## A Hands-On Guide to Key Features of

# 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, memory-intensive, 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 high-performance 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 high-performance 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 price-performance.
- 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 high-speed, 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.

E**C2 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.

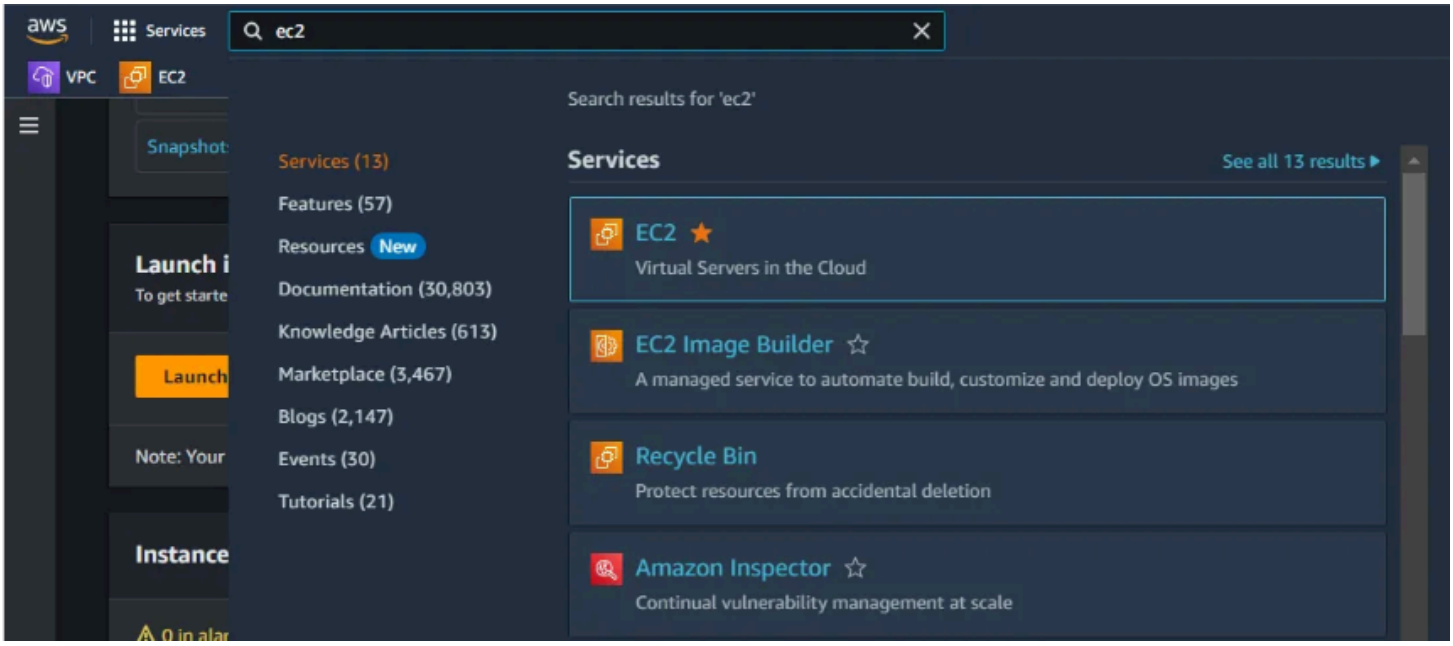
An EC2 instance essentially functions as a virtual machine.

Firstly, log in to your AWS console.

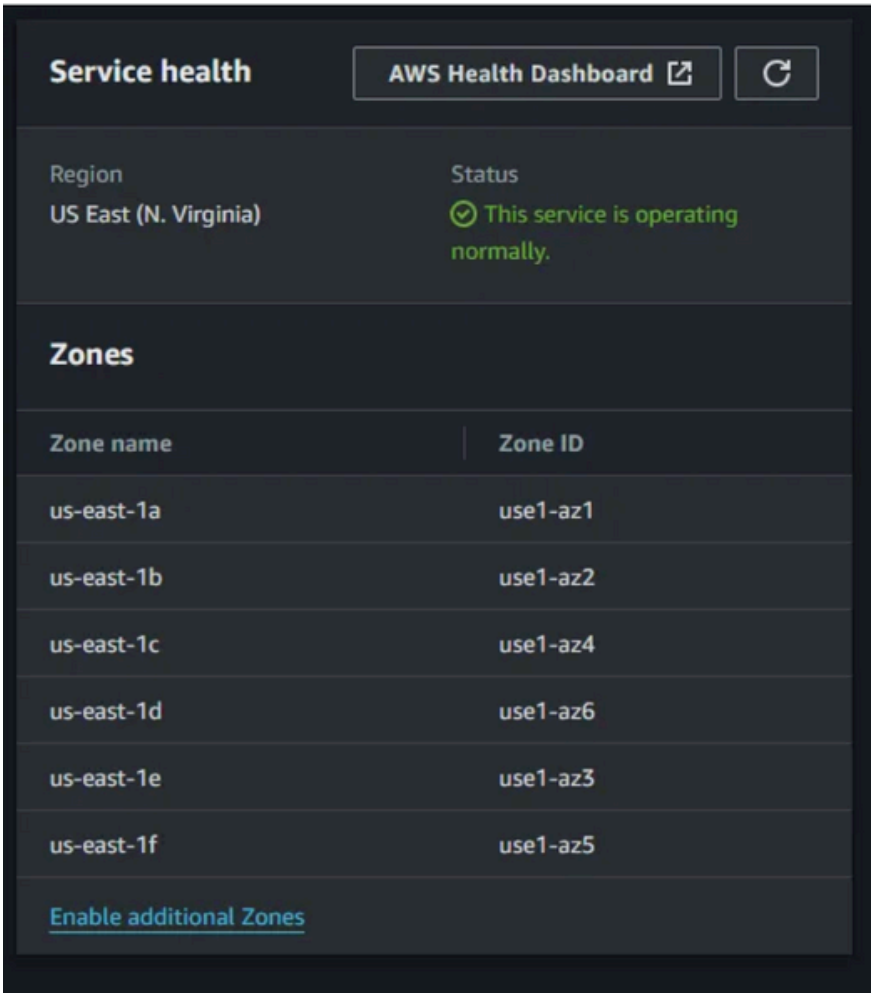
Next, choose your desired region. For instance, I’ll be selecting the North Virginia region. It’s advisable to select a region based on your current location, as some regions may incur higher costs compared to those in the US.

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

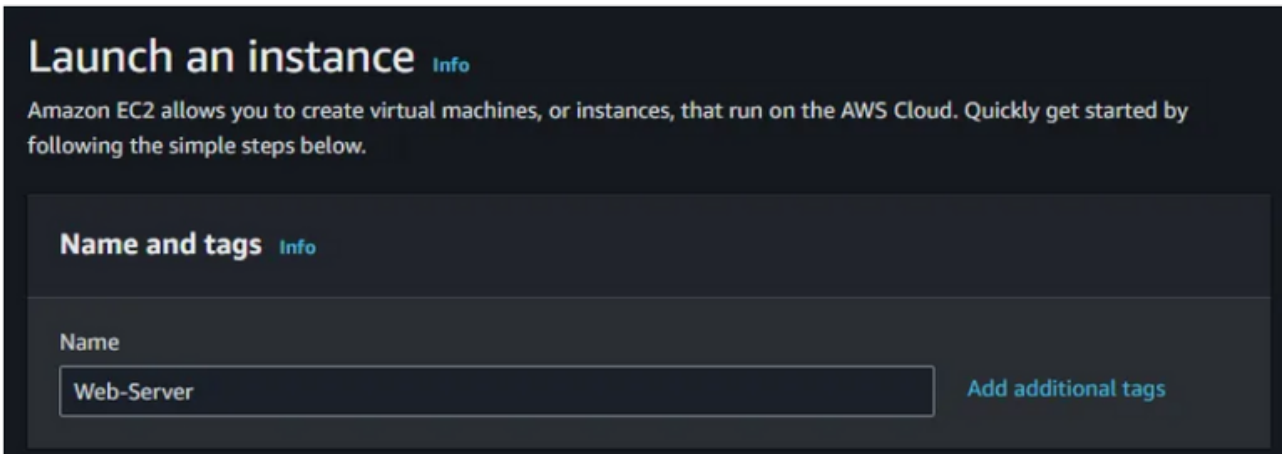
To navigate to the EC2 service, click on “Services” and locate EC2 either by browsing through all services or by directly searching for EC2.



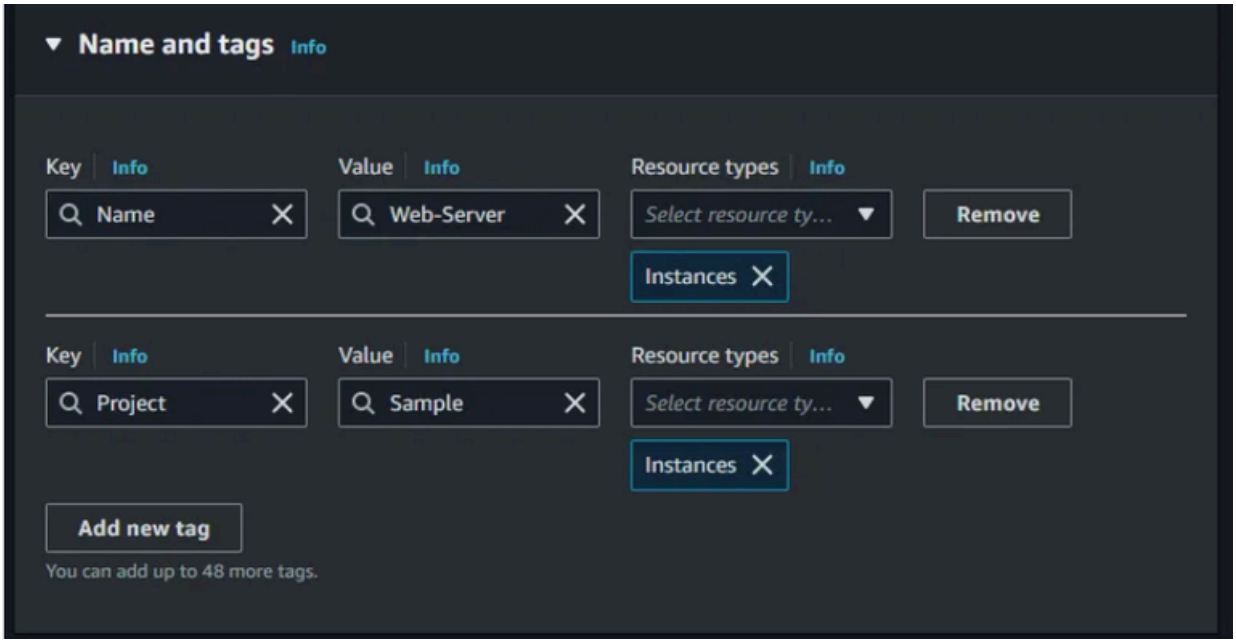

Once in the EC2 dashboard, you'll find an overview of your instances, volumes, and other relevant information. Keep in mind that the dashboard doesn't auto-refresh, so be sure to manually refresh after making any changes.
Scrolling down, you'll notice zones within the region, each identified by a code 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.

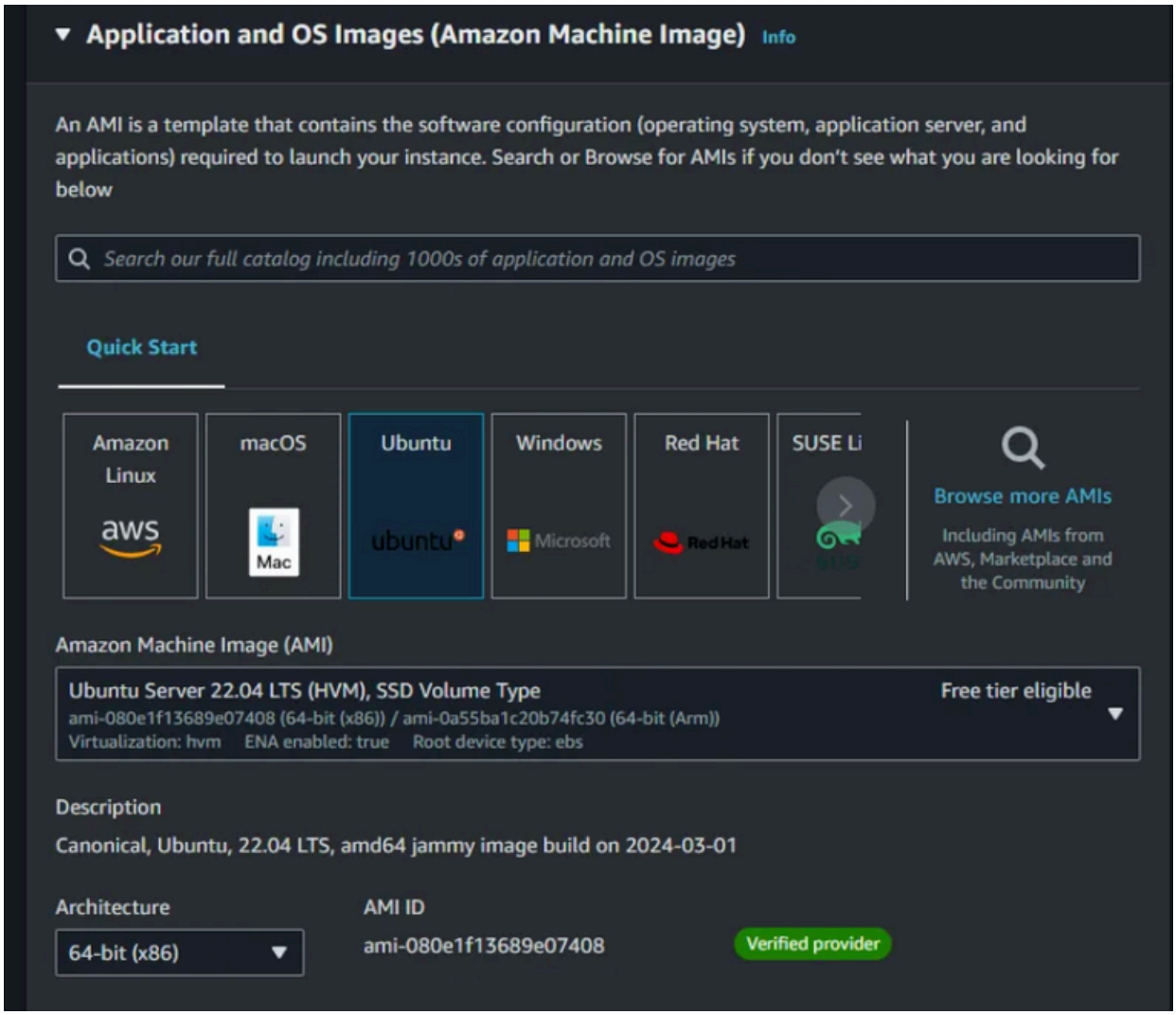



Now, let’s begin by assigning a name and optional tags to the instance. Tags can be useful for filtering and organization purposes.

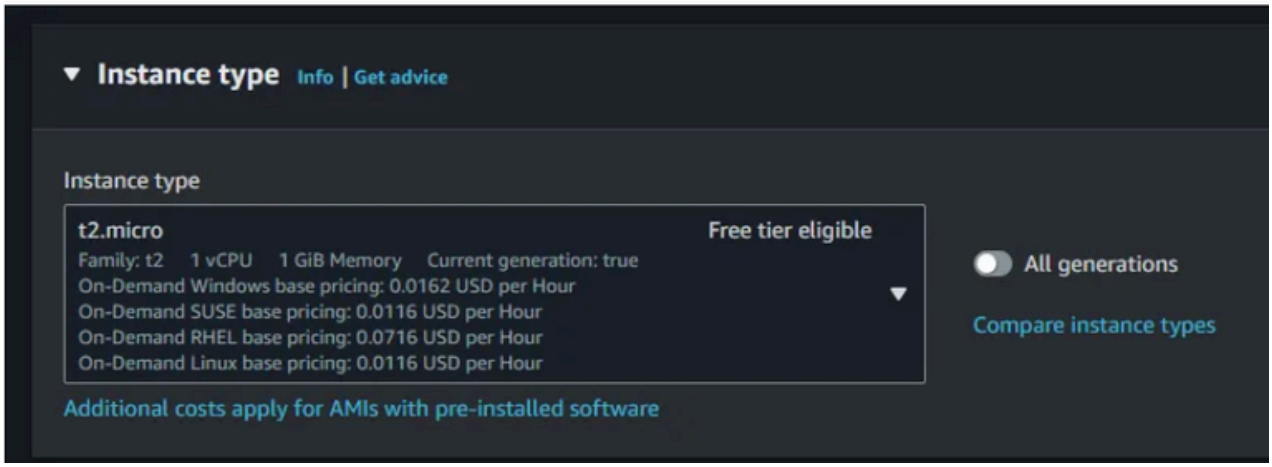



Moving on to selecting the Amazon Machine Image (AMI), you’ll find a variety of options including quick start AMIs like Amazon Linux, Ubuntu, and Windows, as well as community AMIs and those from the AWS Marketplace. Carefully review the details and charges associated with each AMI.

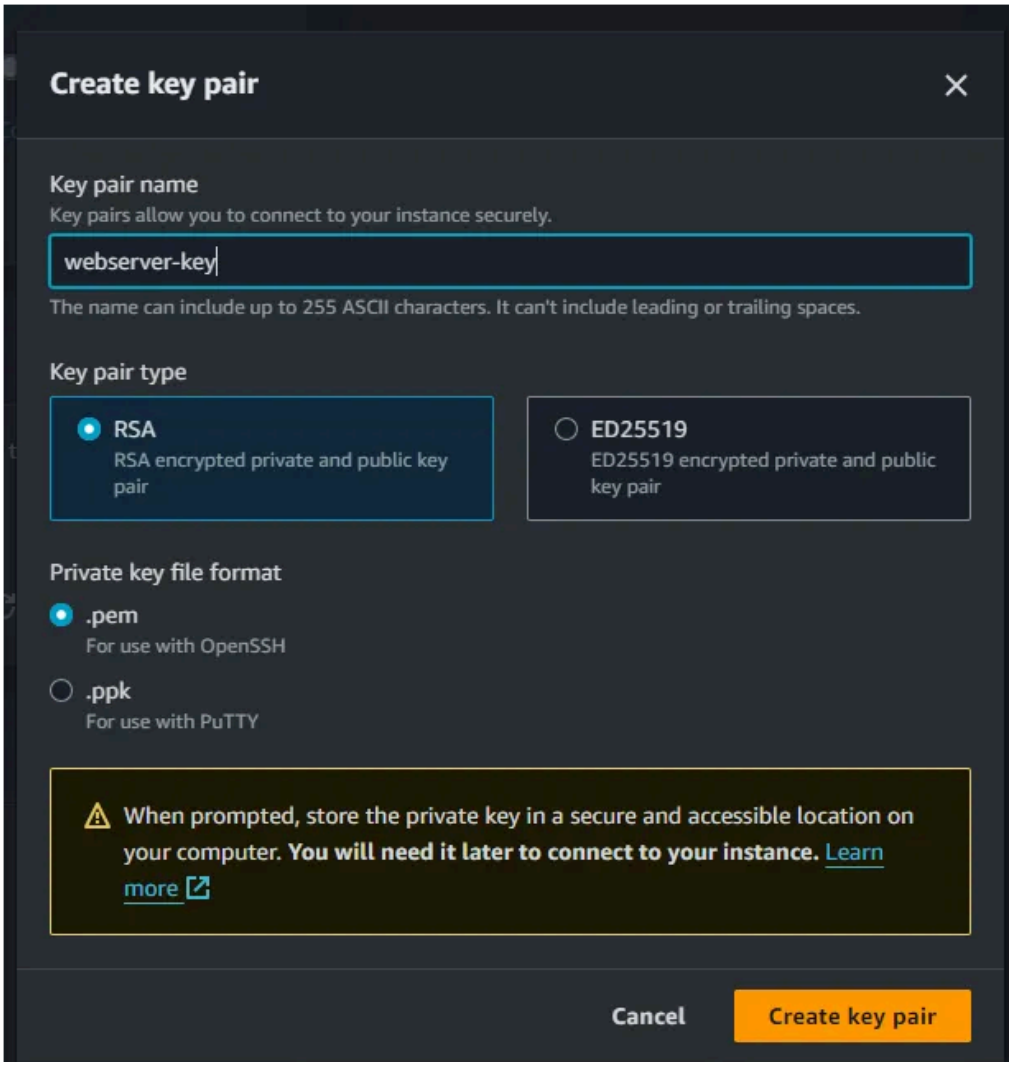
For this example, let’s choose the Ubuntu AMI provided by Amazon Web Services, which is free.



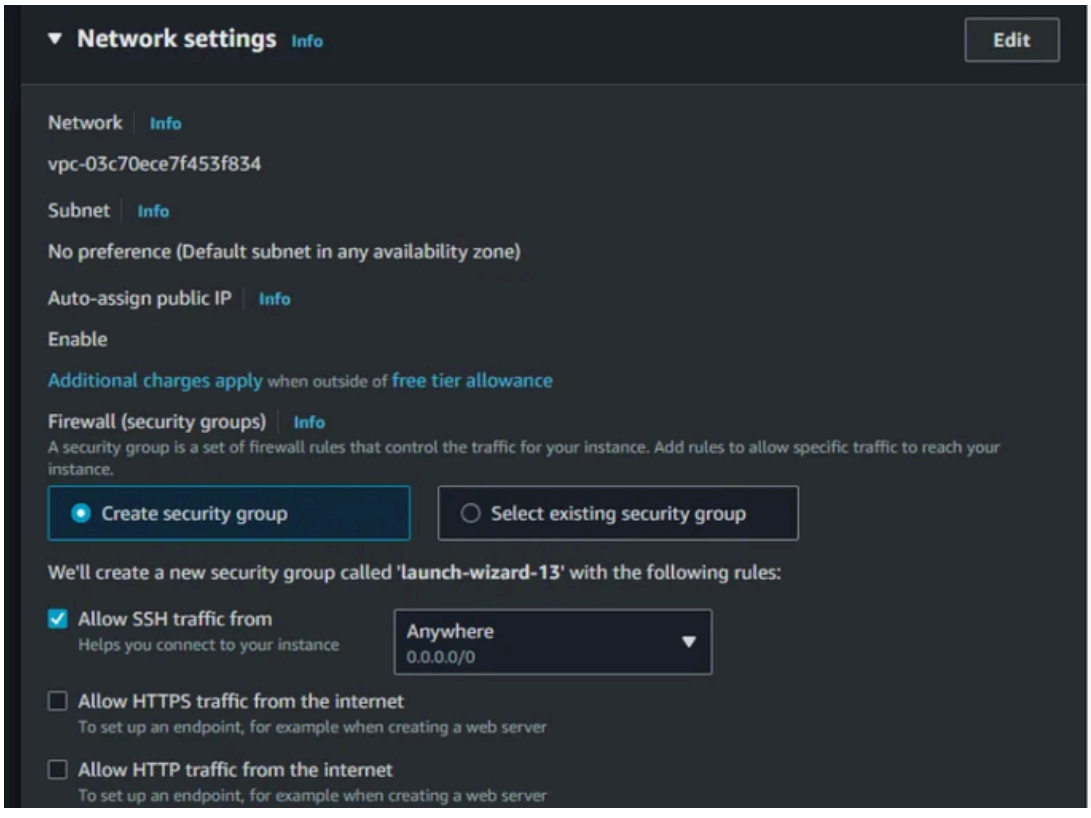

Ensure that the instance type selected aligns with your requirements. For free-tier usage, T2 micro instances are suitable for most learning purposes.



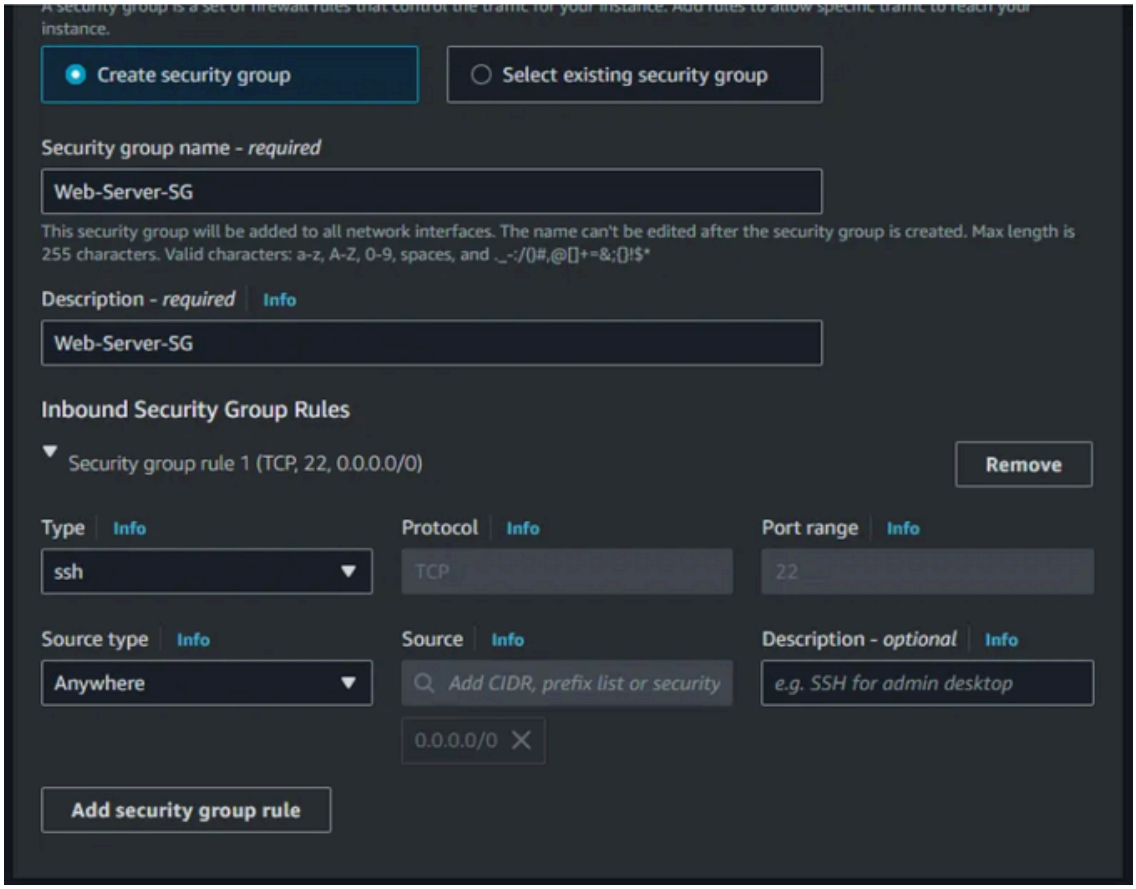

- Key pairs are used primarily for securely accessing EC2 instances. When you create an EC2 instance, you can specify a key pair to use for SSH access to the instance. The key pair consists of a public key that AWS stores, and a private key file that you download to your local machine.
- It's important 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.



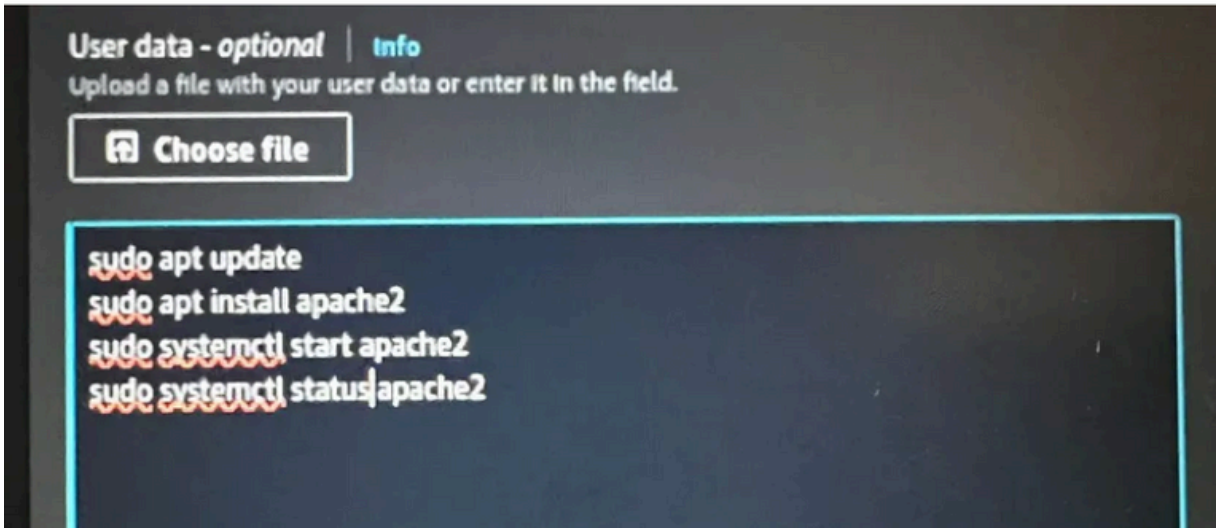
Customize the security group to allow necessary inbound and outbound traffic.




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:

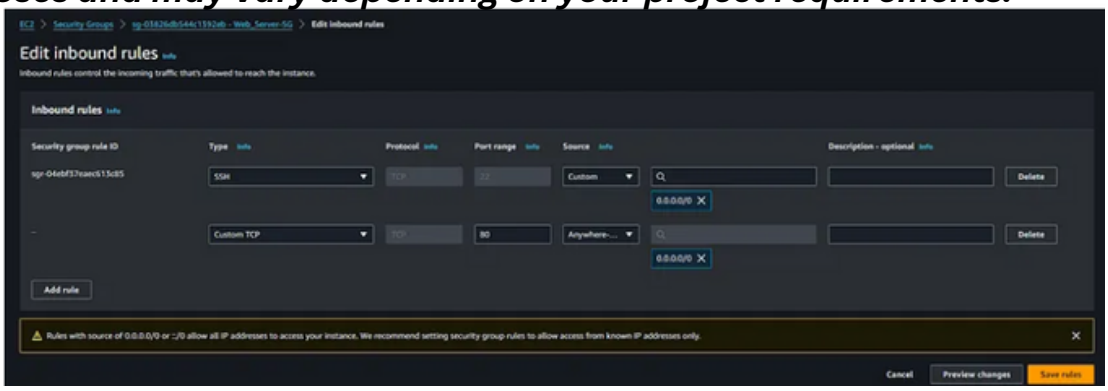
- By incorporating commands in the user data section before initiating the instance launch, as demonstrated below:



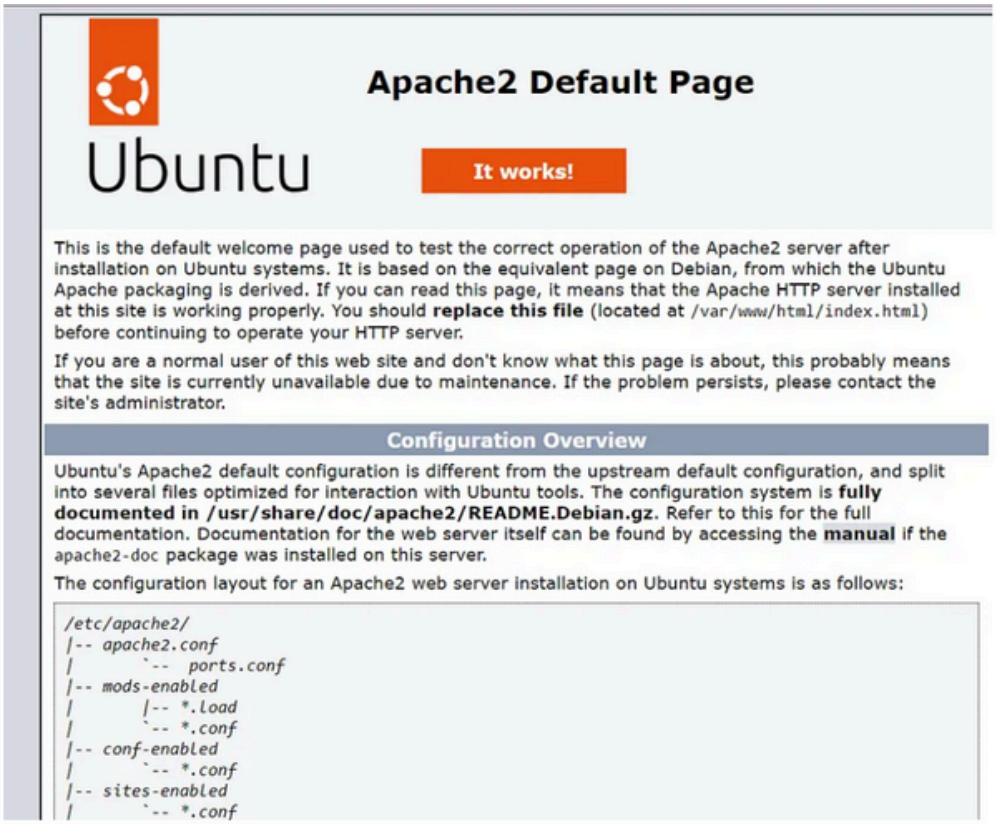
- Or, by executing the commands after connecting the ec2 instance via terminal.

