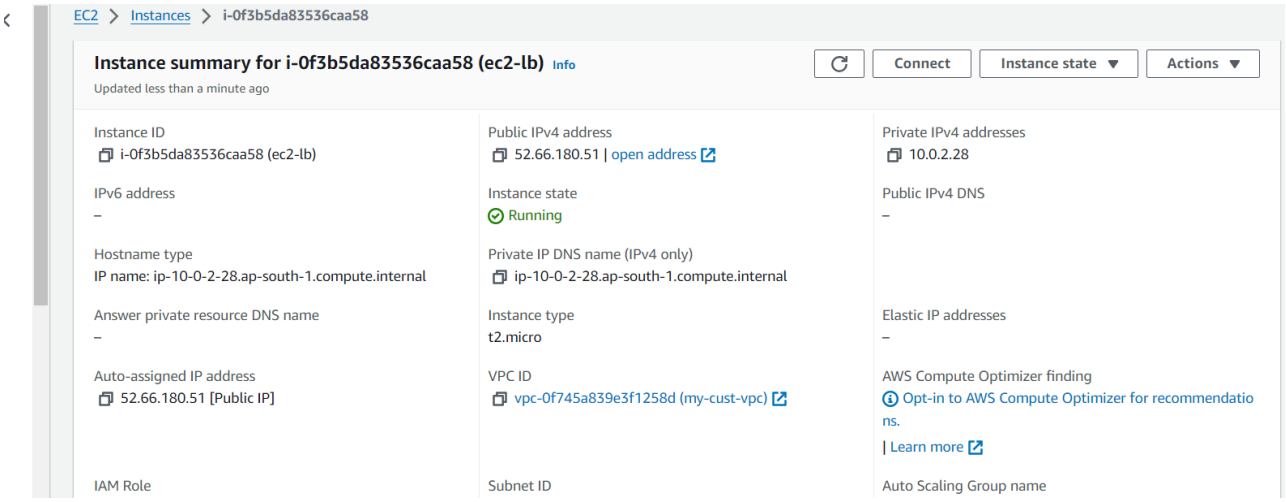
Click on create records

it will be added in to route 53 records list

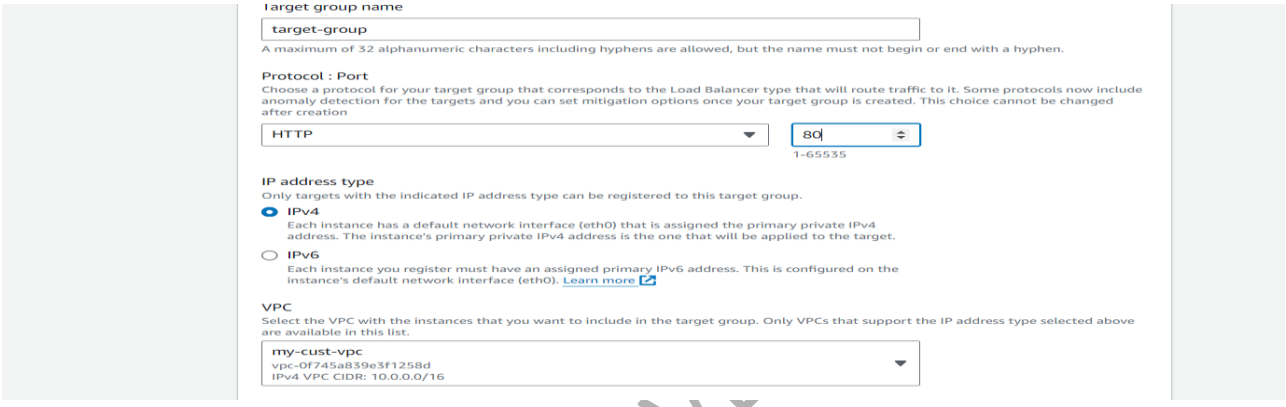




Then create a instance and add user data or manually connect to instance and install httpd server and check with public ip

