AWS ALB and AutoScaling

CS516 – Cloud Computing

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Types of ELB
    - OSI Model
    - Application Load Balancer (ALB)
    - Network Load Balancer (NLB)
- Application Load Balancer (ALB)
    - ALB listener
    - ALB listener rule
    - ALB target groups
- Auto Scaling (with CloudWatch)
- Scaling policies

Scale up issues

Scaling up means making your server or resources bigger and bigger in terms of CPU, memory, etc. The issues with scaling up are:

- Hit the limit – The biggest instance (u-12tb1.112xlarge) on AWS has 448 vCPU and 12,288 GiB memory, charges hourly (on-demand) for $131 (Linux) and $152 (Windows).
- More expensive – Running one big instance charges more than running many smaller instances.
- Single point of failure – If the server goes down, you lose everything.

Because of the issues above, scale out is recommended. Scaling out is to add more servers but runs the same app. For better scaling, you have to design your app properly in a stateless manner in the first place. Serverless services support scaling out out-of-box.

Elastic Load Balancer (ELB)

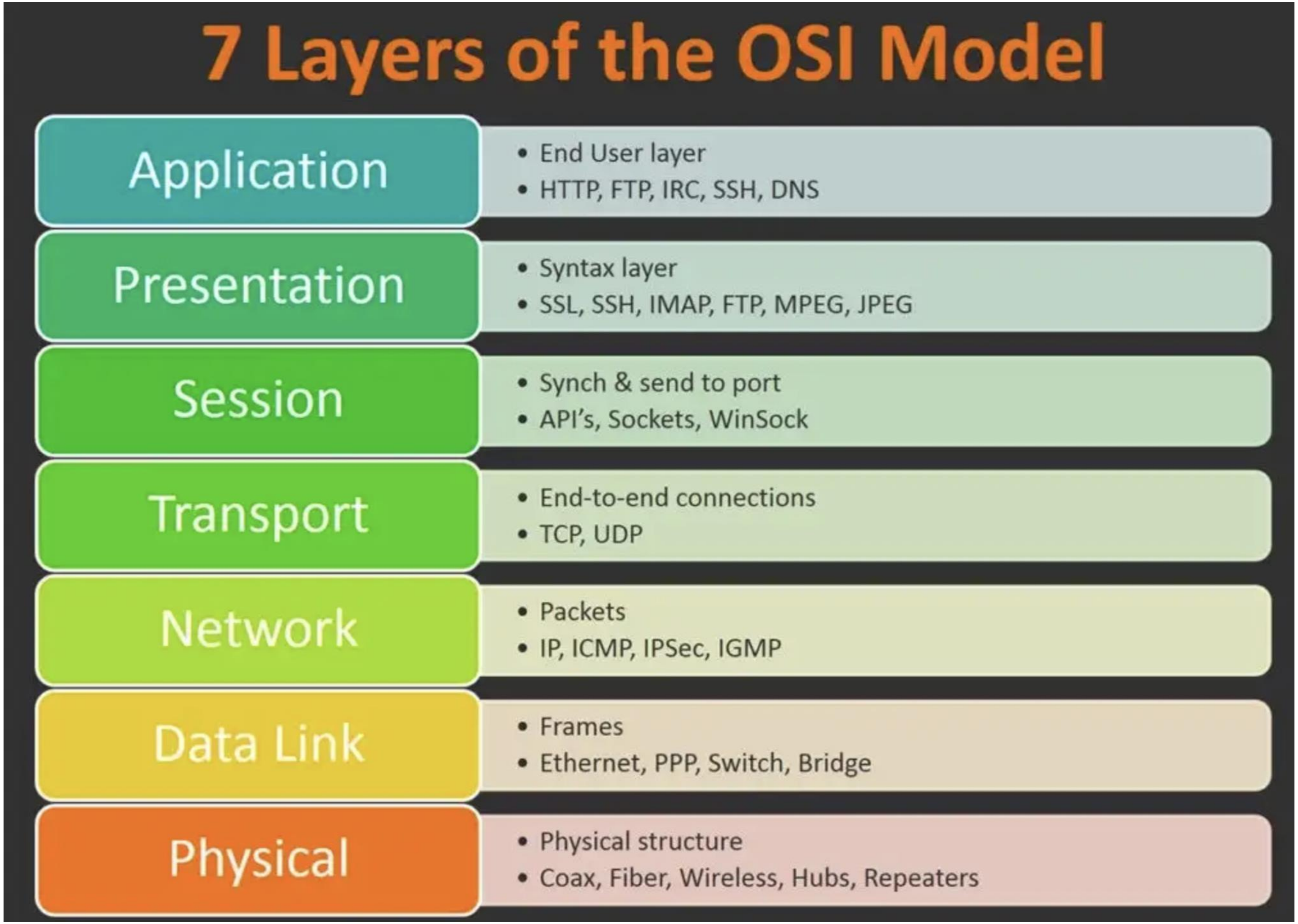
Elastic Load Balancer is a tool that distributes incoming web traffic (visitors to a web site) and equally across multiple EC2 instances that are running the same app. It enables fault tolerance by seamlessly providing the required amount of load-balancing capacity needed to route application traffic.