- *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.***

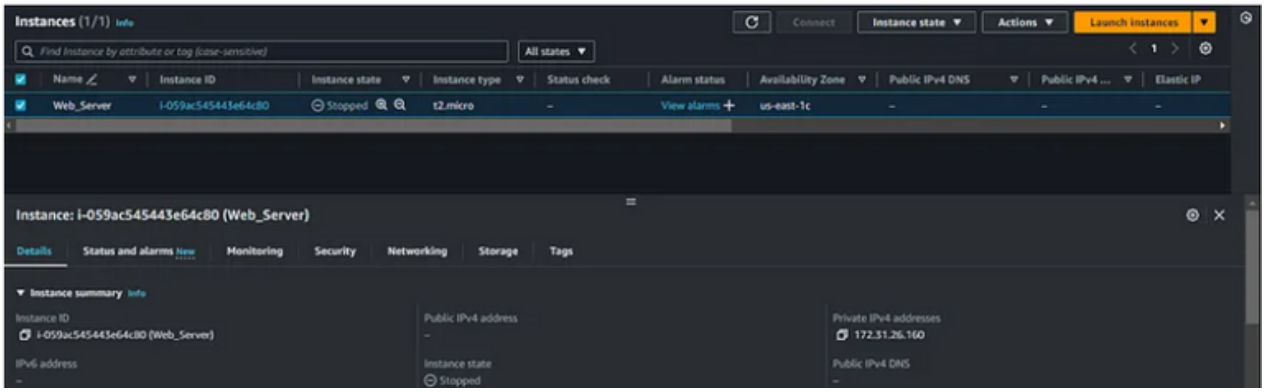



Go to the browser and copy the public IP followed by ':80' to view the Apache page.



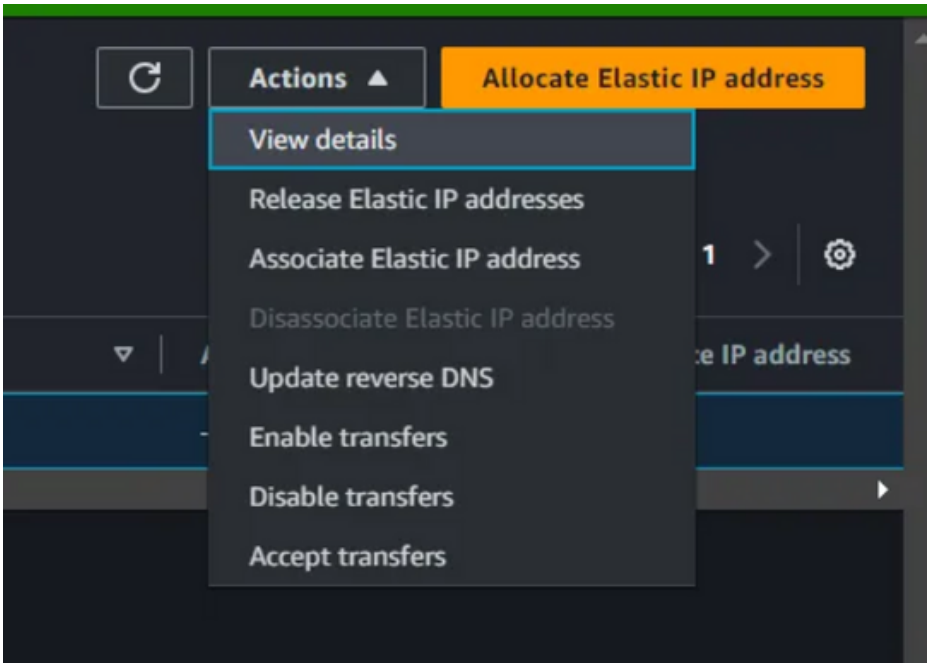
**Elastic IP**

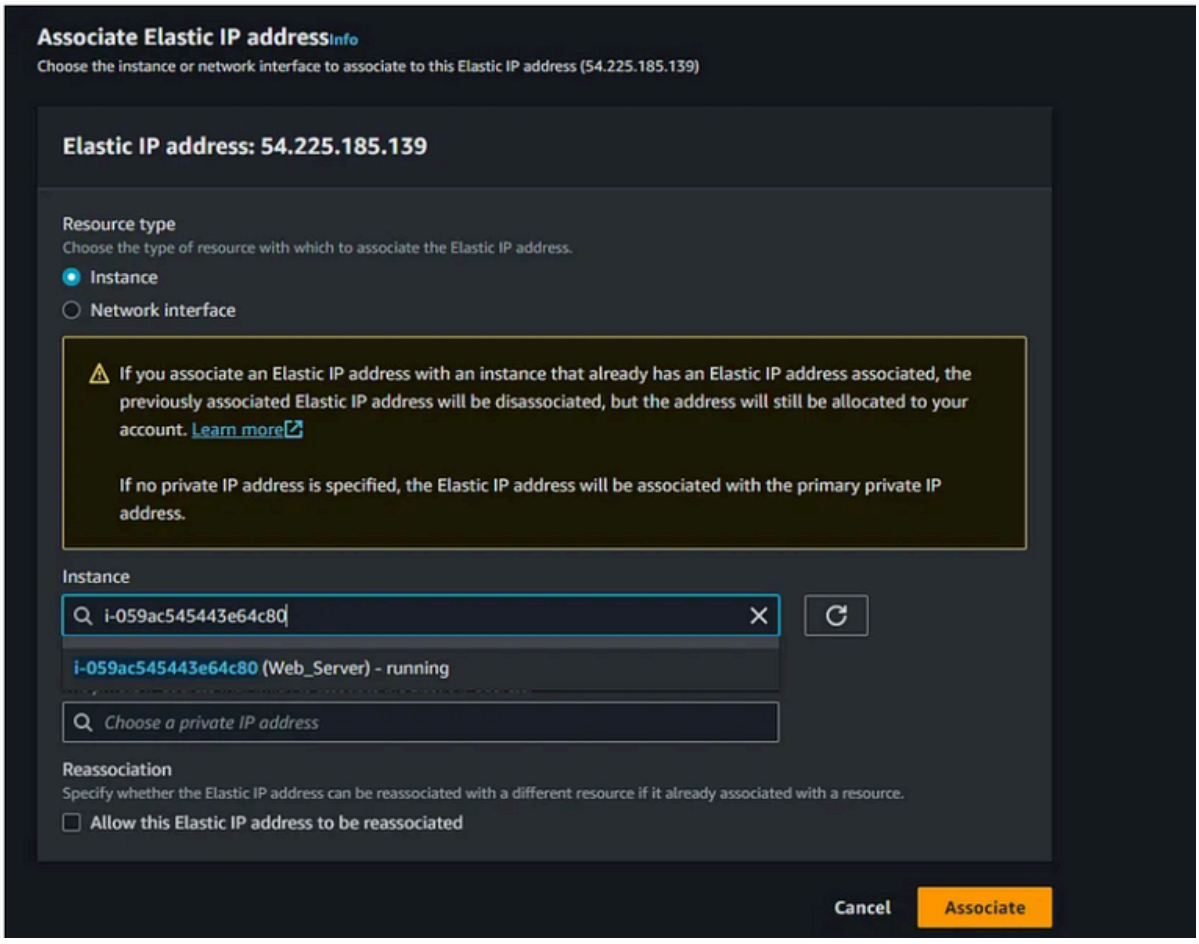
- Powering Off the EC2 Instance:
- Take note of the current public and private IP addresses of the instance.
- Stop the instance using the AWS Management Console.
- Observing IP Address Changes:
- After stopping the instance, observe that the public IP address is released while the private IP address remains unchanged.
- Refresh the console to verify the changes.




**Elastic IP Benefits:**

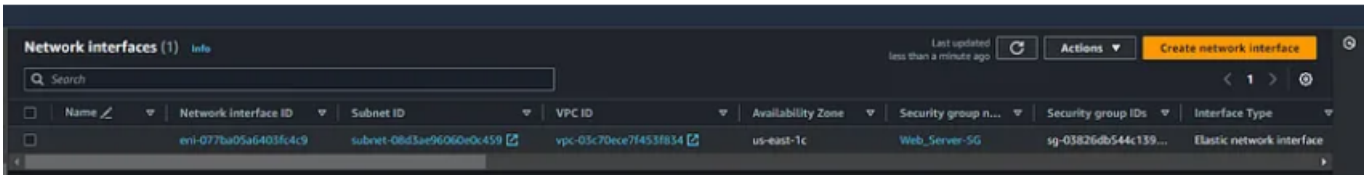
- Elastic IP addresses offer the advantage of maintaining a static public IP address even in scenarios where EC2 instances are restarted or recreated.
- They provide a stable endpoint for applications or services that require a consistent public IP address.
- With Elastic IP, there's no need to update 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.

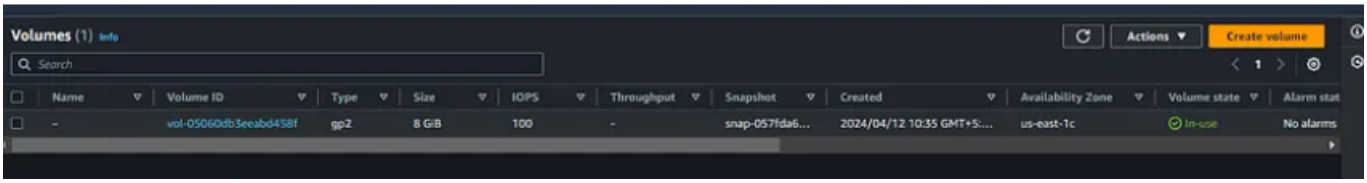



**EC2 Instance Components:**

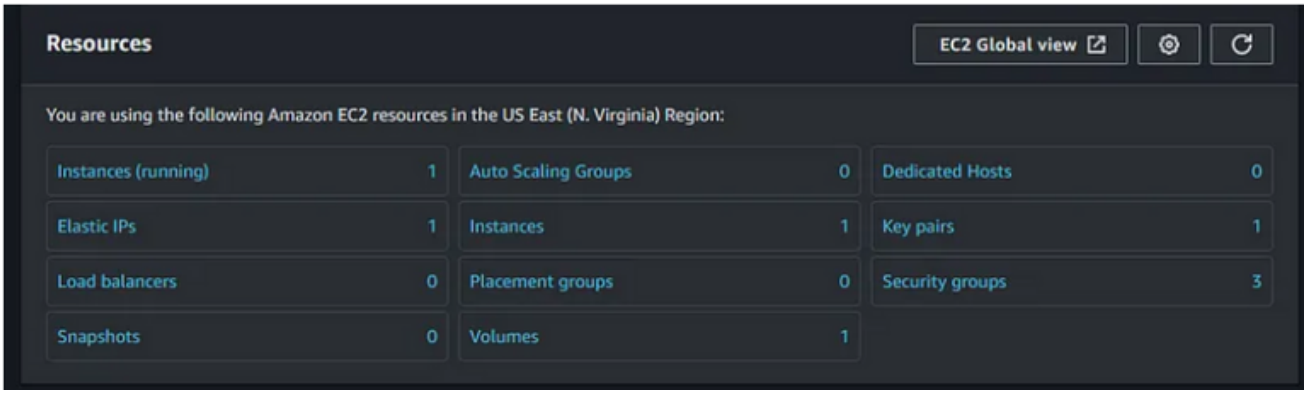
- In addition to the EC2 instance itself, other components include the network interface and volumes (virtual hard disks).
- The network interface manages connectivity and is associated with firewall rules, public IP addresses, and security groups.
- Volumes represent the virtual hard disks attached 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.

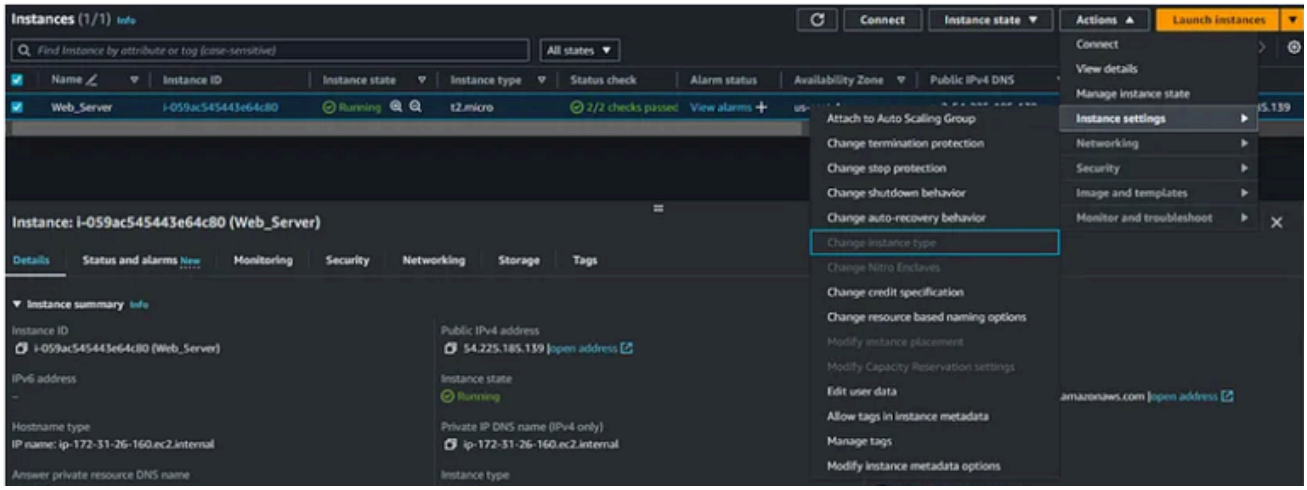



Adjusting Instance Settings:

- 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:**

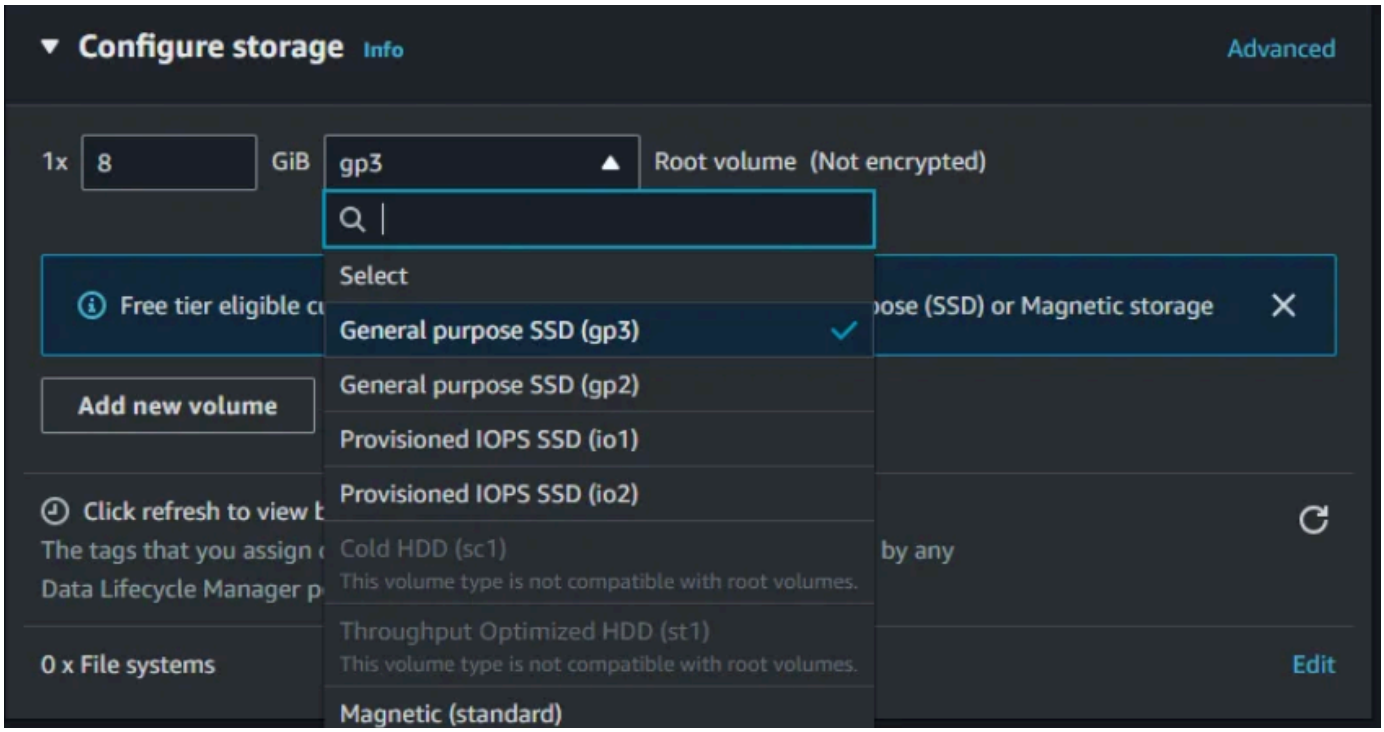
- The selection of an appropriate volume type depends on factors such as workload characteristics, performance requirements, and budget considerations.
- AWS documentation provides detailed information and use cases for each volume type, aiding users in making informed decisions.

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:

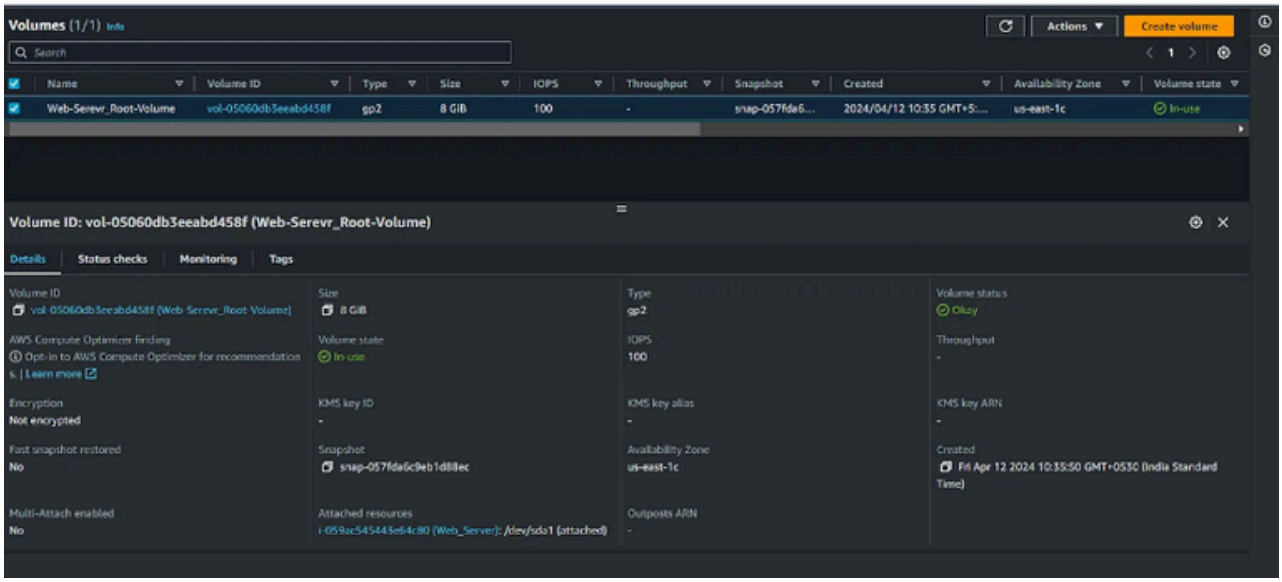
- *systemctl status apache2*
- *cd /tmp*
- *wget https://www.tooplate.com/zip-templates/2119_gymso_fitness.zip*
- *unzip -o 2119_gymso_fitness.zip*
- *cp -r 2119_gymso_fitness/* /var/www/html/*
- *systemctl restart apache2*

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.



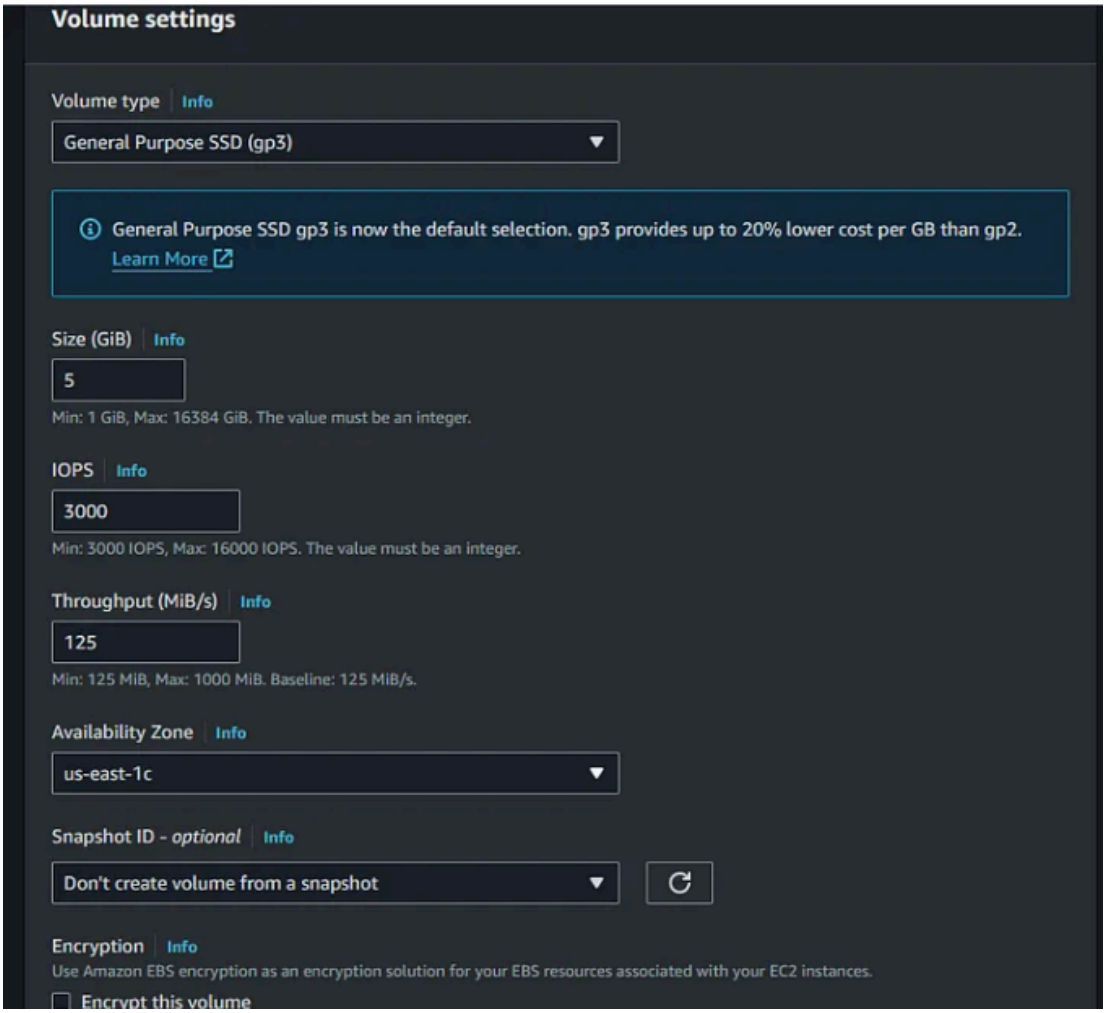
Navigate to the Volumes section located on the left-hand side of the EC2 instances page and update the name of the volume associated with the instance mentioned earlier.



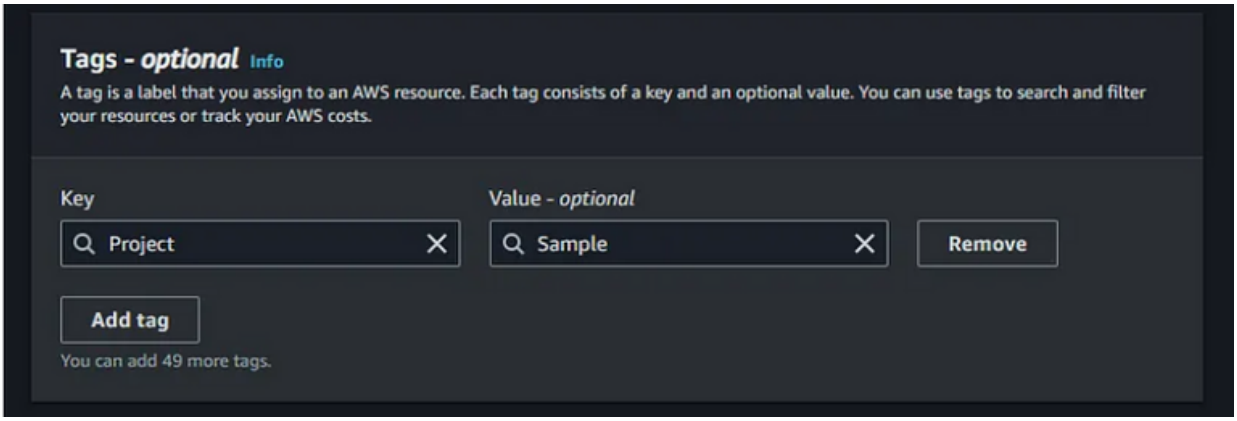

Adding extra storage to an EC2 instance, it's crucial to consider the availability zone. Both the instance and the volume need to be in the same zone to ensure connectivity.

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:

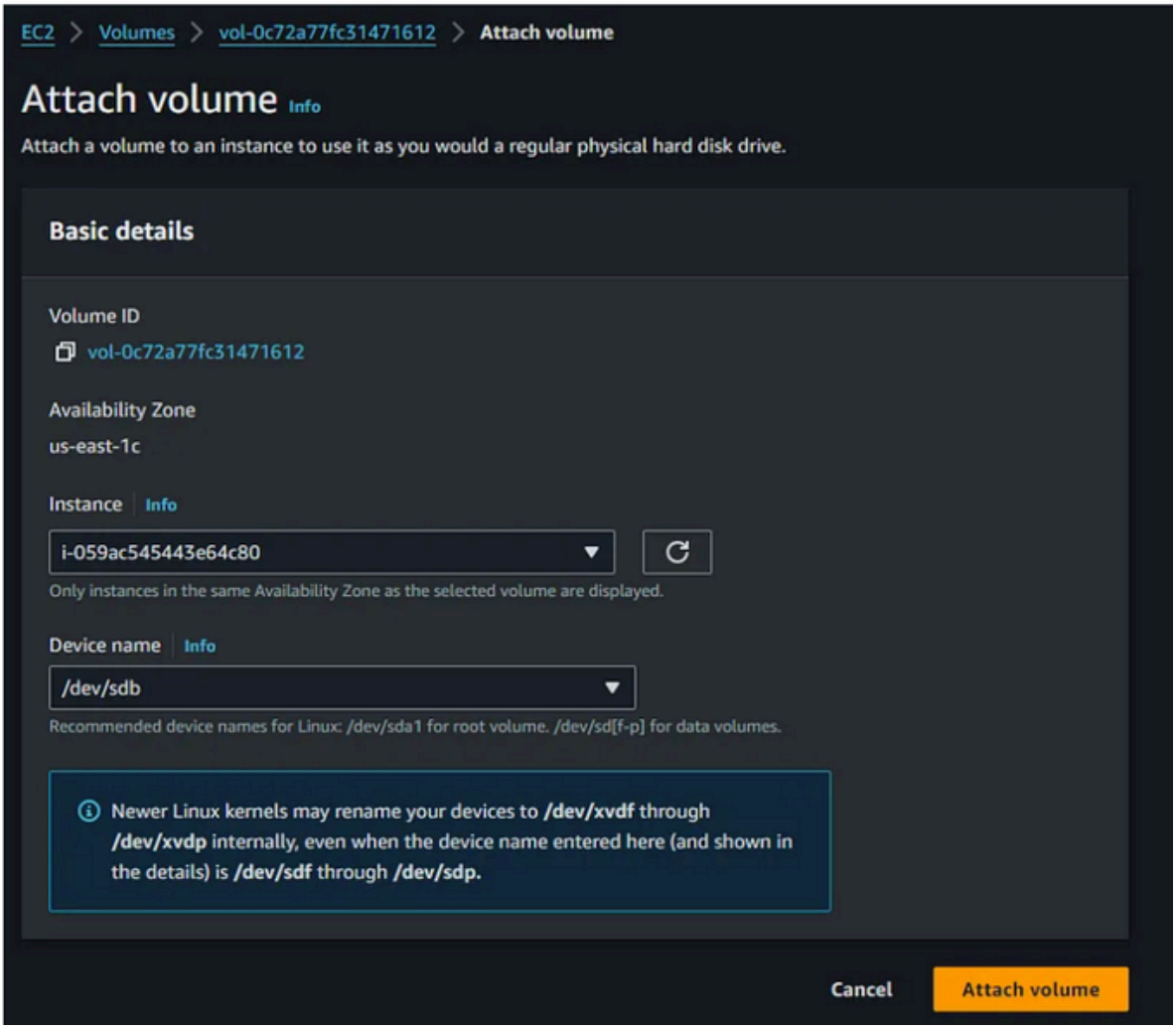



Tags, you can give upon your wish, then click on Create Volume




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.

```
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
root@ip-172-31-26-160:/tmp#
```

- 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
```

- 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.

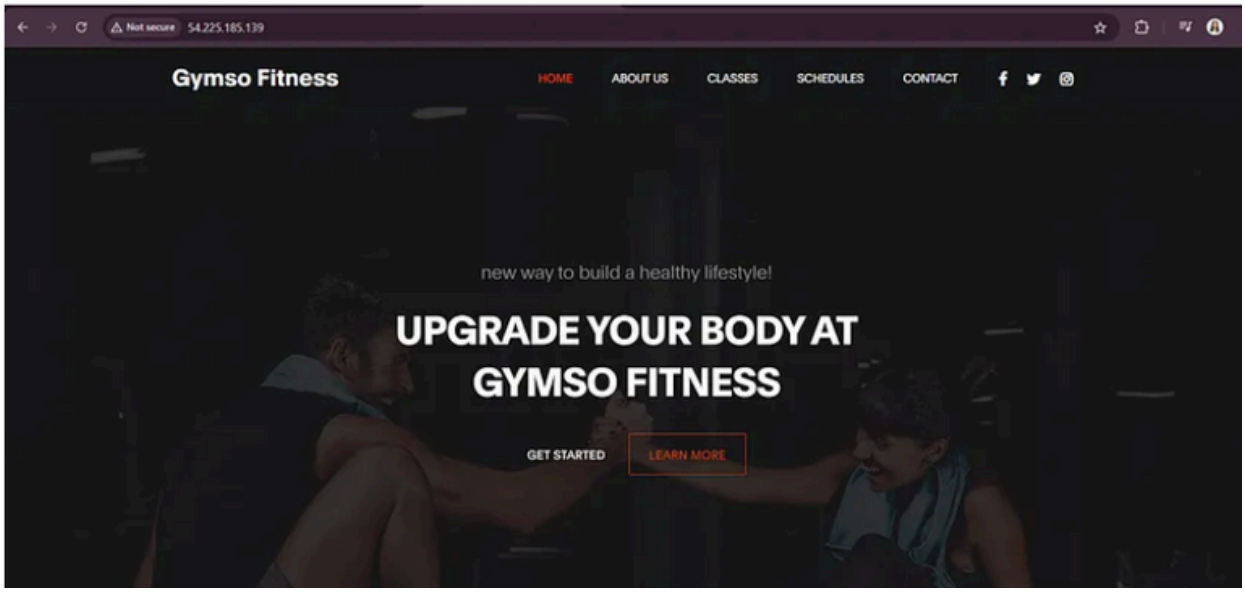
```
root@ip-172-31-26-160:~#
root@ip-172-31-26-160:~# mount -a
root@ip-172-31-26-160:~#
```

```
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.

```
LABEL=cloudimg-rootfs   /        ext4   discard,errors=remount-ro       0 1
LABEL=UEFI      /boot/efi       vfat    umask=0077      0 1
/dev/xvdbf1     /var/lib/mysql  ext4    defaults        0 0
~
~
```

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.

