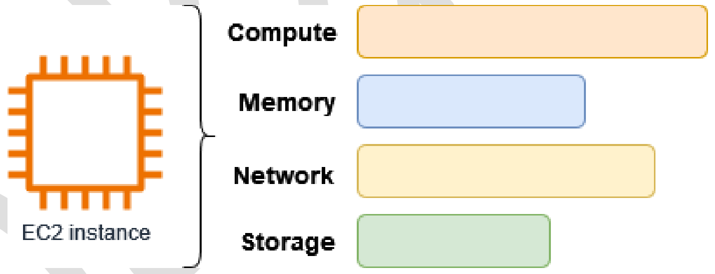
Elastic Compute Cloud
AWS/DevOps notes

Notes by: Prashant Karne

What is EC2:

Amazon EC2 provides scalable computing capacity in the AWS cloud.
You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking and manage storage.
Amazon EC2 enables you to scale up or scale down the instance within a very less time.
Amazon EC2 is having two storage options - **EBS** and **Instance Store.**
Preconfigured templates are available known as amazon machine image.
By default, when you create an ec2 account with amazon your account is limited to a maximum of 20 instances per EC2 region with two default high I/O instances.

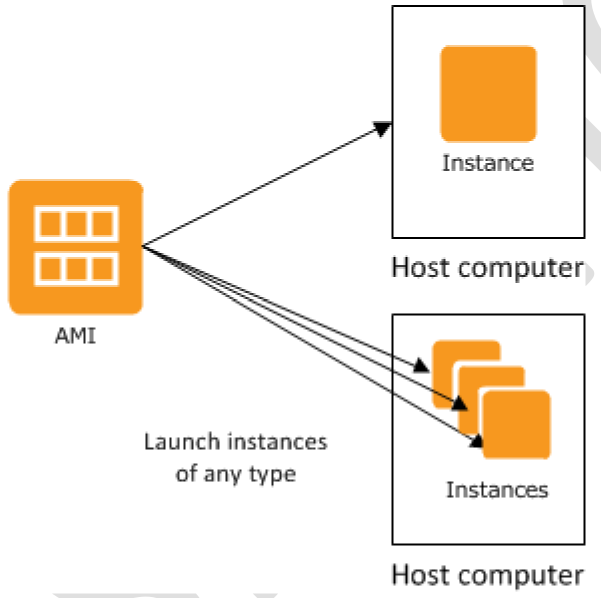
Amazon Elastic Compute Cloud (Amazon EC2) provides on-demand, scalable computing capacity in the Amazon Web Services (AWS) Cloud.
Using Amazon EC2 reduces hardware costs so you can develop and deploy applications faster.
You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage.
You can add capacity (scale up) to handle compute-heavy tasks, such as monthly or yearly processes, or spikes in website traffic. When usage decreases, you can reduce capacity (scale down) again.



An **EC2 instance is a virtual server** in the AWS Cloud. When you launch an EC2 instance, the instance type that you specify determines the hardware available to your instance. Each instance type offers a different balance of compute, memory, network, and storage resources.

AMI – Amazon Machine Image

An Amazon Machine Image (AMI) is a template that contains a software configuration to launch and EC2 instance. Software configuration means - an operating system, an application server, and applications required to launch and instance.



From an AMI, you launch an instance, which is a copy of the AMI running as a virtual server in the cloud. You can launch multiple instances of an AMI, as shown in the following figure.

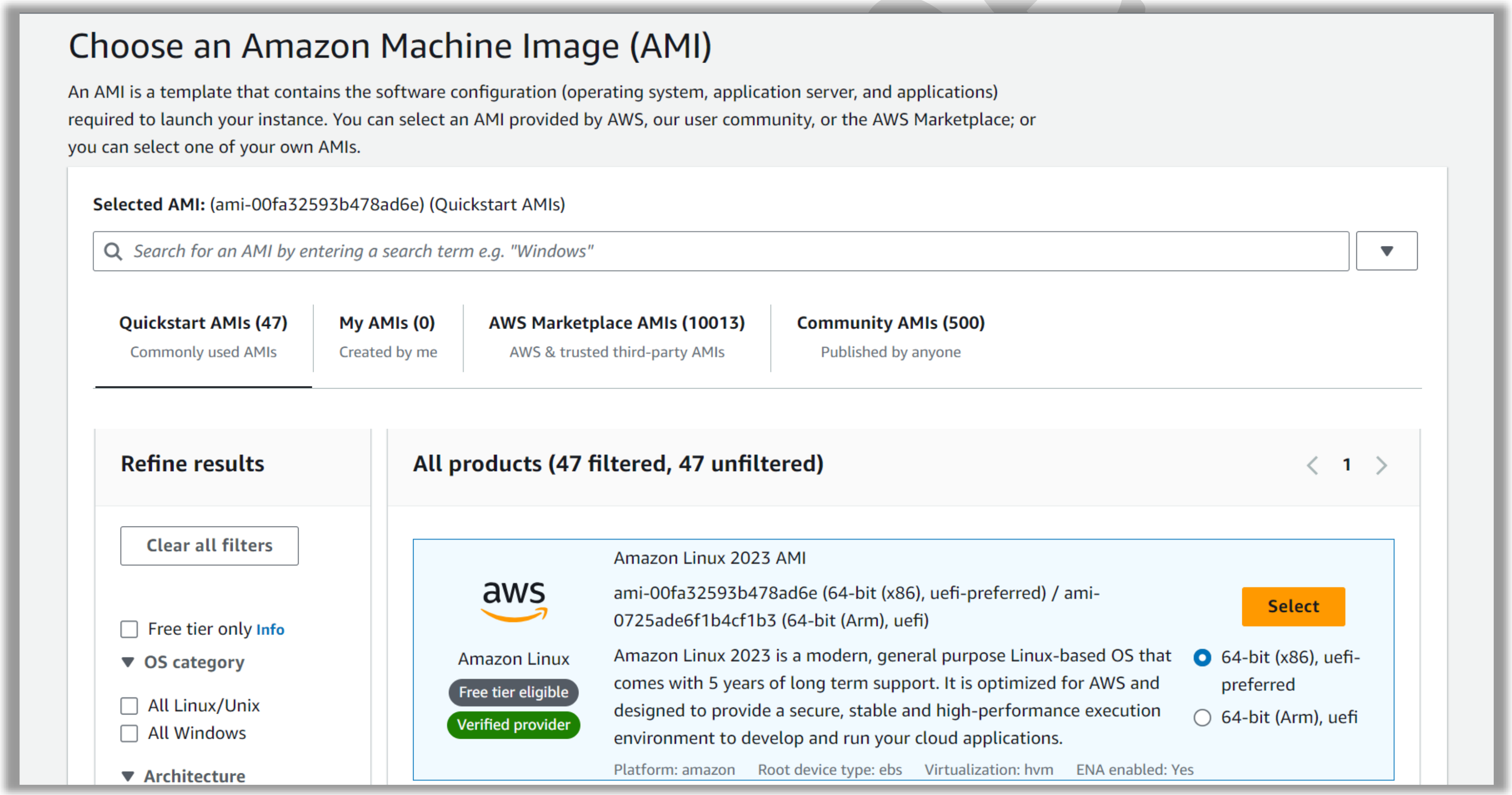
An Amazon Machine Images are supported and maintained by AWS that provides the information required to launch an instance. You must specify an AMI when you launch an instance. You can launch multiple instances from a single AMI when you require multiple instances with the same configuration or you can use different AMIs to launch instances when you require instances with different configurations.

Types of AMIs:

Public AMI: Base AMI – Available to all AWS account users also called as Base Image.

Marketplace: **AMIs in the marketplace are verified by AWS** itself and safe to use. These AMIs are basically used for software vendors to sell their products through AWS. Customers will be billed by AWS only, but then AWS will pay the AMI owner in return. (For Centos this is a bit particular, as this is a free distribution. But been in the marketplace confirms the users that the AMI is safe).

Community AMIs: Whenever you create an AMI, you can add permissions to it to make it public. In that case, it goes to "community AMIs". These are AMIs that comes from AWS users, and are not verified by AWS.



Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Selected AMI: (ami-00fa32593b478ad6e) (Quickstart AMIs)

Search for an AMI by entering a search term e.g. "Windows"

Quickstart AMIs (47) — Commonly used AMIs
My AMIs (0) — Created by me
AWS Marketplace AMIs (10013) — AWS & trusted third-party AMIs
Community AMIs (500) — Published by anyone
Refine results
Clear all filters
Free tier only Info
OS category
All Linux/Unix
All Windows
Architecture
All products (47 filtered, 47 unfiltered)
1
Amazon Linux
Free tier eligible
Verified provider
Select

Amazon Linux 2023 AMI

ami-00fa32593b478ad6e (64-bit (x86), uefi-preferred) / ami-0725ade6f1b4cf1b3 (64-bit (Arm), uefi)

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

[✓ 64-bit (x86), uefi-preferred] 64-bit (Arm), uefi

Platform: amazon   Root device type: ebs   Virtualization: hvm   ENA enabled: Yes

Instance Type:

When you launch an EC2 instance, the instance type that you specify determines the hardware of the host computer used for your instance. Each instance type offers different **compute, memory, networking and storage capabilities**, and is grouped in an **instance family** based on these capabilities. Select an instance type based on the requirements of the application or software that you plan to run on your instance.

Instance Family: These instance types are grouped into families and first letter in the type denotes the family of the Instance type, whereas letter part denotes the size (nano, small, medium, large, extra-large etc.)

Series

For example:

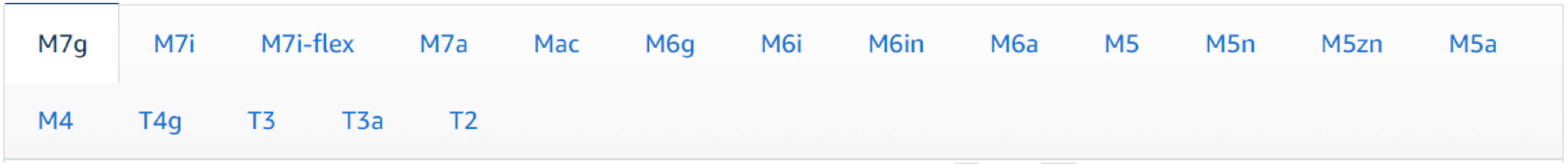
C4 – Compute optimised (for workloads requiring significant processing)
T2 – Lowest cost general **purpose** (webserver/small DBs)
R3 – Memory optimised (for memory intensive workloads)
G2 – GPU-based instances (intended for graphics and general-purpose GPU compute workloads)
I2 – Storage optimised (for workloads requiring high amounts of fast SSD storage)
D2 – Dense storage (File Servers/ Data Warehousing/ Hadoop)

Types of Instances:

1) **General Purpose:** General purpose instances provide a balance of compute, memory and networking resources, and can be used for a variety of diverse workloads. These instances are ideal for applications that use these resources in equal proportions such as web servers and code repositories.

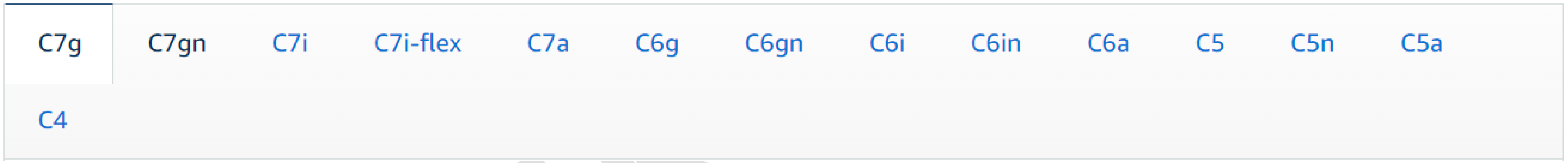
There are 3 series are available in general purpose instance:

a. **A series:** A1 (Previous Generation)
b. **M series**: M4, M5, M5a, M5d, M5ad (large).
c. **T series:** T2 (free tier eligible), T3, T3a Instances are available in four sizes: Nano, Small, Medium, Large.



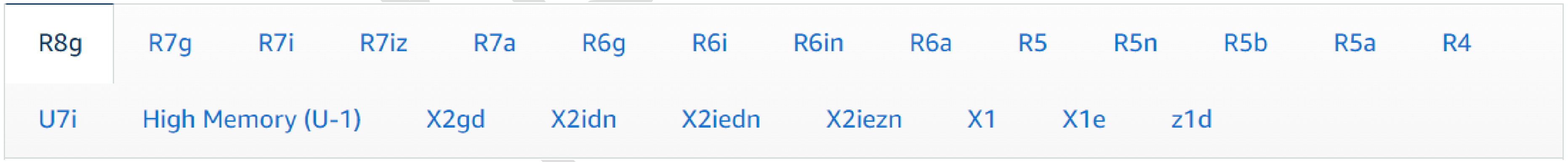
2) **Compute Optimized:** Compute Optimized instances are ideal for compute bound applications that benefit from high performance processors. Instances belonging to this category are well suited for batch processing workloads, media transcoding, high performance web servers, high performance computing (HPC), scientific modelling, dedicated gaming servers and ad server engines, machine learning inference and other compute intensive applications.

C Series: Three types are available: C4, C5, C5n, C3- previous generation instance

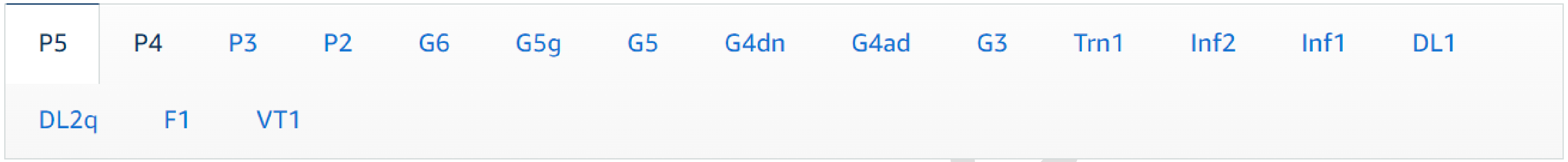


3) **Memory Optimized:** Memory optimized instances are designed to deliver fast performance for workloads that process large data sets in memory.

There are **3 series are available: R series, X series, Z series**

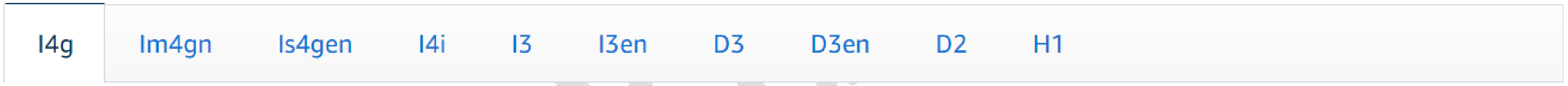


4) **Accelerated Computing:** Accelerated computing instances use hardware accelerators, or co-processors, to perform functions, such as floating-point number calculations, graphics processing, or data pattern matching, more efficiently than is possible in software running on CPUs.

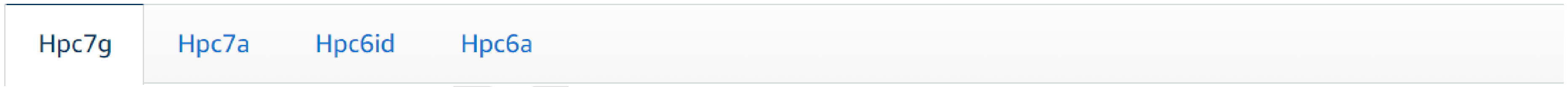

P5 P4 P3 P2 G6 G5g G5 G4dn G4ad G3 Trn1 Inf2 Inf1 DL1
DL2q F1 VT1

5) **Storage Optimized:** Storage optimized instances are designed for workloads that require high, sequential read and write access to very large data sets on local storage. They are optimized to deliver tens of thousands of low-latencies, random I/O operations per second (IOPS) to applications.

It is of three series:
A. D series- D2 instance
B. H series- H1 instance
C. I series- I3 and I3en instance


I4g Im4gn Is4gen I4i I3 I3en D3 D3en D2 H1

6) **High Performance Computing (HPC) Optimized-** High performance computing (HPC) instances are purpose built to offer the best price performance for running HPC workloads at scale on AWS. HPC instances are ideal for applications that benefit from high-performance processors such as large, complex simulations and deep learning workloads.


Hpc7g Hpc7a Hpc6id Hpc6a

EC2 Purchasing Options:

There are 6 ways of purchasing options available for AWS EC2 instances, but there are 3 ways to pay for Amazon EC2 instance i. e On demand, Reserved instance and Spot instance. You can also pay for dedicated host which provide you with EC2 instance capacity on physical servers dedicated for your use.

1. On Demand Instance

2. Dedicated instance

3. Dedicated Host

4. Spot instance

5. Scheduled instance

6. Reserved instance

1. On-Demand Instance:

➢ AWS on demand instances are virtual servers that run in AWS of AWS relational database service (RDS) and are **purchased at a fixed rate per hour.**

➢ AWS recommends using on demand instances for applications with short term irregular workloads that cannot be interrupted.

➢ They also suitable for use during testing and development of applications on EC2.

➢ With on demand instances you only pay for EC2 instances you use.