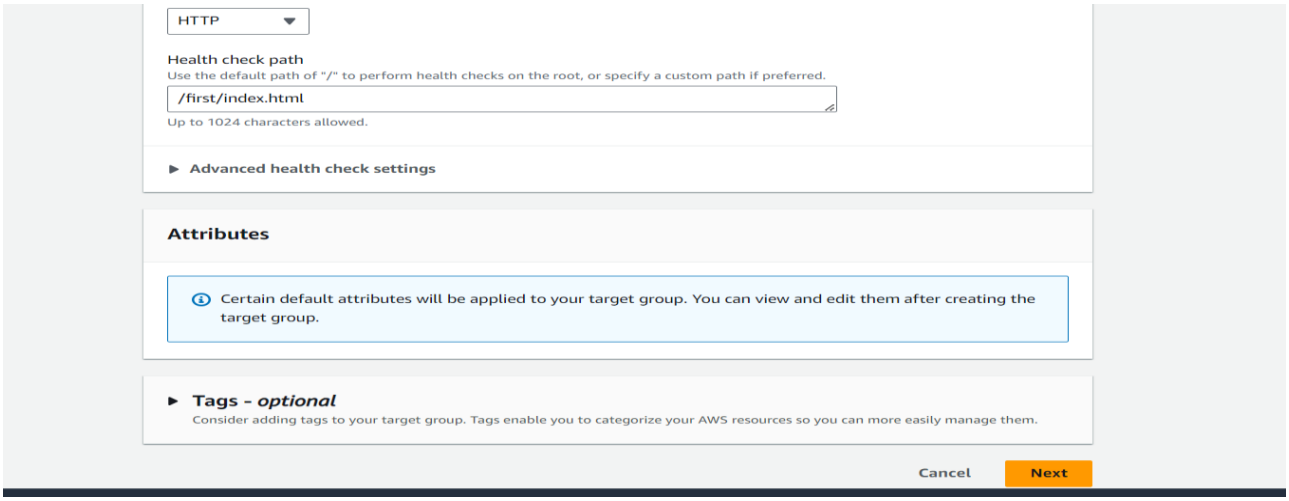


Crete a target group for above instance

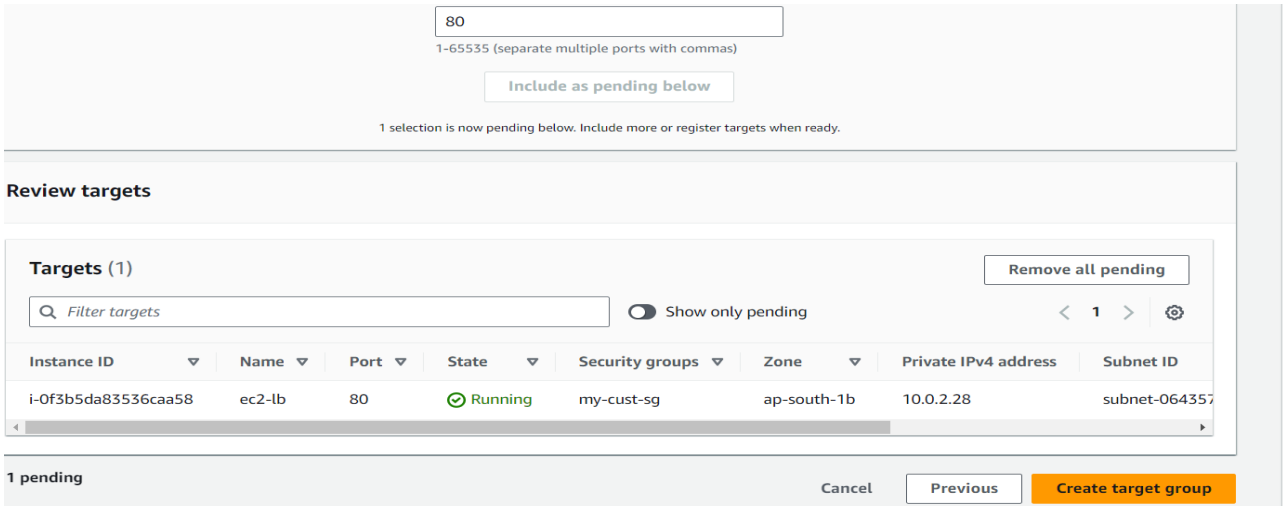


Mention your health check path as your application path