It helps prevent one server from being overloaded while another server can handle more visitors. It makes the app more reliable.

ELB features

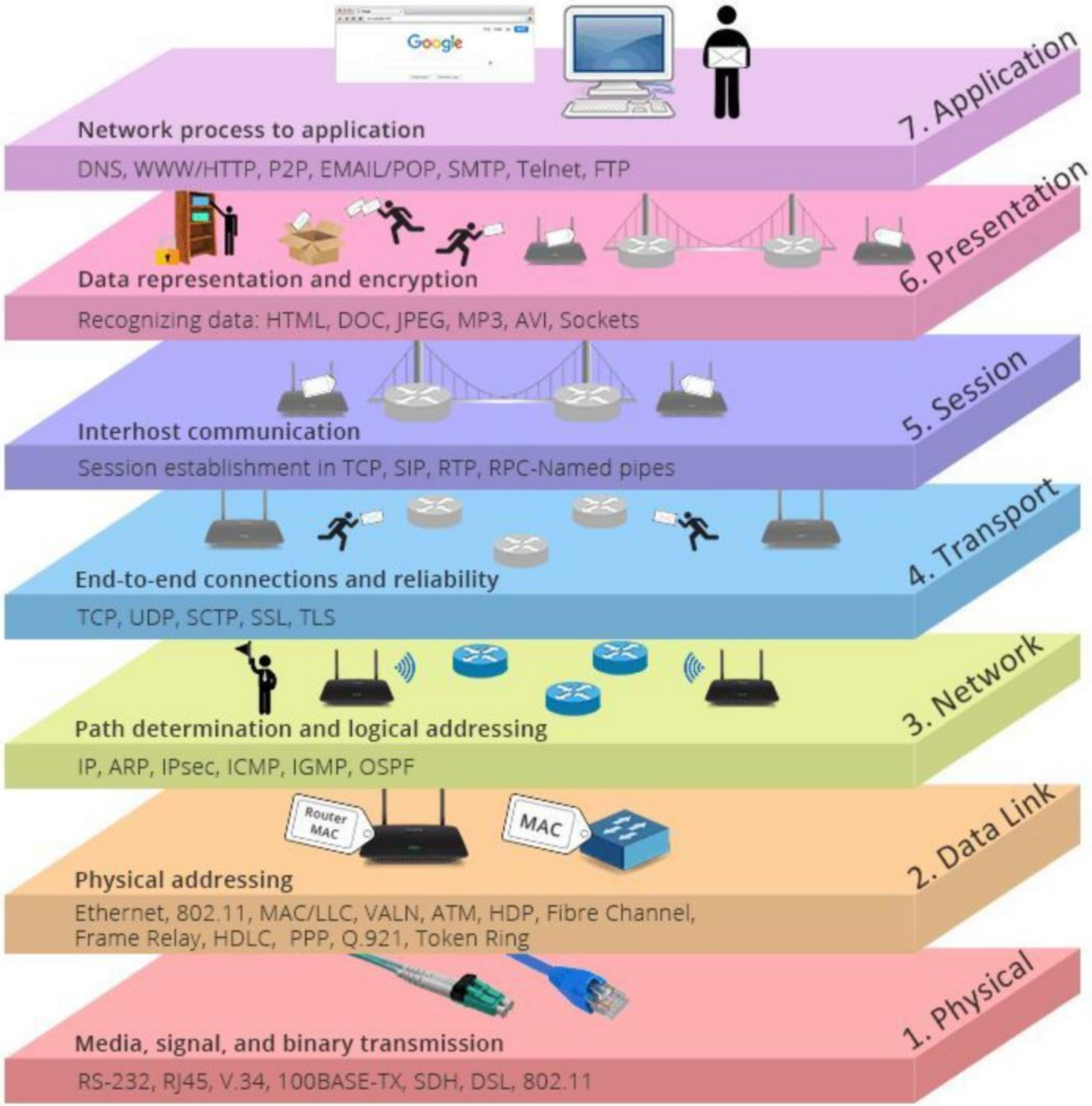
- Helps scaling out. You can run “N” number of servers behind an ELB. All Servers run the same (your) app.
- It is like a gateway to your application(s). There will be one URL for all your servers when using an ELB.
- Improves fault tolerance and reliability. Because it knows if the underlying servers are healthy or not. Then routes traffics to only healthy instances. An ELB is used in conjunction with ASG.