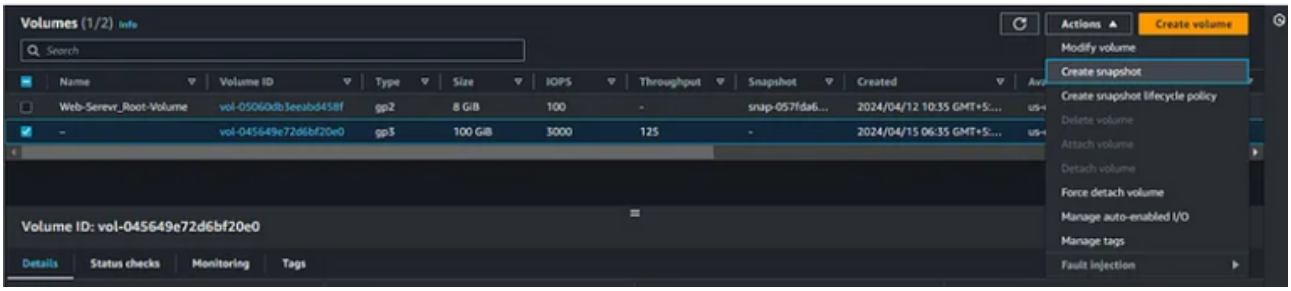
In the event of data loss, the initial step involves unmounting the partition to prevent data overwriting. If the goal is to restore the existing partition, snapshots may not be applicable, as they provide a new volume containing the data. In such cases, unmounting the partition and employing recovery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

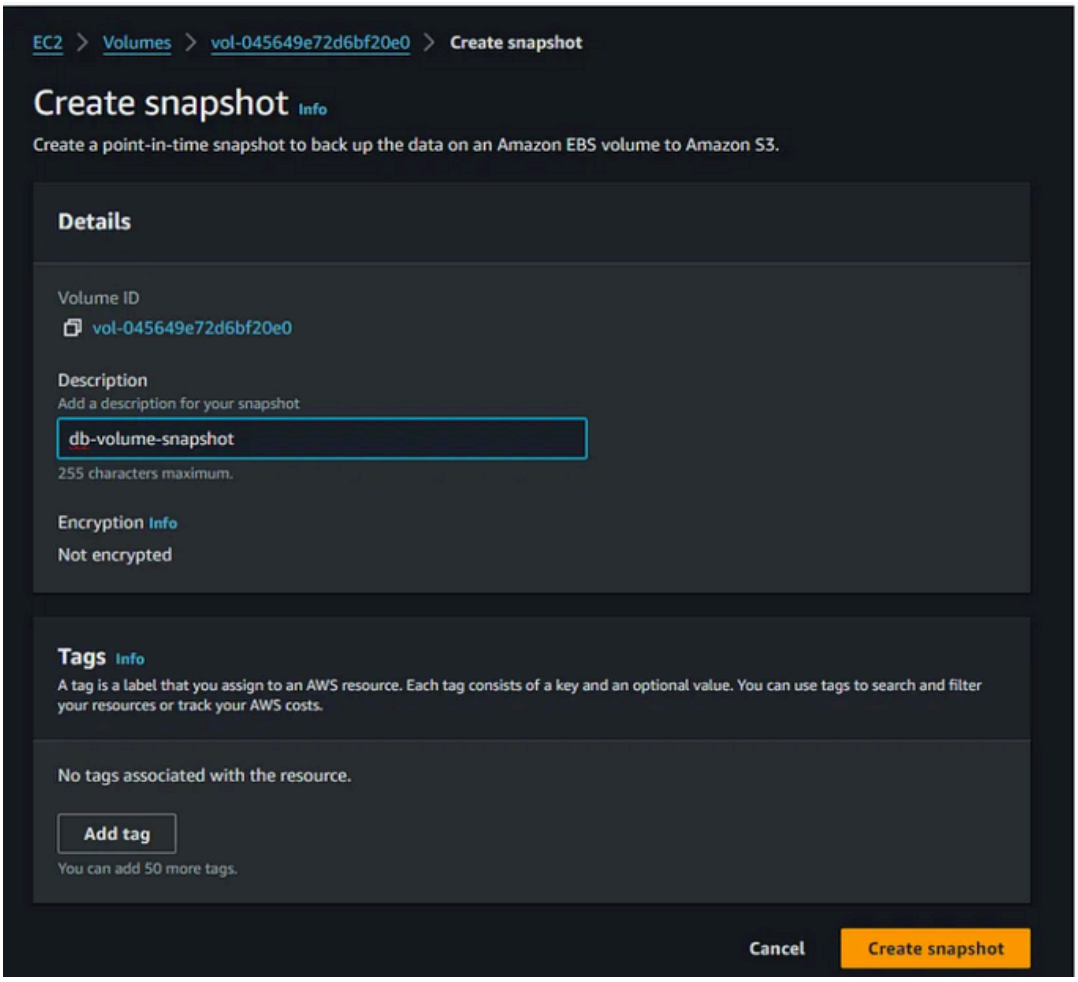
When utilizing snapshots, unmounting the partition is the preliminary action, followed by detaching the corrupted volume. Subsequently, a new volume is created from the snapshot, retaining all data and partition configurations. Upon creation, the new volume is ready for attachment and mounting, effectively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

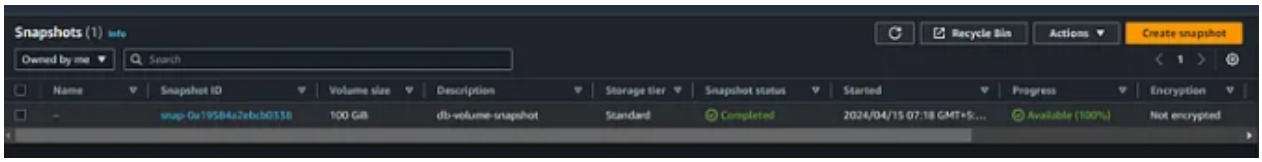
In essence, snapshots facilitate data restoration through volume replacement rather than repair, streamlining the recovery process and ensuring data integrity.

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.



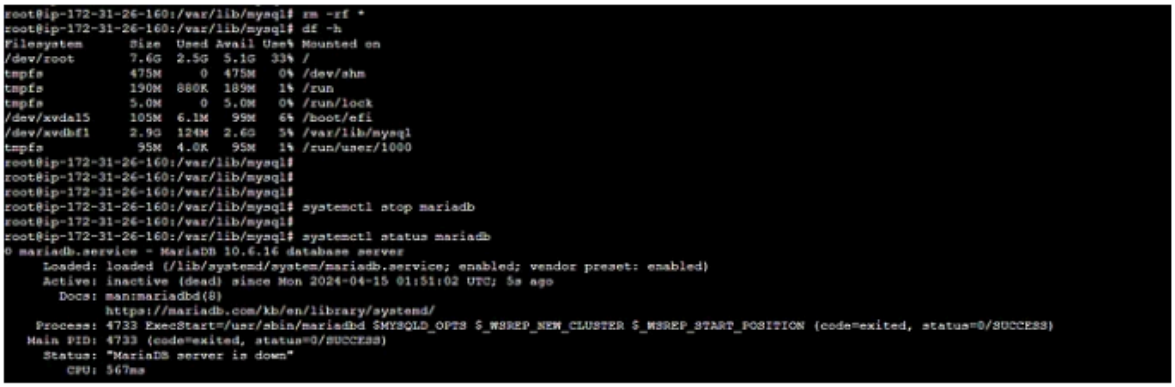


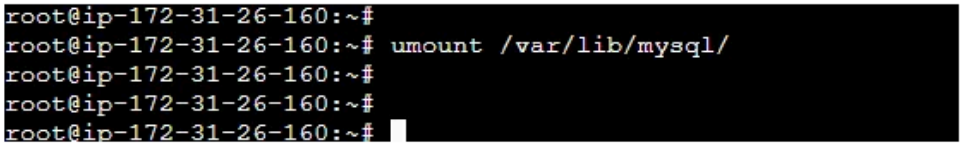

**Delete Data:**

- 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:

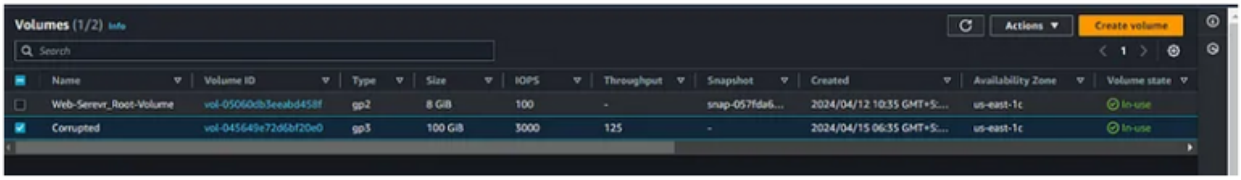
- Stop any relevant services to prevent further data overwrites.
- Unmount the affected directory, e.g., /var/lib/mysql.



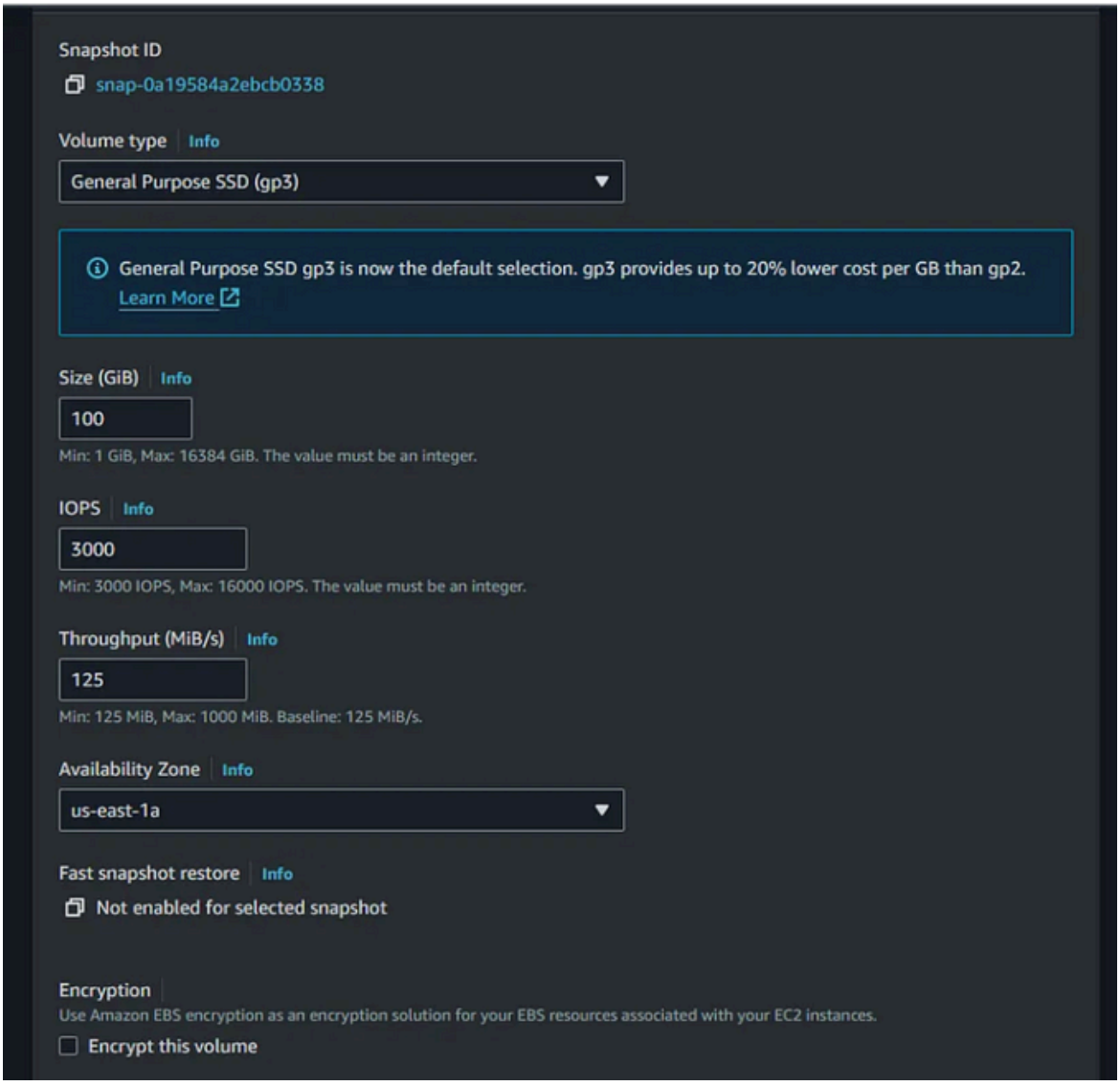

Detach Corrupted Volume:

- 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.



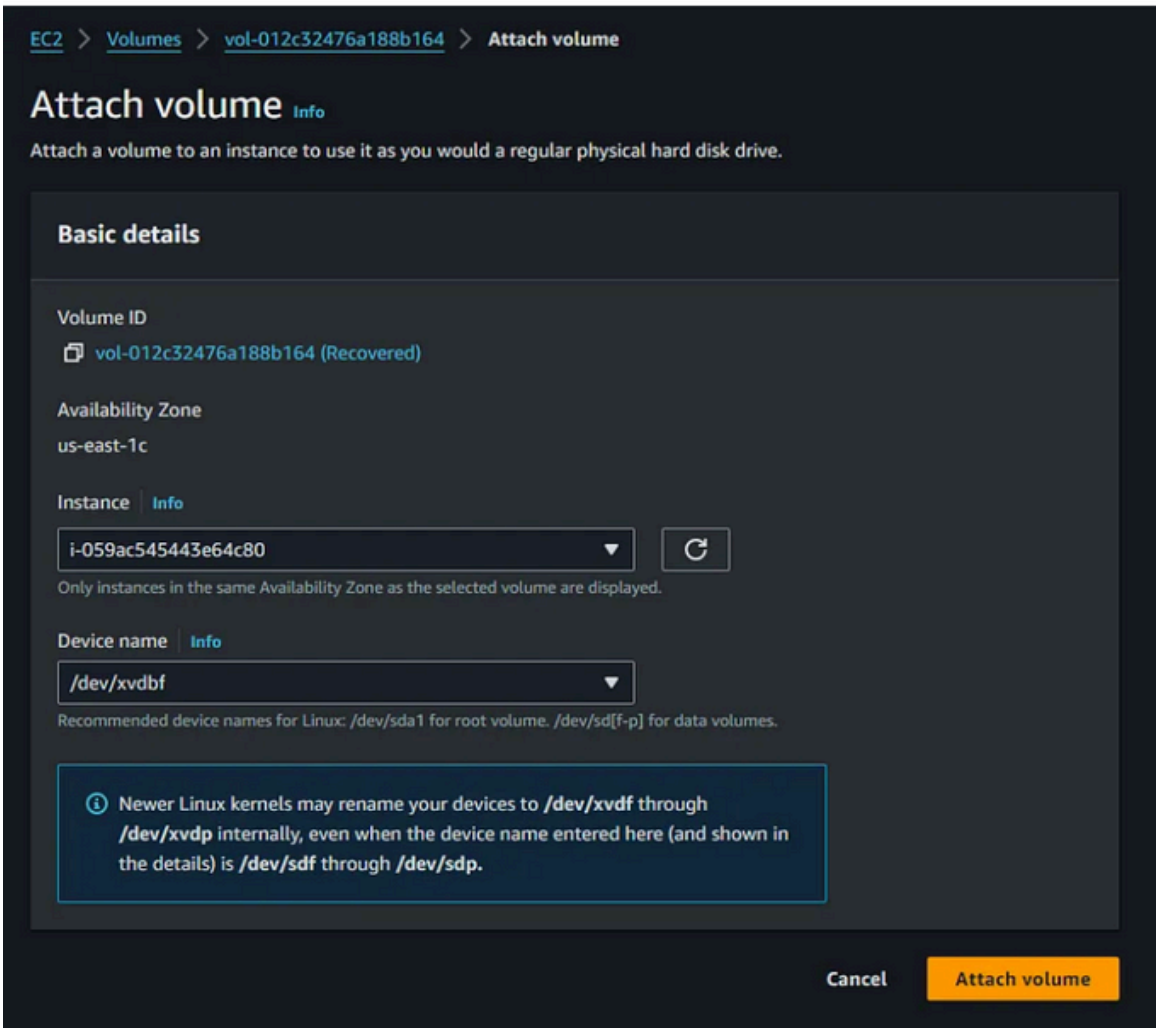

Recover from Snapshot:

- 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 multi-site 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 ultra-low latency, making it suitable for TCP, UDP, and TLS traffic.

■Load Balancer Targets:

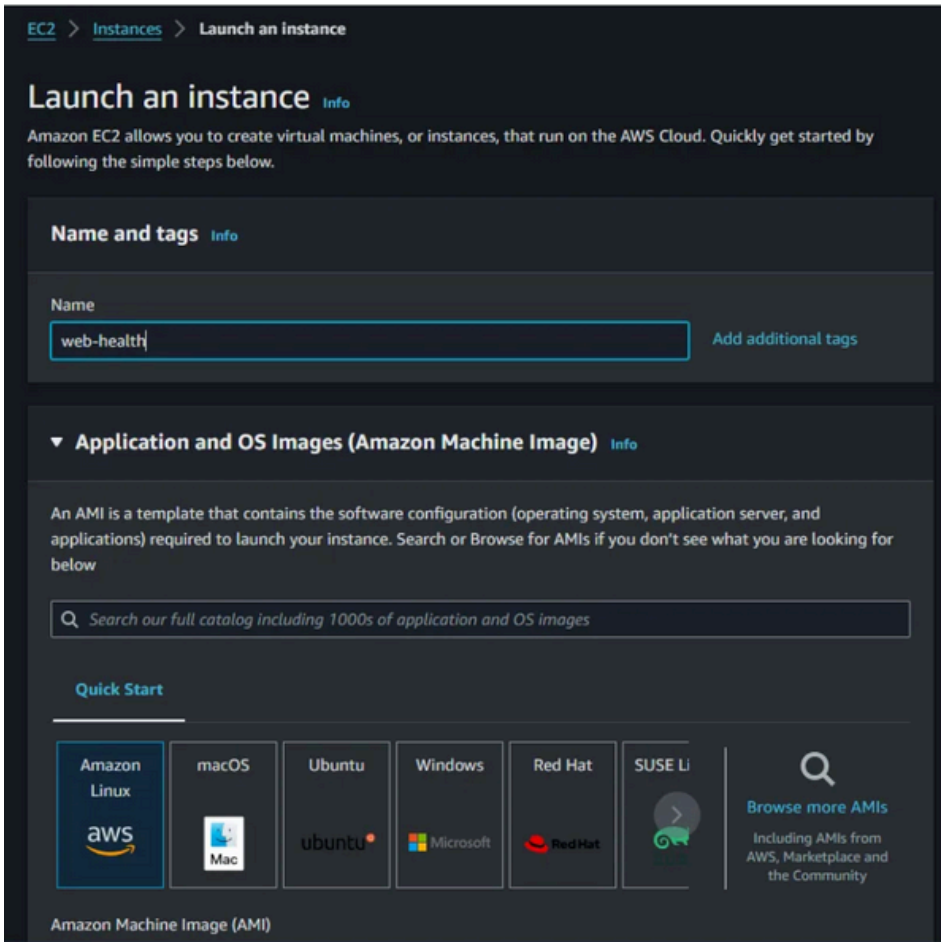
- ELB distributes traffic to backend targets, which can include EC2 instances, containers, IP addresses, or Lambda functions.
- Targets are distributed across multiple availability zones to improve fault tolerance and high availability.

Benefits of ELB:

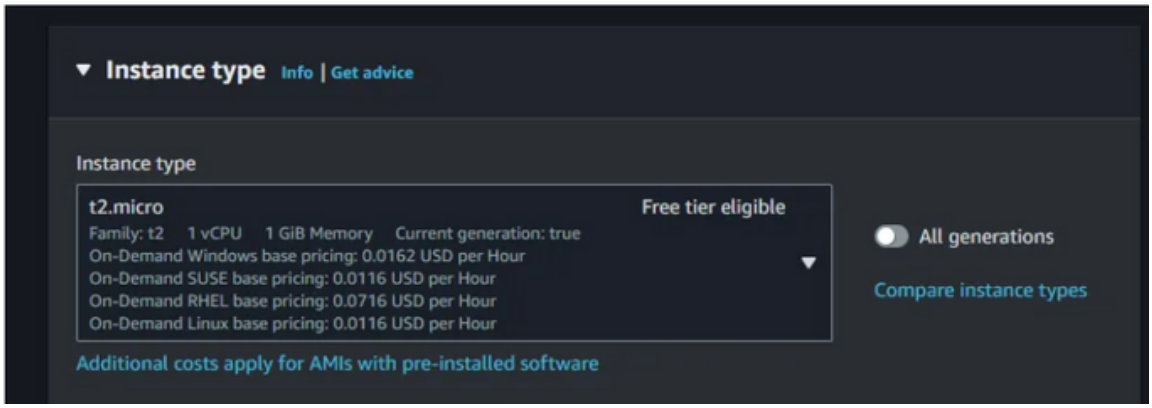
- Scalability: ELB automatically scales in response to incoming traffic, ensuring that applications remain responsive even during traffic spikes.
- Fault Tolerance: By distributing traffic across multiple targets and availability zones, ELB enhances fault tolerance and resiliency.
- Simplified Management: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

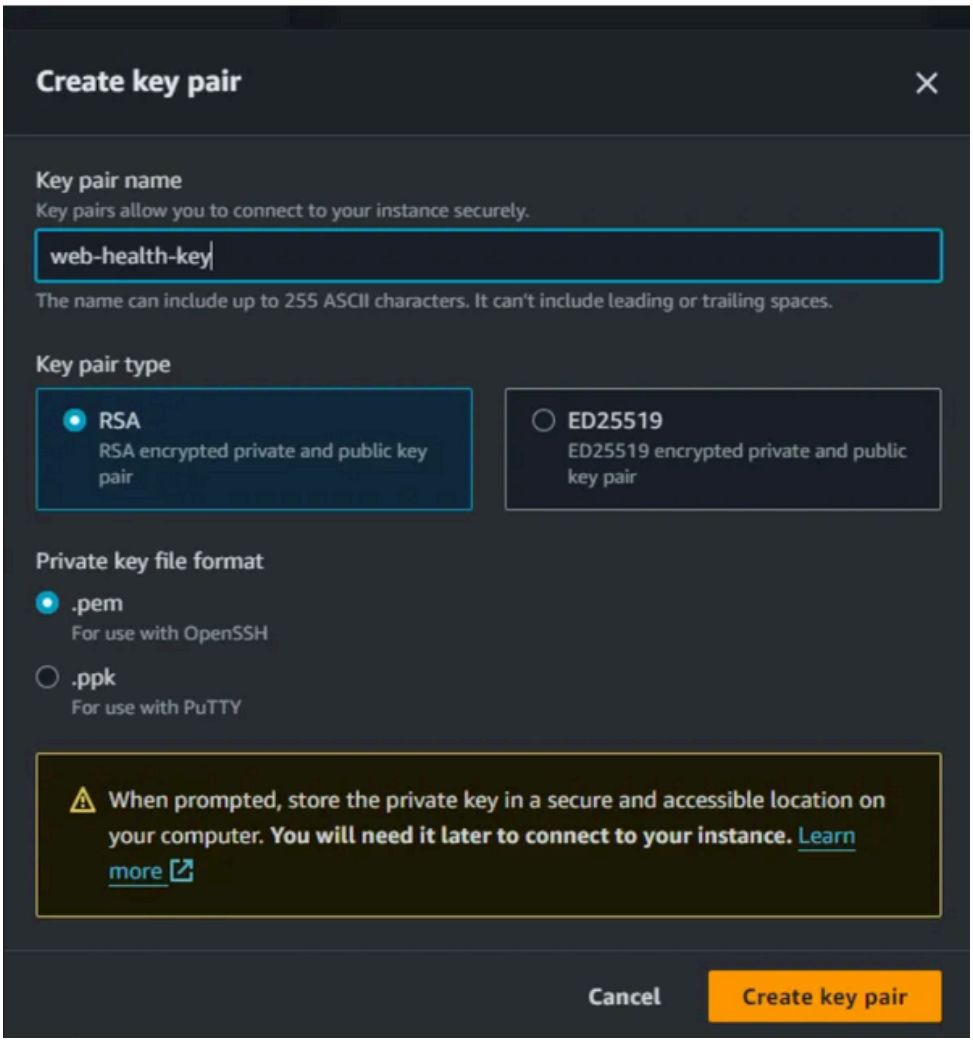
Let's proceed with launching an instance and naming it as "web-health".

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.

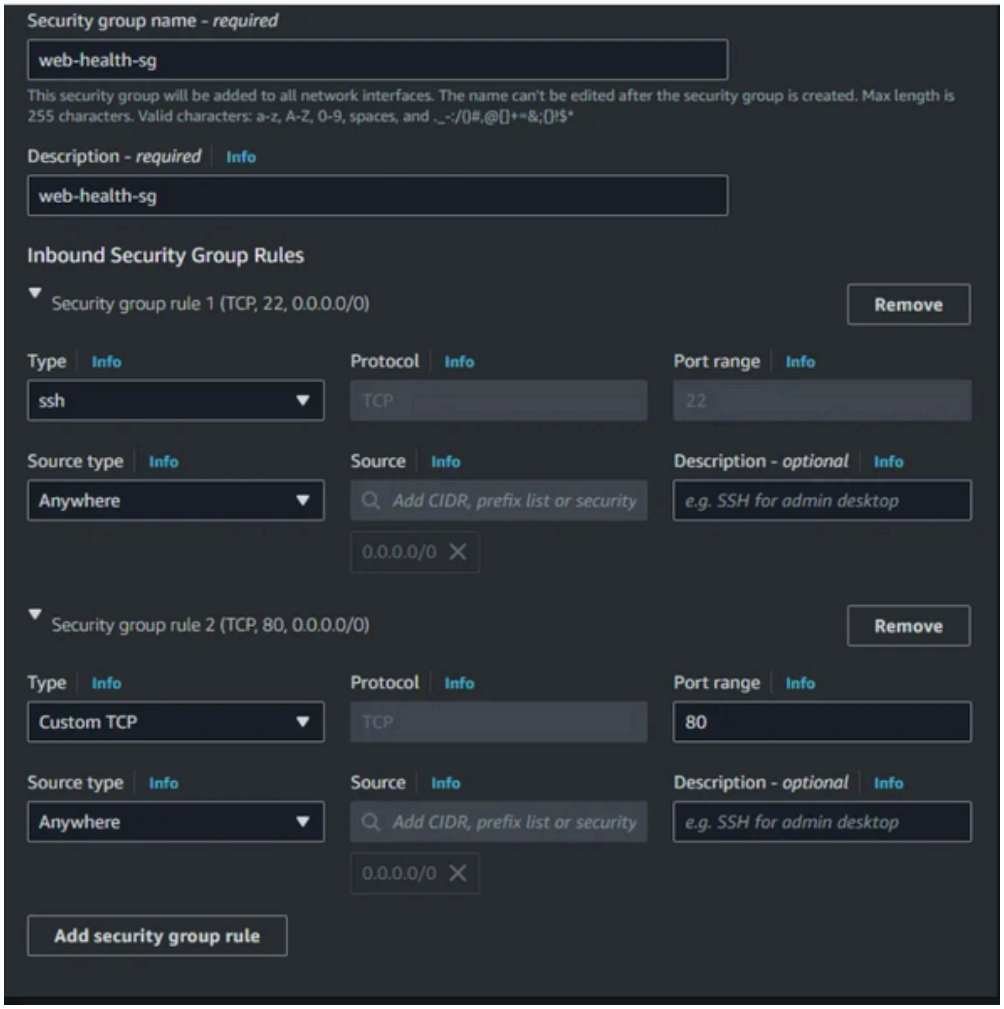


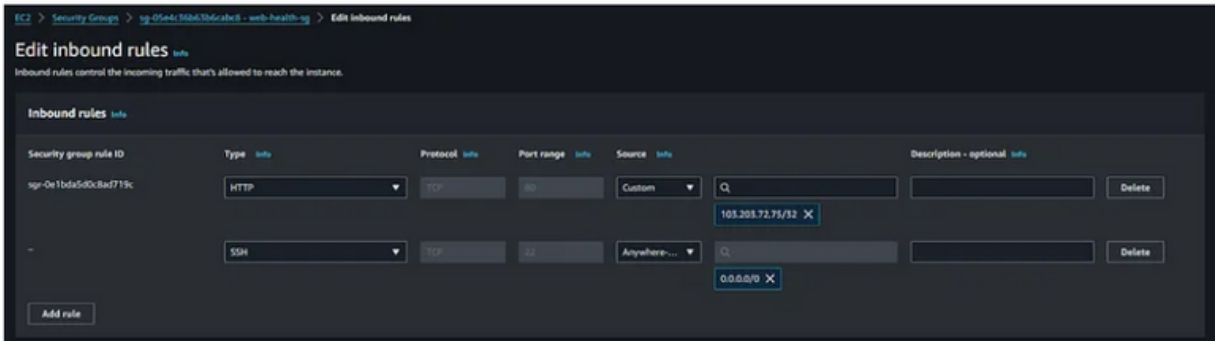
Let’s opt for a t2.micro instance type and create a new key pair named “web-health-key” for secure access.