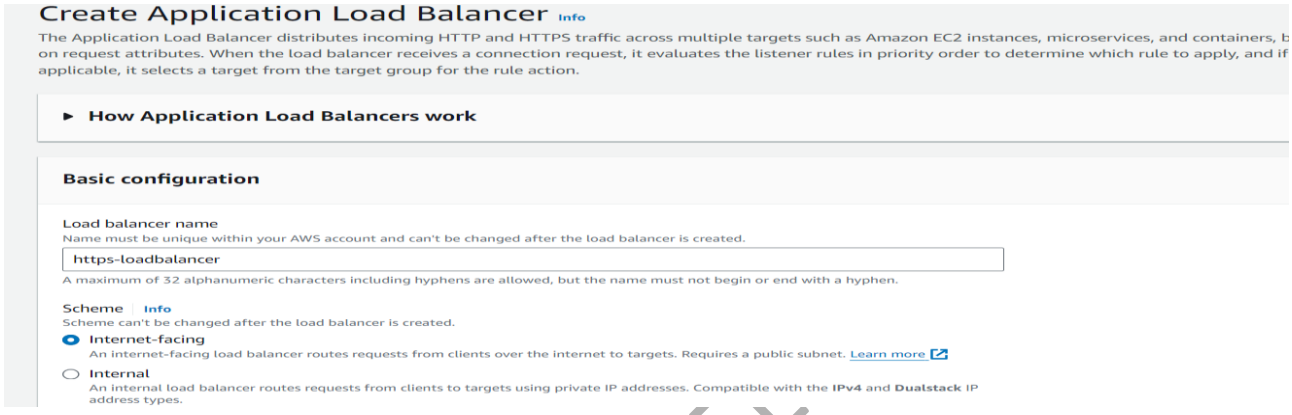
Then click on next



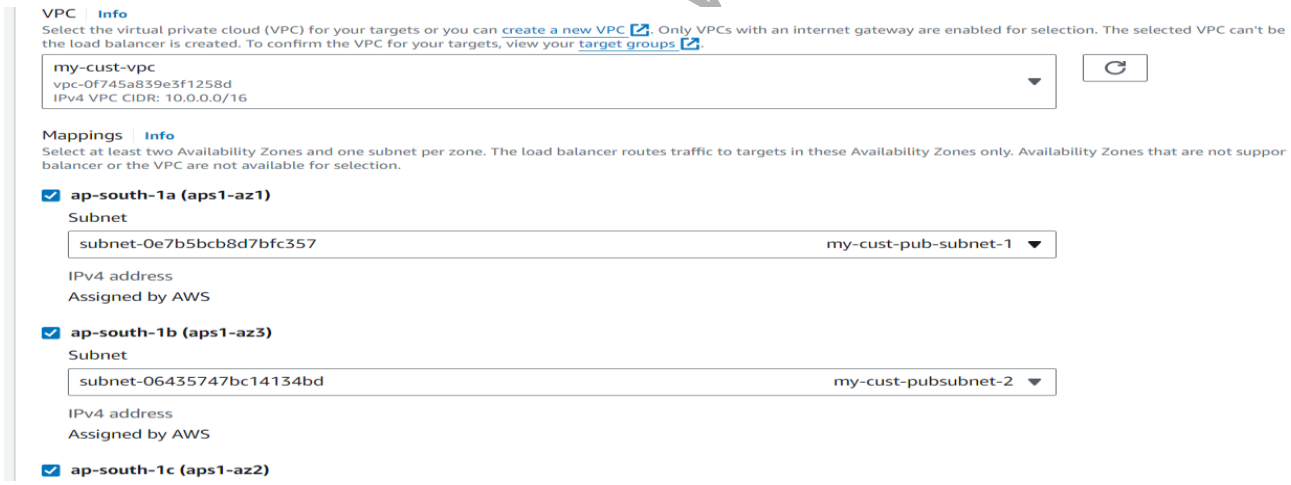
Select your instance and create a target group