There are 4 types of ELBs in AWS. They all operate on different OSI layers. The Open Systems Interconnection (**OSI**) is a model that shows how human-readable data is transferred over the cable or wife.



7 Layers of the OSI Model

Application	• End User layer • HTTP, FTP, IRC, SSH, DNS
Presentation	• Syntax layer • SSL, SSH, IMAP, FTP, MPEG, JPEG
Session	• Synch & send to port • API's, Sockets, WinSock
Transport	• End-to-end connections • TCP, UDP
Network	• Packets • IP, ICMP, IPSec, IGMP
Data Link	• Frames • Ethernet, PPP, Switch, Bridge
Physical	• Physical structure • Coax, Fiber, Wireless, Hubs, Repeaters



7. Application

Network process to application

DNS, WWW/HTTP, P2P, EMAIL/POP, SMTP, Telnet, FTP

6. Presentation

Data representation and encryption

Recognizing data: HTML, DOC, JPEG, MP3, AVI, Sockets

5. Session

Interhost communication

Session establishment in TCP, SIP, RTP, RPC-Named pipes

4. Transport

End-to-end connections and reliability

TCP, UDP, SCTP, SSL, TLS

3. Network

Path determination and logical addressing

IP, ARP, IPsec, ICMP, IGMP, OSPF

2. Data Link

Physical addressing

Ethernet, 802.11, MAC/LLC, VALN, ATM, HDP, Fibre Channel,
Frame Relay, HDLC, PPP, Q.921, Token Ring

1. Physical

Media, signal, and binary transmission

RS-232, RJ45, V.34, 100BASE-TX, SDH, DSL, 802.11

OSI Layers

Learn more: What is the OSI Model?

- Layer 7 (The **application** layer) – It is the layer that directly interacts with a user. These are applications such as email and browsers that shows data in a human-readable format.
- Layer 6 (The **presentation** layer) – This layer prepares the human-readable data by decrypting and decompressing data if required.
- Layer 5 (The **session** layer) – This layer is responsible for opening and closing communication between the local and remote applications. For instance, if a user is downloading 50 MiB data, this layer checks if it is done every time 5 MiB data is downloaded.

OSI Layers

- Layer 4 (The **transport** layer) – Main layer. It receives data from the session layer (5) and breaks the data down into smaller chunks and sends the data to the network layer (3). It also assembles data from layer 3 and passes that to layer 5.