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.

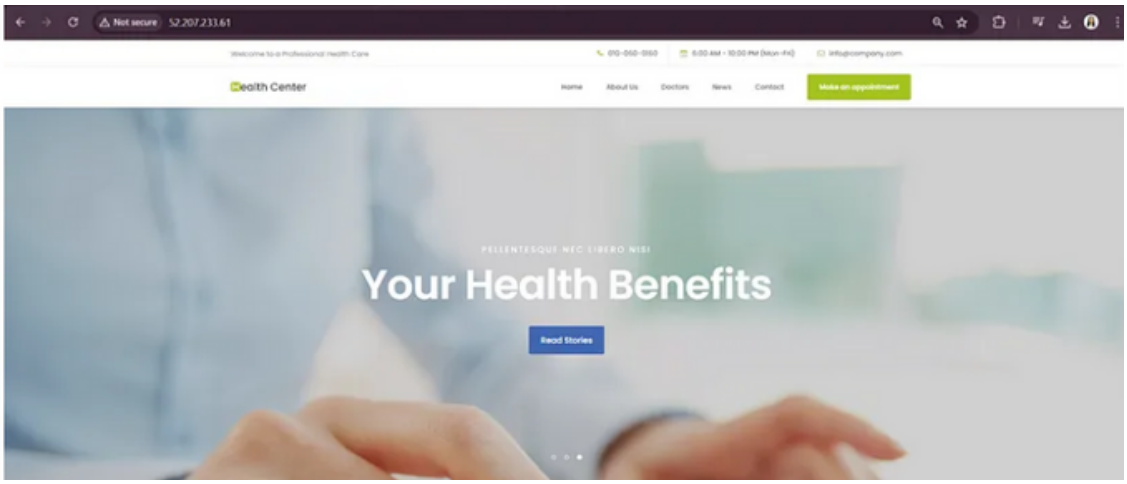
**Script:**

```
#!/bin/bash

# Variable Declaration
#PACKAGE="httpd wget unzip"
#SVC="httpd"
URL='https://www.tooplate.com/zip-templates/2098_health.zip'
ART_NAME='2098_health'
TEMPDIR="/tmp/webfiles"

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 "########################################"
 systemctl restart $SVC
 echo

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

 sudo systemctl status $SVC
 ls /var/www/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/www/html/
```

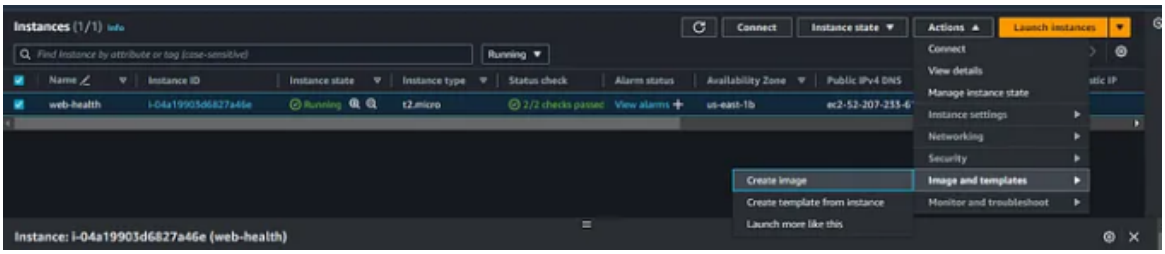
After launching the instance, it typically takes about 5 minutes to initialize. Once it's up, we can check the public IP address and access the website via a browser.



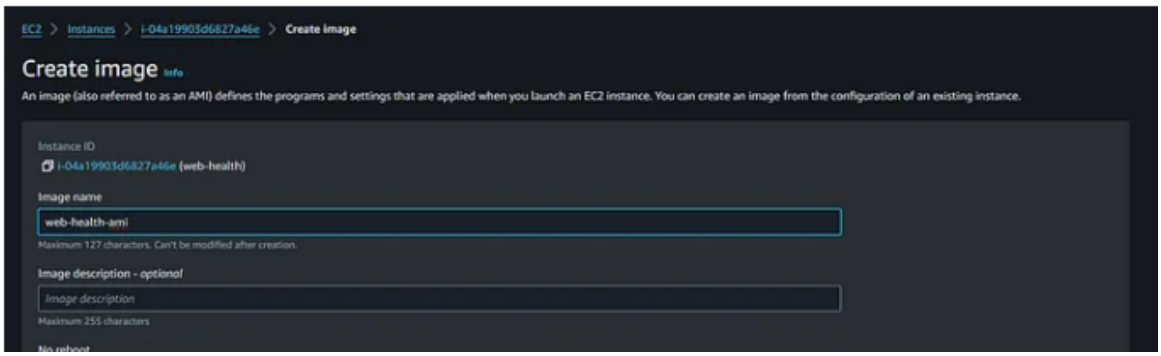
In case you encounter any issues accessing the website, ensure that the security group allows traffic from your IP and that the HTTP service is running on the instance.

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.

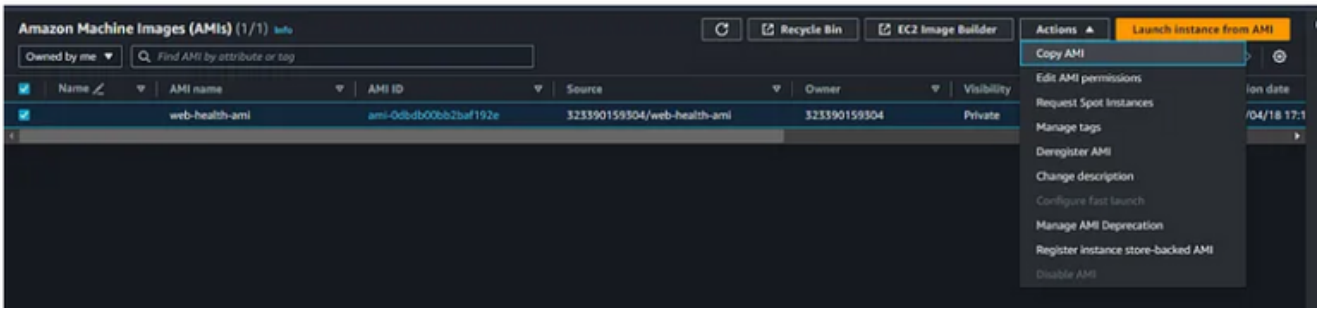



Provide a name for the AMI — Here I gave — "web-health-ami".




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

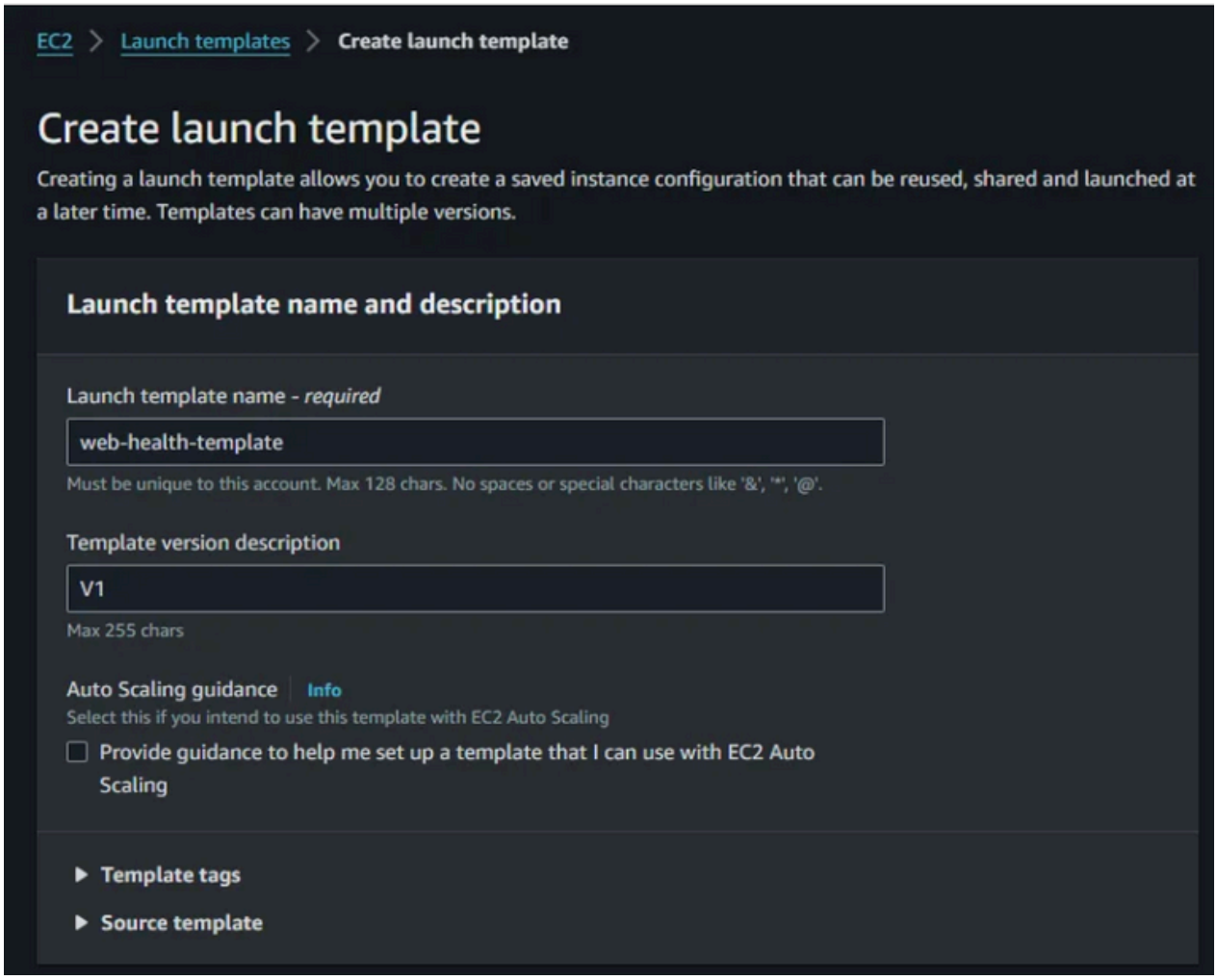
Additionally, we can explore options like copying the AMI to other regions or sharing it with other AWS accounts for collaboration.



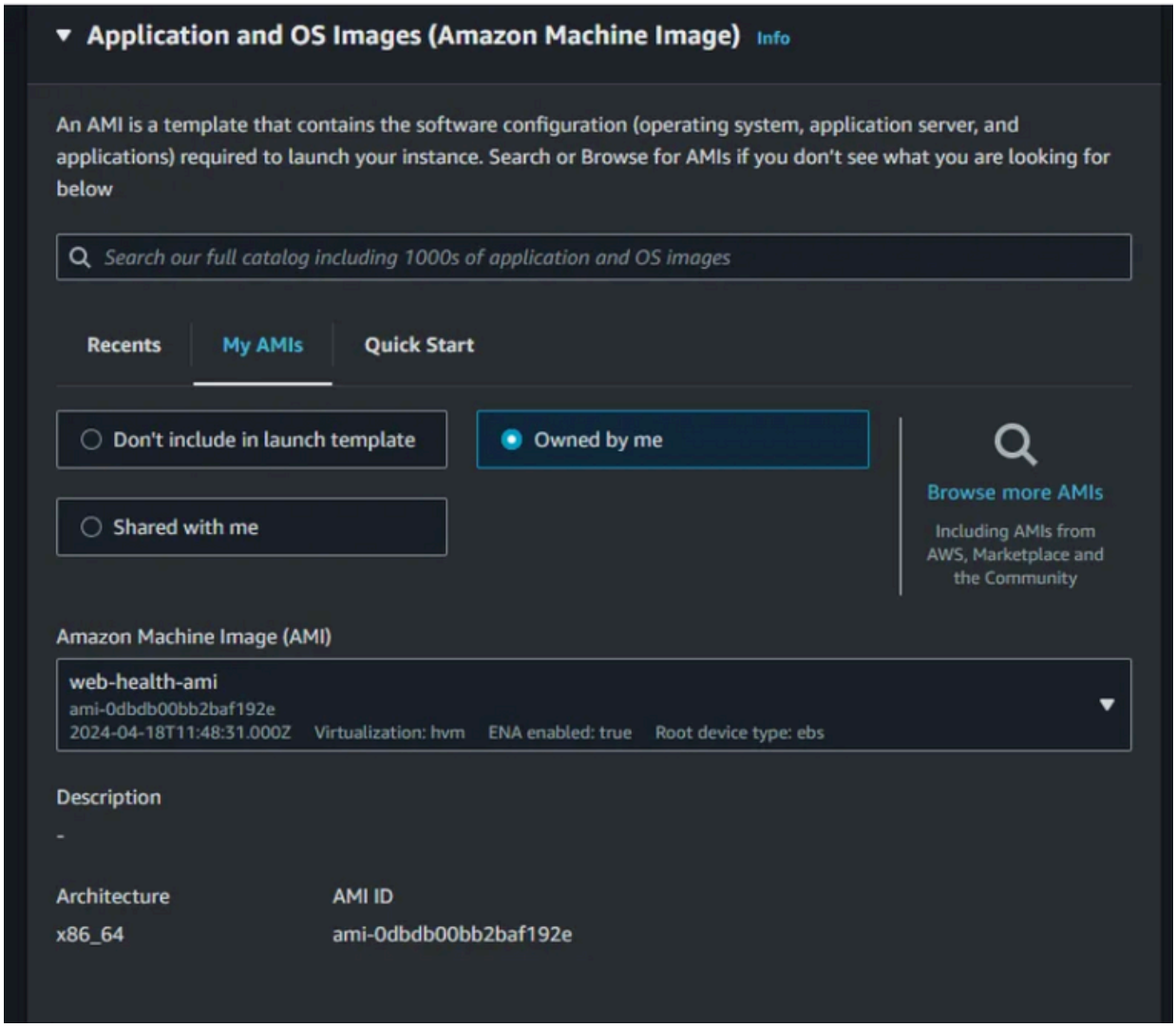
Also, AWS offers a service known as EC2 Image Builder, allowing you to establish automated pipelines for image creation.

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.

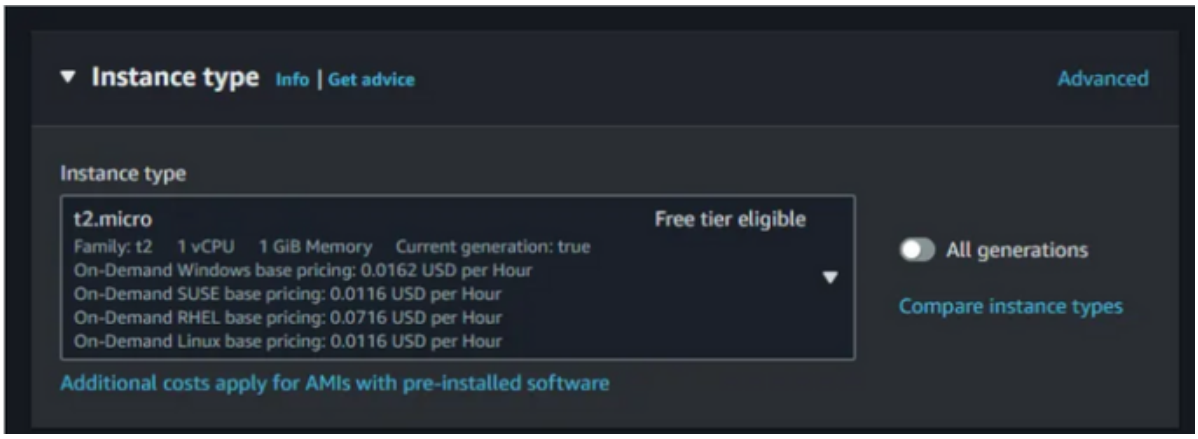
With the AMI in place, we can streamline instance launches by creating a launch template. Given name as “web-health-template” and version as “V1”



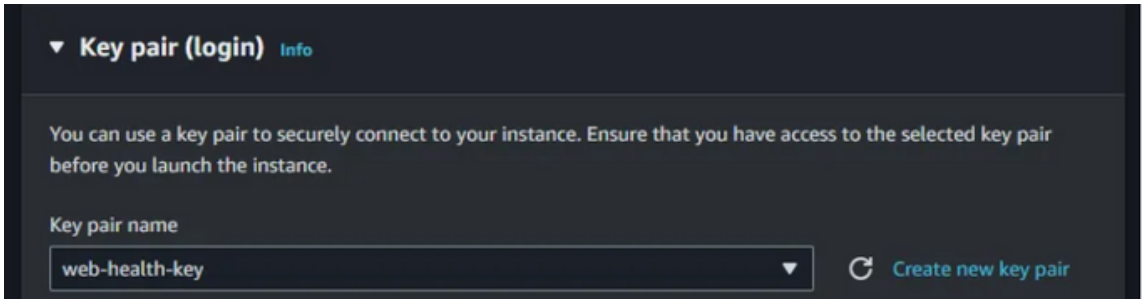

Then choose the “My AMIs” and select the AMI we created just before “web-health-ami”



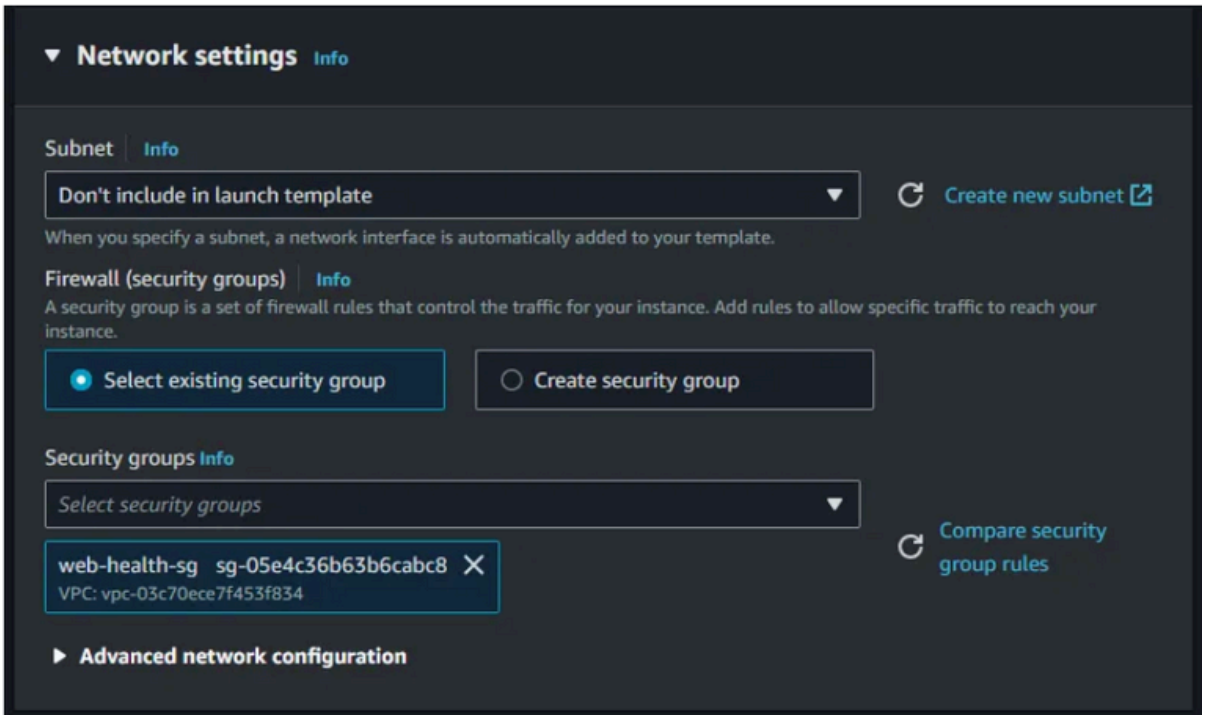

Select the instance type as t2.micro



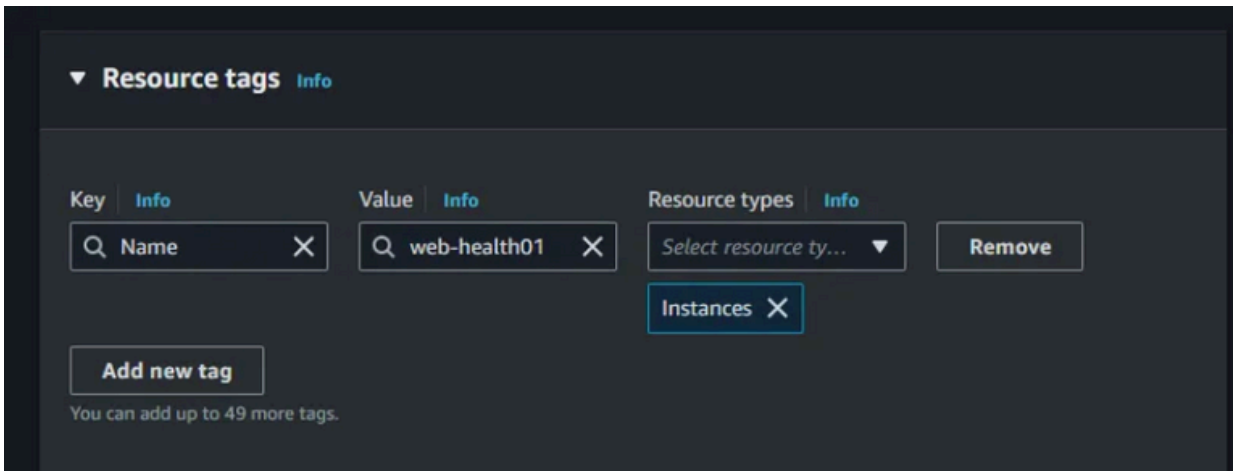

Select our key pair we created for the instance “web-health”

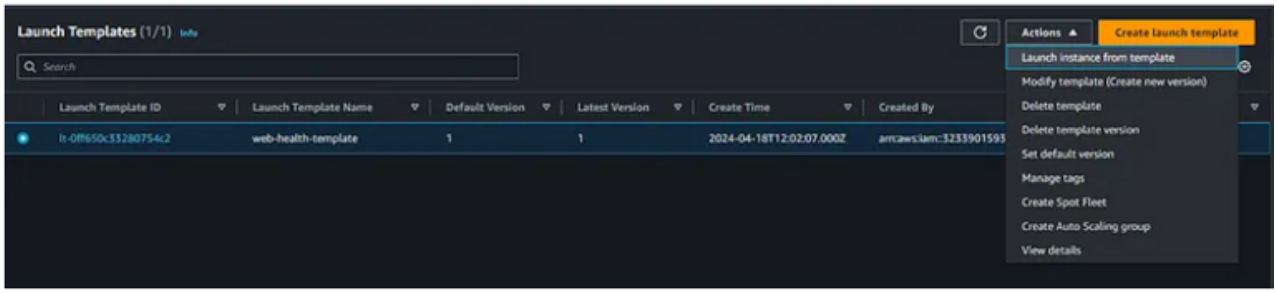


Select the security group as “web-health-sg” we created for our instance “web-health”




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.



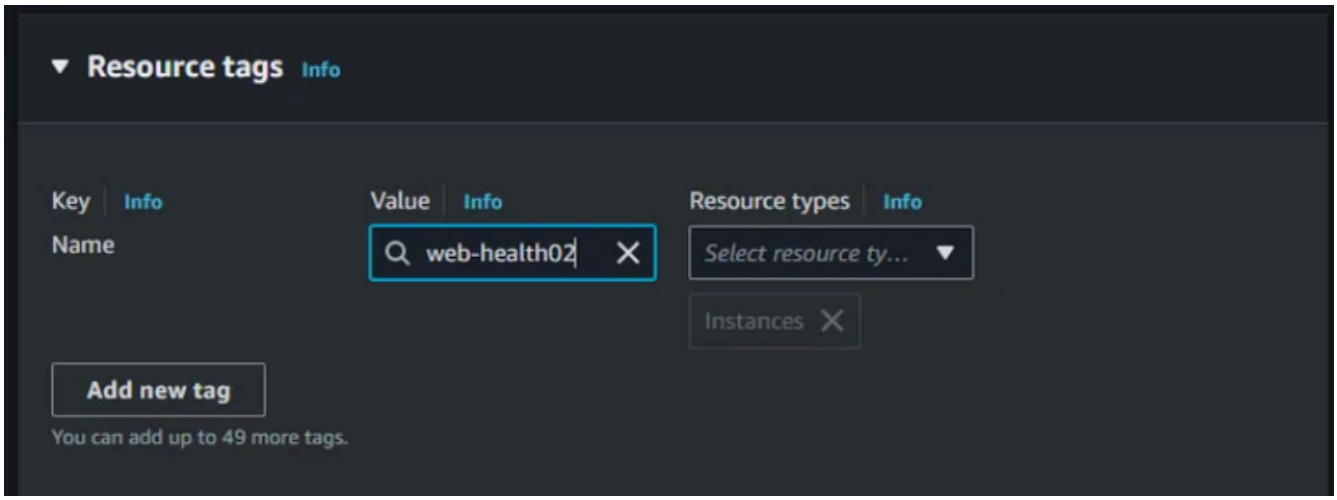




With the AMI in place, we can streamline instance launches by creating a launch template. This template encapsulates all the configuration details, allowing for quick and consistent instance deployments.

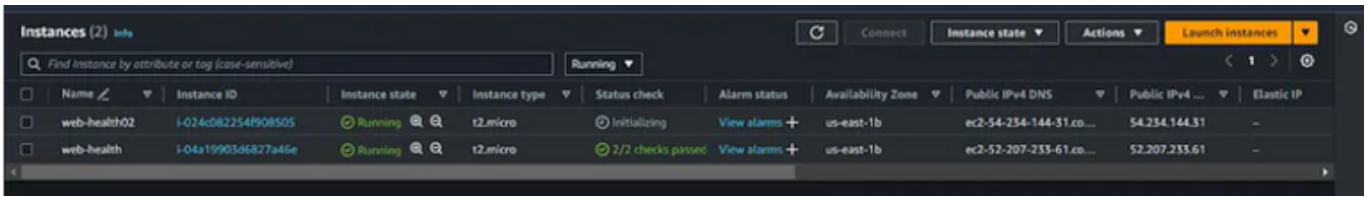
By utilizing launch templates, we can expedite the process of launching instances and maintain standardized configurations across our infrastructure.

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.



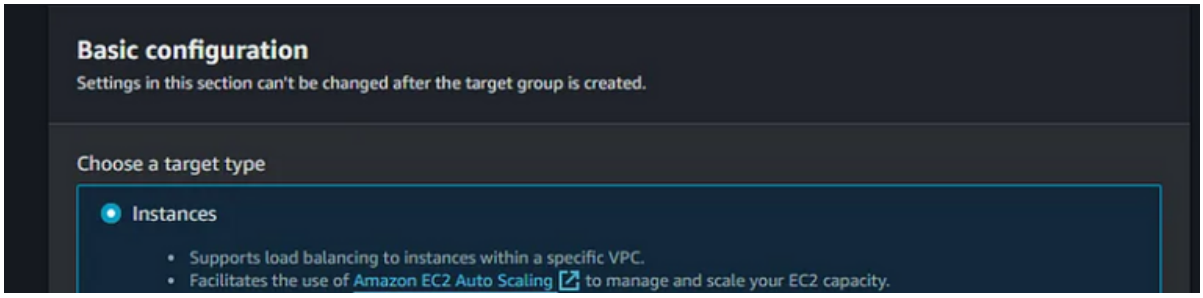

We have successfully created an AMI and established a launch template utilizing this custom AMI.

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.

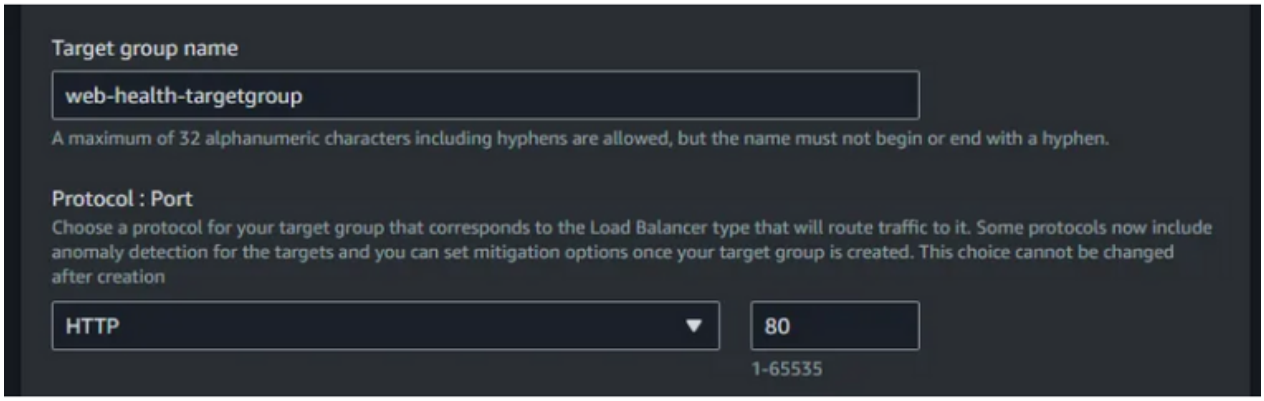
In the Load Balancer section, we will first navigate to target groups. Target groups function as collections of instances, each subject to health checks. Essentially, they are groups comprising instances. Let's proceed to create a target group.

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.

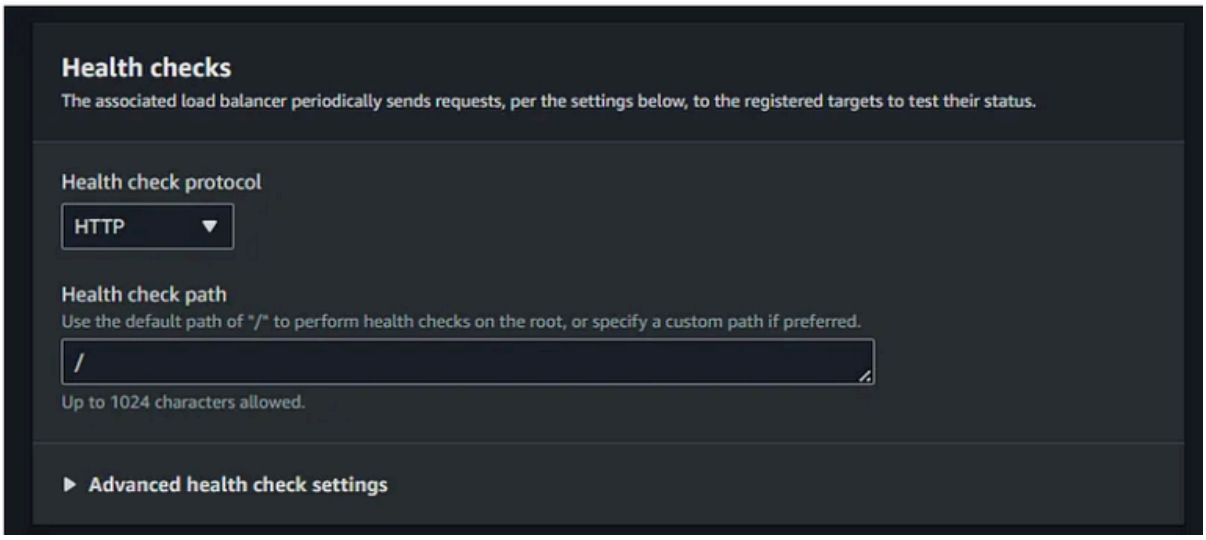



In our setup, we've implemented health checks within the target group. These health checks allow the target group to assess the status of each instance. If an instance is deemed unhealthy during these checks, the target group will refrain from routing requests to that specific instance.