Next create a load balancer



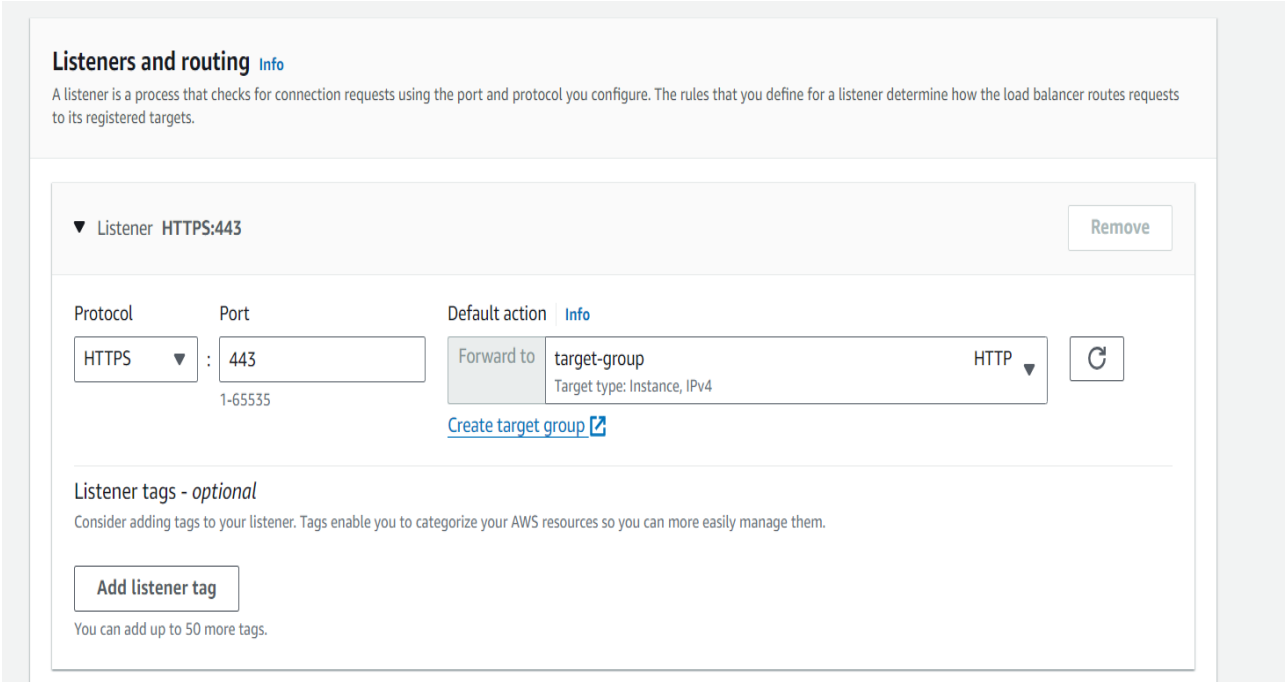
Select you subnets



THIS IS MAJOR STEP FOR HTTPS

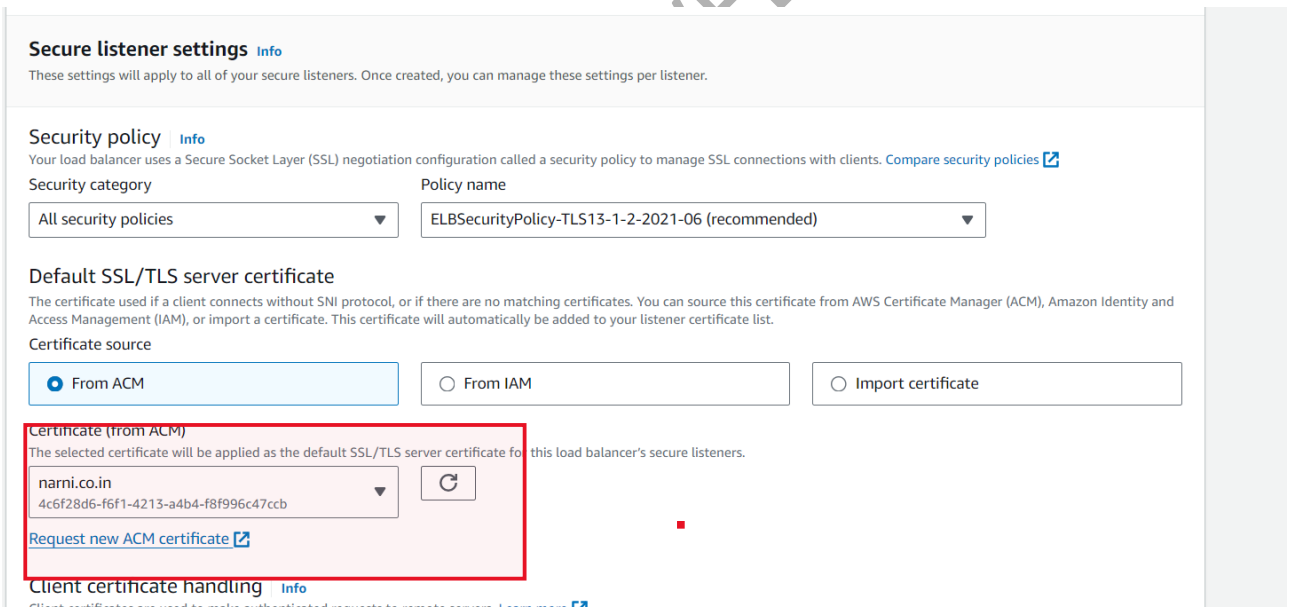
Select protocol as https