➢ The use of on demand instances frees you from the cost and complexities of planning, purchasing, and maintaining hardware and transforms what are commonly large fixed costs into mush smaller variable cost.

➢ Pricing is per instance hour consumed for each instance from the time an instance is launched until if it terminated of stopped.

➢ Each partial instance hour consumed will be billed per second for Linux instances and as a full hour for all other instance types.

2. Dedicated Instance:

➢ Dedicated instances are run in a VPC on hardware that is dedicated to a single customer.

➢ Your dedicated instances are physically isolated at the host hardware level from instances that belong to other AWS account.

➢ Dedicated instances may share hardware with other instances from the same account that are not dedicated instance.

➢ Pay for dedicated instances on demand save up to 70% by purchasing reserved instance or save up to 90% by purchasing spot instances.

3. Dedicated Host:

➢ An Amazon EC2 dedicated host is a physical server with EC2 instance capacity fully dedicated to your use.

➢ Dedicated host can help you address compliance requirement and reduce costs by allowing you to use your existing server bound software licenses.

➢ Pay for a physical host that is fully dedicated to running your instances and bring your existing per socket, per core, per VM software license to reduce cost.
➢ Dedicated host gives you additional visibility and control over how instances are placed in a physical server and you can consistently deploy your instances to the same server over time.
➢ As a result dedicated host enables you to use your existing server bound software license and address corporate compliance and regulatory requirements.
➢ Instances that run on a dedicated host are the same virtualized instances that you had get with traditional EC2 instances that use the XEN Hypervisor.
➢ Each dedicated host supports a single instance size and type (for e.g C3.XLARGE)
➢ Only BYOL, Amazon linux and AWS marketplace AMIs can be launched onto dedicated hosts.

4. Spot Instances:

➢ Amazon EC2 spot instances let you take advantage of unused EC2 capacity in the AWS cloud. Spot instances are available at up to 90% discount compared to on-demand prices.
➢ You can use spot instances for various test and development workloads.
➢ You can also have the options to hibernate, stop or terminate your spot instances when EC2 reclaims the capacity back with two minutes of notice.
➢ Spot instances are spare EC2 capacity that can save you up 90% off of on demand prices that AWS can interrupt with a 2-minute notification. Spot uses the same underlying EC2 instances as on-demand and reserved instances, and is best suited for flexible workloads.
➢ You can request spot instances up to your spot limit for each region.
➢ You can determine the status of your spot request via spot request status code and message. You can access spot request status information on the spot instance page of the EC2 console of the AWS management console.
➢ In case of hibernate, your instance gets hibernated and RAM data persisted. In case of stop, your instance gets shutdown and RAM is cleared.
➢ With hibernate, spot instances will pause and resume around any interruptions so your workloads can pick up from exactly where they left off. Question: when would my spot instance get interrupted? Ans: primary reason would be Amazon EC2 capacity requirement (e.g: on demand or reserved instances). Secondarily, if you have chosen to set a ‘max spot price’ and the spot price rise above.

5. Scheduled Instance:

➢ Scheduled reserve instances enable you to purchase capacity reservations that recur on a daily, weekly or monthly basis, with a specified start time and duration for one year term.
➢ You reserve the capacity in advance so that you know it is available when you need it.
➢ You pay for the time that the instances are scheduled even if you do not use them.

➢ Scheduled instances are a good choice for workloads that do not run continuously but do run on a regular schedule.

➢ Purchase instances that are always available on the specified recurring schedule for one year term. For example: you can use schedule instances for an application that runs during business hours of for batch processing that run at the end of the week.

6. Reserved Instances:

➢ Amazon EC2 RI provides a significant discount up to 75% compared to on demand pricing and provide a capacity reservation when used in a specific availability zone.

➢ Reserved instances give you the option to reserve a DB instance for a one- or three-year term and in turn receive a significant discount compared to the on-demand instance pricing for the DB instance.

There are 3 types of RI are available such as

a. Standard RI: these provide the most significant discount up to 75% off on-demand and are best suited for steady-state usage.
b. Convertible RI: these provide a discount up to 54% and the capability to change the attributes of the RI as long as the exchange results in the creation of reserved instances of greater or equal value.
c. Scheduled RI: these are available to lunch within the time window you reserve.

EC2 Key Pair:

A key pair is combination of private and public key that contains information required to securely login into ec2 instance

IMP: Amazon EC2 stores the public key on instance, and you store the private key in the format of ***pem*** or ***ppk.***

Public key is used to encrypt data and a private key is used to decrypt data.

For Linux instances, the private key allows you to securely SSH into your instance. For Windows instances, the private key is required to decrypt the administrator password, which you then use to connect to your instance.

When you launch an instance, you can specify a key pair. If you plan to connect to the instance using SSH, you must specify a key pair. Depending on how you manage your security, you can specify the same key pair for all your instances or you can specify different key pairs.

IMP: When your instance boots for the first time, the public key that you specified at launch is placed on your Linux instance in an entry within ~/.ssh/authorized_keys. When you connect to your Linux instance using SSH, to log in you must specify the private key that corresponds to the public key.

How to access EC2 instance?

➢ To access instances you need a key and key-pair name.

➢ You can download the private key only once.

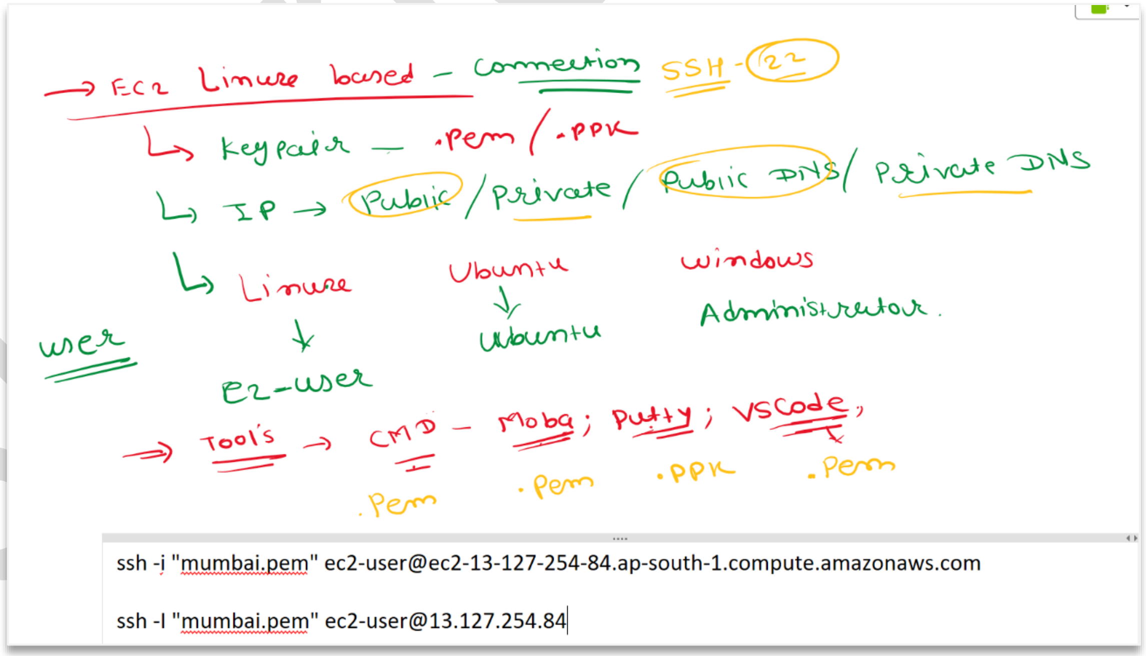
➢ The public key is saved by AWS to match it to the key pair name and private key when you try to login to the EC2 instances.

➢ Without key pair you cannot access instances via RDP or SSH (linux).

➢ There are 20 EC2 instances soft limit per account, you can submit a request to AWS to increase it.

To connect or SSH into Ec2 instance you must have below things with you:

1) **Keypair** -- .pem OR .ppk
2) **SSH Client** – (Putty, MobaXterm, CMD)
3) **Public IP / Public DNS** – to access publicly.
   Private IP / Private DNS – to access ec2 instance form private network only.
4) **Username:** Below are the default user used for login into instance
   a. Linux → ec2-user
   b. Ubuntu → ubuntu
   c. Windows → Administrator


→ EC2 Linux based – connection SSH-22
↳ key pair – .pem / .ppk
↳ IP → Public / Private / Public DNS / Private DNS
↳ Linux
Ubuntu
windows
user
ec2-user
ubuntu
Administrator.
→ Tool's → CMD – Moba; Putty; VSCode;
.pem
.pem
.ppk
.pem

```
ssh -i "mumbai.pem" ec2-user@ec2-13-127-254-84.ap-south-1.compute.amazonaws.com

ssh -I "mumbai.pem" ec2-user@13.127.254.84
```

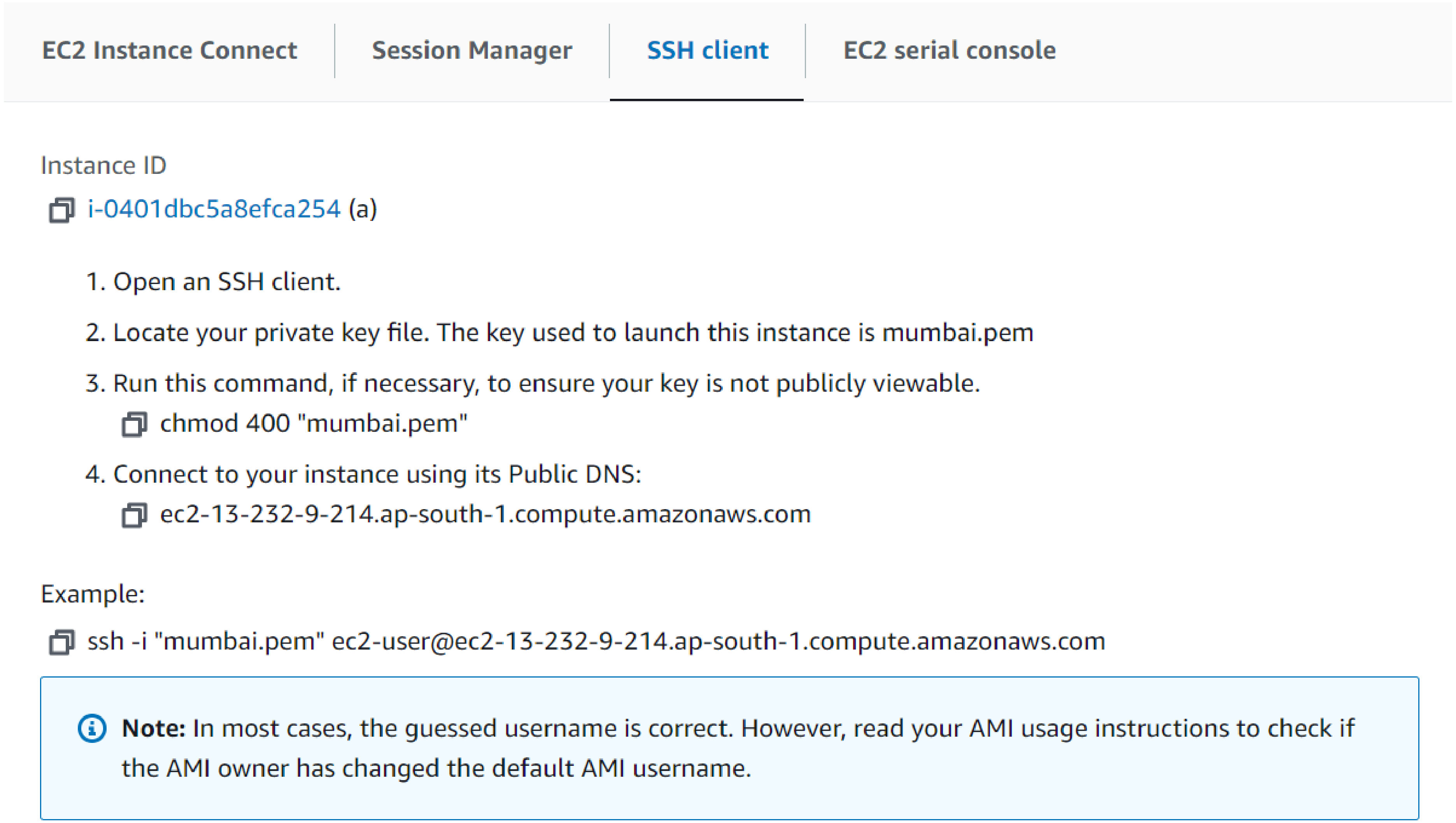
Command to SSH into Linux EC2 Instance:

```
ssh -i “key.pem” ec2-user@pulic-IP/public-DNS
```
(.pem file must be private)

Command to SSH into Linux EC2 Instance:

```
ssh -i “key.pem” ubuntu@pulic-IP/public-DNS
```
(.pem file must be private

Connect to Instance using SSH Client (SSH Client we are using is: MobaXterm)



Instance ID

i-0401dbc5a8efca254 (a)

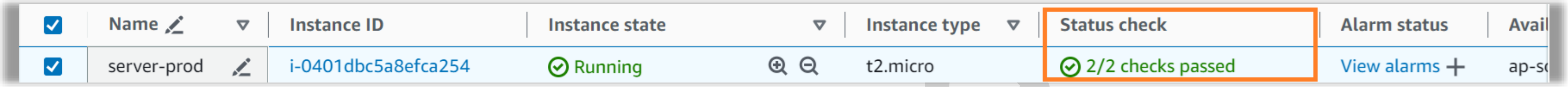
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is mumbai.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   ```
   chmod 400 "mumbai.pem"
   ```
4. Connect to your instance using its Public DNS:
   ```
   ec2-13-232-9-214.ap-south-1.compute.amazonaws.com
   ```

Example:

```
ssh -i "mumbai.pem" ec2-user@ec2-13-232-9-214.ap-south-1.compute.amazonaws.com
```

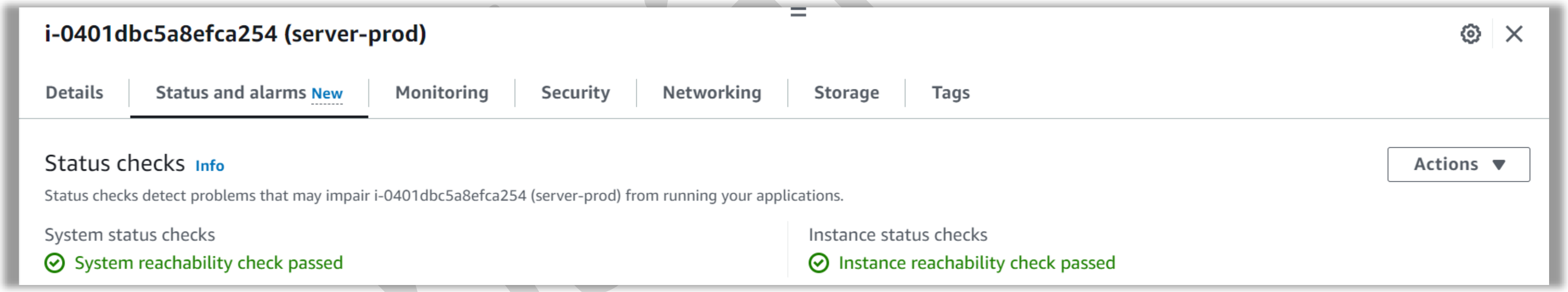
Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

EC2 Status Check:



☑	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Avail
☑	server-prod	i-0401dbc5a8efca254	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +	ap-s

- ➢ By default AWS EC2 instance performs automated status checks every one minute.
- ➢ This is done on every running instance to identify any hardware or software issue.
- ➢ Status check is built into the AWS EC2 instance and it cannot be configured, deleted or disable.
- ➢ EC2 services can send its metric data to AWS CloudWatch every 5 minutes (enable by default)
- ➢ Enable detailed monitoring is chargeable and sends metric in every 1 minute.
- ➢ You are not charged for EC2 instances if they are stopped however attached EBS volumes incur charges.



i-0401dbc5a8efca254 (server-prod)

Status checks

Status checks detect problems that may impair i-0401dbc5a8efca254 (server-prod) from running your applications.

System status checks
✓ System reachability check passed

Instance status checks
✓ Instance reachability check passed

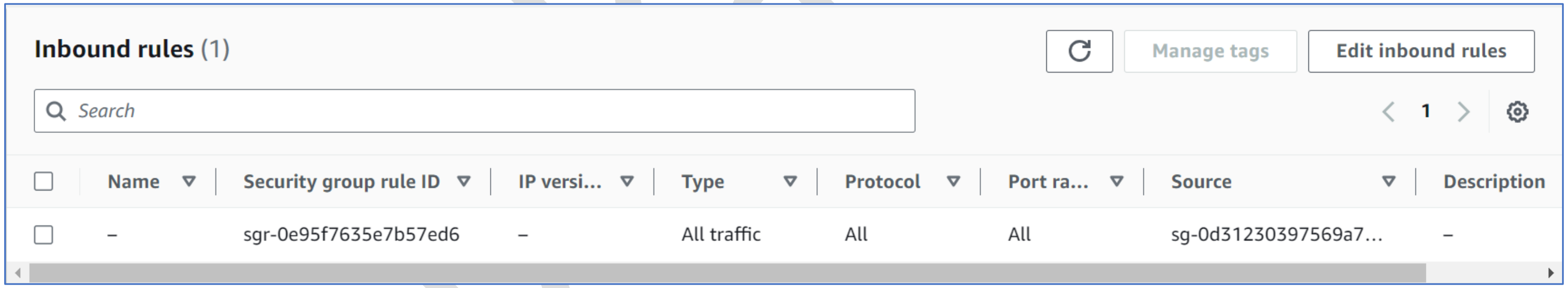
EC2 Security Groups:

- ➢ A security group acts as a virtual firewall for your EC2 instances to control incoming and outgoing traffic.
- ➢ So, with the help of Security Group, we can decide who can connect to our instance.
- ➢ Inbound rules control the incoming traffic to your instance, and outbound rules control the outgoing traffic from your instance.
- ➢ When you launch an instance, you can specify one or more security groups.
- ➢ You can add rules to each security group that allow traffic to or from its associated instances.

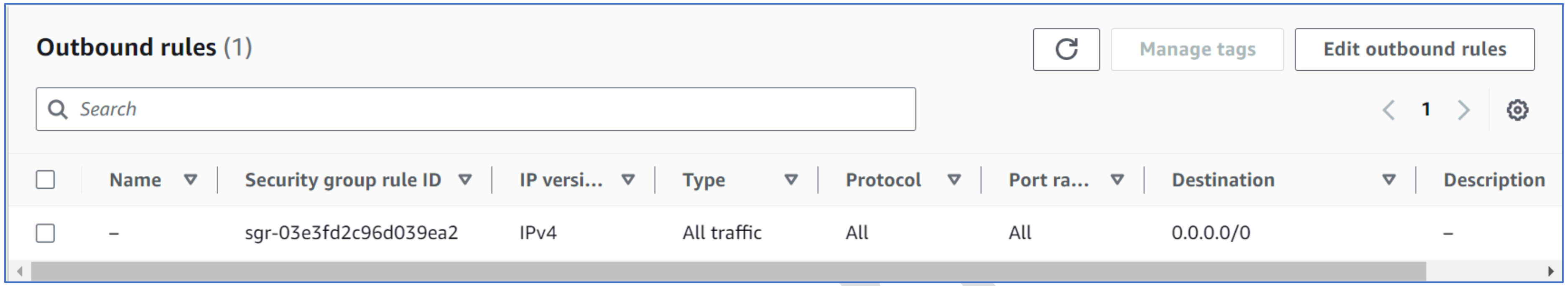
- By default, security groups contain outbound rules that allow all outbound traffic.
- We can only allow the inbound or outbound traffic but we cannot deny the traffic [Security group rules are always permissive; you can't create rules that deny access.
- **IMP: Security Groups are Stateful:**
  Stateful means for any inbound rule that we define in SG doesn't not need an outbound rule defined. For any inbound rule outbound rule we allow is automatically allowed in outbound rule.
- When you associate multiple security groups with an instance, the rules from each security group are effectively aggregated to create one set of rules. We can attach max 5 security groups to our ec2 instance.
- **IMP:** Security Group can be attached on EC2 instance level, if specific on ENI (Elastic Network Interface) level.

Types of Security Groups

1) **Default Security Group:** Each VPC comes with a default security group. **IMP:** In default security group by default, all inbound and outbound traffic is allowed. Below are the default inbound/outbound rules in default SG, off course we can add or remove any rule in default SG as well.


Inbound rules (1)

Name	Security group rule ID	IP versi...	Type	Protocol	Port ra...	Source	Description
–	sgr-0e95f7635e7b57ed6	–	All traffic	All	All	sg-0d31230397569a7...	–

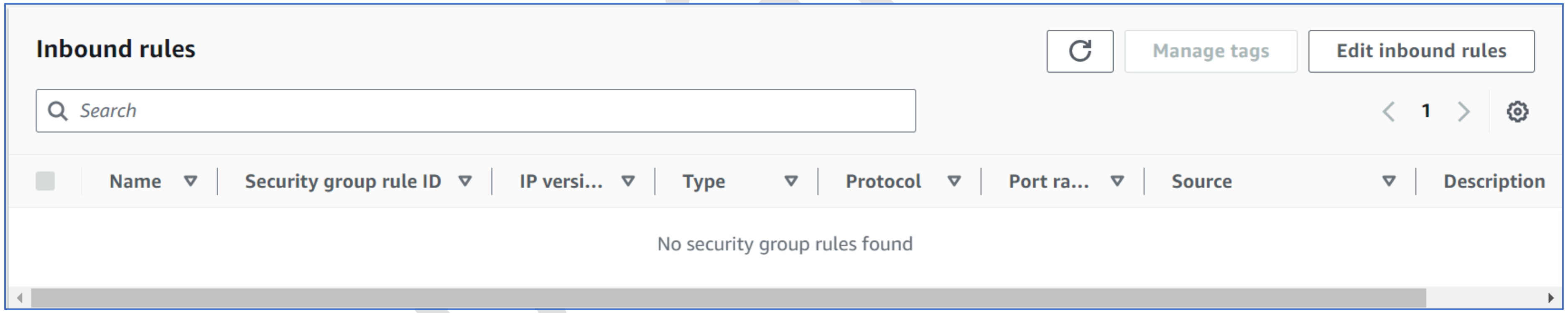

Outbound rules (1)

Manage tags | Edit outbound rules

Search

	Name	Security group rule ID	IP versi...	Type	Protocol	Port ra...	Destination	Description
☐	–	sgr-03e3fd2c96d039ea2	IPv4	All traffic	All	All	0.0.0.0/0	–

2) Custom Security Group: The security groups we create are called as custom security groups. We can create multiple security groups to reflect the different roles that our instances play; for example, web servers or database servers.
IMP: In custom security group by default all inbound traffic is not allowed and all outbound traffic is allowed.

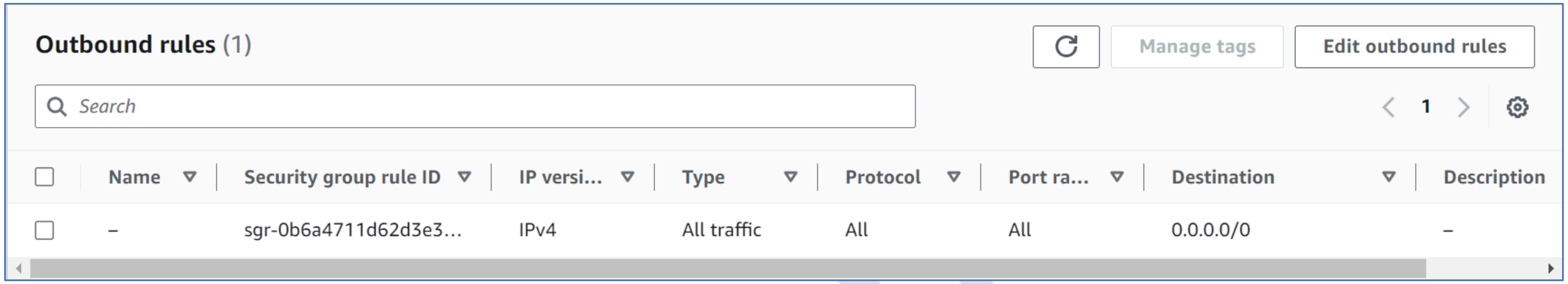

Inbound rules

Manage tags | Edit inbound rules

Search

	Name	Security group rule ID	IP versi...	Type	Protocol	Port ra...	Source	Description

No security group rules found

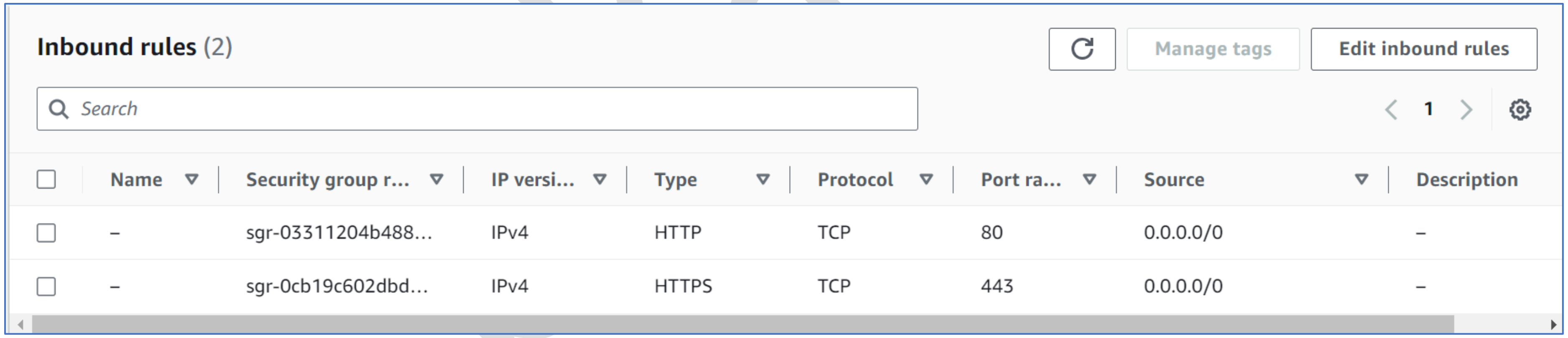


Outbound rules (1)

Name	Security group rule ID	IP versi...	Type	Protocol	Port ra...	Destination	Description
–	sgr-0b6a4711d62d3e3...	IPv4	All traffic	All	All	0.0.0.0/0	–

You can create a security group and add rules that reflect the role of the instance that's associated with the security group. For example, an instance that's configured as a web server needs security group rules that allow inbound HTTP and HTTPS access. Likewise, a database instance needs rules that allow access for the type of database, such as access over port 3306 for MySQL.

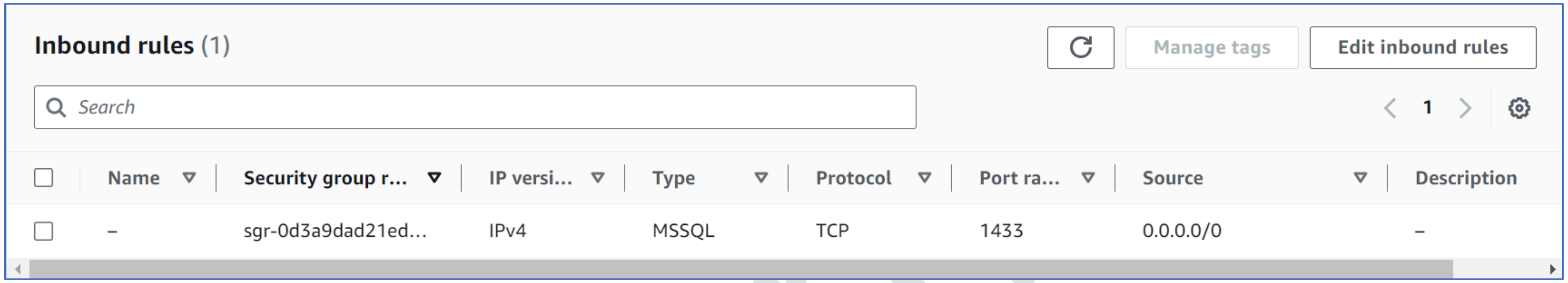
Eg: Security group for WEBSERVER (we can also add PORT 22/3389 to enable the SSH/RDP connection if required)



Inbound rules (2)

Name	Security group r...	IP versi...	Type	Protocol	Port ra...	Source	Description
–	sgr-03311204b488...	IPv4	HTTP	TCP	80	0.0.0.0/0	–
–	sgr-0cb19c602dbd...	IPv4	HTTPS	TCP	443	0.0.0.0/0	–

Eg: Security Group for MSSQL DATABASE Server (we can also add PORT 22/3389 to enable the SSH/RDP connection if required)

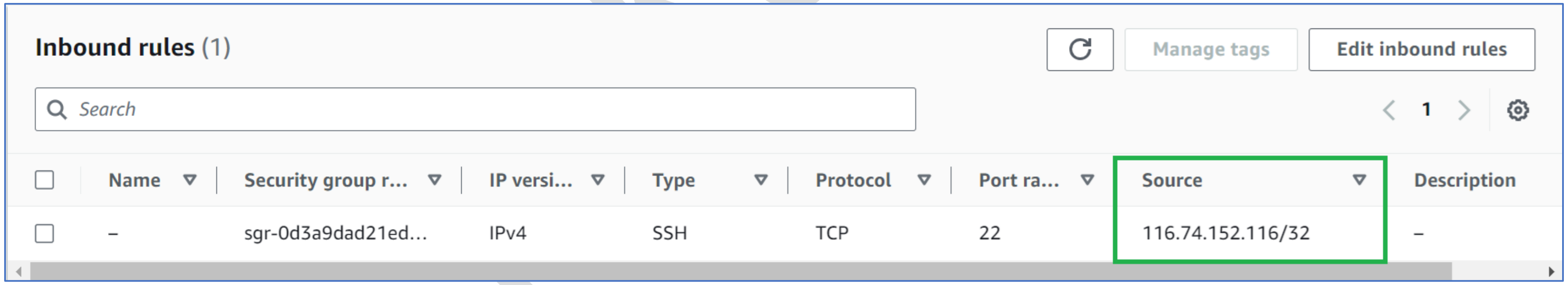


Inbound rules (1)

	Name	Security group r...	IP versi...	Type	Protocol	Port ra...	Source	Description
☐	–	sgr-0d3a9dad21ed...	IPv4	MSSQL	TCP	1433	0.0.0.0/0	–

To enable connection from specific computer/ IP: Suppose we have a case where we want to enable SSH connection to be established from particular group of user/ particular IP only then in this case we can mention the IP of authorised computer in Source. Instead of mentioning the Anywhere IPv4 we can specify 'My IP' option to enable connection from particular IP only. This is one way of enhancing security of our instance.

So in below example IP 116.74.152.116 is the only source who can connect/SSH into ec2 instance to which this SG is attached. Computer with this IP can only connect to our EC2 instance.



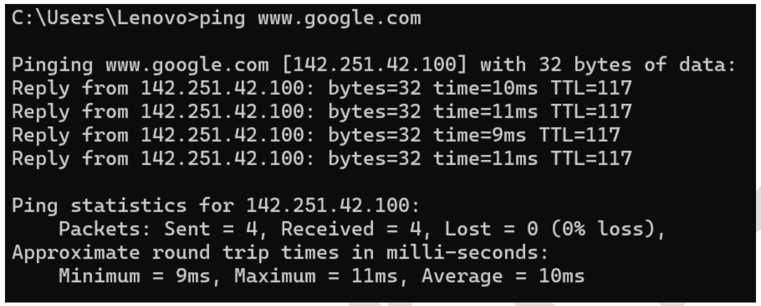
Inbound rules (1)

	Name	Security group r...	IP versi...	Type	Protocol	Port ra...	Source	Description
☐	–	sgr-0d3a9dad21ed...	IPv4	SSH	TCP	22	116.74.152.116/32	–

Networking Concept: Ping?

A ping (Packet Internet or Inter-Network Groper) is a basic Internet program that allows a user to test and verify if a particular destination IP address exists and can accept requests. Basically, we use ping command to check reachability of particular IP or DNS address.

For example, in below screenshot you can see that we are trying to ping google's website with `ping www.google.com` command and as you can see that its reachable as it receives all sent packets.

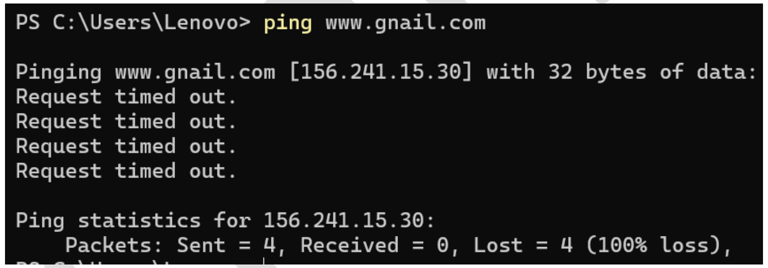


```
C:\Users\Lenovo>ping www.google.com

Pinging www.google.com [142.251.42.100] with 32 bytes of data:
Reply from 142.251.42.100: bytes=32 time=10ms TTL=117
Reply from 142.251.42.100: bytes=32 time=11ms TTL=117
Reply from 142.251.42.100: bytes=32 time=9ms TTL=117
Reply from 142.251.42.100: bytes=32 time=11ms TTL=117

Ping statistics for 142.251.42.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 9ms, Maximum = 11ms, Average = 10ms
```

In below screenshot you can see that the address www.gnail.com is not reachable hence returns no packets:

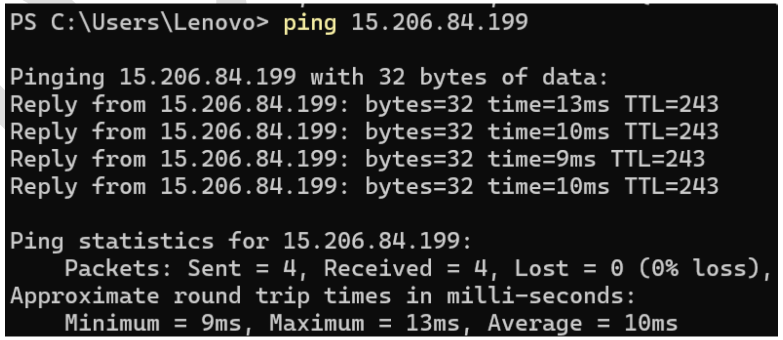


```
PS C:\Users\Lenovo> ping www.gnail.com

Pinging www.gnail.com [156.241.15.30] with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 156.241.15.30:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Likewise, we can ping ou public IP or public DNS of our ec2 instance as well to check if they are reachable exist and can accept the request.



```
PS C:\Users\Lenovo> ping 15.206.84.199

Pinging 15.206.84.199 with 32 bytes of data:
Reply from 15.206.84.199: bytes=32 time=13ms TTL=243
Reply from 15.206.84.199: bytes=32 time=10ms TTL=243
Reply from 15.206.84.199: bytes=32 time=9ms TTL=243
Reply from 15.206.84.199: bytes=32 time=10ms TTL=243

Ping statistics for 15.206.84.199:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 9ms, Maximum = 13ms, Average = 10ms
```

(**IMP Note:** before you try to ping AWS server, make sure that you have enabled **ALL ICMP - IPv4** rule first in security group otherwise you won't be able to ping a server. **Internet Control Message Protocol (ICMP)** is a network layer protocol used by network devices to diagnose reachability and network communication issues. ICMP is mainly used to determine whether or not data is reaching its intended destination in a timely manner)

Public vs Private vs Elastic IP:

Public IP address:

- A public IP address is an IPv4 address that's reachable from the Internet means can be access from public network. You can use public addresses for communication between your instances and the Internet. When you launch an instance in a default VPC, we assign it a public IP address by default.
- A public IP address gets assigned from **Amazon's pool of public IPv4 addresses**, and is not associated with your AWS account.
- Public IP gets changed when we stop ec2 instance. public IP gets released when ec2 instance is stopped, terminated or hibernated.
- AWS also releases default Public IP when we associate Elastic IP address to our Ec2 instance. And when we disassociate the Elastic IP address from instance, it receives a new public IP address.
- If we have a requirement where our ec2 instance needs static public IP which does not change time to time we can use Elastic IP address.
- IMP: Can we have multiple Public IP address assigned with EC2 Instance?