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 (/).

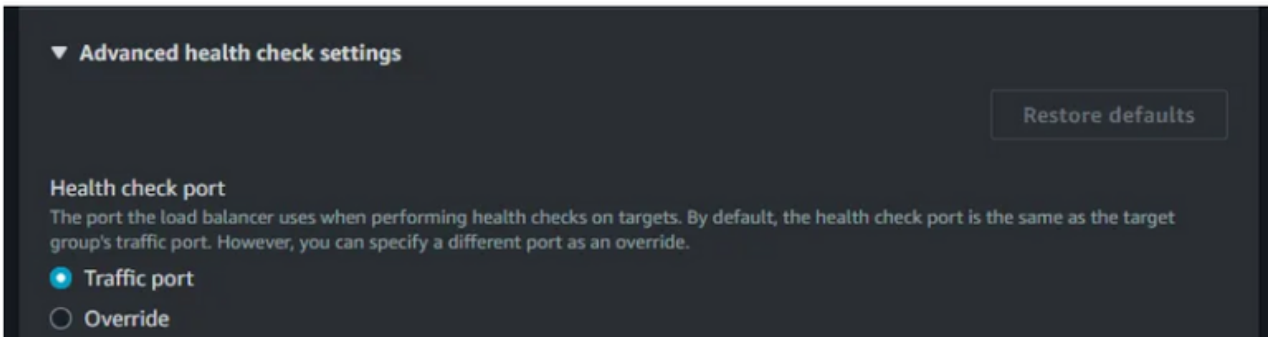
Let's break down what the health check path (/) signifies. Imagine your website's URL is http://IP:80. The slash at the end of this URL represents the root directory of the website. It's where the homepage resides.

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.



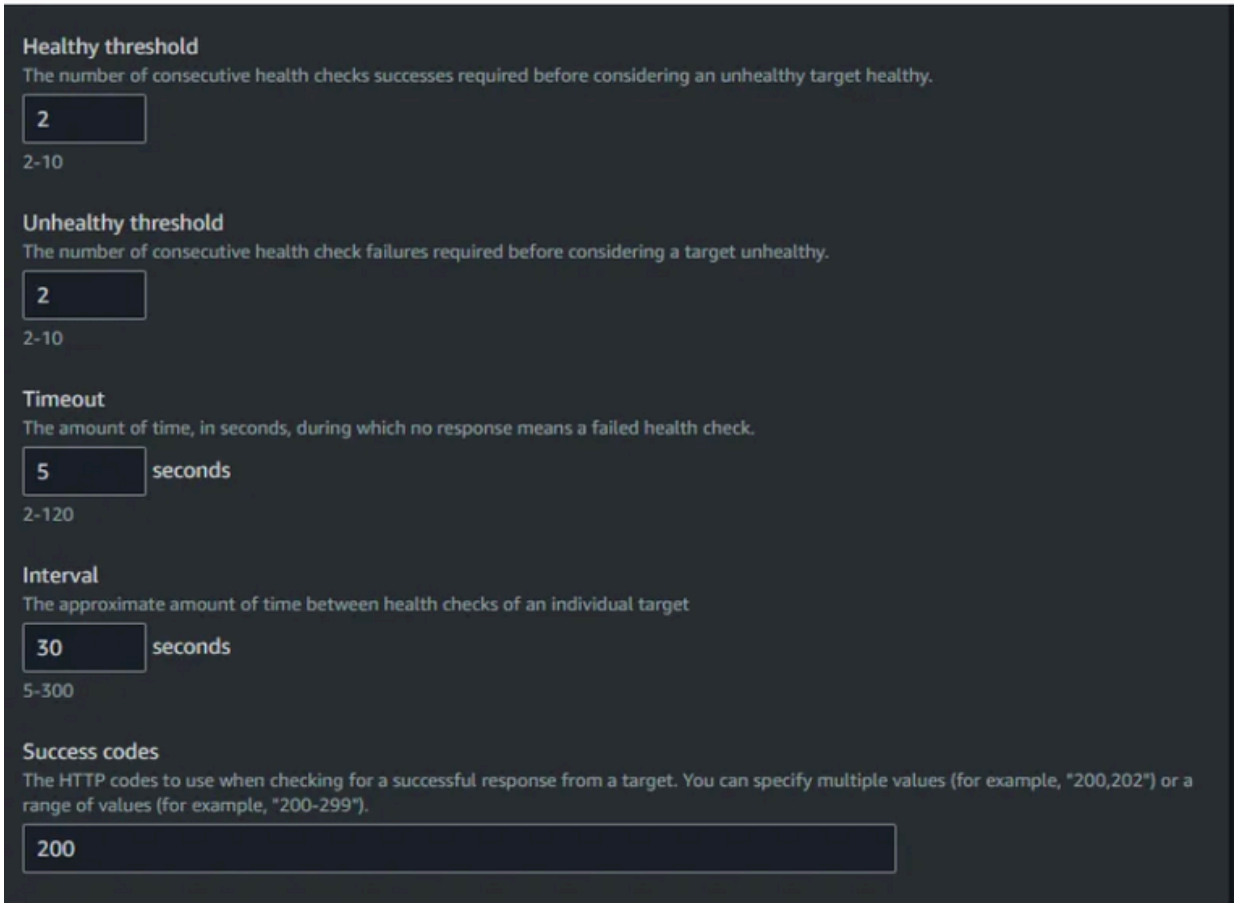

The “Healthy threshold” parameter determines the number of consecutive successful health checks required before declaring the instance as healthy. By default, it is set to five, but you can adjust it to two or ten as per your preference. With a threshold of five, the system checks the instance’s health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

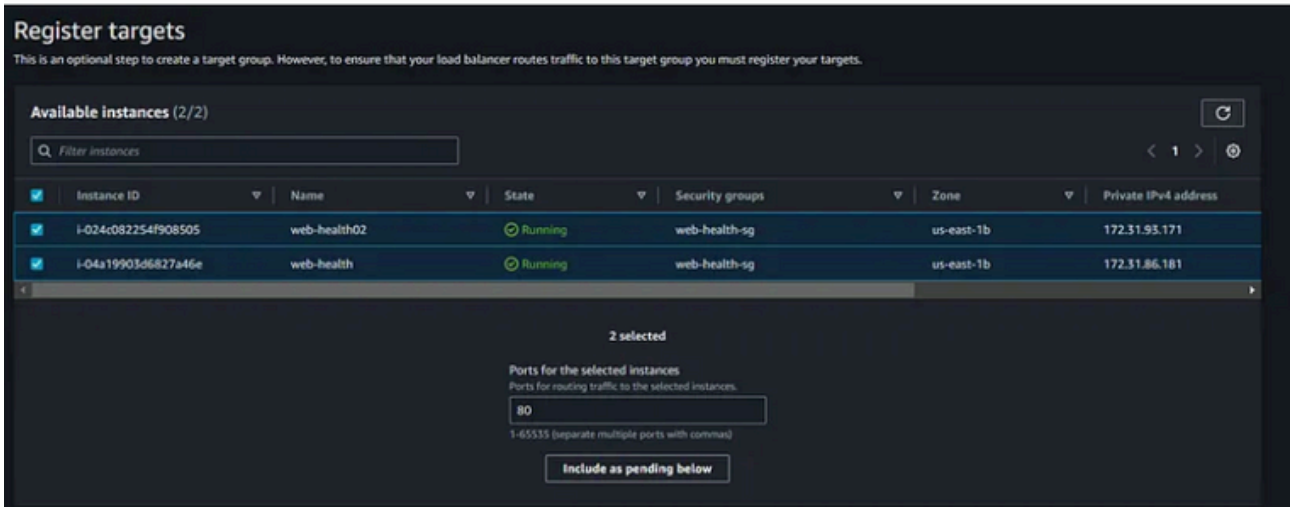
The “Timeout” parameter determines how long the system waits for a response from the website during each health check. If the website does not respond within the specified timeout period (default is 5 seconds), the system moves on to the next check.

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.

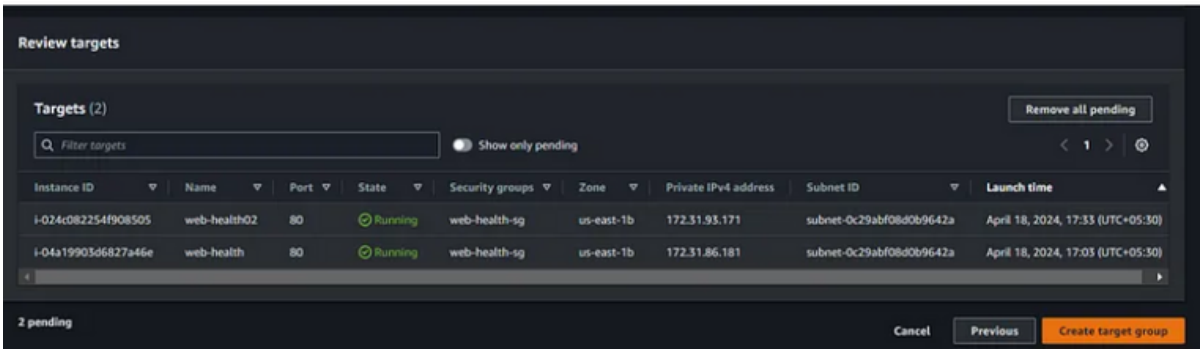
When performing health checks, the system looks for a success exit code, typically HTTP 200, indicating that the website is functioning properly. While we can visually confirm this by checking the website in a browser, the target group relies on these exit codes to determine the instance's health status.




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

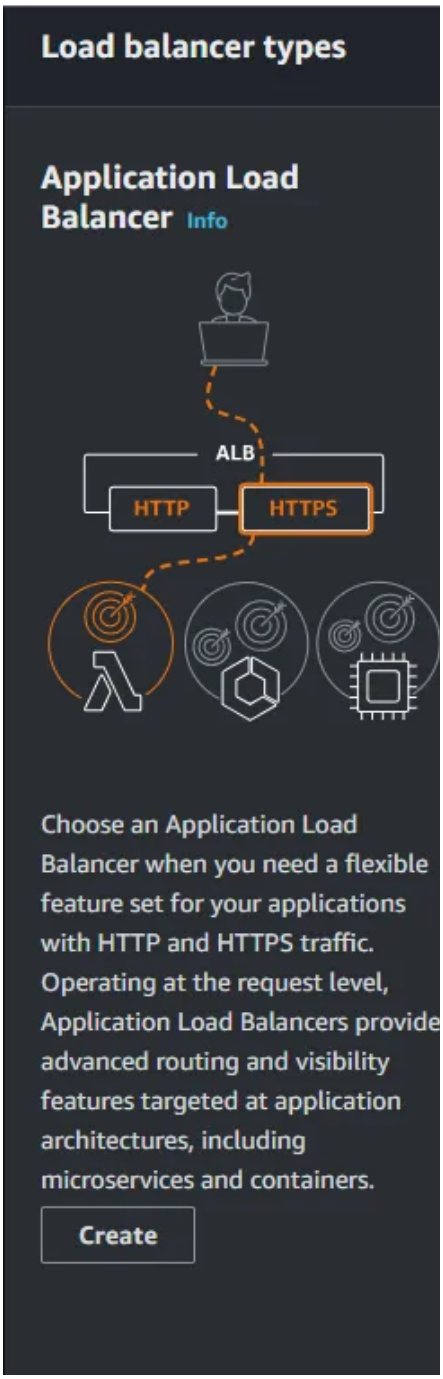



And the targets are in running state.

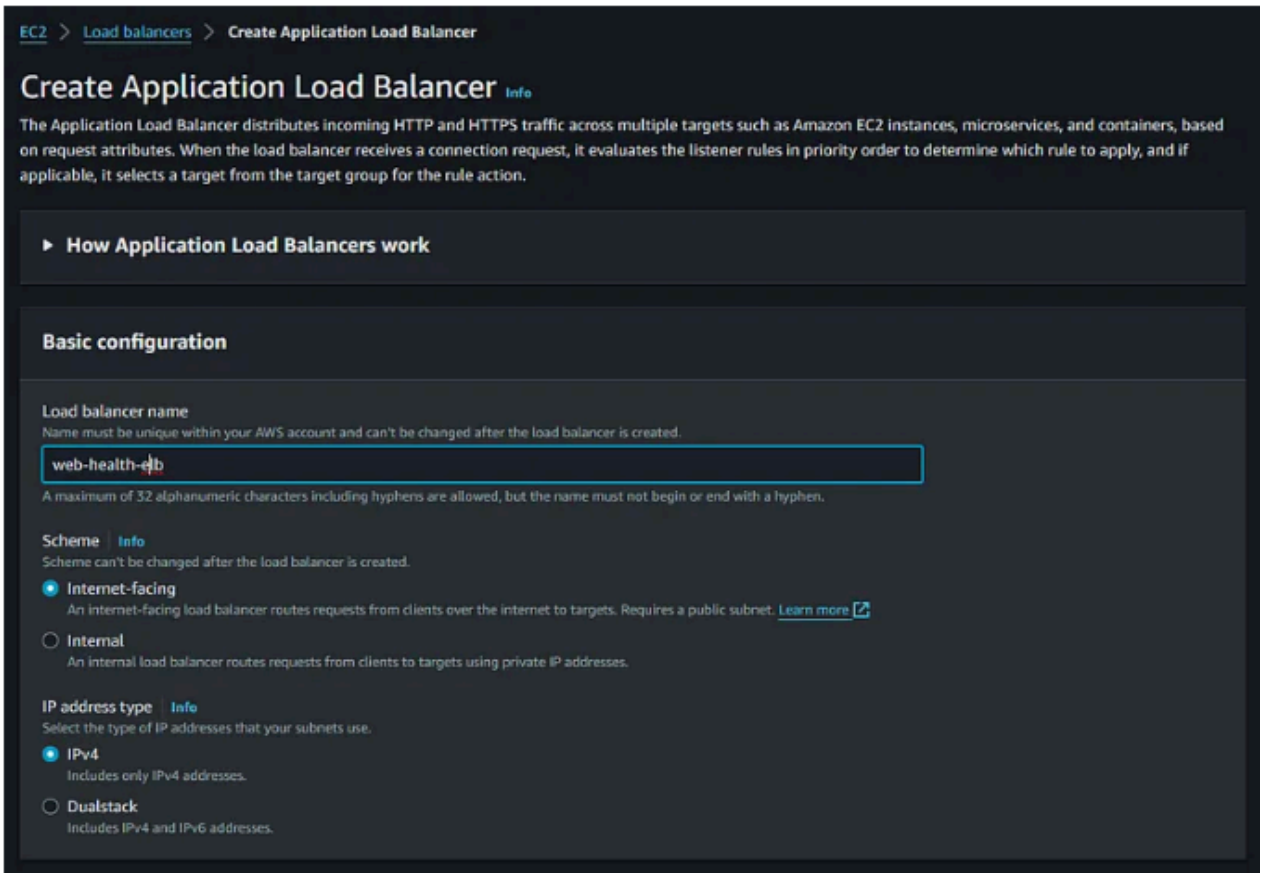



Once the target group is set up, we'll proceed to create an application load balancer (ALB), specifying its name, internet-facing nature, and availability zones.

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



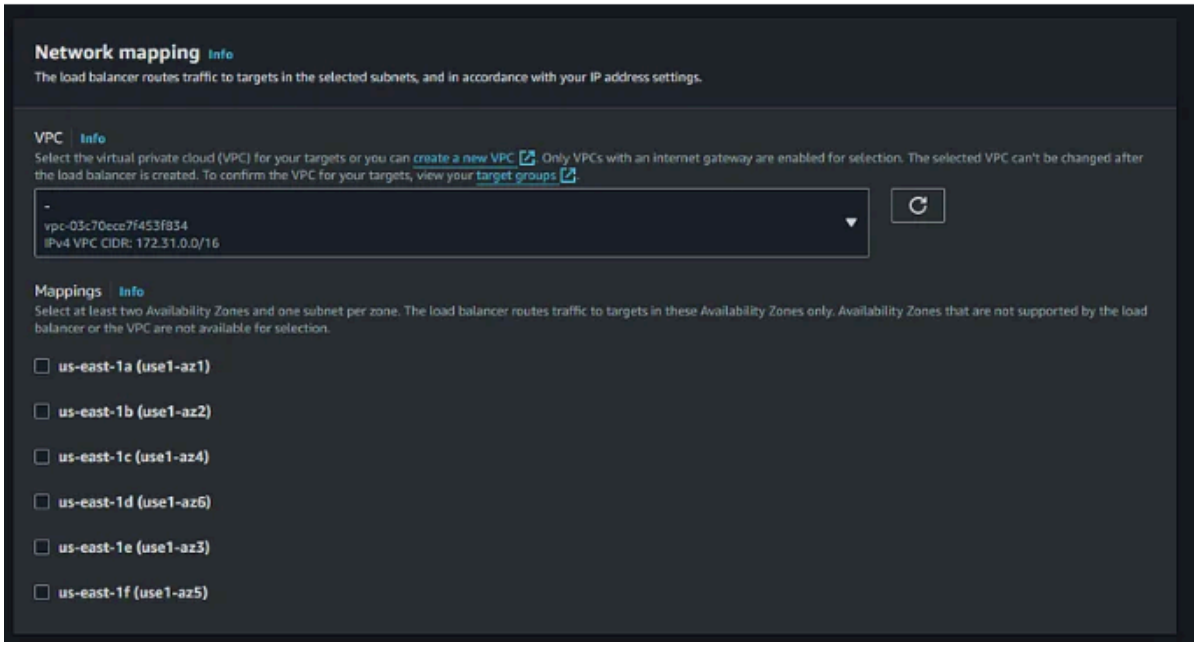
Give a load balancer name “web-health-elb” and We want it to be available on the Internet, so we’ll keep it Internet facing.



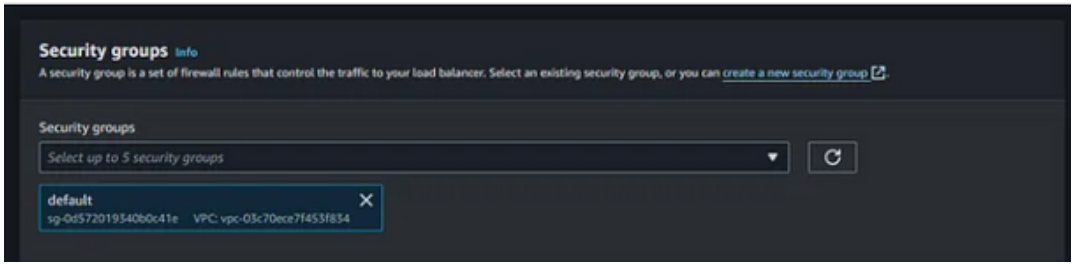

When setting up the load balancer, it’s necessary to specify the availability zones where the load balancer will operate. The system requires a minimum of two zones to ensure redundancy and high availability.

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.

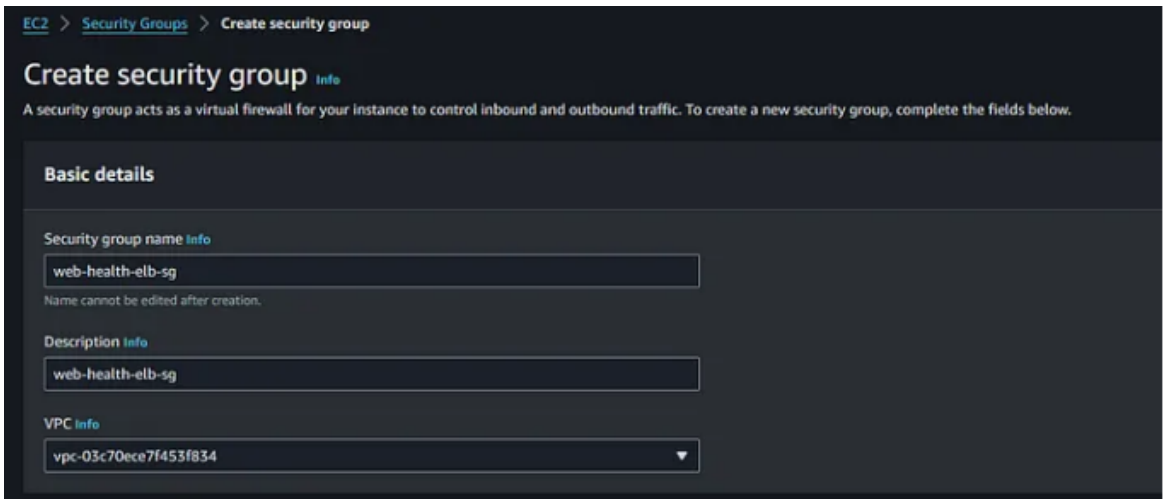



Click on the “Create a new security group”



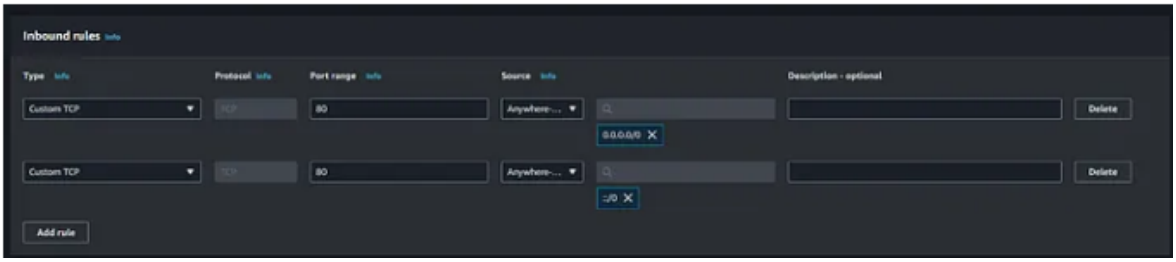

For the security configuration of our load balancer, we’ll create a new security group to define its access permissions. It’s important to name this security group appropriately for easy identification and management.

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.

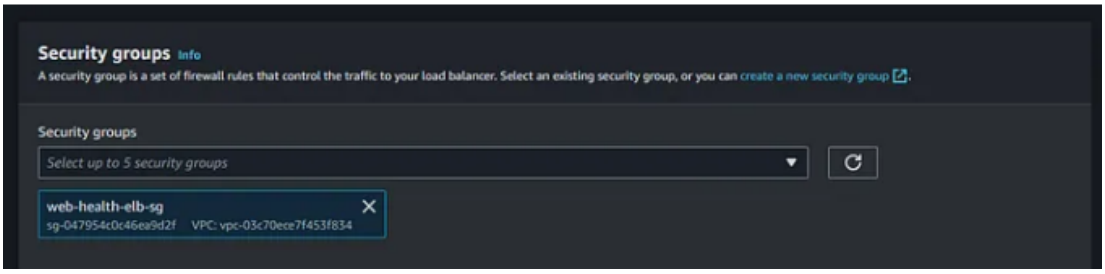


As for the inbound rule, we want to allow access on port 80 from anywhere, making the load balancer accessible to the public. We'll specify the protocol as IPv6 to accommodate modern internet standards, especially considering the increasing adoption of IPv6 by internet service providers, including mobile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

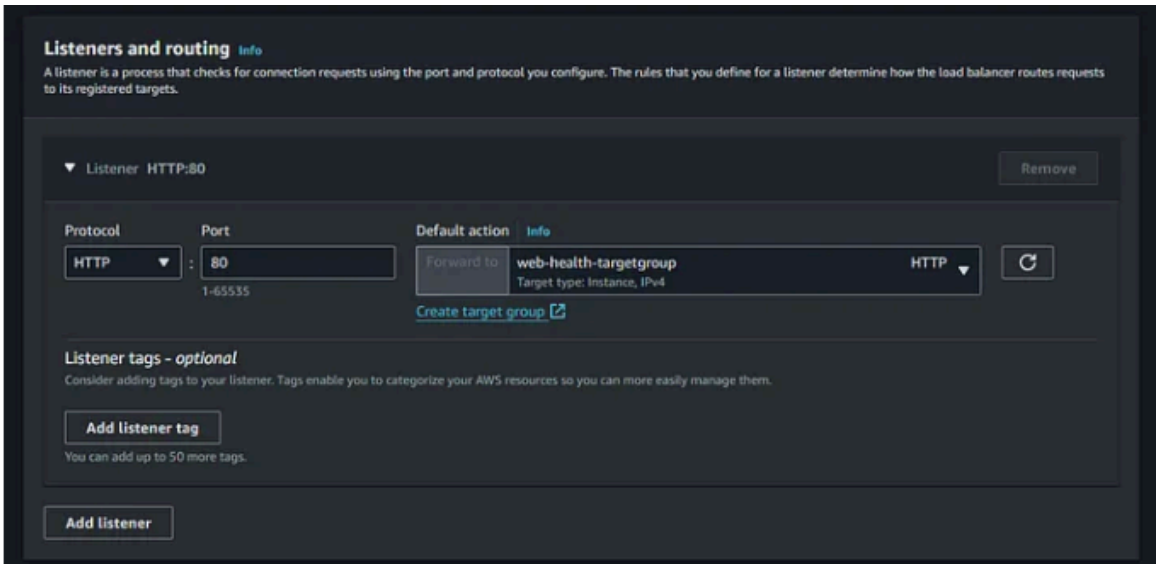
Ensuring broad accessibility while maintaining security is crucial for our load balancer's functionality and usability. Making these considerations and adopting best practices, such as proper naming conventions and access control, contributes to the overall robustness of our infrastructure.



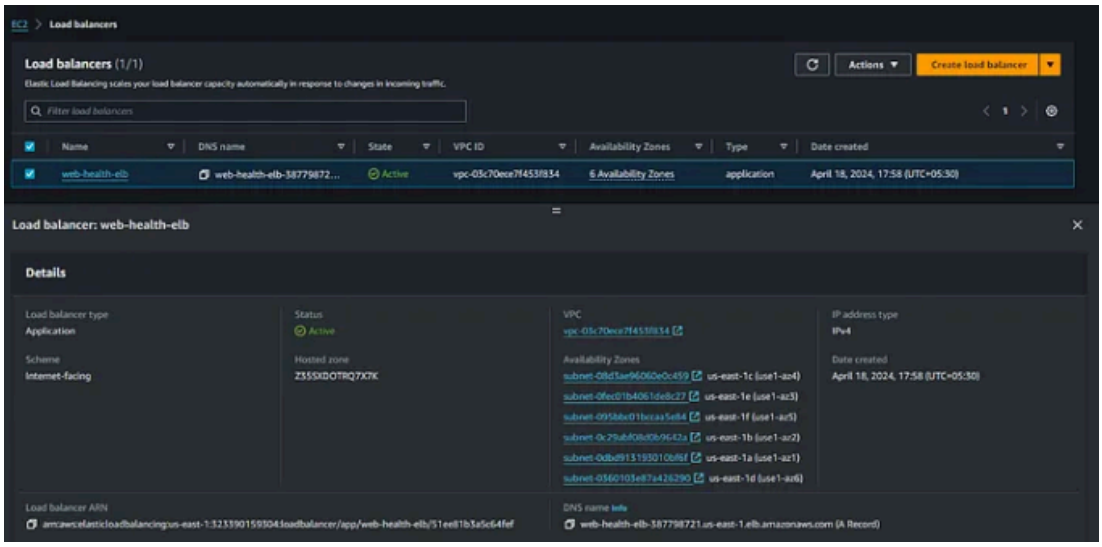

Now select the security group we created just now:




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 network-related 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.

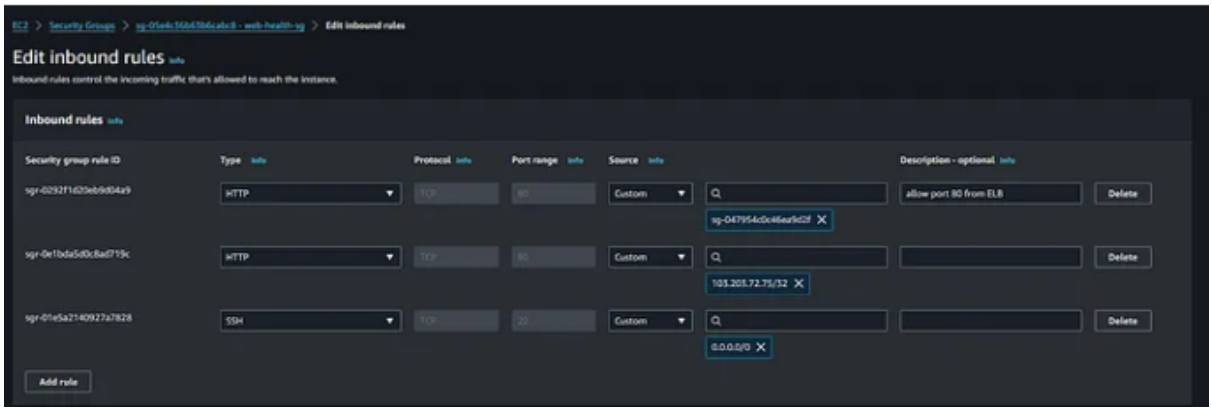
Each instance is shielded by a security group, a vital layer of defense governing inbound and outbound traffic. Upon inspection, we observed that both instances share the same security group. However, upon reviewing its inbound rules, we discovered that port 80 was only permitted from our IP address, 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.