And select you target group

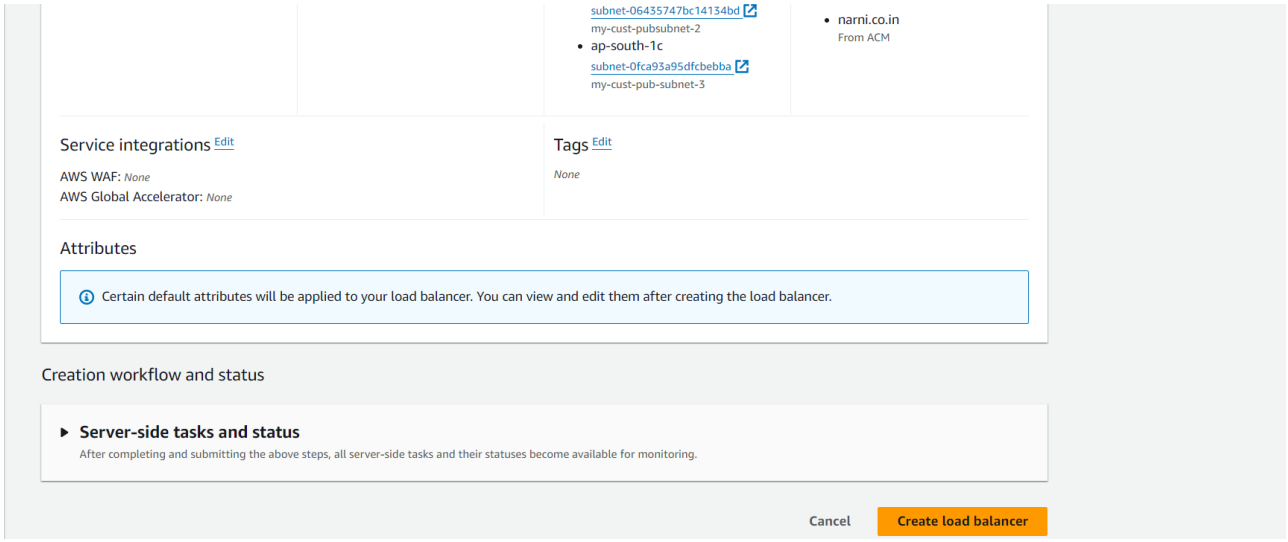


Security policy as it is

Select you certificate here this certificate comes from certificate manager you previously created