  Yes, with the help of – Elastic Network Interface, Elastic IP and Load balancer.

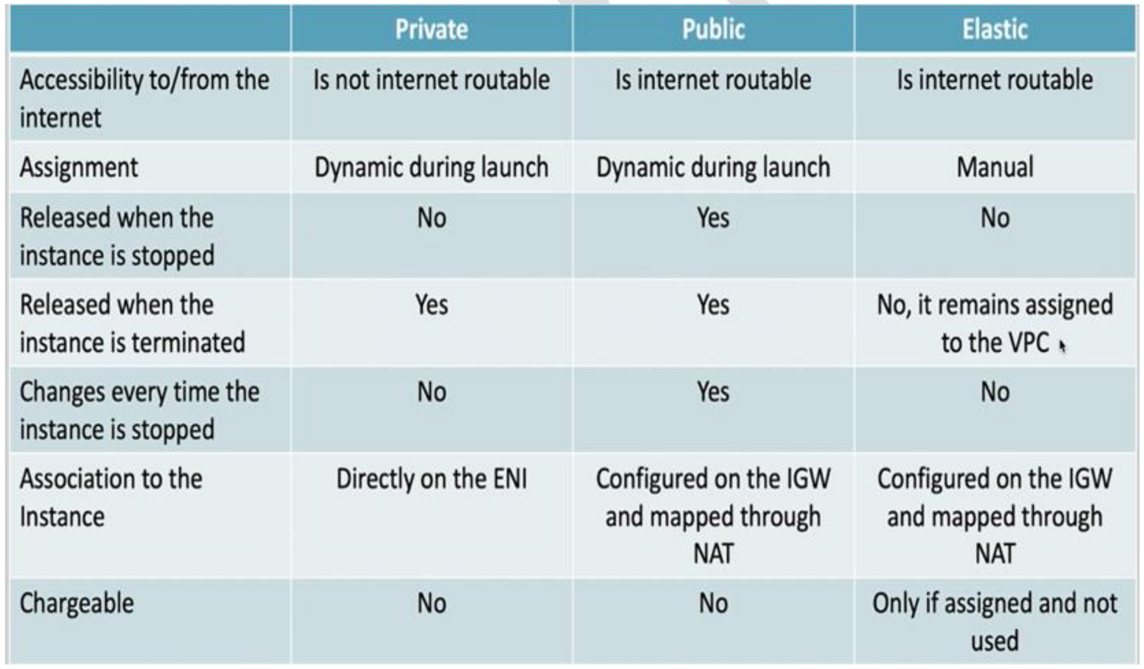
Private IP address:

- A private IPv4 address is an IP address that's not reachable over the Internet. Means cannot be accessible from public network. We can only access private IP from inside the aws private network.
- We can use private IPv4 addresses for communication between instances in the same network (VPC).
- The private IP address gets assigned from **our network (subnet) that we have defined.**
- A private IPv4 address, regardless of whether it is a primary or secondary address, remains associated with the network interface when the instance is stopped and started, or hibernated and started, and is released when the instance is terminated.
- We can have multiple private IP address to our Ec2 instance with the help of Elastic Network Interface (ENI).

Elastic IP address:

- An Elastic IP address is static; it does not change over a time even if we stop or hibernate our ec2 instance.
- Elastic IP is associated to our AWS account and remains with us even if we terminate ec2 instance,
- An Elastic IP address is a public IPv4 address, it is reachable from the internet.

- In case if our instance does not have a public IPv4 address, we can associate an Elastic IP address with your instance to enable communication with the internet.
- An Elastic IP address is for use in a specific Region only, and cannot be moved to a different Region.
- An Elastic IP address comes from Amazon's pool of IPv4 addresses, or from a custom IPv4 address pool that you have brought to your AWS account (BYOI – Bring Your Own IP)
- To use an Elastic IP address, you first allocate one to your account, and then associate it with your instance or a network interface.
- When you associate an Elastic IP address with an instance, it is also associated with the instance's primary network interface. When you associate an Elastic IP address with a network interface that is attached to an instance, it is also associated with the instance.
- You can disassociate an Elastic IP address from a resource, and then associate it with a different resource.
- A disassociated Elastic IP address remains allocated to your account until you explicitly release it.



	Private	Public	Elastic
Accessibility to/from the internet	Is not internet routable	Is internet routable	Is internet routable
Assignment	Dynamic during launch	Dynamic during launch	Manual
Released when the instance is stopped	No	Yes	No
Released when the instance is terminated	Yes	Yes	No, it remains assigned to the VPC
Changes every time the instance is stopped	No	Yes	No
Association to the Instance	Directly on the ENI	Configured on the IGW and mapped through NAT	Configured on the IGW and mapped through NAT
Chargeable	No	No	Only if assigned and not used

```
Practical Steps on Elastic IP (EIP):
====================================

[Steps in short- Allocate Elastic IP address to your account first ==> Associate Elastic IP address to
EC2 instance ==> Disassociate Elastic IP address from EC2 instance ==> Release Elastic IP address to
completely remove from account)

Go to EC2 dashboard ---> on the left hand sides menu scroll down and click on 'Elastic IPs' ---> Click
on 'Allocate Elastic IP address' ---> Keep 'Network border group' as it is [A network border group is
basically a collection of Availability Zones (AZs)] ---> for     Public IPv4 address pool option keep
selected option as 'Amazon's pool of IPv4 addresses' ---> click on 'Allocate' ---> now select EIP and
clock on 'Actions' ---> click on 'Associate Elastic IP address' option ---> select Resource type as
'instance' ---> choose an instance to which you want to attach the EIP from the drop down menu --->
Select private IP address of the instance from the drop down menu ---> enable 'Reassociation' in case if
you want to attach EIP to different resources or else no need ---> click on 'Associate' ---> you can
verify Elastic IP address is now associated to your ec2 instance.

      ➔ try to ping public IP and check if you are able to access over internet:

Now once you done with practical ---> go to Elastic IPs and select the IP address you want to release --
-> click on 'Actions' ---> click on 'Disassociate IP address ---> Click on 'Disassociate' ---> Select IP
again and click on 'Actions' ---> click on 'Release Elastic IP address' ---> Elastic IP is gone now!
```

Service Linked Role: (Service to Service Role)

A role that a service assumes to perform actions on your behalf is called a service role. When a role serves a specialized purpose for a service, it is categorized as a service role for EC2 instances. A service might automatically create or delete the role. It might allow you to create, modify, or delete the role as part of a wizard or process in the service.

It might require that you use IAM to create or delete the role. Regardless of the method, service-linked roles simplify the process of setting up a service because you don't have to manually add permissions for the service to complete actions on your behalf.

Scenario: Let's say we have an application running on EC2 instance and that application need data from S3 bucket so in such a scenario we can create a service linked role for EC2 to S3.

```
To create service to service role (eg: EC2-S3)
==============================================

Go to EC2 dashboard and launch a Linux instance ---> Login (SSH) into ec2 linux machine with 'sudo su -'
and enter command 'aws s3 ls' you can see an error message ---> Now go to IMA dashboard and click on
'Roles' ---> Click on 'Create role' to create new role ---> Keep option 'AWS service' selected ---> in
the Use case select a service for which you want to create role, in this example we want to create a
role for our ec2 instance service so choose a service as 'EC2' from drop down list --->  click on 'Next'
---> In the permission policy select 'AmazonS3FullAccess' policy ---> click on 'Next' ---> enter a name
for role ---> click on 'Create role'

Now go to EC2 instance console and select the instance to which we want to attach the service role we
have created ---> Click on 'Actions' ---> select 'Security' ---> click on 'Modify IAM role ---> click on
'Choose IAM role' and select service role ---> Click on 'Update IAM role' ---> now once agian login on
ec2 instance with 'sudo su -' and run the command 'aws s3 ls' ---> You can see no error and it will also
list the s3 buckets without any error
```

EC2 Volumes:

1. root volume
2. instance store volume
3. EBS – Elastic Block Storage

1. Root Volume

- When you launch an instance, we create a root volume for the instance.
- The root volume contains the image used to boot the instance. Each instance has a single root volume.
- You can add storage volumes to your instances during or after launch.
- The AMI that you use to launch an instance determines the type of root volume.
- You can launch an instance from either an Amazon EBS-backed AMI (Linux and Windows instances) or an instance store-backed AMI (Linux instances only).
- **Amazon EBS-backed AMI** – EBS-backed means the root volume is an EBS volume and storage is persistent.
- **Amazon instance store-backed AMI** – Instance store-backed means the root volume is an instance store volume and storage is not persistent.

2. Instance store volume

- An instance store provides temporary (non-persistent) block-level storage for your instance.
- This is different to EBS which provides persistent storage but is also a block storage service that can be a root or additional volume.
- Instance store storage is located on disks that are physically attached to the host computer.
- Instance store is ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content, or for data that is replicated across a fleet of instances, such as a load-balanced pool of web servers.
- You can specify instance store volumes for an instance only when you launch it.
- You can't detach an instance store volume from one instance and attach it to a different instance.
- The instance type determines the size of the instance store available, and the type of hardware used for the instance store volumes.
- Instance store volumes are included as part of the instance's usage cost.
- Some instance types use NVMe or SATA-based solid-state drives (SSD) to deliver high random I/O performance.
- This is a good option when you need storage with very low latency, but you don't need the data to persist when the instance terminates, or you can take advantage of fault-tolerant architectures.
- **Note:** Instance stores offer very high performance and low latency. If you can afford to lose an instance, i.e. you are replicating your data, these can be a good solution for high performance/low latency requirements. Look out for questions that mention distributed or replicated databases that need high I/O. Also, remember that the cost of instance stores is included in the instance charges so it can also be more cost-effective than EBS Provisioned IOPS.

3. EBS

- EBS is the Amazon Elastic Block Store.

- EBS volumes are network attached storage that can be attached to EC2 instances.
- EBS volume data persists independently of the life of the instance.
- EBS volumes do not need to be attached to an instance.
- You can attach multiple EBS volumes to an instance.
- You can attach an EBS volume to multiple instances with specific constraints.
- For most use cases where you need a shared volume across EC2 instances use Amazon EFS.
- EBS volume data is replicated across multiple servers in an AZ.
- EBS volumes must be in the same AZ as the instances they are attached to.
- EBS is designed for an annual failure rate of 0.1%-0.2% & an SLA of 99.95%.
- Termination protection is turned off by default and must be manually enabled (keeps the volume/data when the instance is terminated).
- Root EBS volumes are deleted on termination by default.
- Extra non-boot volumes are not deleted on termination by default.
- The behaviour can be changed by altering the "DeleteOnTermination" attribute.
- You can now create AMIs with encrypted root/boot volumes as well as data volumes.
- Volume sizes and types can be upgraded without downtime (except for magnetic standard).
- Volume can be increased and cannot be decreases of existing EBS Volume.
- To migrate volumes between AZ's create a snapshot then create a volume in another AZ from the snapshot (possible to change size and type).
- The root device is created under /dev/sda1 or /dev/xvda.
- Magnetic EBS is for workloads that need throughput rather than IOPS.
- Throughput optimized EBS volumes cannot be a boot volume.
- Each instance that you launch has an associated root device volume, either an Amazon EBS volume or an instance store volume.
- You can use block device mapping to specify additional EBS volumes or instance store volumes to attach to an instance when it's launched.
- You can also attach additional EBS volumes to a running instance.
- You cannot decrease an EBS volume size.
- When changing volumes the new volume must be at least the size of the current volume's snapshot.

EBS vs Instance Store

- On an EBS-backed instance, the default action is for the root EBS volume to be deleted upon termination.
- Instance store volumes are sometimes called Ephemeral storage (non-persistent).

- Instance store backed instances cannot be stopped. If the underlying host fails the data will be lost.
- Instance store volume root devices are created from AMI templates stored on S3.
- EBS backed instances can be stopped. You will not lose the data on this instance if it is stopped (persistent).
- EBS volumes can be detached and reattached to other EC2 instances.
- EBS volume root devices are launched from AMI's that are backed by EBS snapshots.
- Instance store volumes cannot be detached/reattached.
- When rebooting the instances for both types data will not be lost.
- By default, both root volumes will be deleted on termination unless you configured otherwise.

EBS Volume Types

1] SSD, General Purpose – gp2/gp3:

- **Volume size from 1 GiB to 16 TiB.**
- **Up to 16,000 IOPS per volume.**
- **Performance:**
  - 3 IOPS/GiB for gp2.
  - Up to 500 IOPS/GiB for gp3.
- **Can be a boot volume.**
- **EBS multi-attach not supported.**
- **Use cases:**
  - Low-latency interactive apps.
  - Development and test environments.

2] SSD, Provisioned IOPS – io1/io2:

- **More than 16,000 IOPS.**
- **Up to 64,000 IOPS per volume (Nitro instances).**
- **Up to 32,000 IOPS per volume for other instance types.**
- **Performance:**
  - Up to 50 IOPS/GiB for io1.
  - Up to 500 IOPS/Gib for io2.
- **Can be a boot volume.**

- **EBS multi-attach is supported – io2**
- **Use cases:**
  - Workloads that require sustained IOPS performance or more than 16,000 IOPS.
  - I/O-intensive database workloads.