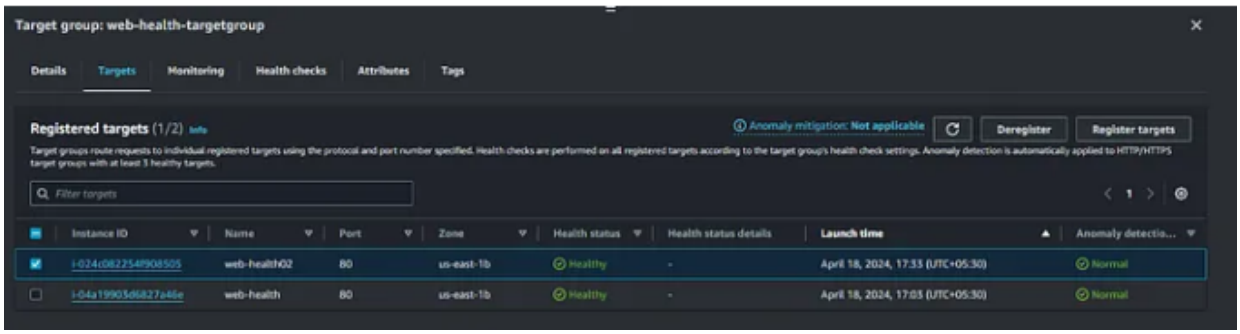
Returning to the security group settings, we locate the security group associated with the instances and modify its inbound rules. Specifically, we add a new rule permitting port 80 access from the security group of the load balancer. By specifying the source as the load balancer's security group, we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

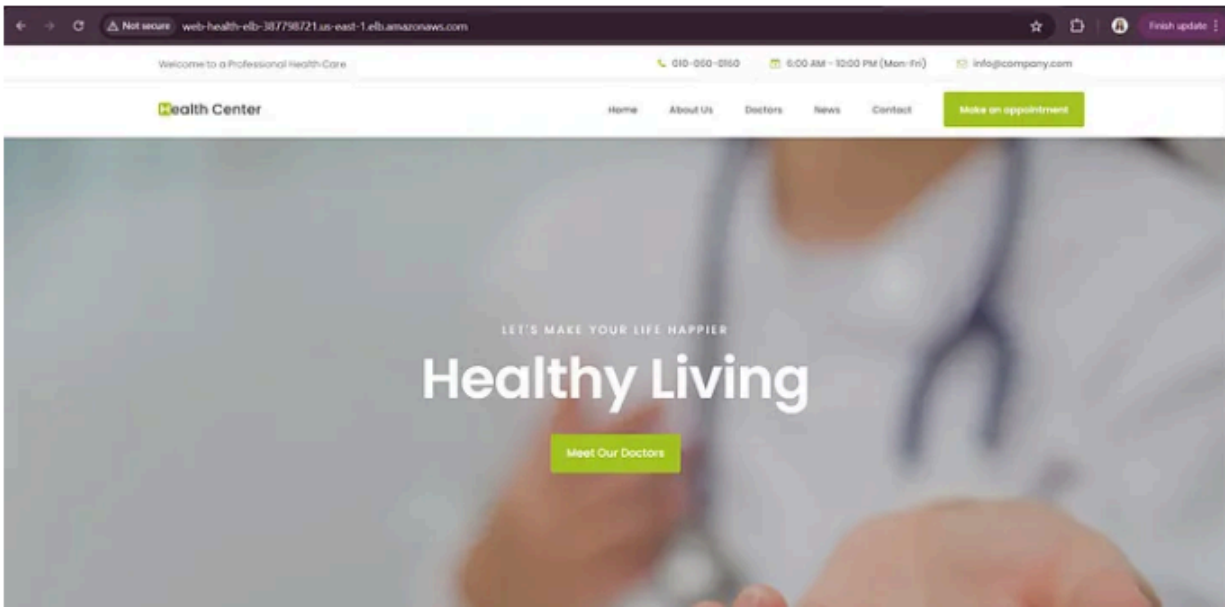
Make the security groups inbound rules as below of "web-health".




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.

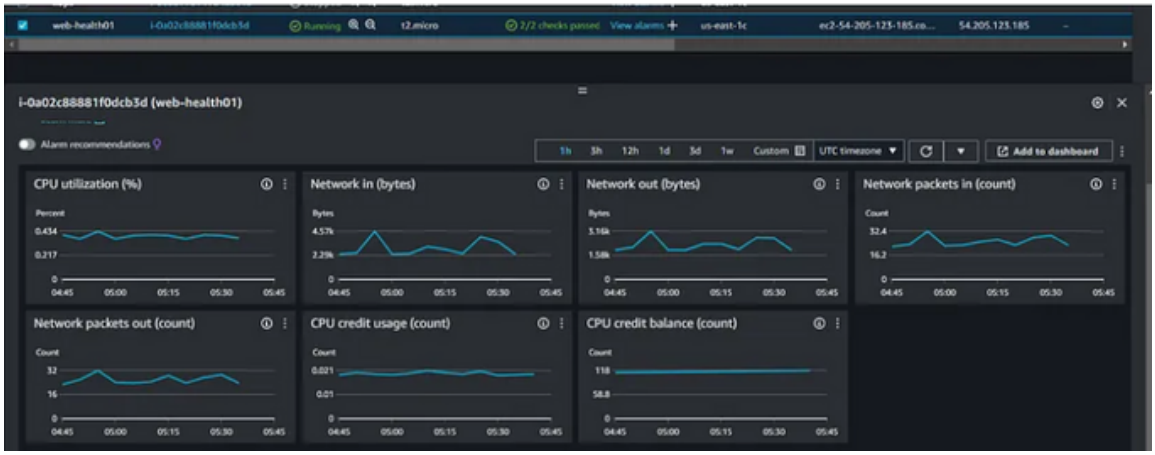
If you've already reviewed the previous blog post, we've provided a template for launching EC2 instances. Kindly review it, or proceed to create an EC2 instance.
Select the EC2 instance and click on monitoring:

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.

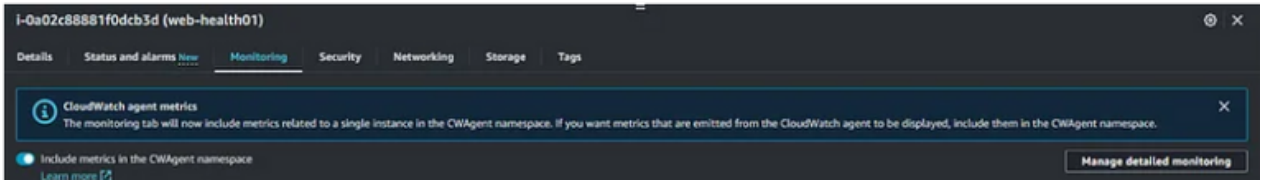
These metrics are automatically generated by CloudWatch when you launch an EC2 instance.

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.

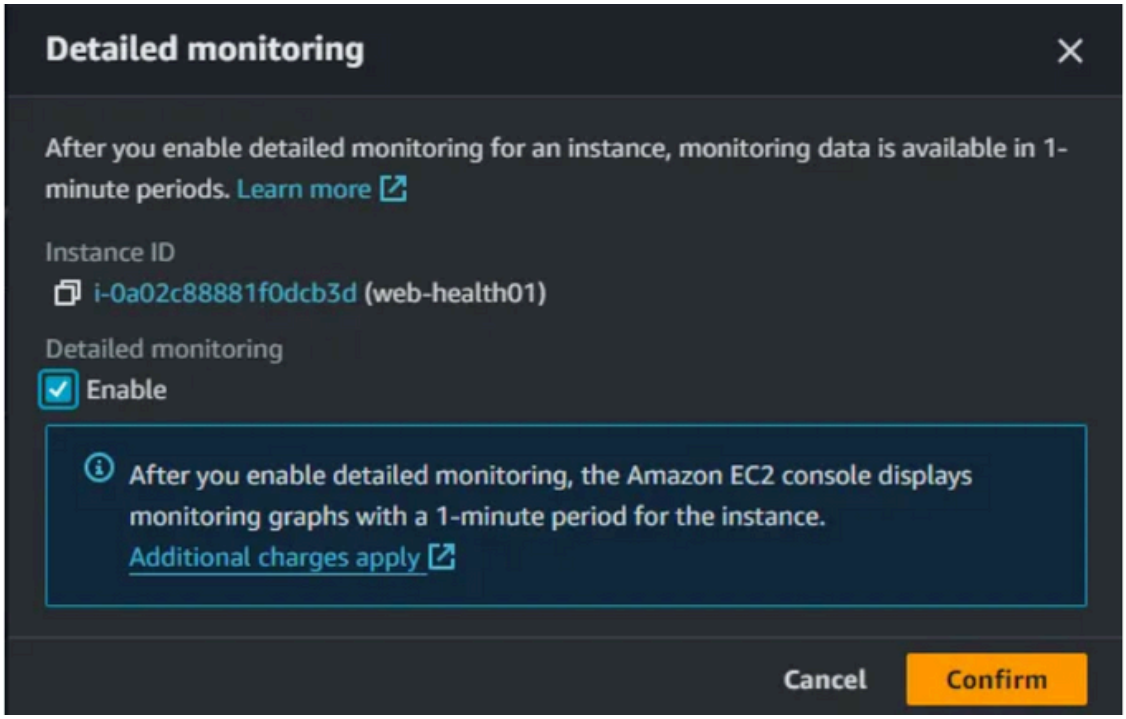
By default, CloudWatch checks and updates these metrics every 5 minutes, populating the corresponding graphs.If you prefer more frequent updates, you can enable detailed monitoring, though it's important to note that this option isn't free.




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.

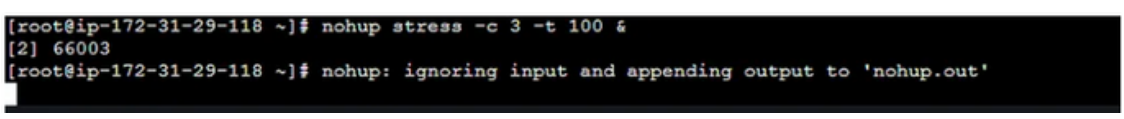



Log in to the terminal of the EC2 instance and switch as root user.

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*



*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.

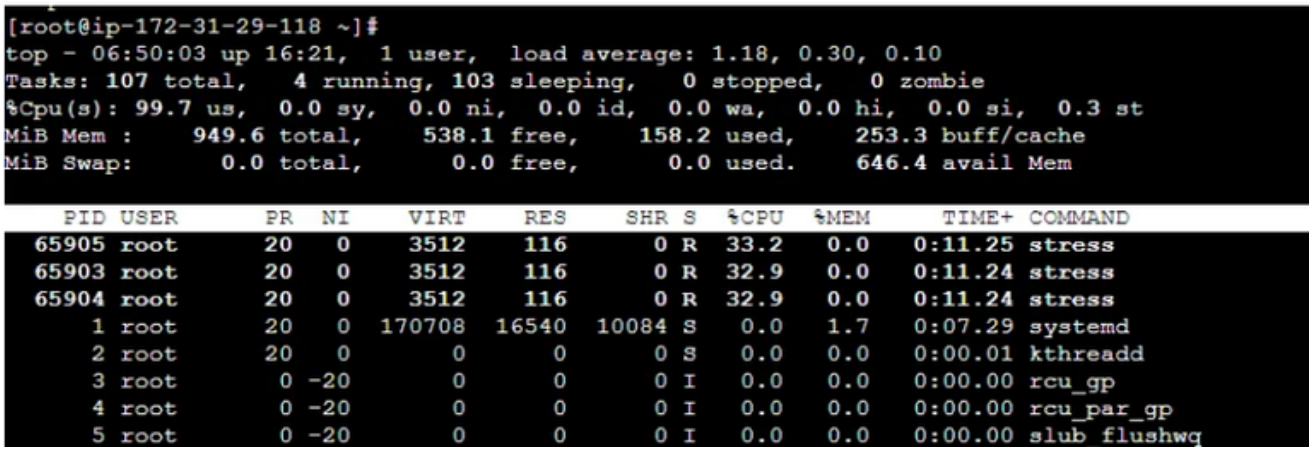
So, this command will run the stress test with 3 worker threads for 100 seconds, and the process will continue running even if the terminal session is closed.
Execute the top command.

*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*

```
[root@ip-172-31-29-118 ~]# nohup ./stress.sh &
[1] 66549
[root@ip-172-31-29-118 ~]# nohup: ignoring input and appending output to 'nohup.out'
```

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

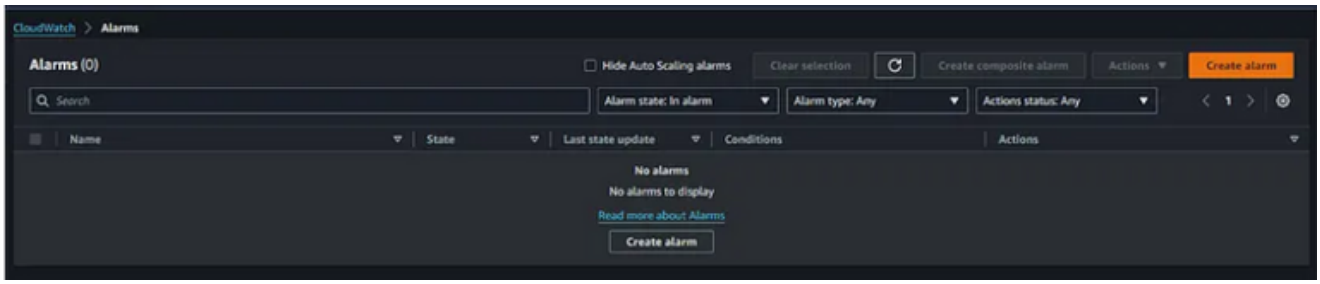
Ensure your script is executable.

*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.

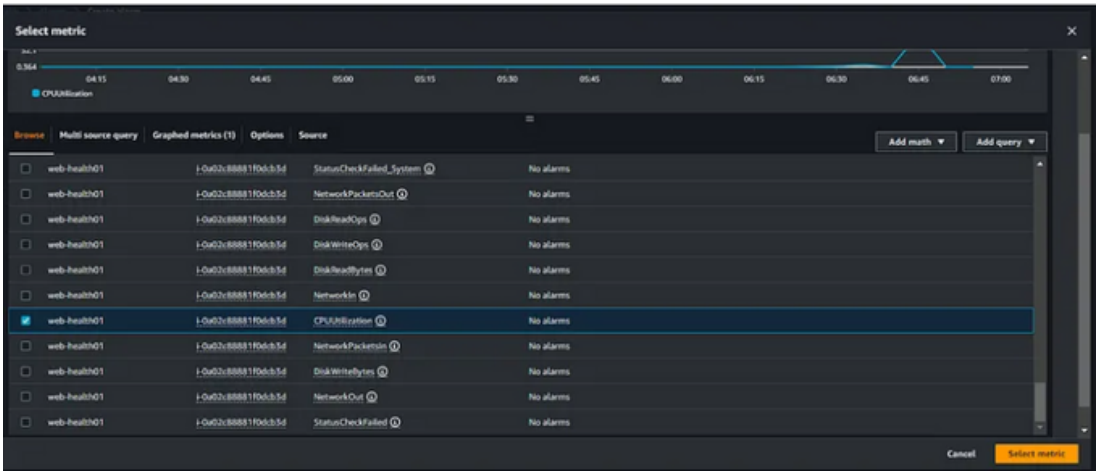



To select the CPU utilization metric, start by navigating to the EC2 service.

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.

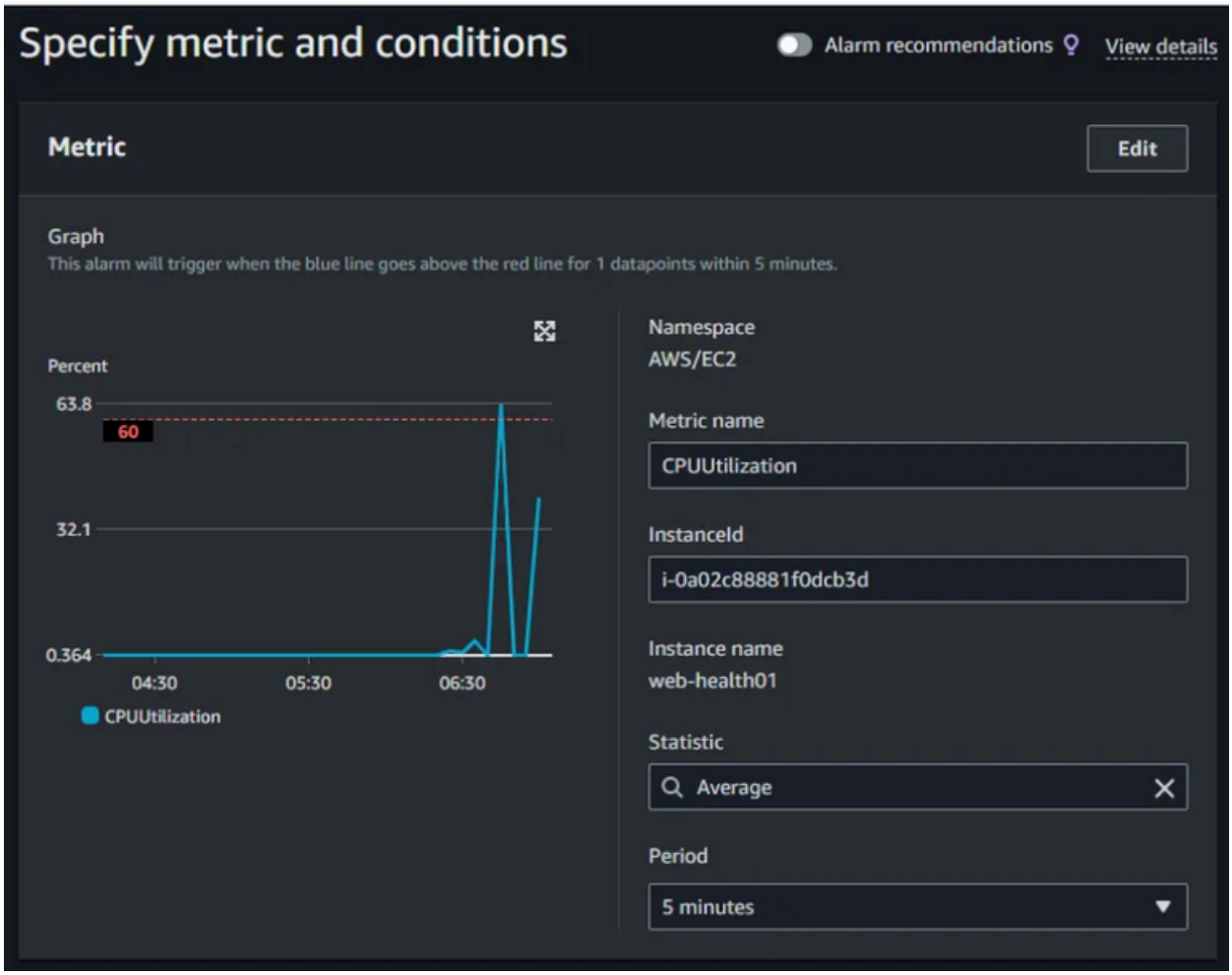
Once you've found your instance, look for the CPU utilization metric.

Select the CPU utilization metric by clicking on it.



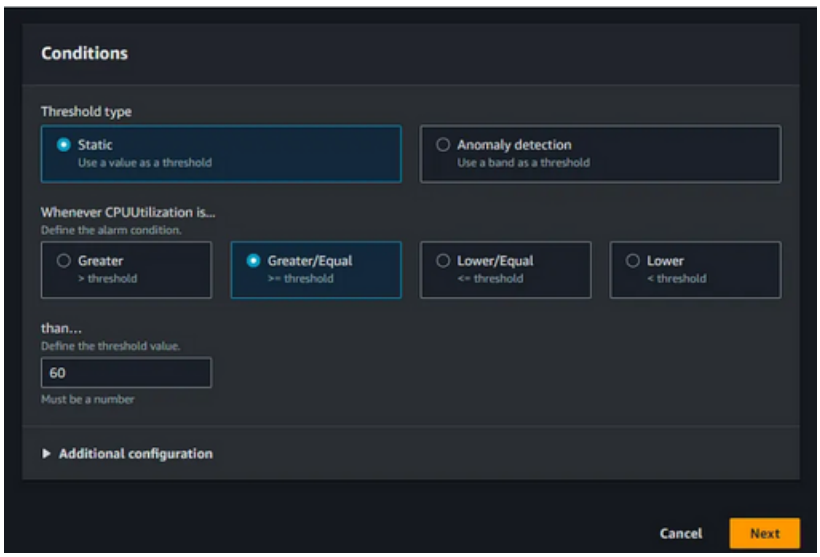
In this step, we can choose the period for which the alarm will be evaluated. For this demonstration, I'll keep the period set to 5 minutes.

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.

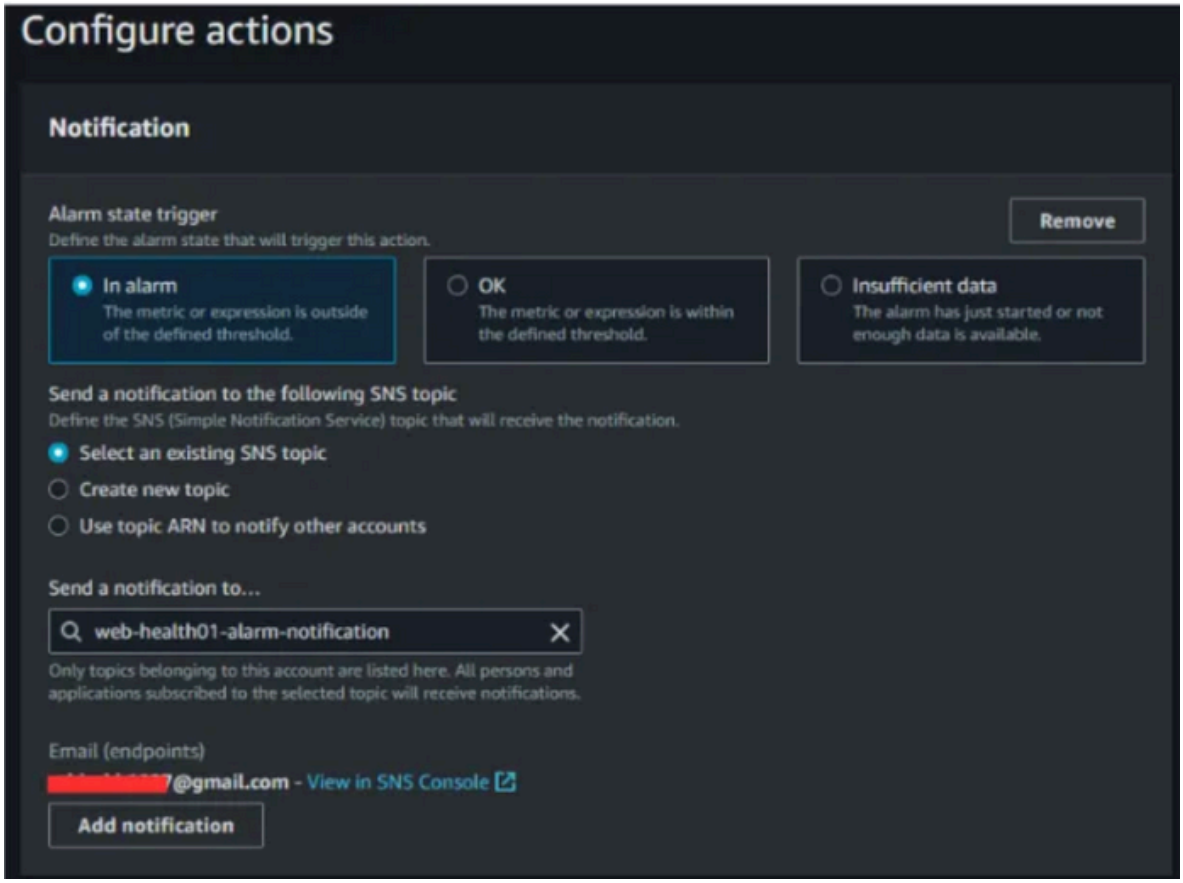
Once we've defined the condition, we proceed to the next step.



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.

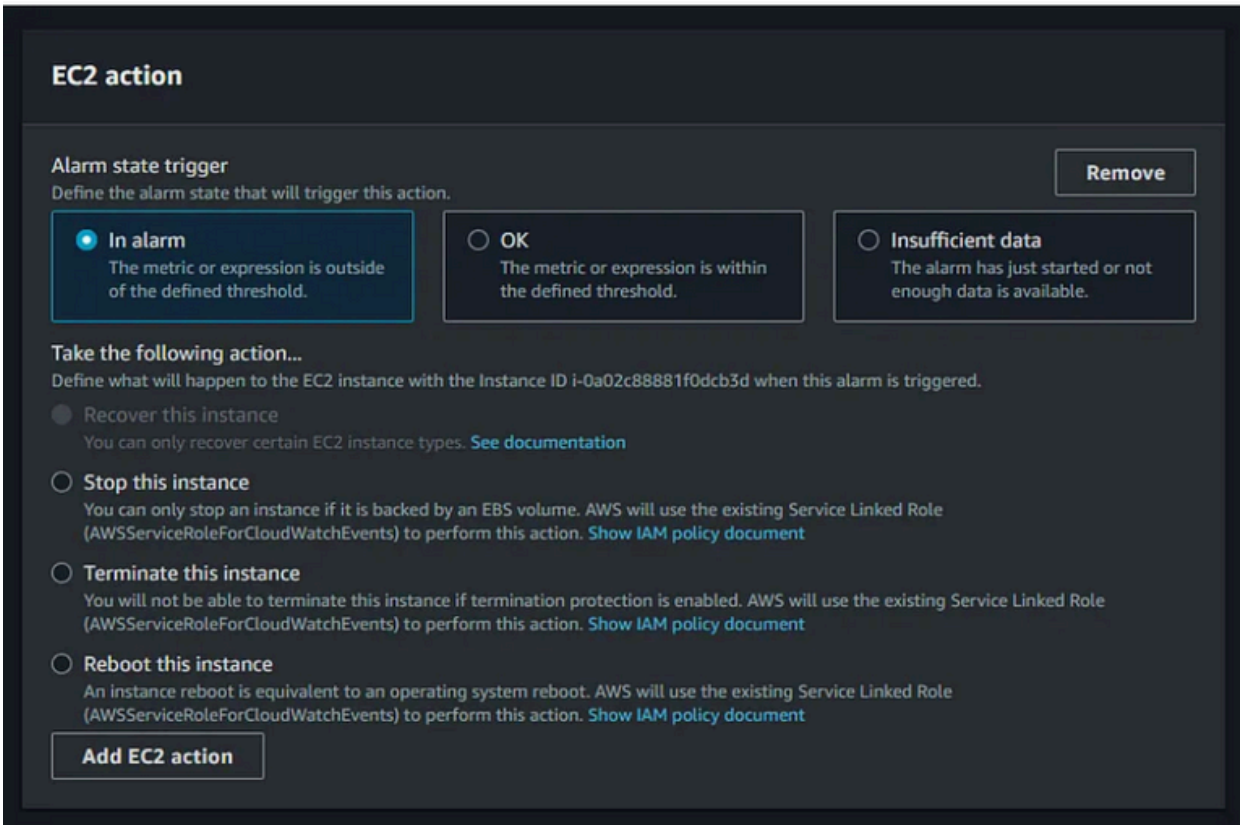
This topic is designated for notifications. So, when the alarm state is triggered, it will send a notification to this topic, ultimately resulting in an email notification being sent to me.




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.

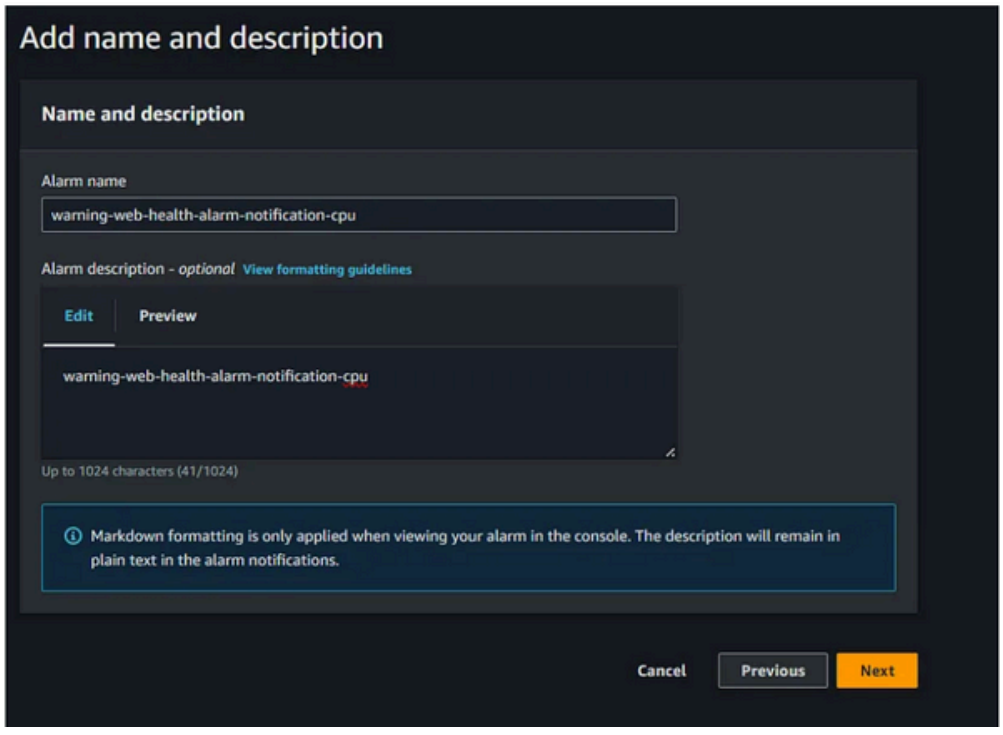
However, for the purpose of this demonstration, we will skip these additional actions. We will stick to email notifications only and proceed to the next step.




Let’s give the alarm a descriptive name.

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

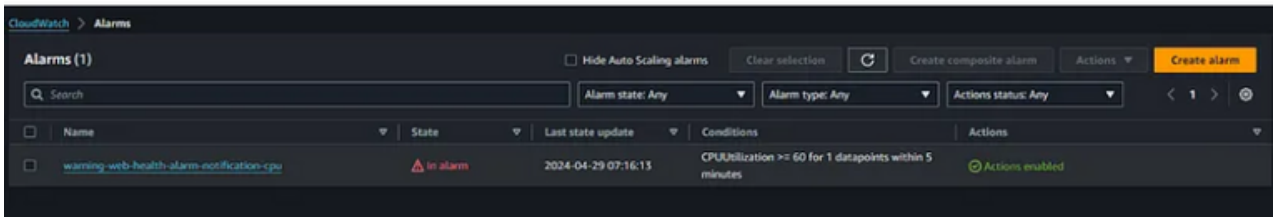
In our organization, a warning is triggered when the utilization exceeds 60%. We could set up additional alarms for critical situations, such as when it surpasses 80%. However, for now, we’ll proceed with just one alarm.
Now, let’s move on to the next step.



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.