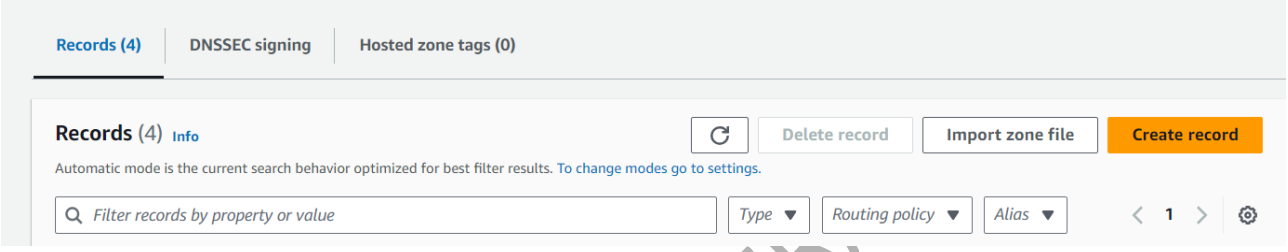


Click on create load balancer



Then open your route 53

Click on create record



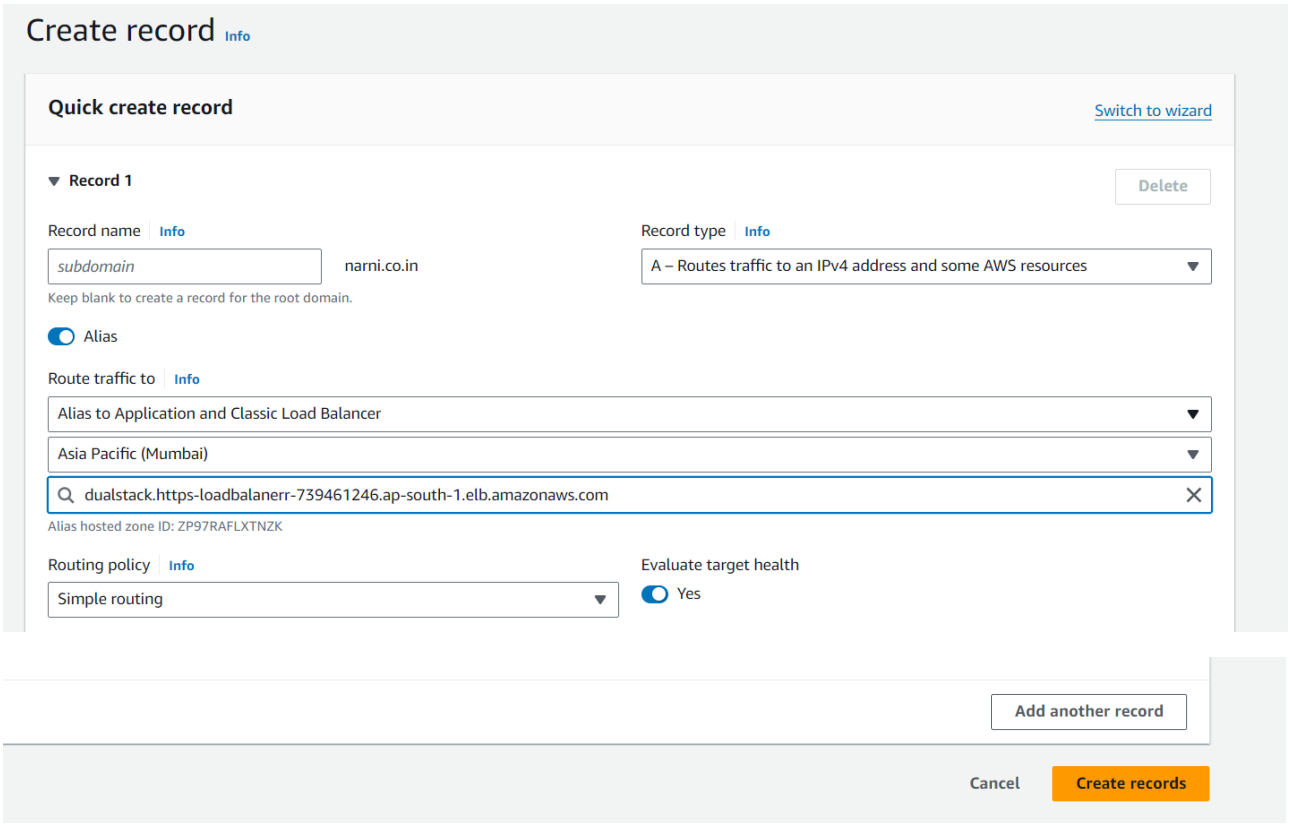
Select alias record

Type = alias to application and classic load balancer

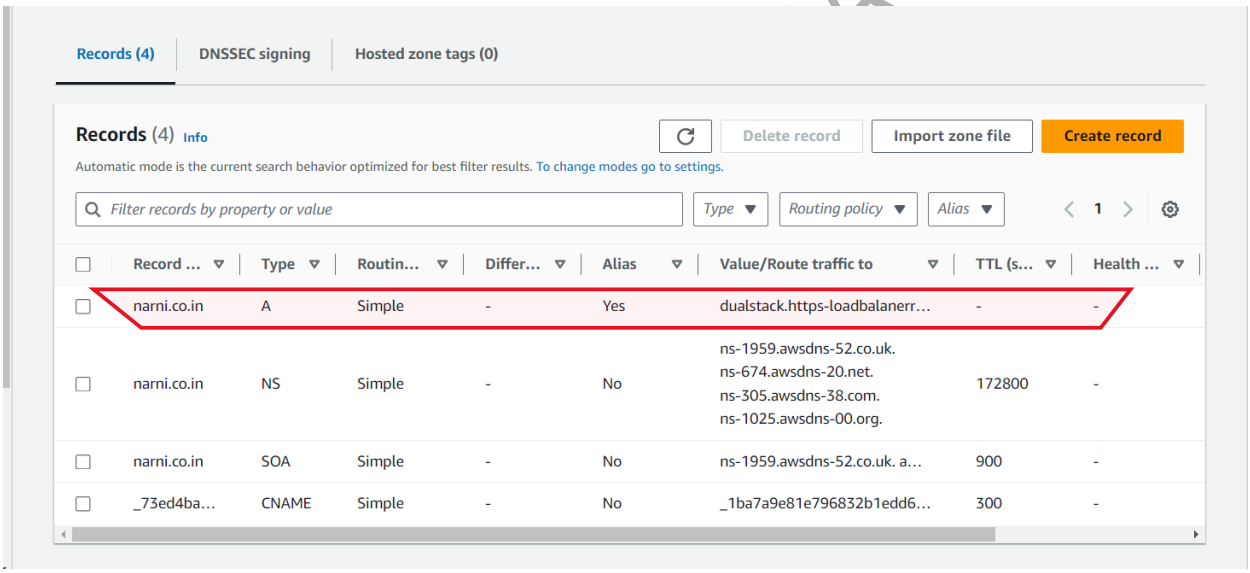
Region= your load balancer region

Select your load balancer dns url

Click on create record

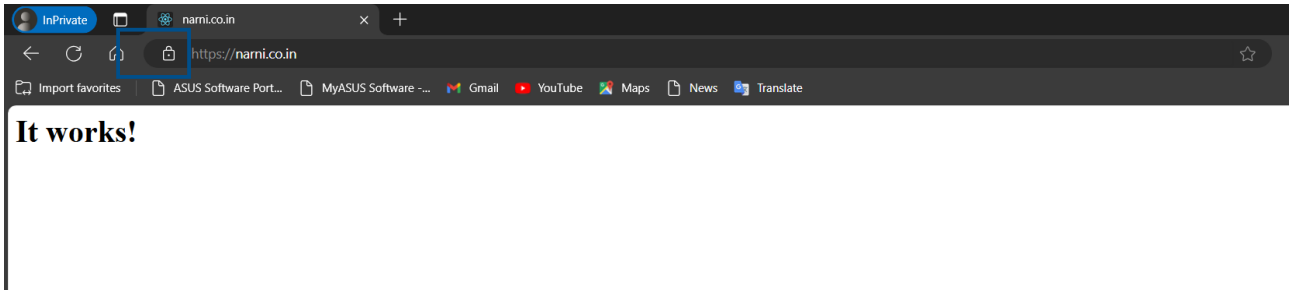


This is your record



Hit with your domain name

https working successfully



Aws lambda

Overview of AWS Lambda

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS) that lets you run code in response to events without provisioning or managing servers. With AWS Lambda, you can run backend code for virtually any application or service, all without worrying about the infrastructure required to run it.

Lambda is one of the foundational services in the serverless computing model, where you don't need to manage servers or infrastructure yourself. Instead, AWS manages all the resources, scaling, and availability for you. You simply upload your code, define the event triggers, and Lambda takes care of execution, scaling, and monitoring.

Key Features of AWS Lambda:

1. Event-Driven Computing:
    - AWS Lambda is event-driven, meaning that your Lambda functions are triggered by specific events or data. These events could come from many AWS services, such as S3, DynamoDB, SNS, API Gateway, CloudWatch, or even HTTP requests from your application.
    - For example, you can trigger a Lambda function every time an object is uploaded to an S3 bucket or whenever an update happens in a DynamoDB table.
2. No Server Management:
    - Lambda abstracts away the underlying infrastructure, so you don't have to worry about provisioning servers, scaling, or handling server maintenance.
    - You upload your code (as a function) and specify the trigger (event). Lambda automatically takes care of scaling the resources to meet demand and managing fault tolerance.
3. Automatic Scaling:
    - AWS Lambda scales automatically by running code in response to each event, and each event is handled by a separate instance of your function.