3] HDD, Throughput Optimized – (st1):

- **Frequently accessed, throughput intensive workloads with large datasets and large I/O sizes, such as MapReduce, Kafka, log processing, data warehouse, and ETL workloads.**
- **Throughput measured in MiB/s and includes the ability to burst up to 250 MiB/s per TB, with a baseline throughput of 40 MB/s per TB and a maximum throughput of 500 MiB/s per volume.**
- **Cannot be a boot volume.**
- **EBS multi-attach not supported.**

4] HDD, Cold – (sc1):

- **Lowest cost storage – cannot be a boot volume.**
- **Less frequently accessed workloads with large, cold datasets.**
- **These volumes can burst up to 80 MiB/s per TiB, with a baseline throughput of 12 MiB/s.**
- **Cannot be a boot volume.**
- **EBS multi-attach not supported.**

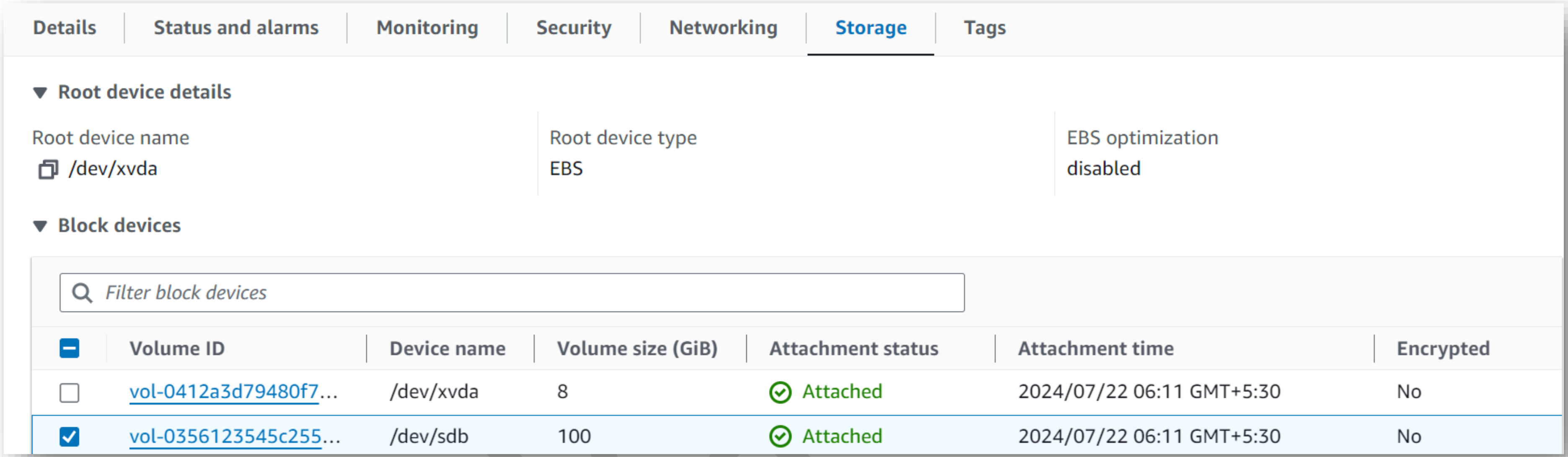
EBS Volumes Practical:

Attach EBS volume at the time of launch of Instance:

➔ Go to EC2 console and select Instances → Click on 'Launch Instances' → Add instance 'name', → Select AMI 'Amazon Linux 2 AMI (HVM) – Kernal 5.10' → Select instance type as 't2.micro' → Select 'Keypair' → In network settings select Security Group → In the Configure Storage settings click on 'Add new volume' option → Enter the size of volume you want → Select type of volume (eg. Gp3, gp2, io1, io2) → Click on 'Launch Instance'

Test: Once instance is launched you can verify attached volume by opening instance's properties and then check 'Storage Tab' to check all attached volumes to instance.

See below screenshot, You can also check 'Root device type' and Root device name as well from storage Tab of instance properties.



Root device details

Root device name: /dev/xvda

Root device type: EBS

EBS optimization: disabled

Block devices

Filter block devices

	Volume ID	Device name	Volume size (GiB)	Attachment status	Attachment time	Encrypted
☐	vol-0412a3d79480f7...	/dev/xvda	8	✓ Attached	2024/07/22 06:11 GMT+5:30	No
☑	vol-0356123545c255...	/dev/sdb	100	✓ Attached	2024/07/22 06:11 GMT+5:30	No

Steps to mount EBS volume to Linux:

First of all, lets create an EBS volume (100 GB): → Go to EC2 management console → Click on 'Volumes' sub-menu option from left sides Manu → Click on 'Create volume' → Select volume type → Specify the size of volume in GB's (eg- 100 GB) → Select 'Availability Zone' to create volume into (make sure to select same AZ as your ec2 instance is in) → click on 'Create Volume'

STEP 1: Connect to your instance using SSH.

STEP 2: Use the **`lsblk`** command to view your available disk devices and their mount points (if applicable) to help you determine the correct device name to use. The output of **`lsblk`** removes the /dev/ prefix from full device paths.

STEP 3: Check if file system is present on the volume. New volumes are raw block devices, and you must create a file system on them before you can mount and use them. Volumes that were created from snapshots likely have a file system on them already; if you create a new file system on top of an existing file system, the operation overwrites your data. Use the **`file -s`** command to get information about a specific device, such as its file system type. If the output shows simply data, as in the following example output, there is no file system on the device

Eg: No file system is present in below volume as output of command **`file -s`** is **data**

```
[ec2-user ~]$ sudo file -s /dev/xvdf
/dev/xvdf: data
```

If device has a file system then output will be like:

```
[ec2-user ~]$ sudo file -s /dev/xvda1
/dev/xvda1: SGI XFS filesystem data (blksz 4096, inosz 512, v2 dirs)
```

STEP 4: This step is **conditional,** If you discovered that there is a file system on the device in the 3rd step, skip this step.

Warning!!!
Do not use this command if you're mounting a volume that already has data on it (for example, a volume that was created from a snapshot). Otherwise, you'll format the volume and delete the existing data.

Use command to create a file system on volume- **`$ sudo mkfs -t xfs /dev/xvdf`**

STEP 5: You can create new directory using mkdir command or you can use existing directory to mount volume:
Command: in below command we are mounting volume to mnt directory:

```
$ sudo mount /dev/xvdf /mnt
```

Test: Now add some data into /mnt directory then detach the volume from instance and attach to another instance and mount volume and check if you can get same data for another instance.

Unmount Volume:

Simply use command **`umount`** and then specify mount point:

```
$ sudo umount /data
```

EC2 Backups & recovery

We have two ways to back up our EC2 instances:

1) AMI
2) Snapshot

1) AMI

- **Creates backup of instance and including the Volumes attached to instance.**
- In Amazon Web Services (AWS), an AMI (Amazon Machine Image) is a pre-configured virtual machine image used to create and launch instances within the Amazon EC2 (Elastic Compute Cloud) environment. It serves as a template for instances, containing all the information necessary to launch a virtual server, including the operating system, application server, and any additional software.
- AMI backups are essentially snapshots of these machine images, capturing the entire state of an EC2 instance at a specific point in time. These backups are useful for various purposes, such as disaster recovery, versioning, and scaling applications.
- When you create an AMI backup, you are essentially creating a copy of the root file system, along with any additional data volumes attached to the instance. This allows you to launch new instances with the same configuration and data as the original instance.

Benefits of AMI:

1. **Disaster Recovery:** AMI backups serve as a foundation for disaster recovery plans. In the event of an instance failure or unexpected data loss, you can quickly launch a new instance from a recent AMI, restoring your system to a known and working state.
2. **Versioning and Rollback:** When you make changes to your system, whether it's installing new software or updating configurations, creating a new AMI allows you to capture a specific version of your environment. If issues arise with the latest changes, you can easily roll back to a previous AMI to revert to a known-good state.
3. **Scalability:** AMI backups simplify the process of scaling your application. When you need to launch additional instances to handle increased traffic or demand, you can do so quickly and consistently by using a pre-configured AMI. This ensures that all new instances have the same configuration as the original.
4. **Consistent Development and Testing Environments:** AMI backups are useful in creating consistent development and testing environments. Development teams can use identical copies of production instances to test new features, updates, or changes in a controlled and isolated environment.

5. **Security and Compliance:** Regularly creating and updating AMI backups ensures that security patches and updates are included in the image. This helps in maintaining a secure and compliant infrastructure, as you can launch instances with the latest security configurations.
6. **Efficient Deployment:** AMIs enable you to quickly deploy instances with specific configurations, reducing the time required to set up new servers. This efficiency is crucial in dynamic environments where rapid scaling and deployment are necessary.
7. **Cost Optimization:** By having consistent and tested AMIs, you can optimize costs by only running instances when needed. You can launch instances from AMIs when there is demand and terminate them when the demand decreases, minimizing unnecessary expenses.
8. **Customization and Configuration Management:** AMIs allow you to capture a specific state of an instance, including the operating system, applications, and configurations. This makes it easy to replicate and deploy the same environment across multiple instances, ensuring consistency in your infrastructure.

In summary, AMI backups are a crucial component of AWS infrastructure management, providing a reliable and efficient way to capture, store, and replicate instances for various purposes, ultimately contributing to the overall stability, security, and scalability of your applications.

An AMI includes the following:

- A template for the root volume for the instance (for example, an operating system, an application server, and applications).
- Launch permissions that control which AWS accounts can use the AMI to launch instances.
- A block device mapping that specifies the volumes to attach to the instance when it's launched.

Copying AMIs:

- You can copy an Amazon Machine Image (AMI) within or across an AWS region using the AWS Management Console, the AWS Command Line Interface or SDKs, or the Amazon EC2 API, all of which support the Copy Image action.
- You can copy both Amazon EBS-backed AMIs and instance store-backed AMIs.
- You can copy encrypted AMIs and AMIs with encrypted snapshots.

AMI Practical Steps:

Create AMI:

➔ Select Ec2 instance → Go to 'Actions' → Click on 'Image & templates' → click on 'Create image' → add Image name (eg: test-image) → add image description (optional) → select *'No reboot' option if you want [here if you not select 'No reboot' option then by default AWS will attempt to shut down and reboot the instance before creating the image. This ensures file system integrity on the created image] → Add the volume → check mark or uncheck 'Delete on termination' option [If you select 'Delete on termination' volume will be deleted when you terminate ec2 instance → Click on 'Create image' → check Status of AMI in AMIs menu.

Create Instance from AMI:

➔ Select AMI → click on 'Launch instance from AMI' → you can see same page opens when you create a new instance → just check all details or change anything if you want to and then click on 'Launch Instance'

Copy AMI to another region:

➔ Select AMI → Click on 'Actions' → give AMI Copy name & description → Select 'Destination region' where you want to copy AMI → click on 'Copy AMI'

Delete AMIs:

➔ Select AMI → click on 'Actions' → click on Deregister AMI

2) Snapshot:

- Snapshots capture a point-in-time state of a volume and instance
- Cost-effective and easy backup strategy.
- Share data sets with other users or accounts.
- Can be used to migrate a system to a new AZ or region.
- Snapshots are stored on Amazon S3 from backend but you won't be able to see actual bucket and data in s3.
- If you make periodic snapshots of a volume, the snapshots are incremental, which means that only the blocks on the device that have changed after your last snapshot are saved in the new snapshot.
- Snapshots can only be accessed through the EC2 APIs.
- EBS volumes are AZ specific, but snapshots are region specific, you can choose in which AZ you have to create volume from snapshot.
- Volumes can be created from EBS snapshots that are the same size or larger.
- Snapshots can be taken of non-root EBS volumes while running.
- You are charged for data traffic to S3 and storage costs on S3.
- You are billed only for the changed blocks.
- You can resize volumes through restoring snapshots with different sizes (configured when taking the snapshot).
- Snapshots can be copied between regions (and be encrypted). Images are then created from the snapshot in the other region which creates an AMI that can be used to boot an instance.

- You can create volumes from snapshots and choose the availability zone within the region.

Snapshot Practical steps:

Create snapshot from volume:

➔ Click on 'Snapshot' from left sides menu → Click on 'Create snapshot' → Select resource type 'Volume' or 'Instance' → if volume is selected then select 'Volume ID' or if you have selected Instance then select 'Instance ID' → add description for snapshot → click on 'Create snapshot' →

*You can create volume or image from a snapshot. Image can be used to launch an ec2 instance then.

Create volume from snapshot:

➔ Select snapshot → Click on 'Actions' → then select 'Create volume from snapshot' option → Select 'Volume type' → Select 'Availability zone' where you want to create Volume → Click on 'Create Volume'

Create Image from snapshot:

➔ Select snapshot → Click on 'Actions' → then select 'Create image from snapshot' option → Add 'Image name' & description of image → Click on 'Create image'

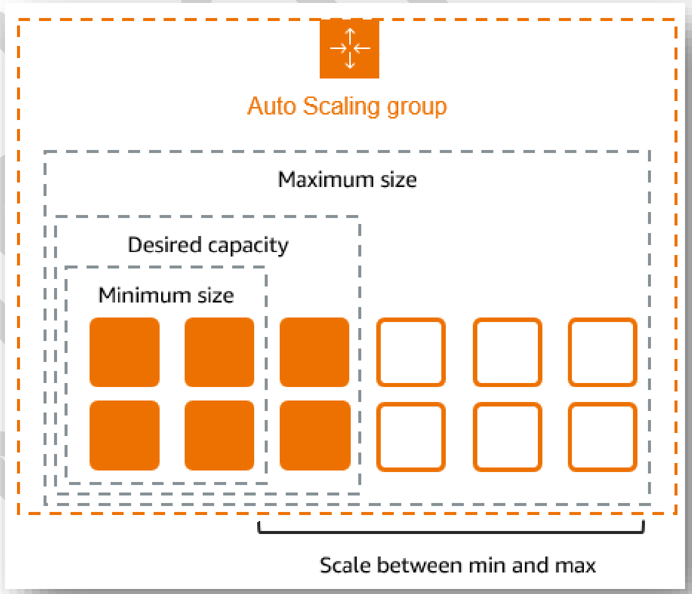
Copy snapshot into another region:

➔ Select snapshot → Go to 'Actions' → Click on 'Copy snapshot' → Add 'Description' → Select 'Destination Region' → click on 'Copy snapshot'

Snapshots	AMI
It is used as a backup of a single EBS volume attached to the EC2 instance	It is used as a backup of an EC2 instance.
Select this option when the instance contains multiple EBS volumes	This is widely used to replace a failed EC2 instance.
Here, pay only for the storage of the modified data	Here, pay only for the storage that you use.
It is a non-bootable image on EBS volume	It is a bootable image on an EC2 instance.

Autoscaling:

- Amazon EC2 Auto Scaling helps you ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application. You create collections of EC2 instances, called *Auto Scaling groups*. You can specify the minimum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes below this size. You can specify the maximum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes above this size. If you specify the desired capacity, either when you create the group or at any time thereafter, Amazon EC2 Auto Scaling ensures that your group has this many instances. If you specify scaling policies, then Amazon EC2 Auto Scaling can launch or terminate instances as demand on your application increases or decreases.
- For example, the following Auto Scaling group has a minimum size of four instances, a desired capacity of six instances, and a maximum size of twelve instances. The scaling policies that you define adjust the number of instances, within your minimum and maximum number of instances, based on the criteria that you specify.



- AWS Auto Scaling lets you build scaling plans that automate how groups of different resources respond to changes in demand. You can optimize availability, costs, or a balance of both. AWS Auto Scaling automatically creates all of the scaling policies and sets targets for you based on your preference. AWS Auto Scaling monitors your application and automatically adds or removes capacity from your resource groups in real-time as demands change.
- One of the primary reasons companies move to the cloud has been the facility to scale up based on client-requirement and scale back when that requirement has been met. With the help of AWS autoscaling, every individual can not only maintain application performance in a single unified interface but can also maintain those applications at the lowest possible price.
- AWS Auto Scaling is a service that helps the user to monitor applications and automatically adjusts the capacity to maintain steady, predictable performance at the lowest possible cost.

Benefits of Auto Scaling:

Auto Scaling your application leads to the following benefits:

- *Better fault tolerance.*
- *High availability of resources.*
- *Better cost management.*
- *High reliability of resources.*
- *The high flexibility of resources.*

Features of Auto Scaling:

Auto Scaling provides the following features:

1. Scaling: Automatically adds or removes instances based on demand.
2. Group Management: Manages a group of instances as a single unit.
3. Scaling Policies: Defines rules for scaling, such as when to scale out or in.
4. Scheduled Scaling: Scales based on a schedule, e.g., increasing instances during peak hours.
5. Dynamic Scaling: Scales based on real-time metrics, such as CPU utilization or request latency.
6. Predictive Scaling: Uses machine learning to predict future demand and scale accordingly.
7**. Load Balancing:** Distributes traffic across instances to ensure high availability.
8. Health Checks: Monitors instance health and replaces unhealthy instances.

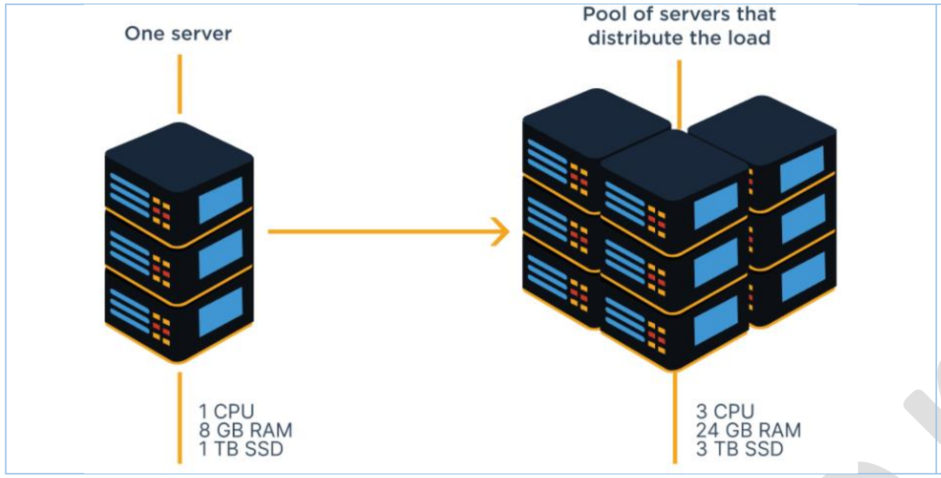
9. Notification: Sends alerts when scaling events occur.
10. Integration: Works with other AWS services, such as CloudWatch, ELB, and EC2.
11. Customization: Allows customization of scaling settings and policies.
12. Cost Optimization: Helps reduce costs by scaling down during low demand.
13. High Availability: Ensures high availability by maintaining a minimum number of instances.
14. Flexibility: Supports various scaling strategies, such as step scaling, target tracking, and scheduled scaling.
15. Monitoring: Provides detailed monitoring and logging of scaling activities.