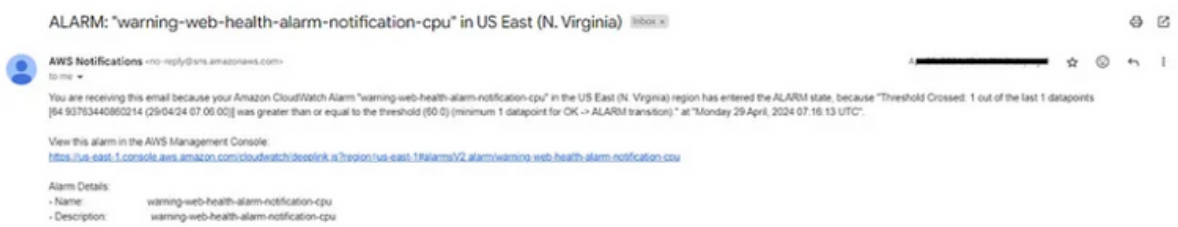
Whether it's Prometheus, Nagios, Icinga, Zenos, or others, the concept remains similar. However, with CloudWatch, monitoring is already configured, and you only need to set up alarms. In contrast, with other tools, such as those you set up yourself, the entire monitoring system needs configuration, typically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

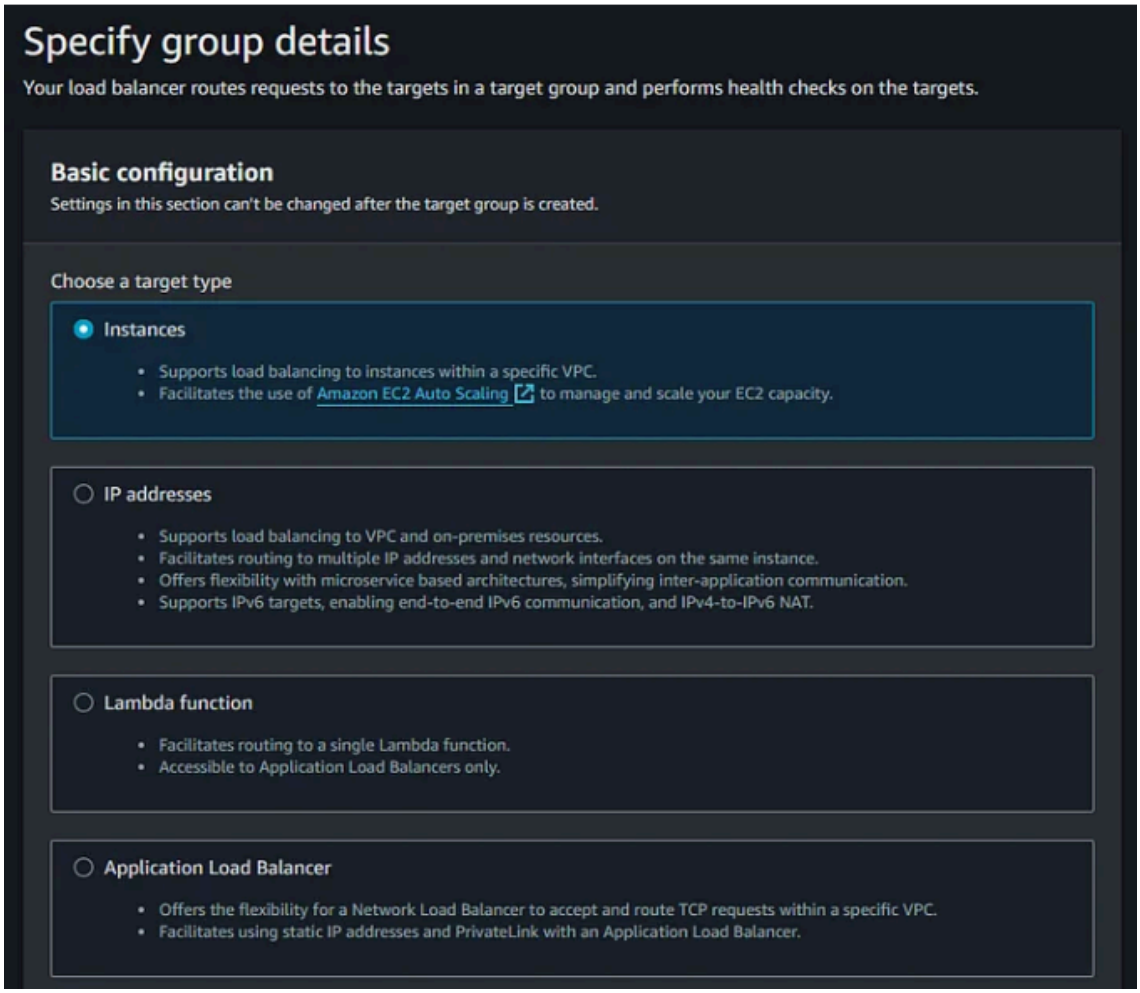
In practice, ASG operates by creating a group powered by launch configurations. When a metric breach triggers a CloudWatch alarm, scaling policies kick in to launch or terminate instances accordingly, ensuring optimal performance and resource utilization.

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.

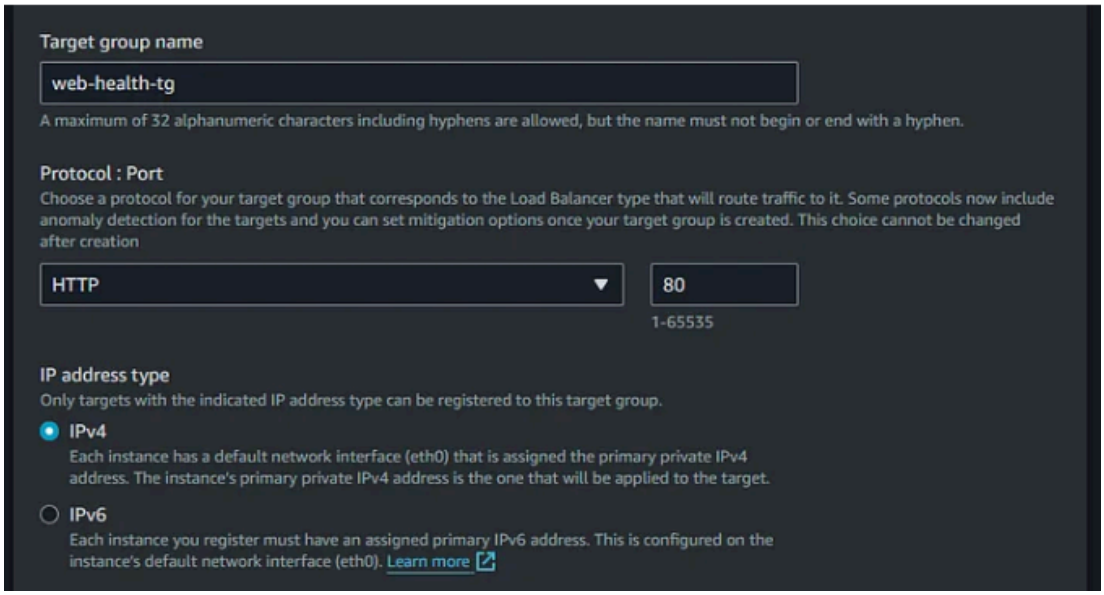
Firstly, we’ll need the launch template, which should already be created as per the instructions provided in the load balancer section. If the launch template has been removed or needs to be recreated, refer back to the load balancer section for detailed steps on how to create it.

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

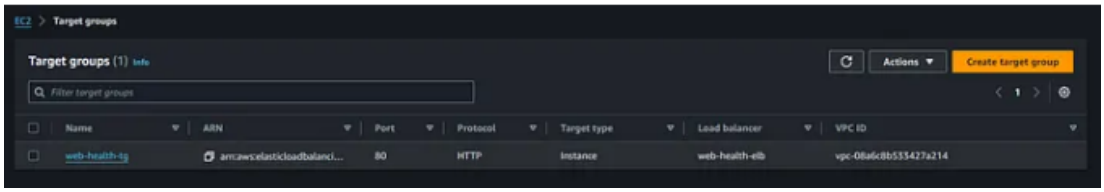



Configure the target group settings, such as name. For example, let’s name it “web-health-tg”.

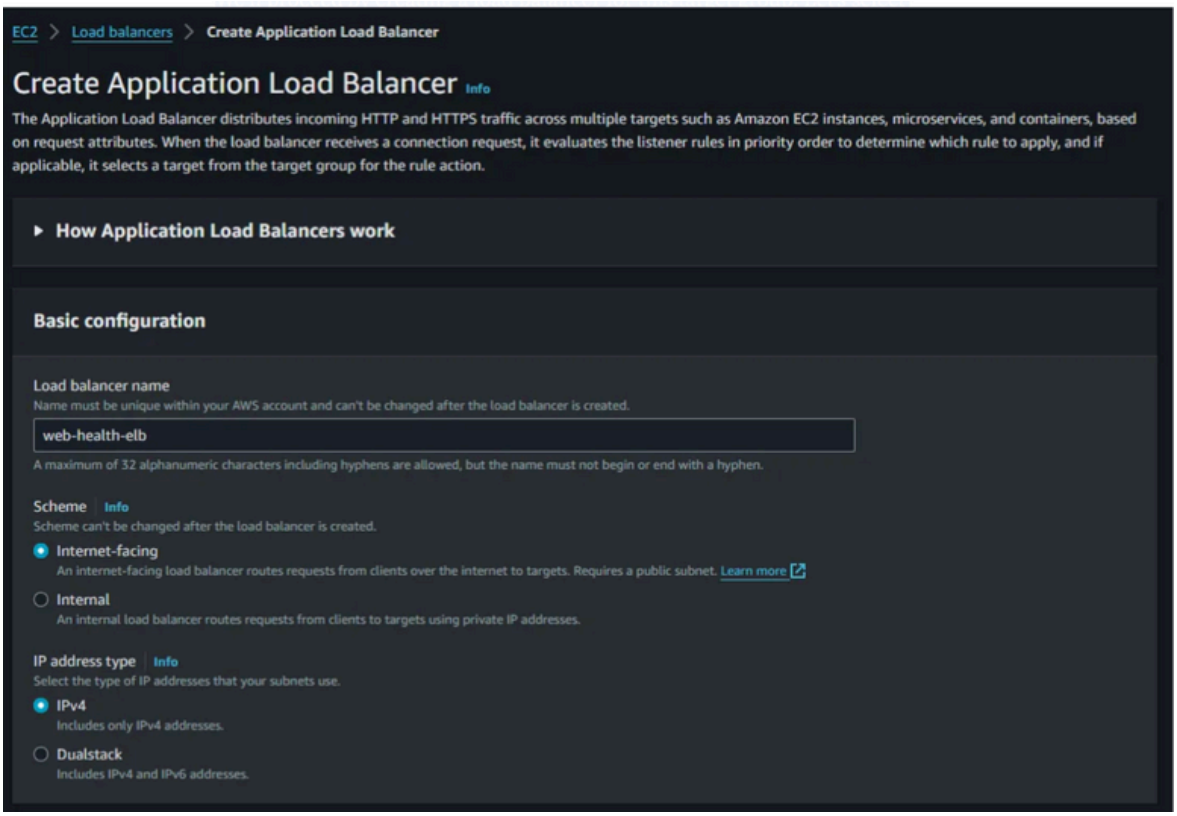
■Keep the other configurations as default and proceed by clicking “Next”.
■Since we don’t have any instances yet, select the option to create an empty target group.
■Complete the target group creation process by following the prompts.



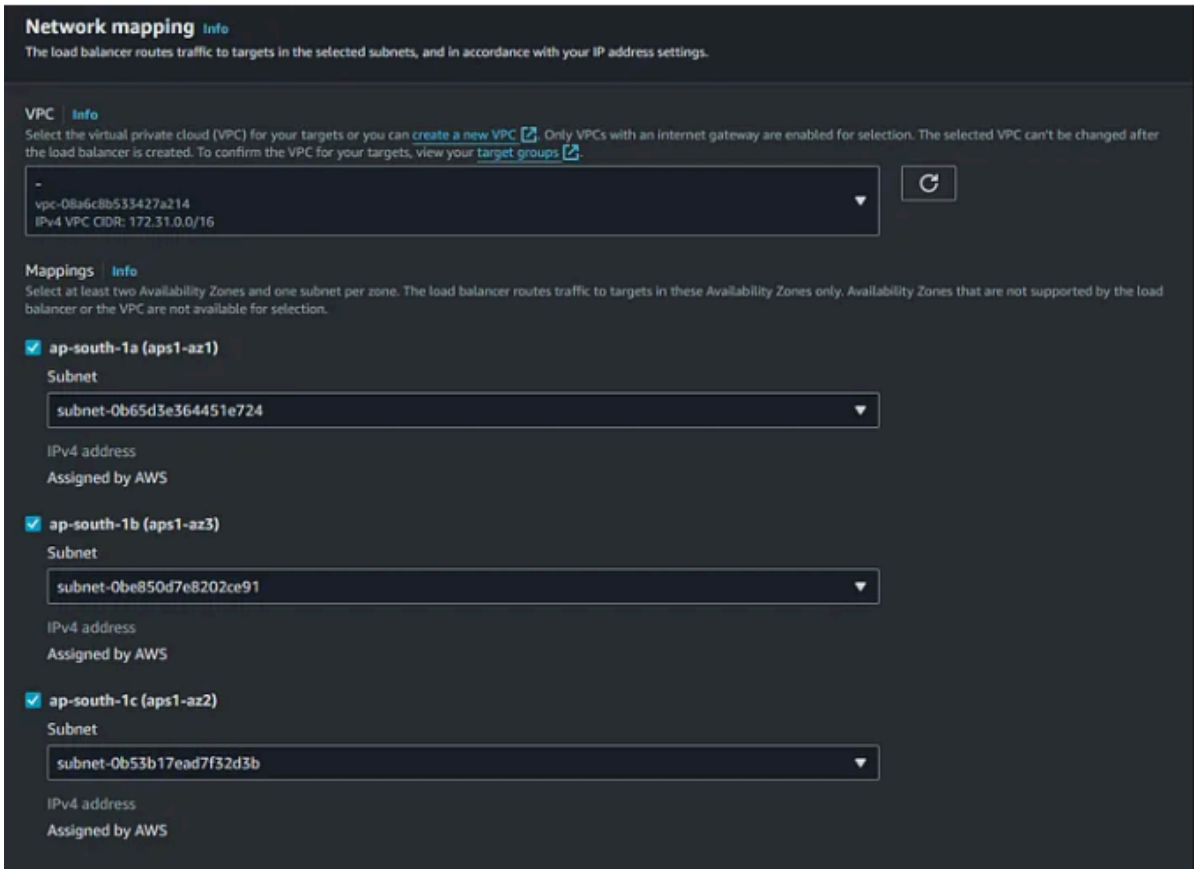

Indeed, the Auto Scaling group will automatically add instances to this target group as needed based on your scaling policies and the metrics you've configured, ensuring seamless scalability and efficient resource management.



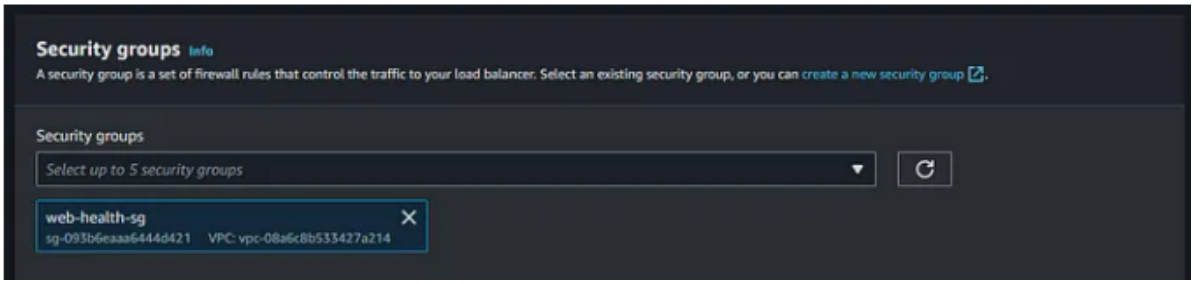

We'll proceed by creating a load balancer for this target group. Let's select "Create load balancer" and opt for the application load balancer, maintaining consistency with our previous setup.

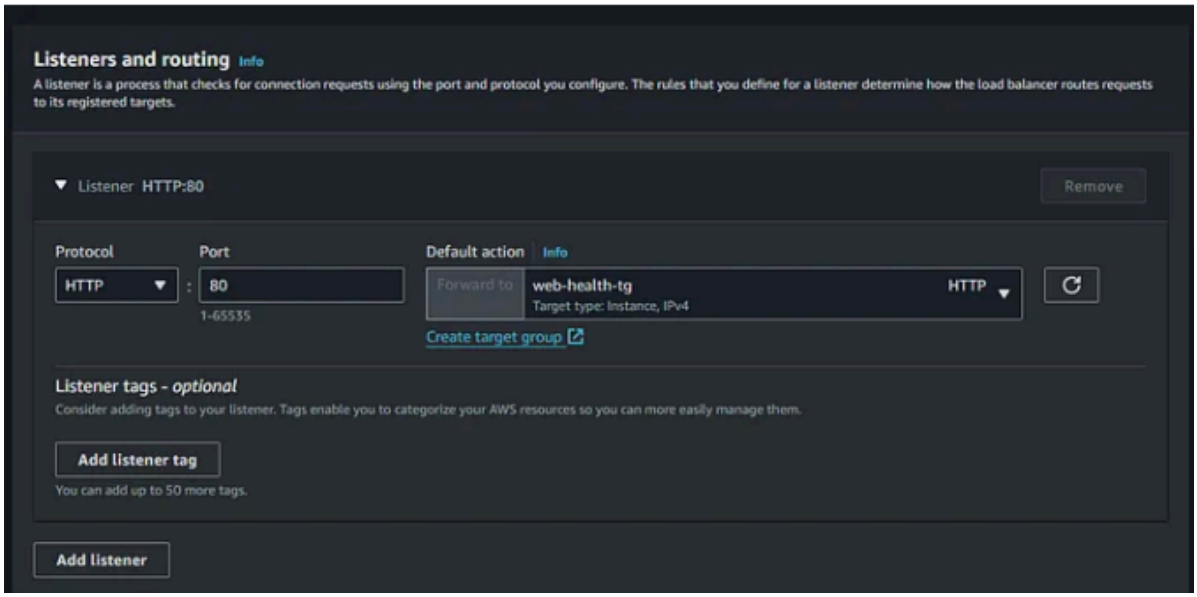


We'll name it "health-elb" and choose the same availability zones and security groups used previously.

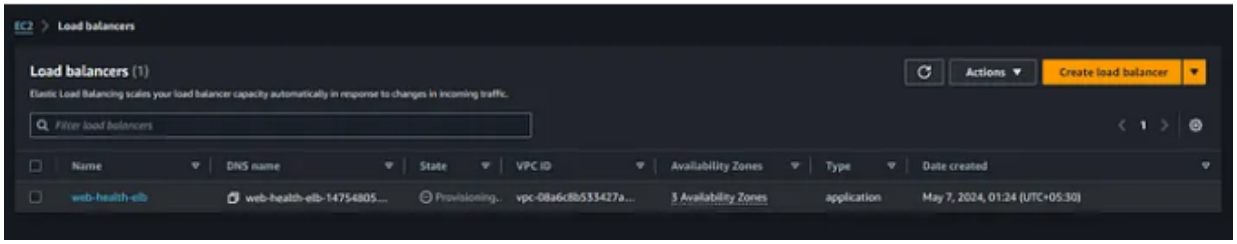



Remember to remove the Default Security Group. Next, we'll configure it to forward traffic to the target group we just created.



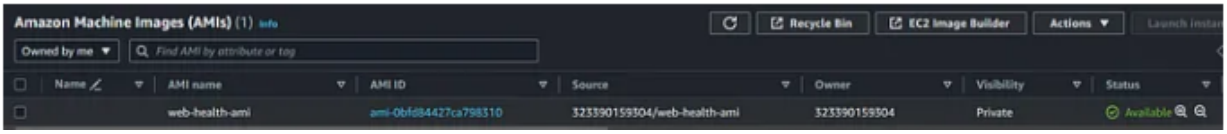



If you already have these resources set up, you can reuse them; otherwise, go ahead and create them now. Here's our load balancer.

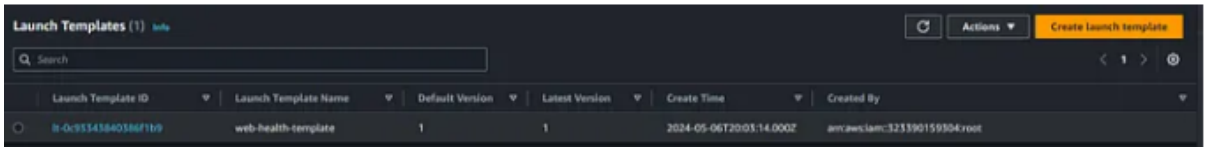



Let's revisit the launch template. This template is essential as it defines the configuration for the instances that will be launched by the auto scaling group.

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.

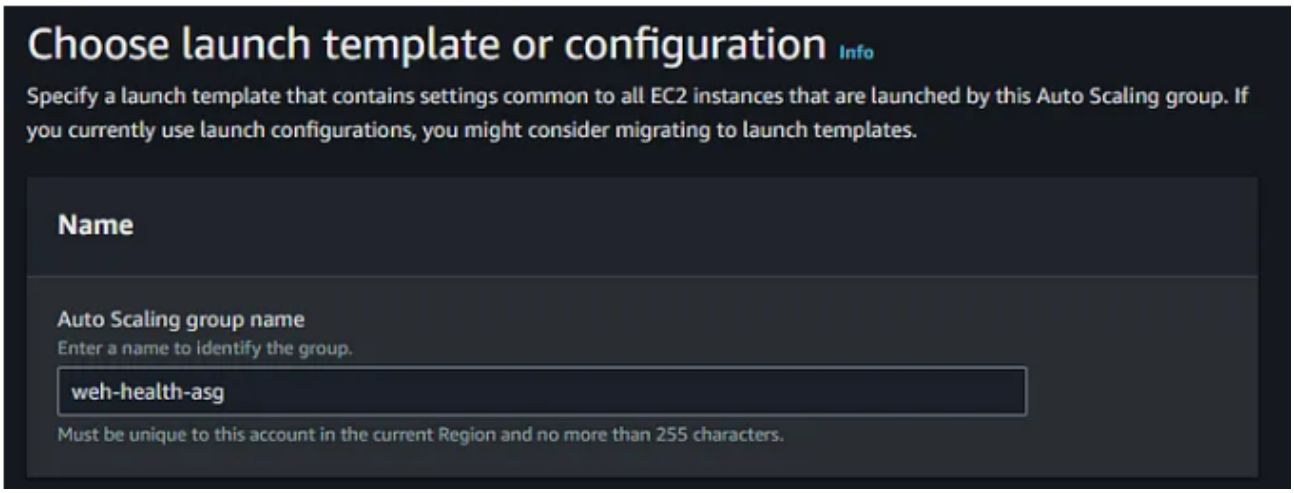



The launch template specifies details such as the Amazon Machine Image (AMI) to be used for the instances.




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".

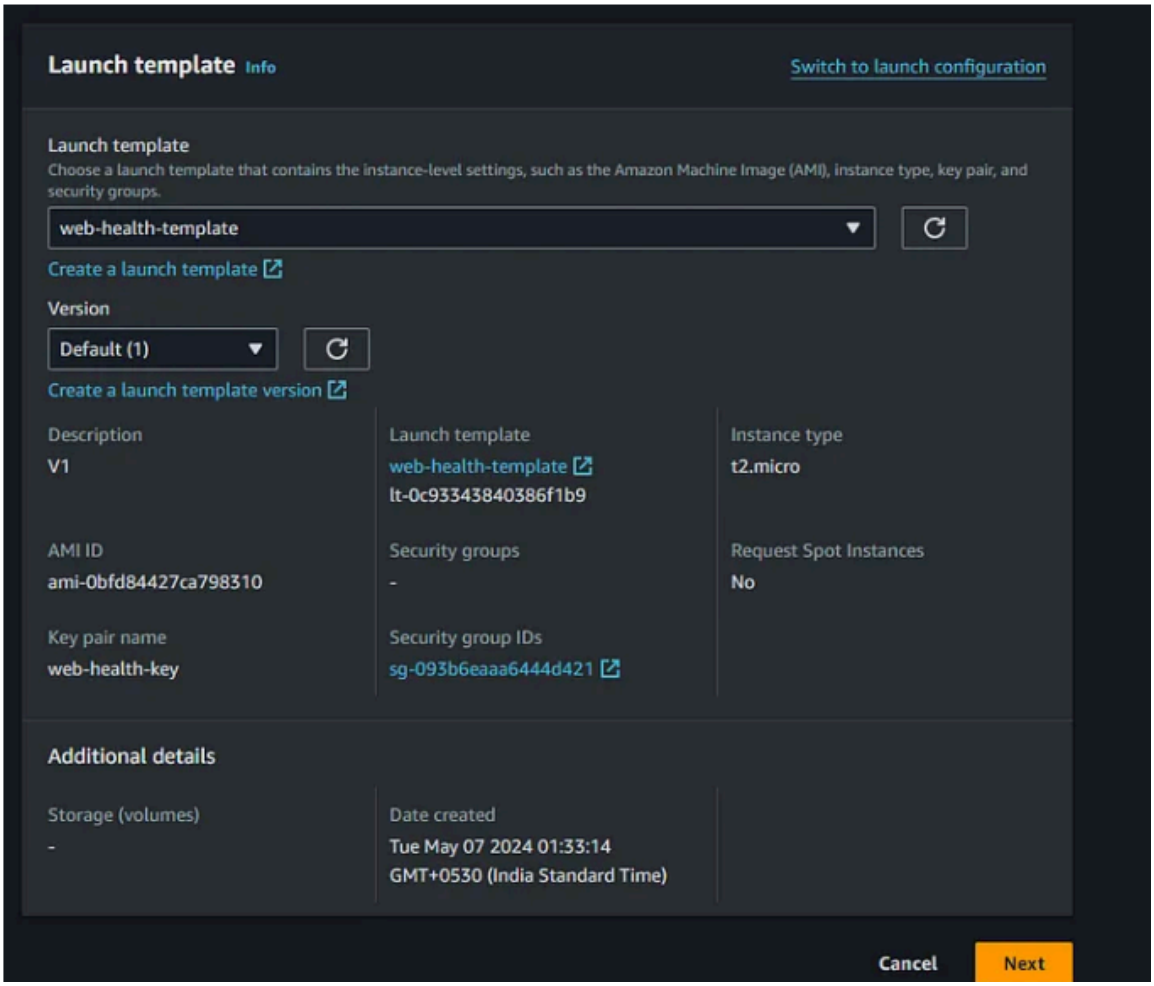


Next, we'll select our launch template for the auto scaling group.

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.

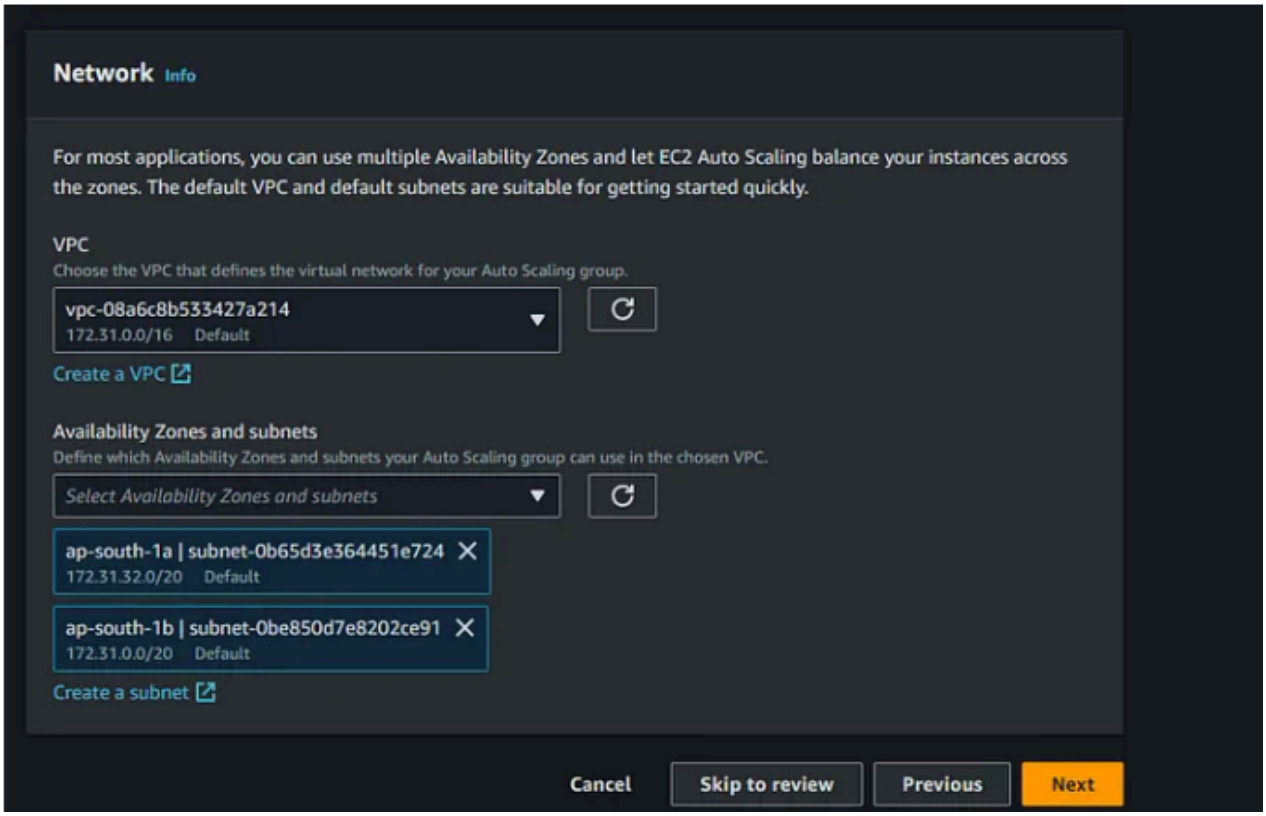
As we proceed, we'll opt for the launch template, leveraging its benefits for our auto scaling group. You'll notice the option to select different versions of the launch template, providing flexibility in configuration management.




Choose instance launch options:

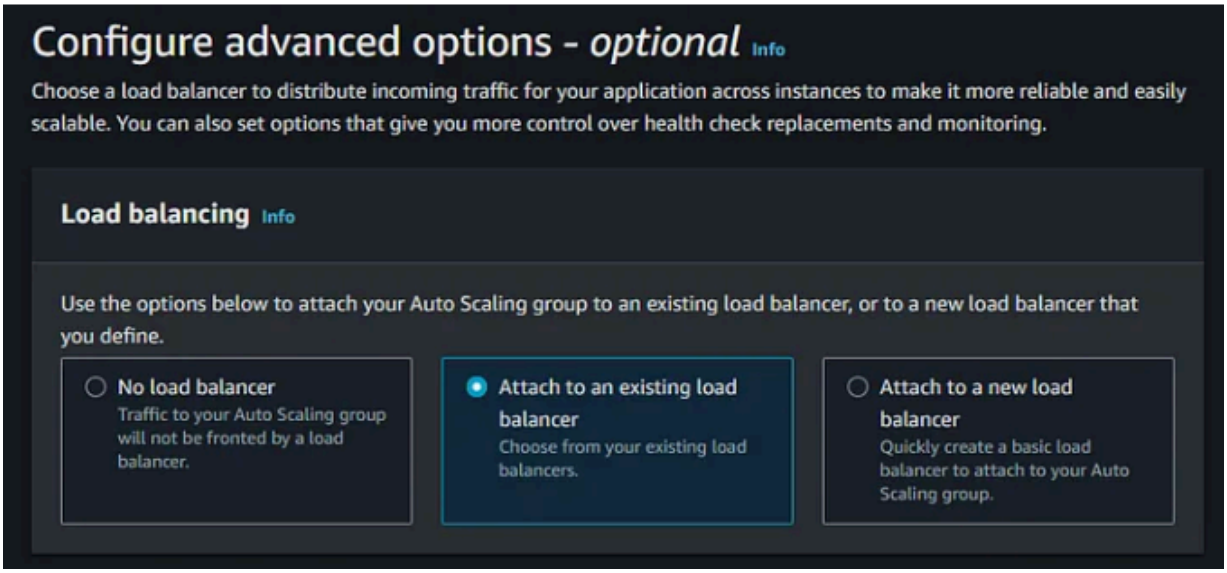
In this step, we need to specify the availability zones where we want our instances to be launched.
You have the option to select specific zones or allow AWS to choose based on availability. It's generally recommended to choose at least two zones for redundancy and fault tolerance.

Let’s proceed by selecting all zones and moving to the next step.