    - If there are a high number of incoming events (e.g., many users accessing your application), Lambda will automatically create more instances of the function to handle the increased load, without any intervention from you.
4. Short-Running Tasks:
    - Lambda functions are designed to handle short, stateless tasks. They typically run for a few seconds to a few minutes at most.
    - AWS Lambda has a maximum execution timeout of 15 minutes per function invocation, but for most use cases, tasks are completed in less than a minute.

5. Supports Multiple Programming Languages:
    - AWS Lambda supports several popular programming languages, including:
        - Node.js
        - Python
        - Java
        - Go
        - C# (.NET Core)
        - Ruby
        - Custom Runtime: Lambda also allows you to create custom runtimes if you want to run code written in other languages.
    - This flexibility lets developers choose the best programming language for their specific needs.
6. Pay-Per-Use Pricing:
    - Lambda follows a pay-per-use pricing model. You only pay for the compute time your functions consume when they are triggered, and there are no charges when your code is idle.
    - You are billed based on the number of requests and the duration of code execution, which is calculated in milliseconds.
    - There is also a free tier that provides 1 million free requests and 400,000 GB-seconds of compute time per month.
7. Integrated with AWS Services:
    - AWS Lambda integrates seamlessly with other AWS services like Amazon S3, DynamoDB, API Gateway, SNS, CloudWatch, SQS, and more.
    - This enables developers to build complex workflows and event-driven applications without managing server infrastructure.
8. Stateless and Immutable:
    - Lambda functions are stateless by design, meaning that each invocation of the function is independent, and there's no stored state between function calls.
    - However, you can use external storage services like Amazon S3, DynamoDB, or ElastiCache to store data if needed.
9. Versioning and Aliases:

- AWS Lambda supports function versioning, allowing you to publish and manage different versions of a Lambda function. You can then create aliases to point to specific versions of a function, providing flexibility for testing, deployment, and version management.