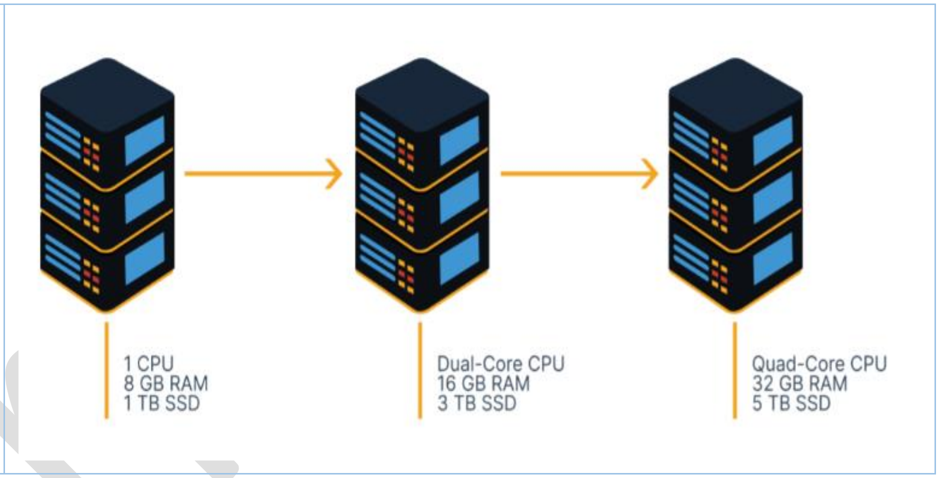
Scale up/Scale out vs Scale down/Scale in:

Scale Up or Scale Out	Scale Down or Scale In
Increases the number of instances as demand increases	**Decreases the number of instances as demand decreases**

Horizonal vs Vertical Scaling:

Horizontal Scaling	Vertical Scaling
Adds or removes instances to handle changes in demand.	Increases or decreases the power of single instance (e.g., CPU, RAM).





Important Points to remember:

Auto Scaling is a region-specific service.
Auto Scaling can span multiple AZs within the same AWS region.
Auto Scaling can be configured from the Console, CLI, SDKs and APIs.
There is no additional cost for Auto Scaling, you just pay for the resources (EC2 instances) provisioned.
Auto Scaling works with ELB, CloudWatch and CloudTrail.
You can determine which subnets Auto Scaling will launch new instances into.
Auto Scaling will try to distribute EC2 instances evenly across AZs.
Launch template used to create new EC2 instances and includes parameters such as instance family, instance type, AMI, key pair, and security groups.
You can create multiple launch template for different versions of the Application.
An ASG can be edited once defined.
You can attach one or more classic ELBs to your existing ASG.
You can attach one or more Target Groups to your ASG to include instances behind an ALB.
The ELBs must be in the same region.
Once you do this any EC2 instance existing or added by the ASG will be automatically registered with the ASG defined ELBs.
If adding an instance to an ASG would result in exceeding the maximum capacity of the ASG the request will fail.
You can add a running instance to an ASG if the following conditions are met:

- *The instance is in a running state.*

- *The AMI used to launch the instance still exists.*
- *The instance is not part of another ASG.*
- *The instance is in the same AZs for the ASG.*

You can also detach an instance from autoscaling group as well.

Launch Template:

A Launch Template in an Autoscaling group is a template that defines the configuration for launching new instances in the group. It provides a way to specify the instance type, AMI, security groups, and other settings that are used when launching new instances.

A Launch Template typically includes the following information:

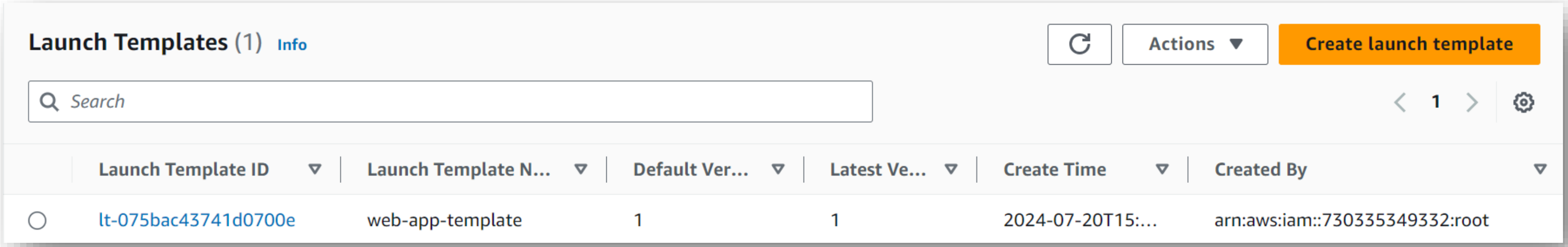
1. Instance type: The type of instance to launch (e.g., t2.micro, c5.large, etc.)
2. AMI: The ID of the Amazon Machine Image (AMI) to use for the instance
3. Security groups: The security groups to associate with the instance
4. Key pair: The key pair to use for SSH access to the instance
5. User data: Optional user data to pass to the instance during launch
6. Instance market: The market type (e.g., spot, on-demand) and optional spot price
7. Launch template version: The version of the launch template to use

By using a Launch Template, you can:

- Simplify the process of launching new instances in your AutoScaling group
- Ensure consistency in the configuration of new instances
- Easily update the configuration of new instances by updating the Launch Template

When you create an AutoScaling group, you can specify a Launch Template to use for launching new instances. You can also override the Launch Template settings when creating an AutoScaling group.
Note that Launch Templates are a separate resource from AutoScaling groups, and you can create and manage them independently. To create an autoscaling group you need to create a launch template first.

- Templates can be created from the AWS console or CLI.
- You cannot make changes to launch configuration once it is created, you need to create new launch template with new version with modified changes.
- You can use a launch templated with multiple Auto Scaling Groups (ASG).



	Launch Template ID	Launch Template N...	Default Ver...	Latest Ve...	Create Time	Created By
○	lt-075bac43741d0700e	web-app-template	1	1	2024-07-20T15:...	arn:aws:iam::730335349332:root

IMP: Terminologies in AUTOSCALING:

1] **AutoScaling Group (ASG):** A collection of instances that are managed by Autoscaling.
2] **Launch Template:** A template that defines the configuration for launching new instances in an ASG.
3] **Scaling Policy:** A policy that defines how Autoscaling should scale the ASG based on certain metrics or conditions.
4] **Scaling Plan:** A plan that defines the scaling strategy for an ASG.
5] **Instance Type:** The type of instance to launch in an ASG (e.g., t2.micro, c5.large, etc.).
6] **Desired Capacity:** refers to the desired number of instances that you want to run in your Auto Scaling group. It's a key concept in Auto Scaling, as it determines the number of instances that Auto Scaling will aim to maintain in the group.
7] **MinSize:** The minimum number of instances, ASG scaling will never go down below this capacity no matter of what.
8] **MaxSize:** The maximum number of instances in an ASG, ASG will never scale out above this capacity in any condition.
9] **Capacity:** The current number of instances in an ASG.
10] **Scaling Activity:** An event that occurs when Autoscaling launches or terminates instances in an ASG.
11] **Cooldown Period:** A period of time after a scaling activity during which Autoscaling will not launch or terminate instances.
12] **Health Check:** A check that determines the health of an instance in an ASG.
13] **Standby State:** A state where instances are not serving traffic but are still running.
14] **Protected State:** A state where instances are protected from termination by Autoscaling.
15] **Scheduled Scaling:** Scaling based on a schedule.
16] **Dynamic Scaling:** Scaling based on metrics or conditions.
17] **Predictive Scaling:** Scaling based on predictive analytics.
18] **Load Balancer:** A service that distributes traffic to instances in an ASG.

19] **Target Group:** A group of instances that receive traffic from a load balancer.
20] **AutoScaling Lifecycle Hook:** A hook that allows you to run scripts or commands during instance launch or termination.

Autoscaling policies: IMP

1] Target tracking policy:

- Automatically adjust the number of instances (servers) in a group based on a target value for a specific metric (like CPU usage or response time).
- In simple words:
  - *You set a target value for a metric (e.g., CPU usage = 50%)*
  - *Autoscaling monitors the metric and adjusts the instance count to achieve the target value*
  - *If the metric goes above the target, Autoscaling adds more instances*
  - *If the metric goes below the target, Autoscaling removes instances*
- This policy helps maintain a consistent level of performance and efficiency for your application, without manual intervention.

2] Simple Scaling policy:

- Simple scaling relies on a metric as a basis for scaling.
- It does not provide any fine-grained control to scaling in and scaling out.
- It is based on a single scaling adjustment, with a cooldown period between each scaling activity.

3] Step Scaling policy:

- Step scaling applies “step adjustments,” which means you can set multiple actions to vary the scaling depending on the size of the alarm breach.
- It provides more fine-grained control over the scaling process.
- It can continue to respond to additional alarms even in the middle of the scaling event.

Scaling Adjustment Types:

- Step scaling and simple scaling support the following adjustment types:
  - **ChangeInCapacity:** Increment or decrement the current capacity of the group by the specified value.
  - **ExactCapacity:** Change the current capacity of the group to the specified value.
  - **PercentChangeInCapacity:** Increment or decrement the current capacity of the group by the specified percentage.

Use Cases:

- Simple scaling is suitable for applications with simple scaling needs.

- Step scaling is suitable for applications that require more fine-grained control over scaling and have multiple scaling needs.

AWS AUTO SCALING SUMMARY

Creating group of EC2 instances that scale up or down depending on the conditions you set.

- Enable elasticity by scaling horizontally through adding or terminating EC2 instances.
- Auto scaling ensures that you have the right number of AWS EC2 instances for your needs at all time.
- Auto scaling helps you to save cost by cutting down the number of EC2 instances when not needed and scaling out to add more instances only it is required.

Auto Scaling Components:

1. Launch Configuration : like instance type, AMI, key-pair, security group
2. Auto Scaling Group: group name, group size, VPC, subnet, health check period
3. Scaling Policy : metric type, target value

How to Balance, Attach and Detach EC2 Instances:

Balance:

- If auto scaling finds that the number of EC2 instances launched by ASG into subjects AZs is not balanced (EC2 instances are not evenly distributed), auto scaling do rebalancing activity by itself.
- AS always tries to balance the instances distribution across AZs.
- While rebalancing, ASG launches new EC2 instances where there are less EC2 at present and then terminates the instances from the AZs that had more instances.

What causes imbalance of EC2?

- If we add or remove same subnets/AZ form auto scaling.
- If we manually request for EC2 termination from our ASG.
- An AZ that did not have enough EC2 capacity now has enough capacity and it is one of the auto scaling group.

Attach:

- We can attach a running EC2 instance to an ASG by using AWS console or CLI if the below conditions are meet:
- Instances must be on running state.
- AMI used to launch the EC2 still exists.
- Instances is not the part of another auto scaling group.

- Instances must be in the same AZ of the same group.

Detach:

- If the existing EC2 instances under the ASG, plus the one to be needed, exceeds the maximum capacity of the ASG, the request will fail, EC2 instance would not be added.
- You can manually remove EC2 instances from an ASG using AWS console of CLI.
- You can then manage the detached instances independently or attach it to another ASG.
- When you detach an instance you have the option to decrement the ASG desired capacity.
- If you do not, ASG will launch another instance to replace the one detached.
- When you delete an ASG its parameter like maximum, minimum and desired capacity are all set to zero. Hence it terminates it's all EC2 instances.
- If you want to keep the EC2 instances and manage them independently you can manually detach them first then delete ASG.

We can attach one more Elastic Load Balancer (ELB) to our ASG.

- The ELB must be in the same region as the ASG.
- Once you do this any EC2 instance existing or added by ASG will be automatically registered with the ASG defined ELB.
- Instances and the ELB must be in the same VPC.
- Auto scaling classifies its EC2 instance health check or unhealthy.
- By default, as uses EC2 status checks only to determine the health status of an instance.
- When you have one or more ELB defined with the ASG you can configure auto scaling to use both the EC2 status check and SLB health check to determine the instances health check.
- Health check grace period is 300sec by default.
- If we set zero in grace period the instance health is checked once it is in service.
- Until the grace period timer expires any unhealthy status reported by EC22 status check of the ELB attached to the ASG will not be acted upon.
- After grace period expires ASG consider an instance unhealthy in any of the following cases:
- EC2 status check report to ASG an instance other than running.
- If ELB health check are configured to be used by the auto scaling then if the ELB report the instance as 'out of service'.
- Unlike AZ rebalancing, termination of unhealthy instances happen first then auto scaling attempt to launch new instance to replace the ones terminated. Elastic IP and EBS volumes gets detached from the terminated instances you need to manually attach there to the new instance.

Types of Auto Scaling Policies:
In four situations, ASG sends a SNS email notification:
i. an instance is launched

ii. an instance is terminated
iii. an instance fails to launch
iv. an instance fails to terminate

StandBy State:

- You can manually move an EC2 instance form an ASG and put it in standby state.
- Instances in standby state are still managed by auto scaling.
- Instances in standby state are charged as normal in-service instances.
- They do not count towards available EC2 instances for workload/app use.
- Auto scaling does not perform health check on instance in standby state.

Scaling Policies: Generally scaling policy is of two types such as:
1. Manual
2. Dynamic

Dynamic policy is divided into three categories as follows:
A. Target Tracking
B. Simple Scaling Policy
C. Step Scaling Policy

Define how much you want to scale based on defined conditions.
ASG uses alarms and policies to determine scaling.
For Simple or Step scaling a scaling adjustment can't change the capacity of the group above the maximum group or below the minimum group.

Predictive/Scheduled/Cycle Scaling: it looks at historic pattern and forecast them into the future to schedule change in the number of EC2 instances. It uses machine learning model to forecast daily and weekly pattern.

Target Tracking Policies: increase or decrease the current capacity of the group based on a target value for specific metric. This is similar to the way that your thermostatic maintain the temperature of your home.

Step Scaling: increase or decrease the current capacity of the group based on a set of scaling adjustment known as step adjustment that vary based on the size of the alarm breach. It does not support/ wait for cool down times. It supports warm-up timer: time taken by newly launched instance to be ready and contribute to the watched metric.

Simple Scaling: single adjustment (up or down) in response to an alarm (cool down timer- 300sec by default)

Schedule Scaling: used for predictable load change. You need to configure a schedule action for a scale out at a specific date/time and to a required capacity. A scheduled action must have a unique data/time. You cannot configure two schedule activities at the same date/time.

Autoscaling Practical's:

Create launch template:

➔ Go to EC2 console → Click on 'Launch Templates' option from left sides Manu → click on 'Create launch template' → add 'Launch template name' & 'Template version description' → Select 'AMI' → Select 'Instance type' → Select 'Key pair name' → then select 'subnet' & 'security group' → click on 'Advanced details' → add user data script to install httpd and deploy our web application → Click on **'Create launch template'**

Create Autoscaling Group:

➔ Go to EC2 console → Click on 'Auto Scaling Groups' Manu option from the left side Manu → click on 'Create Auto Scaling Group' → Add 'Auto Scaling group name' of your choice → Select 'Launch template' that you have created or else click on 'Create a launch template' to create new one → select version of application if you have multiple versions → click on 'Next' → in Network select 'VPC' & 'Subnet' to launch your instances in → Click 'Next' → Select 'No load balancer' → click on 'Next' → Now enter 'Desired capacity' value → Also mention 'Min desired capacity' & 'Max desired capacity' for scaling → Click on Next-Next until you get 'Review' page → now review all autoscaling configuration → click **'Create Auto Scaling Group'**

Testing- Once autoscaling group is created go to instances and check if new instances are being launched, Now terminate one/two instances manually, see if new instances are automatically being launched or not. For every terminated instance there will be new instance being created by auto scaling group to replace the terminated instance. Hit public IP of newly created instance you can see newly created instances has all the settings and properties like the instances terminated by you.

Create Auto Scaling group with Target Tracking Policy:

➔ Go to EC2 console → Click on 'Auto Scaling Groups' Manu option from the left side Manu → click on 'Create Auto Scaling Group' → Add 'Auto Scaling group name' of your choice → Select 'Launch template' that you have created or else click on 'Create a launch template' to create new one → select version of application if you have multiple versions → click on 'Next' → in Network select 'VPC' & 'Subnet' to launch your instances in → Click 'Next' → Select 'No load balancer' → click on 'Next' → Now enter 'Desired capacity' value - 3 → Also mention 'Min desired capacity' value – 1 &

'Max desired capacity' value as – 5 for scaling → Select 'Target tracking scaling policy' option → Add 'Scaling policy name' → For Metric type select 'Average CPU Utilization' Enter target value as 70 and Instance warmup time as 300 seconds → Click Next-Next until you get 'Review' page → now review all autoscaling configuration → click **'Create Auto Scaling Group'**

Testing- To check target traffic policy with Average CPU utilization matric we need to install stress tool on Linux to increase the CPU utilization, so SSH on Linux and enter below command to install stress tool on system-

```
$ sudo amazon-linux-extras install epel -y
$ sudo yum install stress -y
$ stress –cpu 10
```
---> This command is used to stress the CPU by spawning 10 workers that continuously perform calculations.