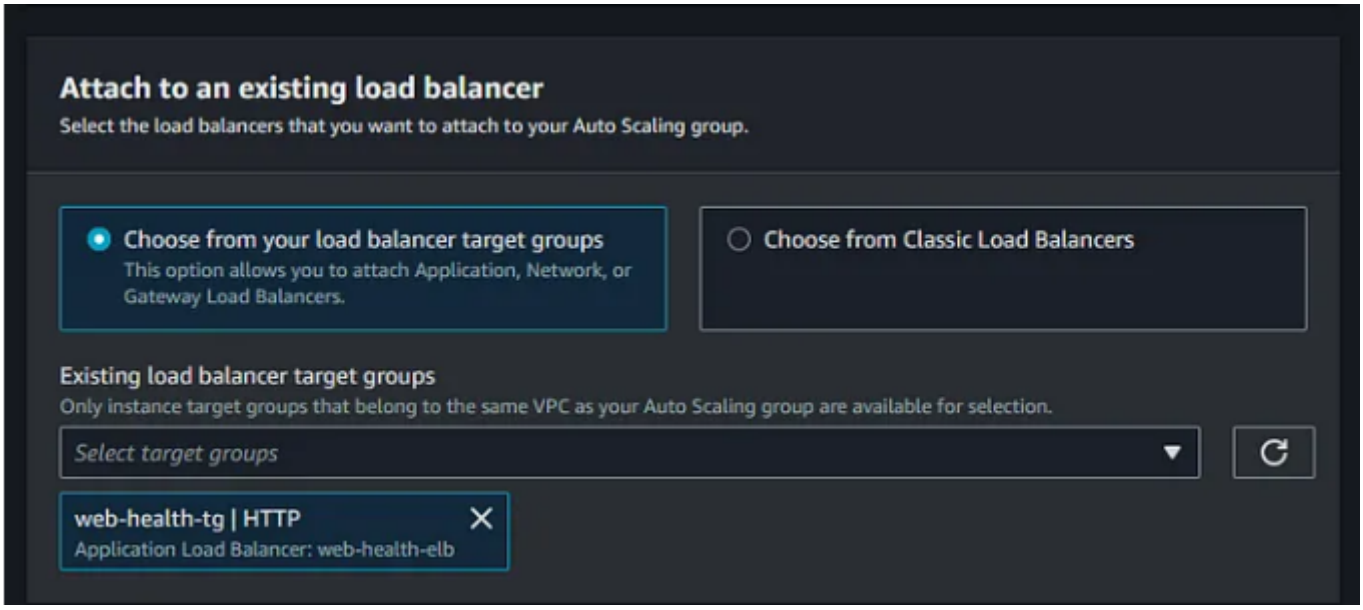



Configure advanced options

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.

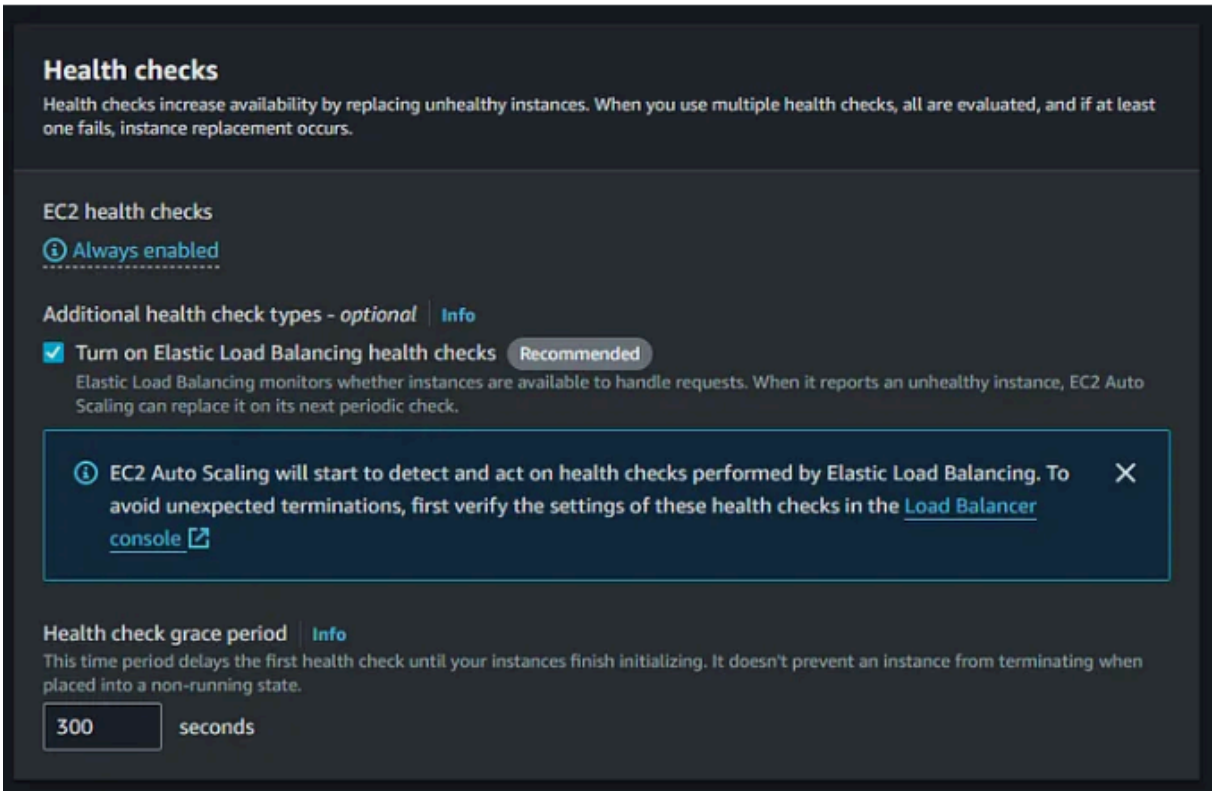
The primary responsibility of the auto scaling group is to maintain the desired number of healthy instances, ensuring optimal performance and reliability.

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.

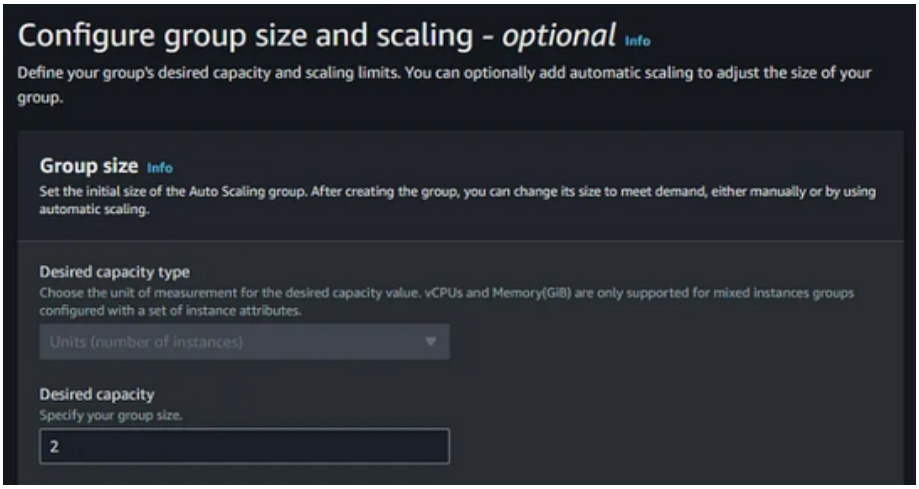
Upon detecting an unhealthy instance, the auto scaling group takes corrective action by terminating the problematic instance and launching a replacement. This ensures that the overall system remains resilient and capable of handling fluctuations in workload and application health effectively.



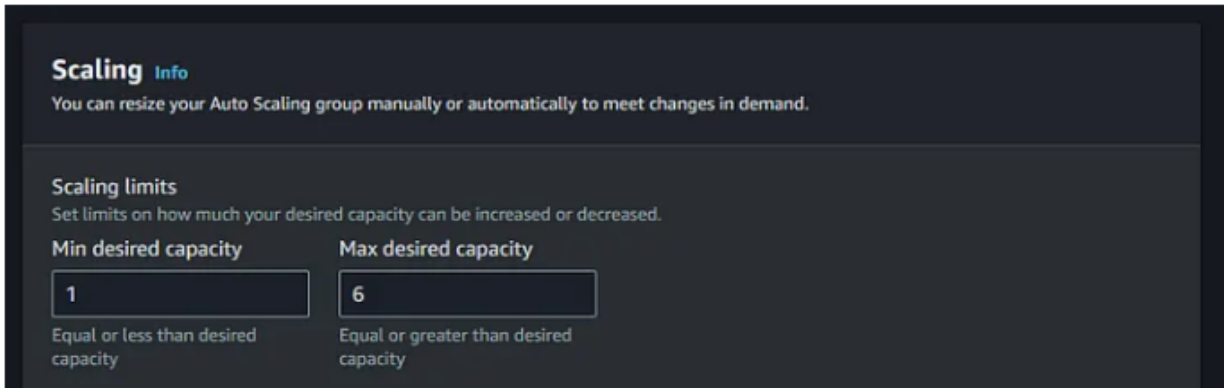

Configure group size and scaling

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.

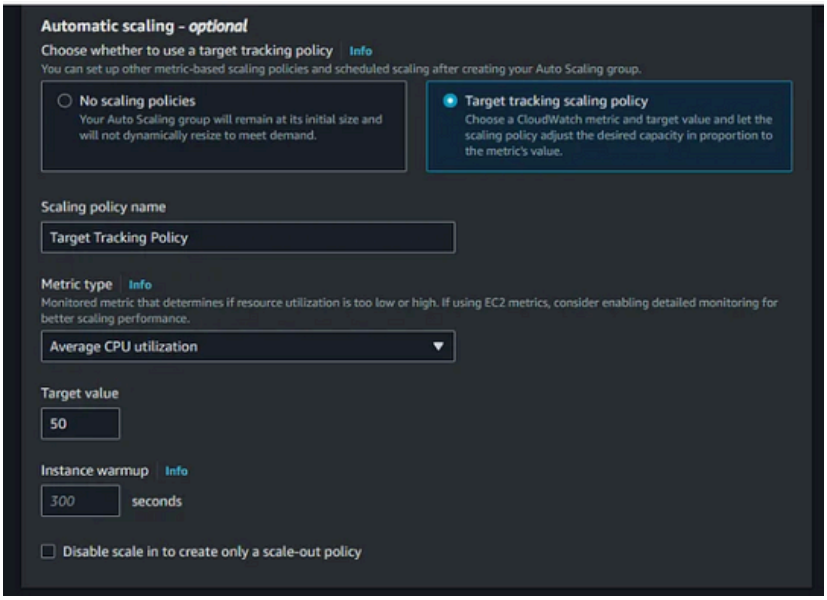



Next, we delve into scaling policies, which dictate how the auto scaling group responds to changes in workload or system metrics.

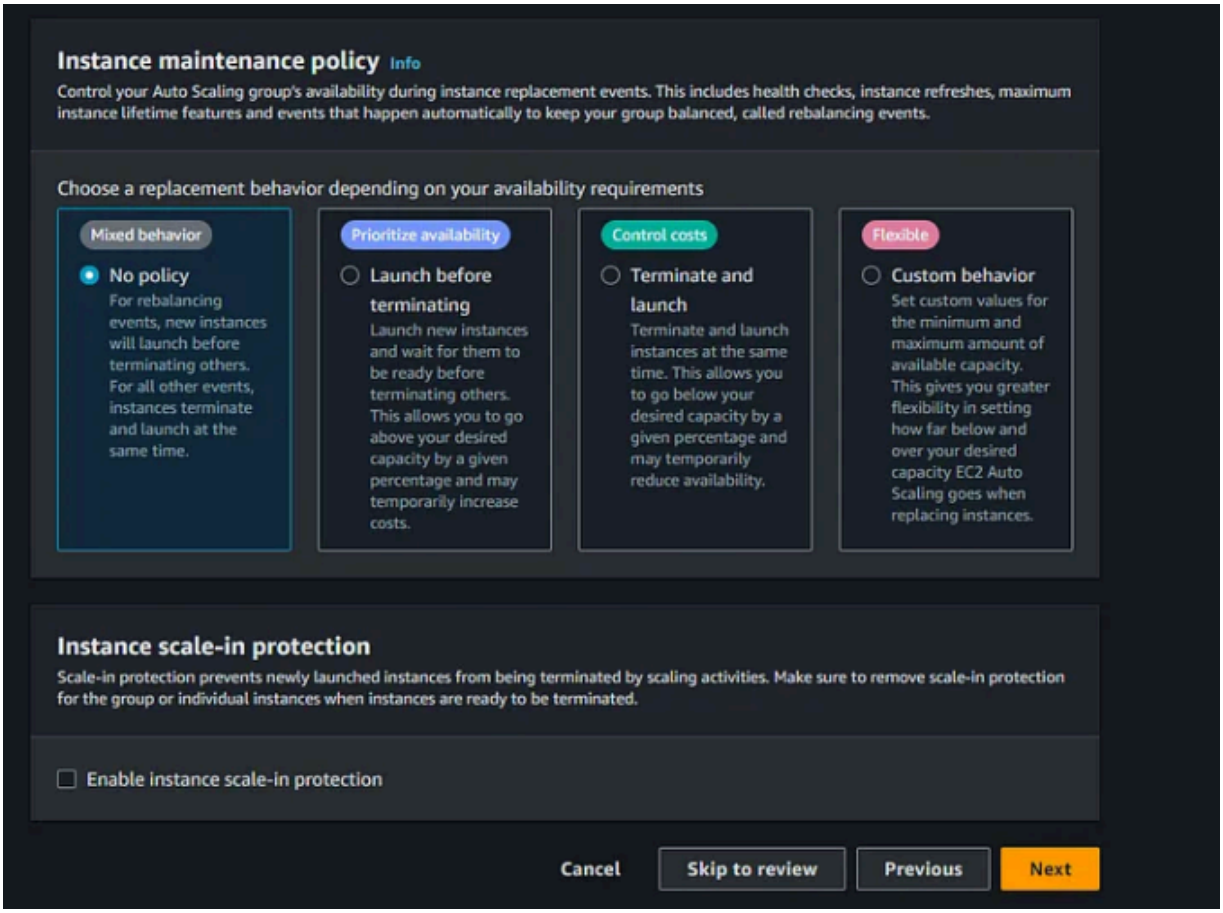
- Scaling policies: These policies govern when and how the auto scaling group should scale out (add instances) or scale in (remove instances) based on predefined conditions. We can choose from various scaling policy types, such as target tracking scaling policies or simple scaling policies.

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.



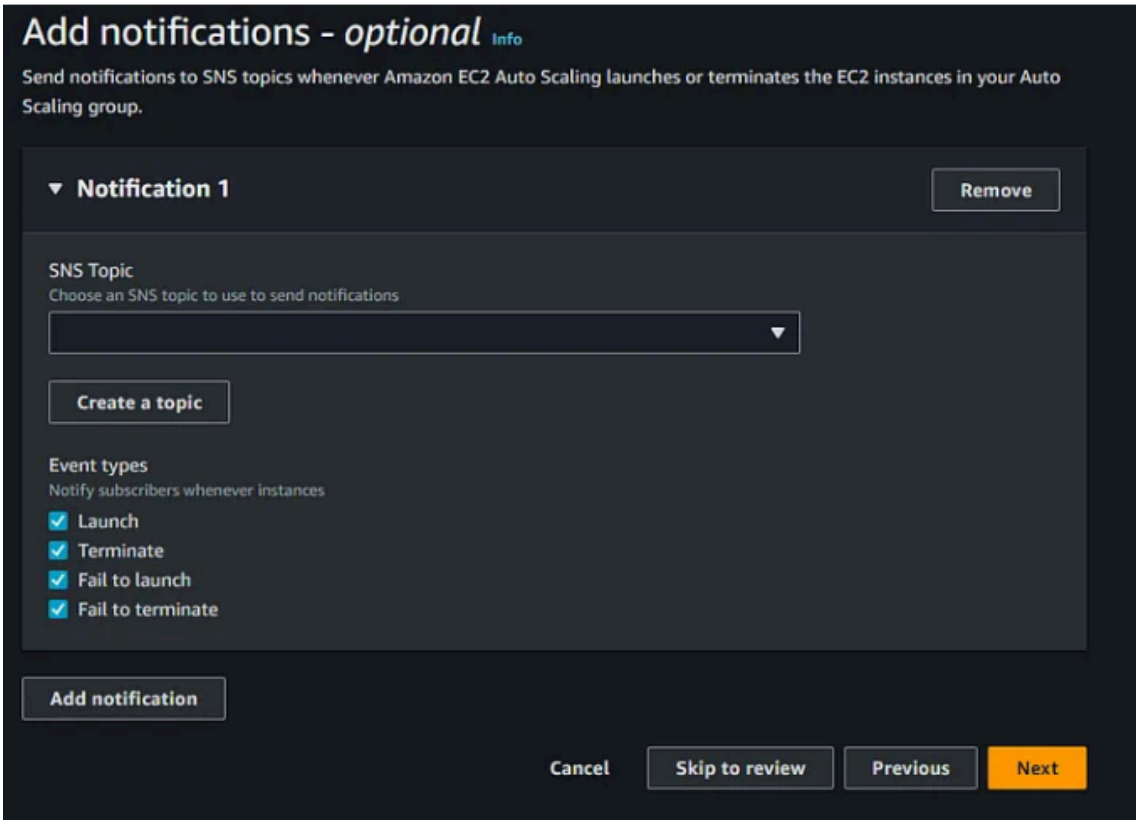

Add notifications

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.

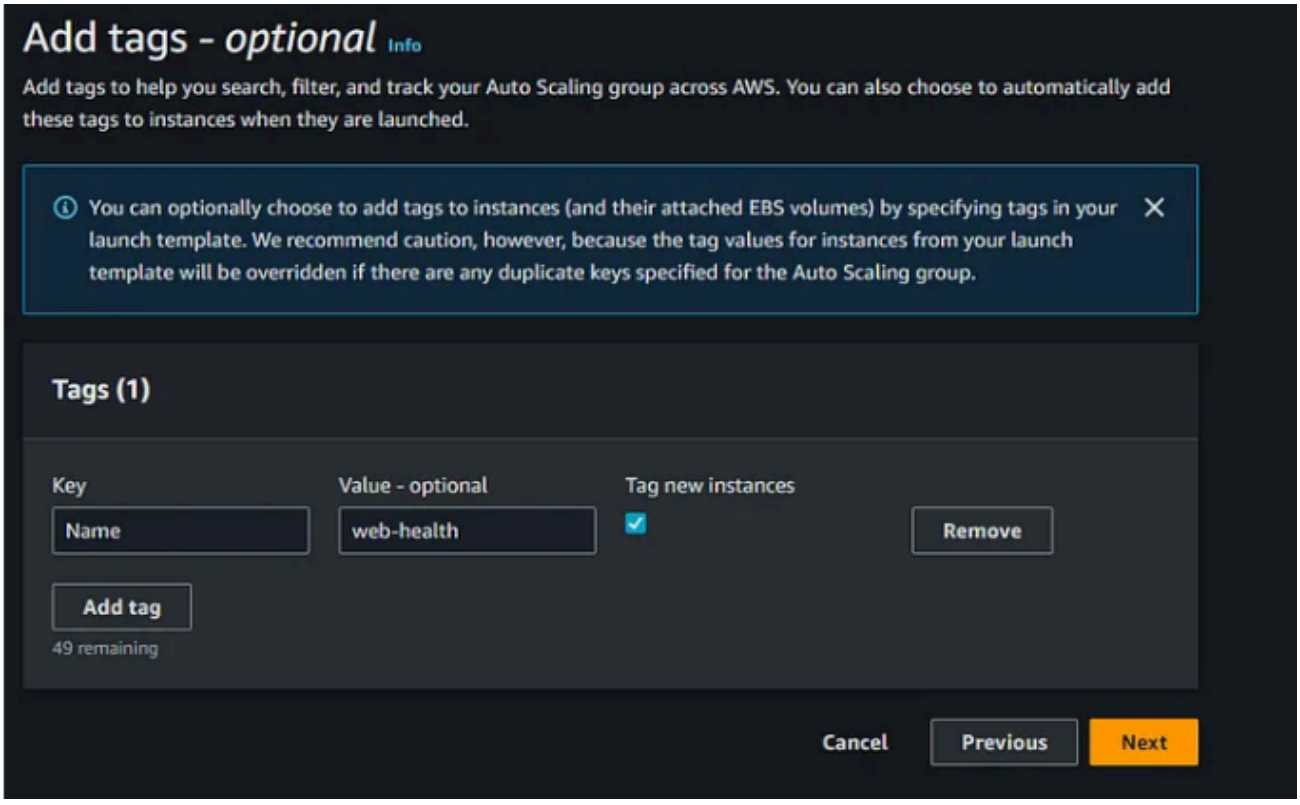



Add tags

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.

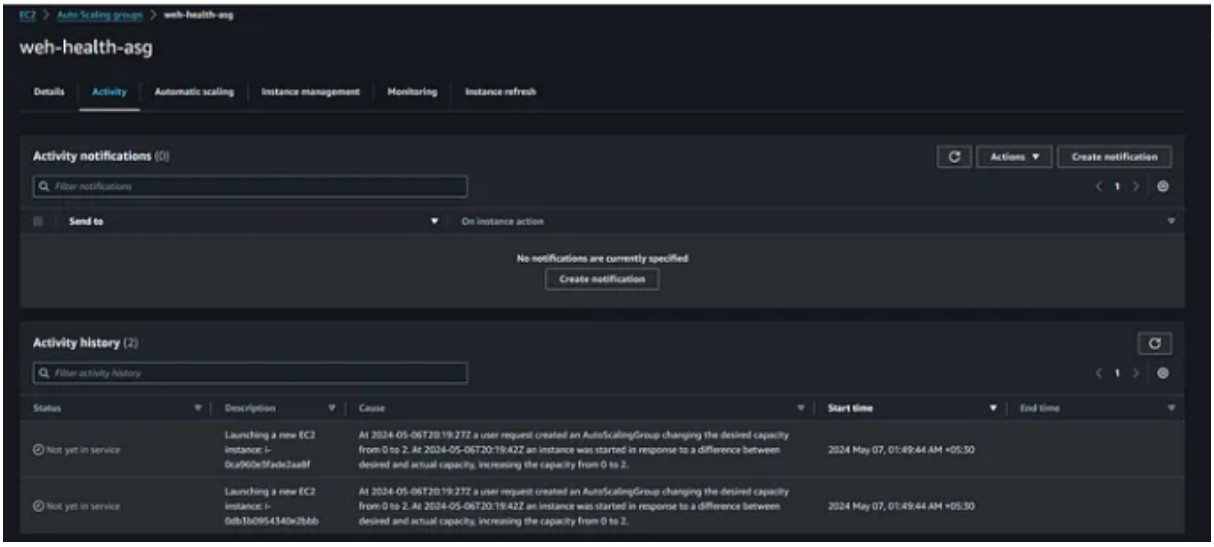


Now, let's proceed by clicking on "Next" and review all the provided information before finalizing the creation of the auto scaling group. Once verified, we can proceed with the creation process.

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.

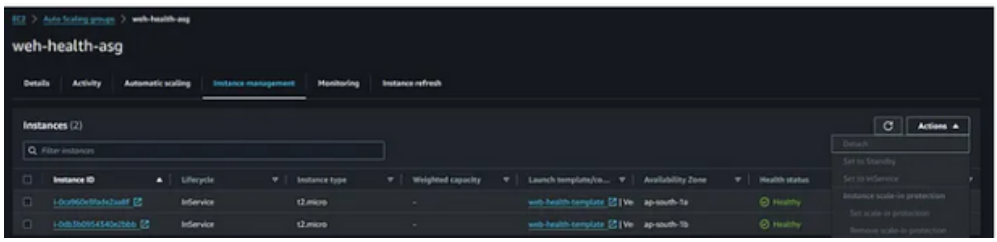
You can observe that two instances are being created to fulfill the desired capacity. Additionally, there's a scaling policy in place. If the CPU utilization drops below 50%, it will terminate instances accordingly. However, it will always maintain a minimum of one instance.

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.

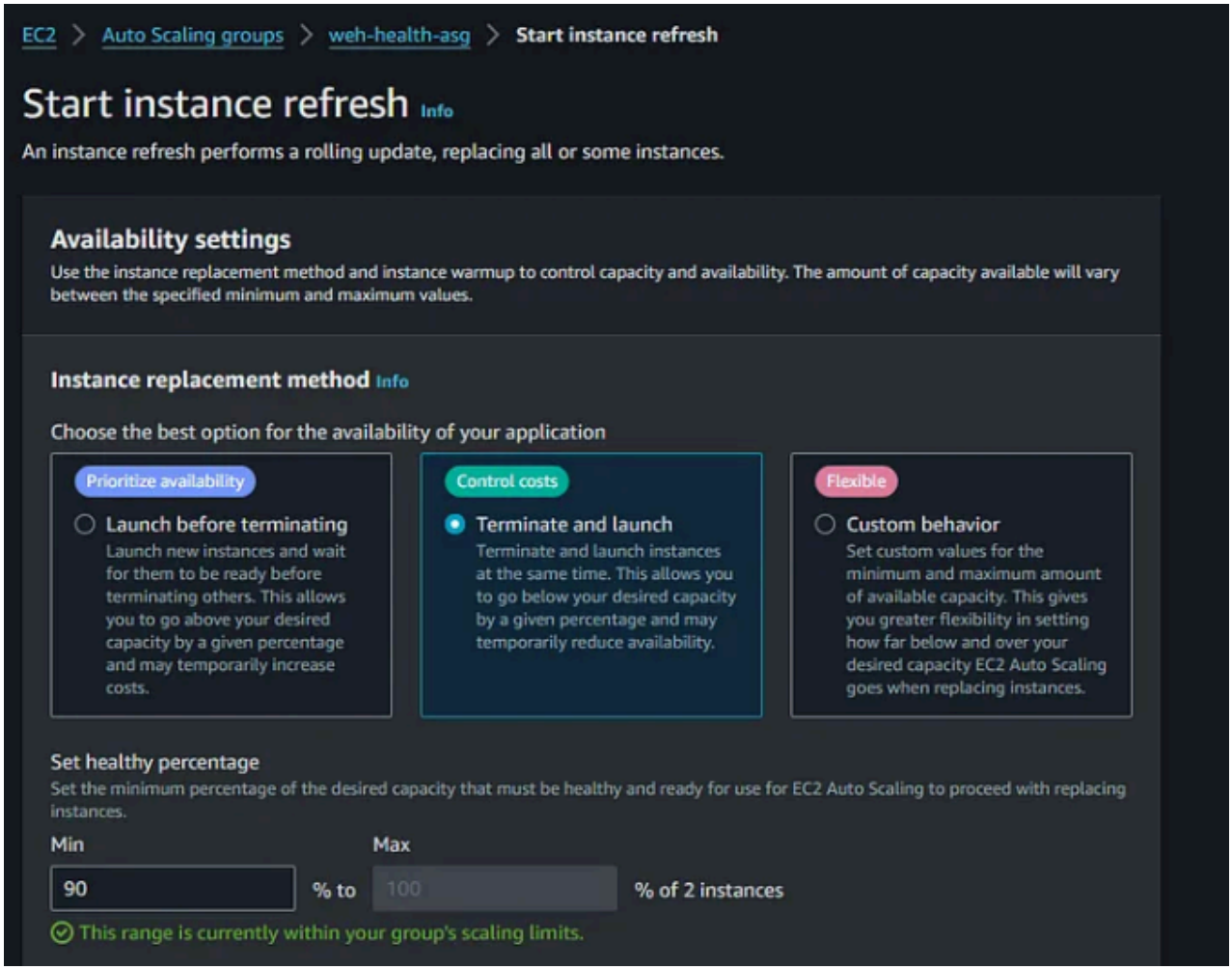
We'll wait for a few minutes to ensure that all instances are in a healthy state. Once they are marked as healthy, we'll explore some additional options. Currently, both instances are running and healthy. You should also see them updated in the target group.



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 auto-scaling 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.

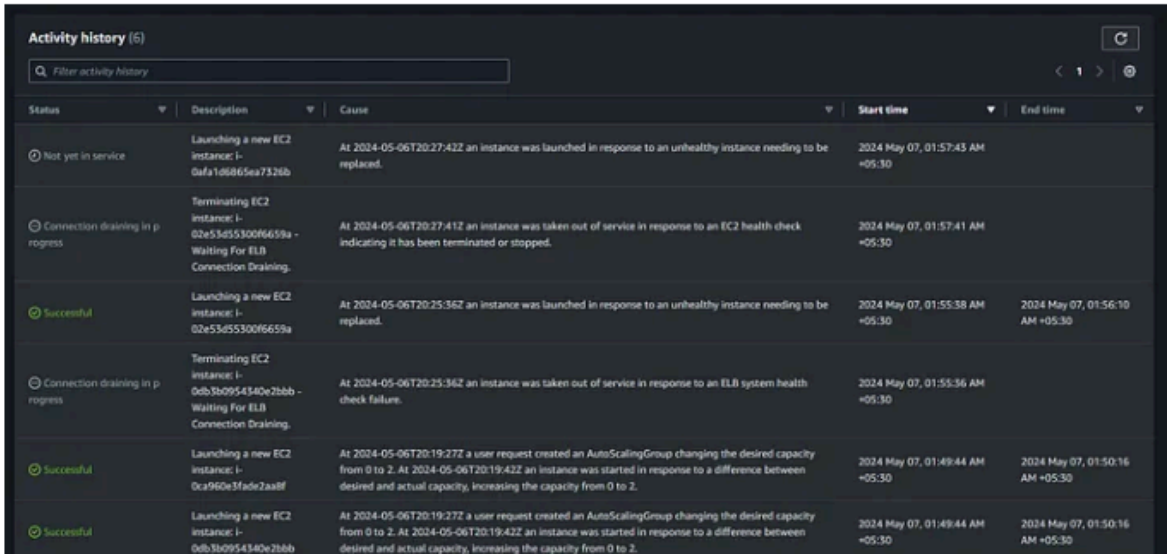
To apply changes to instances, you can edit the auto-scaling group and select the new launch template. Once the changes are saved, you must initiate an instance refresh to apply the changes to existing instances. During instance refresh, instances are terminated and recreated based on the updated launch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



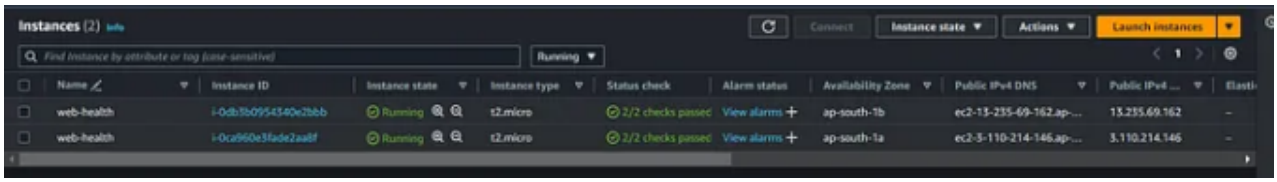

When initiating an instance refresh, ensure that you specify a minimum healthy percentage. This ensures that a certain percentage of instances remain healthy during the refresh process, preventing disruptions to your application. It's recommended to set a higher healthy percentage to maintain application 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.

We're concluding our discussion on EC2 concepts.....