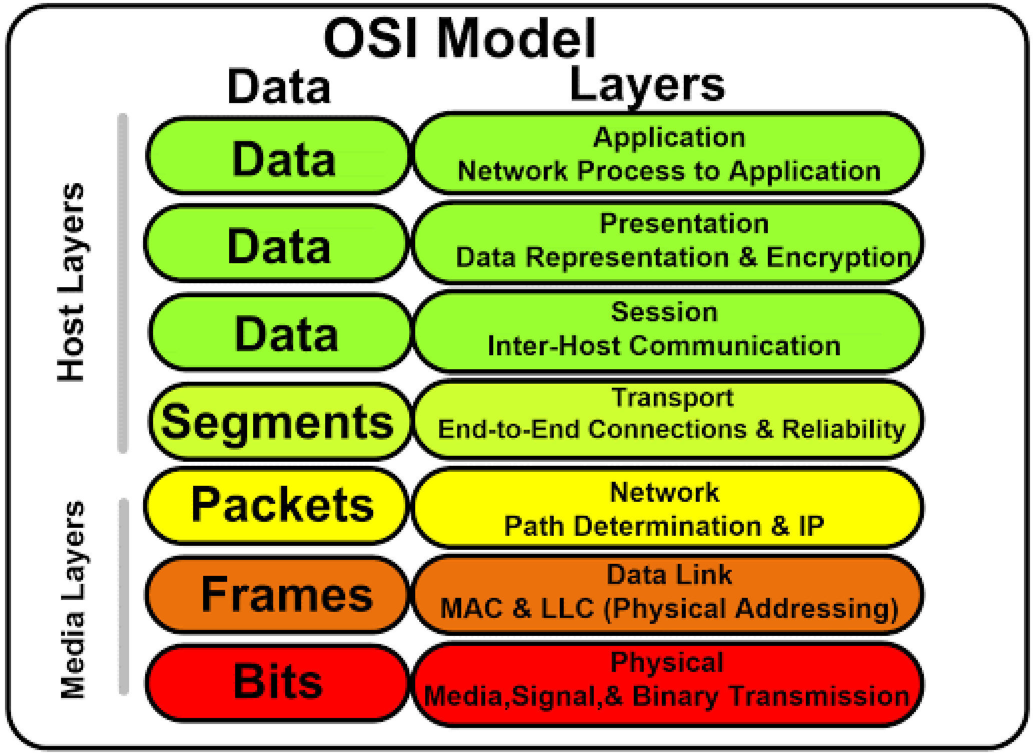
This layer **transports data between 2 devices**. The connection-oriented TCP protocol is used in this layer that makes sure all data is transferred and is more reliable. There is also a connectionless UDP protocol in this layer. UDP is faster than TCP used in streaming and gaming applications.

OSI Layers

- Layer 3 (The **network** layer) – This layer is used to **connect different networks**. Common protocols in this layer are IP and ICMP. In real-life, developers need to check the connection between 2 servers in 2 different networks. In that case, they check the connection by making an ICMP call. You will see the ICMP protocol is open in some SG groups and now you understand what the rule is for.
- Layer 2 (The **data link** layer) – This layer is similar to the network layer but **between devices in the same network.**
- Layer 1 (The **physical** layer) – It is the physical networking matter you see all around you such as cables and switches. It transfers machine data in bits (one-zero, light-dark, high-low frequencies).

Types of ELB

1. **Network Load Balancer** (NLB) – Routes connections based on IP protocol data (layer 4). Provides ultra-high performance and low latency. Supports UDP and static IP addresses as targets.
2. **Application Load Balancer** (ALB) – Routes based on the content of the request (layer 7). Supports path-based, host-based, query string, parameter-based, and source IP-based routing. Supports IP addresses, Lambda, and containers as targets.
3. **Classic Load Balancer** (CLB) – Old generation, not recommended for new apps. Performs routing at Layer 4 and Layer 7.
4. **Gateway Load Balancer** – Operates at Layer 3 (Network). Gateway Load Balancers enable you to deploy, scale, and manage virtual appliances, such as firewalls, intrusion detection and prevention systems, and deep packet inspection systems.