Wait for the CPU utilization to go above 70%, you can check in instance monitor graph for CPU Utilization, also turn on 'detailed monitoring' option to get 1 minute's monitoring data.
Once the CPU Utilization goes above 70% then wait for 5 – 6 minutes to check if ASG launces a new instance automatically if not then confirm all the steps are performed correctly.

Detach existing ec2 instance to an Auto Scaling group:

You can detach instances from your Auto Scaling group. After an instance is detached, that instance becomes independent and can either be managed on its own or attached to a different Auto Scaling group, separate from the original group it belonged to. This can be useful, for example, when you want to perform testing using existing instances that are already running your application.
IMP - You can detach an instance only when it's in the InService state.

STEPS ➔ Select and open auto scaling group → Click on 'Instance Management' → Select the instance you want to detach from ASG → Click on 'Actions' → now click on 'Detach' → Check mark 'Replace Instance' option if you want replace the detached instance in ASG → Type *detach* and click on 'Detach Instance'

TEST: selected instance is now free from ASG group and can be controlled manually, also confirm new instance must be created in place of the detached instance

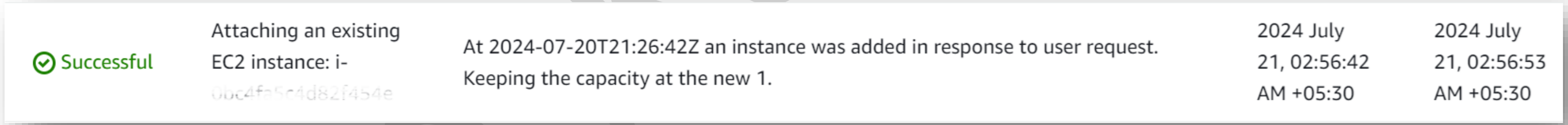
Attach an instance into Auto Scaling Group:

Amazon EC2 Auto Scaling treats attached instances the same as instances launched by the group itself. This means that attached instances can be terminated during scale-in events if they're selected. When you attach instances, the desired capacity of the group increases by the number of instances being attached.

IMP: For an instance to be attached, it must meet the following criteria:
The instance is in the running state with Amazon EC2.
The AMI used to launch the instance must still exist.
The instance is not a member of another Auto Scaling group.
The instance is launched into one of the Availability Zones defined in the Auto Scaling group.
If the Auto Scaling group has an attached load balancer target group or Classic Load Balancer, the instance and the load balancer must both be in the same VPC.

STEPS → Open EC2 console and go to instances → Select an instance that you want to add to ASG → go to 'Actions' → click on option 'Attach to Auto Scaling Group' → Select an 'Auto Scaling Group' to which you want to attach instance to → Finally click on 'Attach'

TEST: Go to auto scaling group and check 'Instance Management' tab you can see attached instance is listed here or not. You can also check 'Activity' Tab to confirm if instance attach activity is successful or not, for example see below screenshot-



✓ Successful	Attaching an existing EC2 instance: i-0bc4fa5c4d82f454e	At 2024-07-20T21:26:42Z an instance was added in response to user request. Keeping the capacity at the new 1.	2024 July 21, 02:56:42 AM +05:30	2024 July 21, 02:56:53 AM +05:30

Elastic Load balancers (ELB):

- Elastic Load Balancing automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. It monitors the health of its registered targets, and routes traffic only to the healthy targets. Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.
- Network traffic can be distributed across a single or multiple Availability Zones (AZs) within an AWS Region.
- Elastic Load Balancing provides fault tolerance for applications by automatically balancing traffic across targets.
- Targets can be Amazon EC2 instances, containers, IP addresses, and Lambda functions.
- ELB distributes traffic across Availability Zones while ensuring only healthy targets receive traffic.
- Only 1 subnet per AZ can be enabled for each ELB.
- ELBs can be **Internet facing** or **internal-only:**
  - **Internet facing ELB:**
    - ELB nodes have public IPs.
    - Routes traffic to the private IP addresses of the EC2 instances.
    - Need one public subnet in each AZ where the ELB is defined.
  - **Internal only ELB:**
    - ELB nodes have private IPs.
    - Routes traffic to the private IP addresses of the EC2 instances.
    - Internal-only load balancers do not need an Internet gateway.

types of Elastic Load Balancer (ELB) on AWS:

1. **Classic Load Balancer (CLB)** – this is the oldest of the three and provides basic load balancing at both layer 4 and layer 7.
2. **Application Load Balancer (ALB)** – layer 7 load balancer that routes connections based on the content of the request.
3. **Network Load Balancer (NLB)** – layer 4 load balancer that routes connections based on IP protocol data.
4. **Gateway Load Balancer (GLB)** – layer 3/4 load balancer used in front of virtual appliances such as firewalls and IDS/IPS systems.

Note: *The CLB is not covered in detail on this page as it is on old generation load balancer and is no longer featured on most AWS interviews.*

IMP: Listeners in ELB:

An ELB listener is the process that checks for connection requests:

- CLB listeners support the TCP and HTTP/HTTPS protocols.
- ALB listeners support the HTTP and HTTPS protocols.
- NLB listeners support the TCP, UDP and TLS protocols.
- GLB listeners support the IP protocol.

IMP Points:

- Deleting an ELB does not affect the instances registered against it (they won't be deleted; they just won't receive any more requests).
- For ALB at least 2 subnets must be specified.
- For NLB only one subnet must be specified (recommended to add at least 2).
- ELB uses a DNS record TTL of 60 seconds to ensure new ELB node IP addresses are used to service clients.
- By default, the ELB has an idle connection timeout of 60 seconds, set the idle timeout for applications to at least 60 seconds.

ELB Security Groups

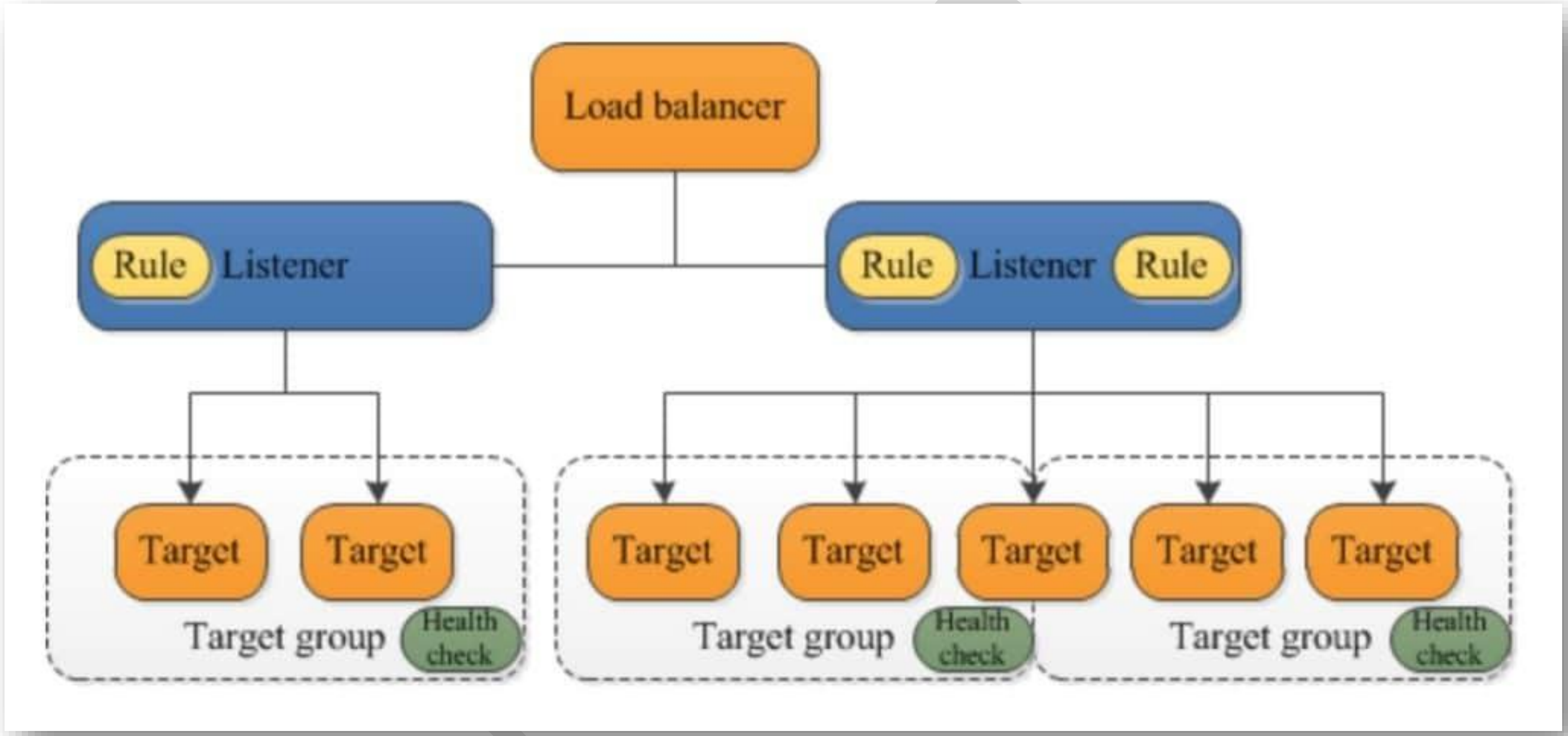
Security groups control the ports and protocols that can reach the front-end listener.
In non-default VPCs you can choose which security group to assign.
You must assign a security group for the ports and protocols on the front-end listener.
You need to also allow the ports and protocols for the health check ports and back-end listeners.

Target groups

Target groups are used for registering instances against an ALB, NLB, or GLB.

- Target groups are a logical grouping of targets and are used with ALB, NLB, and GLB.
- Targets are the endpoints and can be EC2 instances, ECS containers, IP addresses, Lambda functions, and other load balancers.
- Target groups can exist independently from the ELB.
- A single target can be in multiple target groups.
- Only one protocol and one port can be defined per target group.
- You cannot use public IP addresses as targets.
- You cannot use instance IDs and IP address targets within the same target group.

- A target group can only be associated with one load balancer.
- The following diagram illustrates the basic components. Notice that each listener contains a default rule, and one listener contains another rule that routes requests to a different target group. One target is registered with two target groups.



Health checks in Target Groups:

- We can use the default path of "/" to perform health checks on the root, or specify a custom path of our application for health check

- The load balancer periodically sends requests, per specified settings to the registered targets to test their status.
- Health checks are defined per target group.

You can only use Auto Scaling with the load balancer if using instance IDs in your target group.
ALB/NLB/GLB can route to multiple target groups.
You can register the same EC2 instance or IP address with the same target group multiple times using different ports (used for routing requests to microservices).
If you register by instance ID the traffic is routed using the primary private IP address of the primary network interface.
If you register by IP address you can route traffic to an instance using any private address from one or more network interfaces.
You cannot mix different types within a target group (EC2, ECS, IP, Lambda function).

Let's see load balancer types in detail,

1] Application Load Balancer (ALB)

- The Application Load Balancer works at the OSI Layer 7 (Application layer), routing traffic to targets – EC2 instances, containers and IP addresses based on the content of the request.
- Distributes traffic to multiple targets (e.g., EC2 instances, containers) based on-
  - Request content (e.g., URL, headers)
  - Application-specific logic
- Operates at Layer 7 (application layer) of the OSI model
- Supports advanced features like path-based routing, host-based routing, and query string-based routing
- Typically used for web applications, APIs, and microservices
- The ALB supports content-based routing which allows the routing of requests to a service based on the content of the request. For example:
  - **Host-based routing** routes client requests based on the Host field of the HTTP header allowing you to route to multiple domains from the same load balancer.
  - **Path-based routing** routes a client request based on the URL path of the HTTP header (e.g. /images or /orders).

Health Checks:

- Can have custom response codes in health checks (200-399).
- Details provided in the API and management console for health check failures.

- Reason codes are returned with failed health checks.

Deregistration delay is like connection draining.

Sticky Sessions:

- Uses cookies to ensure a client is bound to an individual back-end instance for the duration of the cookie lifetime.
- Sticky sessions are enabled at the target group level.
- WebSockets connections are inherently sticky (following the upgrade process).

Listeners and Rules

Listeners:

- Each ALB needs at least one listener and can have up to 10.
- Listeners define the port and protocol to listen on.
- Can add one or more listeners.
- Cannot have the same port in multiple listeners.

Listener rules:

- Rules determine how the load balancer routes requests to the targets in one or more target groups.
- Each rule consists of a priority, one or more actions, an optional host condition, and an optional path condition.
- Only one action can be configured per rule.
- One or more rules are required.
- Each listener has a default rule, and you can optionally define additional rules.
- Rules determine what action is taken when the rule matches the client request.
- Rules are defined on listeners.
- You can add rules that specify different target groups based on the content of the request (content-based routing).
- If no rules are found the default rule will be followed which directs traffic to the default target groups.

Default rules:

- When you create a listener you define an action for the default rule.
- Default rules cannot have conditions.
- You can delete the non-default rules for a listener at any time.
- You cannot delete the default rule for a listener.

- When you delete a listener all its rules are deleted.
- If no conditions for any of a listener's rules are met, the action for the default rule is taken.

Rule priority:

- Each rule has a priority, and they are evaluated in order of lowest to highest.
- The default rule is evaluated last.
- You can change the value of a non-default rule at any time.
- You cannot change the value of the default rule.

Rule action:

- Only one target group per action.
- Each rule has a type and a target group.
- The only supported action type is forward, which forwards requests to the target group.
- You can change the target group for a rule at any time.

Rule conditions:

- There are two types of rule condition: host and path.
- When the conditions for a rule are met the action is taken.
- Each rule can have up to 2 conditions, 1 path condition and 1 host condition.
- Optional condition is the path pattern you want the ALB to evaluate for it to route requests.

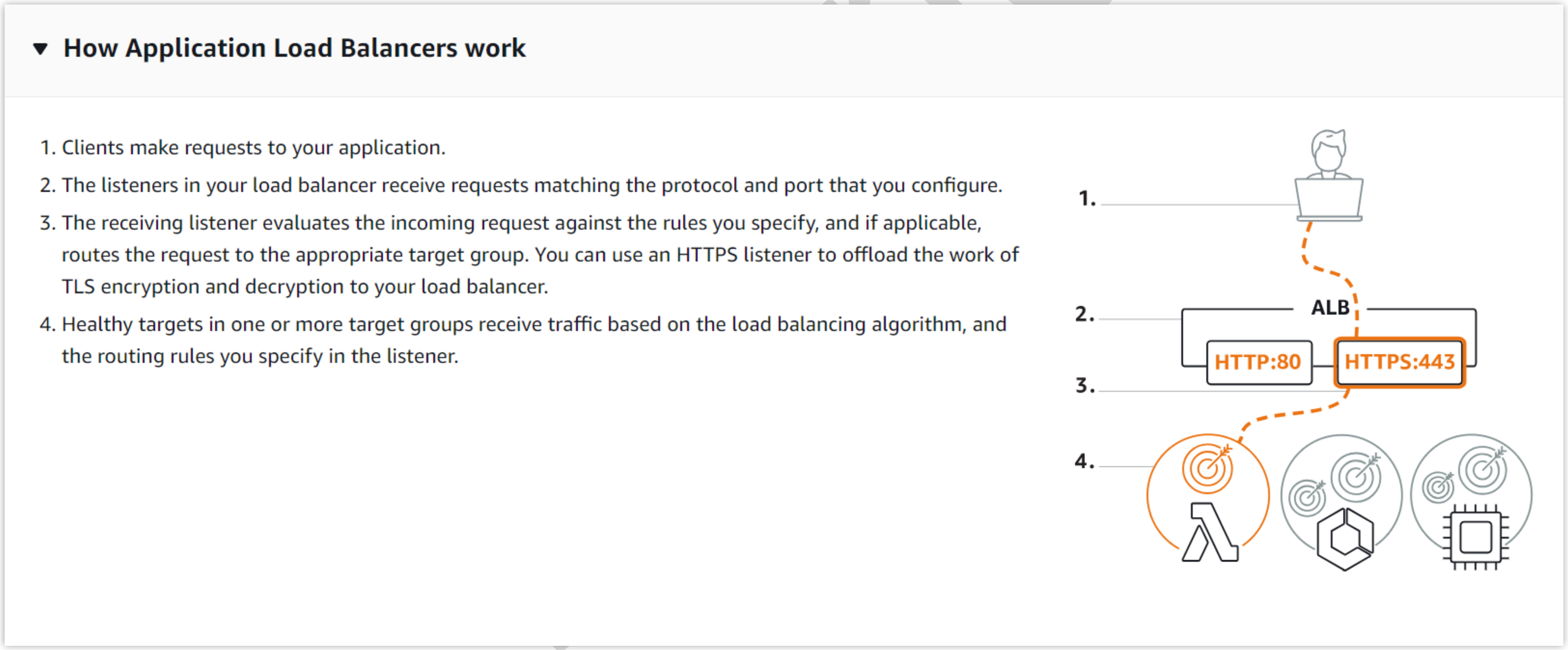
Request routing:

- After the load balancer receives a request it evaluates the listener rules in priority order to determine which rule to apply, and then selects a target from the target group for the rule action using the round robin routing algorithm.
- Routing is performed independently for each target group even when a target is registered with multiple target groups.
- You can configure listener rules to route requests to different target groups based on the content of the application traffic.