10. Monitoring and Logging:
    - AWS Lambda integrates with Amazon CloudWatch for monitoring and logging. Lambda automatically creates logs for every function invocation.
    - You can monitor the function's performance, success/failure rates, and error logs via CloudWatch, making it easy to track function behavior and diagnose issues.

How AWS Lambda Works:

1. Upload Your Code:
    - You write your function and upload it to AWS Lambda, or you can store it in a zip file or a container image.
    - You also specify the runtime environment (e.g., Node.js, Python) and any required permissions (via IAM roles) for your function to access other AWS services.
2. Define Event Triggers:
    - You configure your Lambda function to be triggered by specific events (such as changes in S3 buckets, DynamoDB streams, SNS notifications, etc.).
    - For example, an image uploaded to an S3 bucket can trigger a Lambda function to process the image or generate a thumbnail.
3. Execution:
    - When the specified event occurs, AWS Lambda automatically runs the function.
    - Lambda runs the function in a stateless container and allocates resources such as CPU and memory based on the configuration you provide.
4. Scaling:
    - If multiple events are triggered at once (e.g., many users accessing your app), Lambda automatically scales to handle the load by running multiple instances of your function in parallel.
    - If the event volume decreases, Lambda reduces the number of active function instances accordingly, without you needing to manage the infrastructure.
5. Pay for Usage:

- You are billed for the total compute time your function uses, based on the number of invocations and how long each function takes to execute.

Use Cases for AWS Lambda:

1. Real-Time File Processing:
   - Lambda is often used for processing files in real-time. For example, if you upload files to an S3 bucket, Lambda can automatically trigger a function to process, transform, or analyze the files.
   - Example: Image resizing, video transcoding, log file analysis.
2. Web Application Backends:
   - You can use AWS Lambda to build scalable web application backends. By integrating with API Gateway, Lambda can respond to HTTP requests (GET, POST, etc.) without managing web servers.
   - Example: User authentication, data processing, and serving API responses.
3. Data Pipelines:
   - Lambda can help in building data pipelines where data is processed in real-time. For example, triggering Lambda functions to process data as it is inserted into DynamoDB or Kafka.
   - Example: Real-time analytics, data transformation, or ETL jobs.
4. Serverless Microservices:
   - Lambda is a perfect fit for implementing microservices architectures. Each Lambda function can perform a specific task (e.g., payment processing, user management), and together they form a serverless application.
5. IoT Applications:
   - Lambda is commonly used in IoT applications where devices trigger events that Lambda processes.
   - Example: A temperature sensor sends data to an S3 bucket or DynamoDB, and Lambda triggers processing or sends alerts.
6. Automating AWS Management Tasks:
   - Lambda can be used to automate operational tasks like resource provisioning, backup operations, security monitoring, and scaling of AWS resources.
   - Example: Automatically stop unused EC2 instances during off-hours.

7. Scheduled Jobs and Cron Jobs:
    - You can run Lambda functions at scheduled intervals using CloudWatch Events. This is similar to running cron jobs, but with no need for provisioning servers.
    - Example: Periodic reports, system health checks, and backups.

Pricing for AWS Lambda:

- Requests: You are charged for the total number of requests to your Lambda function. The first 1 million requests per month are free.
- Duration: Lambda is charged based on the duration of your function execution, measured in milliseconds. You specify the amount of memory allocated to the function, and the cost is based on the resources consumed during execution.
- Free Tier: AWS offers a free tier that includes 1 million requests and 400,000 GB-seconds of compute time per month for free.

Advantages of AWS Lambda:

1. Serverless Architecture: You don't need to manage infrastructure, making it easier and cheaper to scale applications.
2. Automatic Scaling: Lambda automatically scales based on incoming events, without needing you to configure or manage servers.
3. Cost-Effective: You only pay for what you use, which makes Lambda cost-effective, especially for intermittent workloads.
4. Speed of Development: You can focus on writing code without worrying about managing servers or infrastructure.
5. High Availability: AWS Lambda is highly available by default, as AWS manages the infrastructure across multiple availability zones.

Limitations of AWS Lambda:

1. Execution Time Limit: Lambda functions have a maximum execution time of 15 minutes. If your function requires more time, you may need to use an alternative like EC2 or Fargate.
2. Statelessness: Lambda functions are stateless by design, meaning they don't retain data between invocations. You'll need external storage for any persistent data (e.g., S3, DynamoDB).

3. Cold Starts: While Lambda can scale automatically, the first time a function is invoked after a period of inactivity (called a cold start), it can experience some latency.