Content-based routing:

- ALB can route requests based on the content of the request in the host field: **host-based** or **path-based.**
- **Host-based**:
    - It's a domain name-based routing.
    - e.g. example.com or app1.example.com.

  - The host field contains the domain name and optionally the port number.
- **Path-based**
  - It’s URL based routing
  - e.g. example.com/images, example.com/app1.
- You can also create rules that combine host-based and path-based routing.
- Anything that doesn’t match content routing rules will be sent to a default target group.
- The ALB can also route based on other information in the HTTP header including query string parameters, request method, and source IP addresses.



▼ **How Application Load Balancers work**

1. Clients make requests to your application.
2. The listeners in your load balancer receive requests matching the protocol and port that you configure.
3. The receiving listener evaluates the incoming request against the rules you specify, and if applicable, routes the request to the appropriate target group. You can use an HTTPS listener to offload the work of TLS encryption and decryption to your load balancer.
4. Healthy targets in one or more target groups receive traffic based on the load balancing algorithm, and the routing rules you specify in the listener.

2] Network Load Balancer

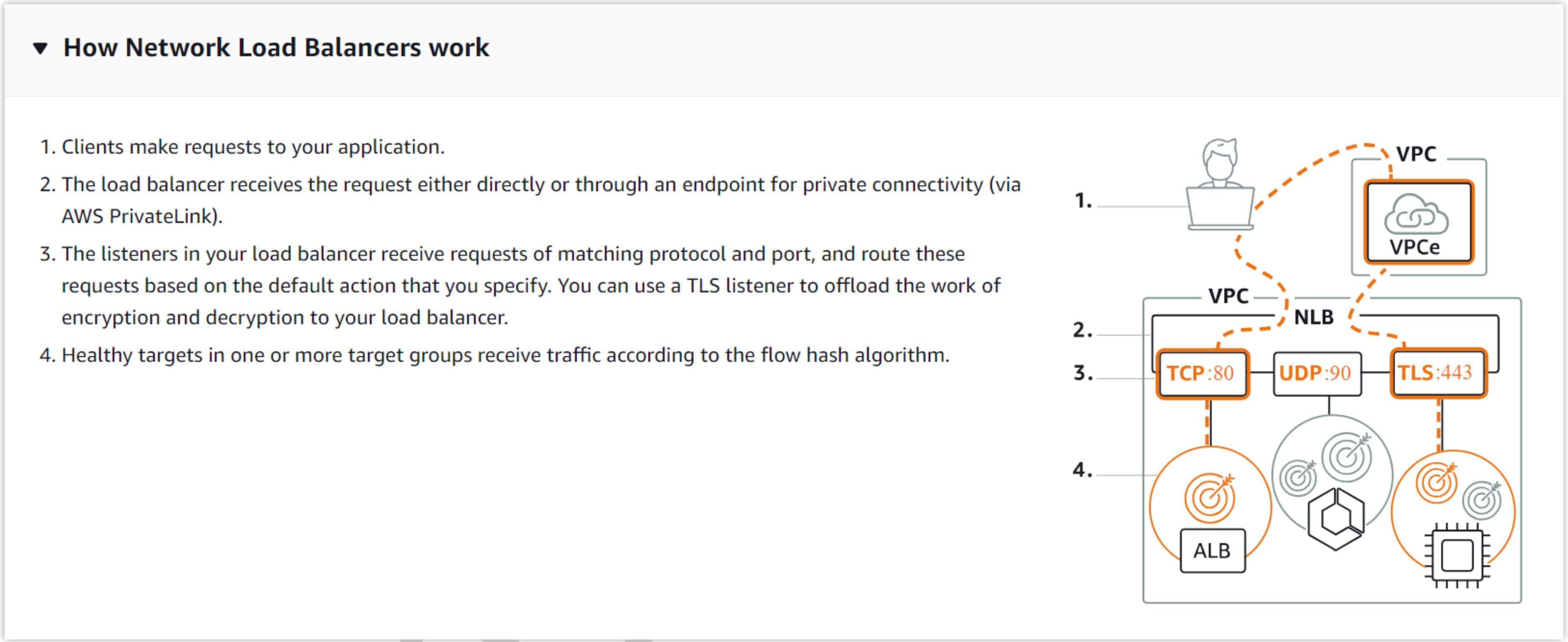
- Network Load Balancer works at the Layer 4 of the OSI model (Transport layer), routing connections to targets – Amazon EC2 instances, containers and IP addresses based on IP protocol data.
- Distributes traffic to multiple targets (e.g., EC2 instances, IP addresses) based on:
  - IP address and port
  - Protocol (TCP, UDP, etc.)
- *Supports Low Latency with high throughput
- **Typically used for:**
  - Load balancing for non-HTTP traffic.
  - Examples: Database connections, Gaming servers etc.
  - High-performance computing applications
- The NLB is designed to handle millions of requests/seconds and to support sudden volatile traffic patterns at extremely low latencies.
- The NLB can be configured with a single static/Elastic IP address for each Availability Zone.
- You can load balance any application hosted in AWS or on-premises using IP addresses of the application back-ends as targets.
- NLB supports both network and application target health checks.
- NLB supports failover between IP addresses within and across AWS Regions (uses Amazon Route 53 health checks).

Target groups for NLBs support the following protocols and ports:

- **Protocols:** TCP, TLS, UDP, TCP_UDP.
- **Ports:** 1-65535.

The table below summarizes the supported listener and protocol combinations and target group settings:

Listener Protocol	Target Group Protocol	Target Group Type	Health Check Protocol
TCP	TCP | TCP_UDP	instance | ip	HTTP | HTTPS | TCP
TLS	TCP | TLS	Instance | ip	HTTP | HTTPS | TCP
UDP	UDP | TCP_UDP	instance	HTTP | HTTPS | TCP
TCP_UDP	TCP_UDP	instance	HTTP | HTTPS | TCP



▼ **How Network Load Balancers work**

1. Clients make requests to your application.
2. The load balancer receives the request either directly or through an endpoint for private connectivity (via AWS PrivateLink).
3. The listeners in your load balancer receive requests of matching protocol and port, and route these requests based on the default action that you specify. You can use a TLS listener to offload the work of encryption and decryption to your load balancer.
4. Healthy targets in one or more target groups receive traffic according to the flow hash algorithm.

ELASTIC LOAD BALANCER (ELB) SUMMARY

Load balancer distributes the web traffic to the available server equally. Load balancing refers to efficient distributing incoming traffic across a group of backend server.

Load Balancer is of 4 types:

1. Classic Load Balancer
2. Application Load Balancer
3. Network Load Balancer
4. Gateway Load Balancers

- An internet facing load balancer has a publicly resolvable DNS name.
- Domain names for content on the EC2 instances served by the ELB, is resolved by the internet DNS server to the ELB DNS name (and hence IP address). This is how traffic from the internet is directed to the ELB front-end.
- Classic load balancer service support: http, https, TCP, SSL.
- Protocols ports supported are 1-65535.
- It supports IPV4, IPV6 and Dual Stack.
- Application load balancer distributes incoming application traffic across multiple targets such as EC2 instances in multiple availability zone. This increases the availability of your application.
- Network load balancer has ability to handle volatile workloads and scale to millions of requests per seconds.
- An ELB listener is the process that checks for connection request. You can configure the protocol/ port number on which your ELB listener listen for connection request.
- Fronted listeners check for traffic from client to the listener.
- Backend listeners are configured with protocol/port to check for traffic from the ELB to the EC2 instances.
- It may take some time for the registration of the EC2 instances under the ELB to complete.
- Registered EC2 instances are those are defined under the ELB.
- ELB has nothing to do with the outbound traffic that is initiated/generated from the registered EC2 instances destined to the internet or to any other instances within the VPC.
- ELB only has to do with inbound traffic destined to the EC2 registered instances (as the destination) and the respective return traffic.
- You start to be charged hourly (also for partial hours) once your ELB is active.
- If you do not want to be charged as you so not need the ELB anymore, you can delete it.
- Deleting the ELB does not affect or delete the EC2 instance registered with it.

- ELB forwards traffic to “eth0” (Primary private IP)of your registered instances.
- In case the EC2 registered instances has multiple IP address on eth0, ELB will route the traffic to its primary IP address.
- Elastic load balancer supports IPV4 address only in VPC.
- To ensure that the ELB service can scale ELB nodes in each AZ, ensure that the subnet defined for the load balancer is at least /27 in scale size and has at least 8 available IP address the ELB nodes can use to scale.
- For fault tolerance it is recommended that you distribute your registered EC2 instances across multiple AZ with in the VPC region.
- If possible, try to allocate same number of registered instances in each AZ.
- The load balancer also monitors the health of its registered instances and ensures that it routes traffic only to healthy instances.
- A healthy instance shows as healthy under the ELB.

- When the ELB detects an unhealthy instance it stops routing traffic to that instance.
- An unhealthy instance shows as unhealthy under the ELB.
- By default, AWS console uses ping http (port 80) for healthy check.
- Registered instances must respond with an http “200 OK” message within the timeout period else it will be considered as unhealthy.

- AWS API uses ping TCP (port-80) for health check.
- Response time-out is 5 seconds (range is 2-60 sec).
- Health check internet.
- Period of time between health check (default 30 and range is 5 to 300 sec)

- **Unhealthy Threshold:** number of consecutive failed health check that should occur before the instance is declared unhealthy.
  Range is 2 to 10
  Default is 2
- **Healthy Threshold:** number of consecutive successful health checks that must occur before the instance considered unhealthy.
  Range is 2 to 10
  Default is 10
- By default the ELB distributes traffic evenly between the AZ, it is defined in without consideration to the number of registered EC2 instances in each AZ.

Cross Zone Load Balancing:

- Disabled by default.
- When enabled, the ELB will distribute traffic evenly between registered EC2 instances.
- If you have 7 EC2 instances in one AZ, and 3 in another AZ, and you enabled cross zone

- Load balancing each registered EC2 instances will be getting the same amount of traffic load from the ELB.
- ELB name you choose must be unique within the account.
- ELB is region specific, so all registered EC2 instances must be in the same region, but can be in different AZs.
- To define your ELB in an AZ you can select one subnet in that AZ. Subnet can be public or private.
- Only one subnet can be defined for the ELB in an AZ.
- If you try and select another one in the same AZ, it will replace the former one.
- If you register instance in an AZ with ELB but do not define a subnet in that AZ for the ELB, these instances will not receive traffic form the ELB.
- ELB should always be accessed using DNS and not IP.

An ELB can be internet facing or internal ELB:
Internet Facing:

- ELB nodes will have public IP address.
- DNS will resolve the ELB DNS name to these IP address.
- If routes traffic to the private IP address of your registered EC2 instances.
- You need one public subnet in each AZ where the internet facing ELB will be defined such that the ELB will be able to route internet traffic.

- An ELB listener is the process that checks for connection request.
- Each network load balancer needs at least one listener to accept traffic.
- You must assign a security group to your ELB. This will control traffic that can reach your ELB front end listeners.

Target Group:

- Logical grouping of targets behind the load balancer.
- Target groups can be existed independently from the load balancer.
- Target group can be associated with an auto scaling group.
- Target group can contain up to 200 targets.

Elastic Load Balancers practical steps:

Create Application Load balancer:

Let's create a Target Group (test-target-group) first to create an Application Load Balancer, also launch few ec2 instances in 2 to 3 Availability Zones, install httpd and web application on all instances. Make changes in index.html file as – server-1, server-2, ………., server-n, so that we can differentiate between server's letter.

➔ Login on the EC2 console → On the navigation pane, under Load Balancing, choose `'Target Groups'` → Click on `'Create target group'` → Choose a target type `'Instances'` → Add 'Target group name' (i.e. test-target-group) → Set 'Protocol : Port' setting as `HTTP : 80` or `HTTPS : 443` (As application load balancer works on application layer- http & https port) → Keep 'IP address type' `IPv4` → select `'VPC'` → Set 'Protocol Version' to `'HTTP1'` → Select

'Health check protocol' as 'HTTP' → Add health check path for your application '/index.html' or you can simply keep default path as '/' → Click 'Next' → Select available instances to include in Target group → Click on 'Include as pending below' → Click on 'Create target group'

We have created a target group now let's create an Application Load balancer:

➔ Now go to 'Load Balancers' option available on the navigation pane → Click on 'Create load balancer' → Select 'Application Load Balancer' → Add 'Load balancer name' (i.e. test-load-balancer) → select Scheme as 'Internet facing' → In the 'Network mapping' section select 'VPC', 'Availability Zones' & 'Subnets' to cover for load balancing → Select 'Security group' → Keep listener port settings as – HTTP : 80 → Select Target Group that you have created before 'test-target-group' → Click on 'Create load balancer'

You have successfully created Application Load Balancer, wait for the state of Load Balancer to change from 'Provisioning' to 'Active'

Test: Once Application Load Balancer state becomes 'Active' copy the DNS Name of Load Balancer and access via browser check if you are able to access application, hit refresh button few times and you can see that request is being routed between all instances added in Target Group.

Create Network Load balancer:

Let's create a Target Group (test-target-group) first to create a Network Load Balancer, also launch few ec2 instances in 2 to 3 Availability Zones, install httpd and web application on all instances. Make changes in index.html file as – server-1, server-2, ………, server-n, so that we can differentiate between server's letter.

➔ Login on the EC2 console → On the navigation pane, under Load Balancing, choose 'Target Groups' → Click on 'Create target group' → Choose a target type 'Instances' → Add 'Target group name' (i.e. test-target-group) → Keep 'Protocol : Port' setting as TCP : 80 OR TLS : 443 OR UDP : 53 (As application load balancer works on Transport layer, on TCP & UDP port)→ Keep 'IP address type' IPv4 → select 'VPC' → Set 'Protocol Version' to 'HTTP1' → Select 'Health check protocol' as 'TCP' → Add health check path for your application '/index.html' or you can simply keep default path as '/' (if you select UDP or TLS port then no need to add health check path) → Click 'Next' → Select available instances to be included in Target group → Click on 'Include as pending below' → Click on 'Create target group'

We have created a target group now let's create an Application Load balancer:

➔ Now go to 'Load Balancers' option available on the navigation pane → Click on 'Create load balancer' → Select 'Network Load Balancer' → Add 'Load balancer name' (i.e. test-load-balancer) → select Scheme as 'Internet facing' → In the 'Network mapping' section select 'VPC', 'Availability Zones' & 'Subnets' to cover for load balancing → Select 'Security group' → Keep listener port settings as – TCP : 80 (Same as Target Group) → Select Target Group that you have created before 'test-target-group' → Click on 'Create load balancer'

You have successfully created Application Load Balancer, wait for the state of Load Balancer to change from `‘Provisioning’` to `‘Active’`

Test: Once Network Load Balancer state becomes ‘Active’ copy the DNS Name of Load Balancer and access via browser check if you are able to access application, hit refresh button few times and you can see that request is being routed between all instances added in Target Group.

Autoscaling with Load Balancer:

Let’s create a Target Group (test-target-group) first to create a Autoscaling group with Load balancers, launch few ec2 instances in 2 to 3 Availability Zones, install httpd and web application on all instances. Make changes in index.html file as – server-1, server-2, ………, server-n, so that we can differentiate between server’s letter.

➔ First of all, follow above steps to create an application load balancer → Once application load balancer is ready create a ‘Launch Template’:

➔ Go to EC2 console → Click on ‘Launch Templates’ option from left sides Manu → click on ‘Create launch template’ → add ‘Launch template name’ & ‘Template version description’ → Select ‘AMI’ → Select ‘Instance type’ → Select ‘Key pair name’ → then select ‘subnet’ & ‘security group’ → click on ‘Advanced details’ → add user data script to install httpd and deploy our web application → Click on **‘Create launch template’**

Once the launch template is created let’s begin with auto scaling group creation ➔ On EC2 console on left sides menu go to option ‘Auto Scaling Group’ → Select ‘Create auto scaling group’ option → Add Auto scaling group name → Select launch template and then click on ‘Next’ → Select VPC and Subnets for ASG → Click on ‘Next’ → in Load Balancing select option ‘Attach to an existing load balancer’ option → Select option ‘Choose from your load balancer groups’ → Then ‘Select target group’ → click on ‘Next’ → Mention ‘Desired capacity - 3’, ‘Minimum desired capacity- 2’ & ‘Max desired capacity- 5’ → Click on ‘Next’ → ‘Next’ → ‘Next’ → At final click on **‘Create auto scaling group’**

TEST: Now go to ‘Target Group’ you can see that Ec2 instances launched by Auto scaling group are now being added in Target Group which is being used by Load balancer to equally distribute the traffic.

Cleanup sequence:

- Delete Auto Scaling Group
- Delete Launch template
- Delete Load balancer
- Delete Autoscaling group
- And at final terminate manually created instances.