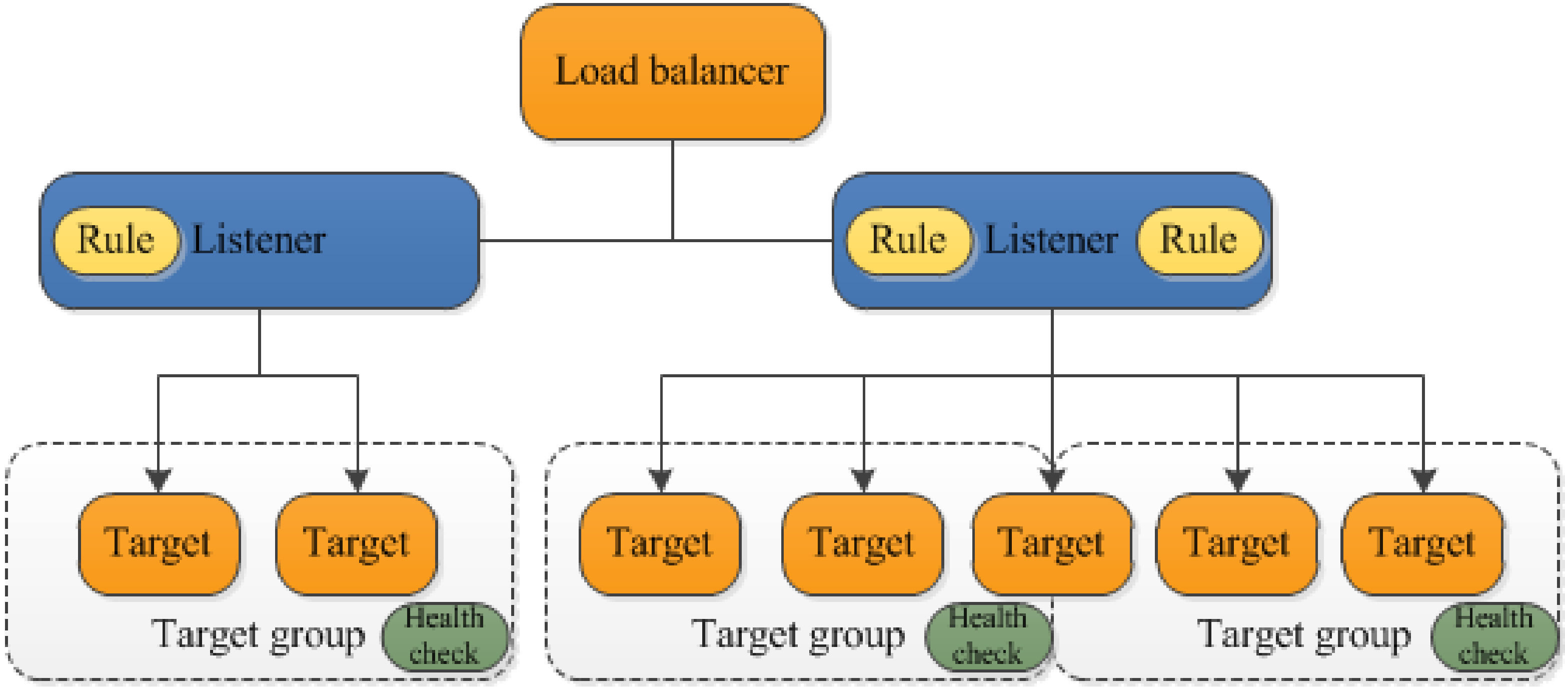


OSI Model

	Data	Layers
Host Layers	Data	Application Network Process to Application
	Data	Presentation Data Representation & Encryption
	Data	Session Inter-Host Communication
	Segments	Transport End-to-End Connections & Reliability
Media Layers	Packets	Network Path Determination & IP
	Frames	Data Link MAC & LLC (Physical Addressing)
	Bits	Physical Media,Signal,& Binary Transmission

Application Load Balancer (ALB)

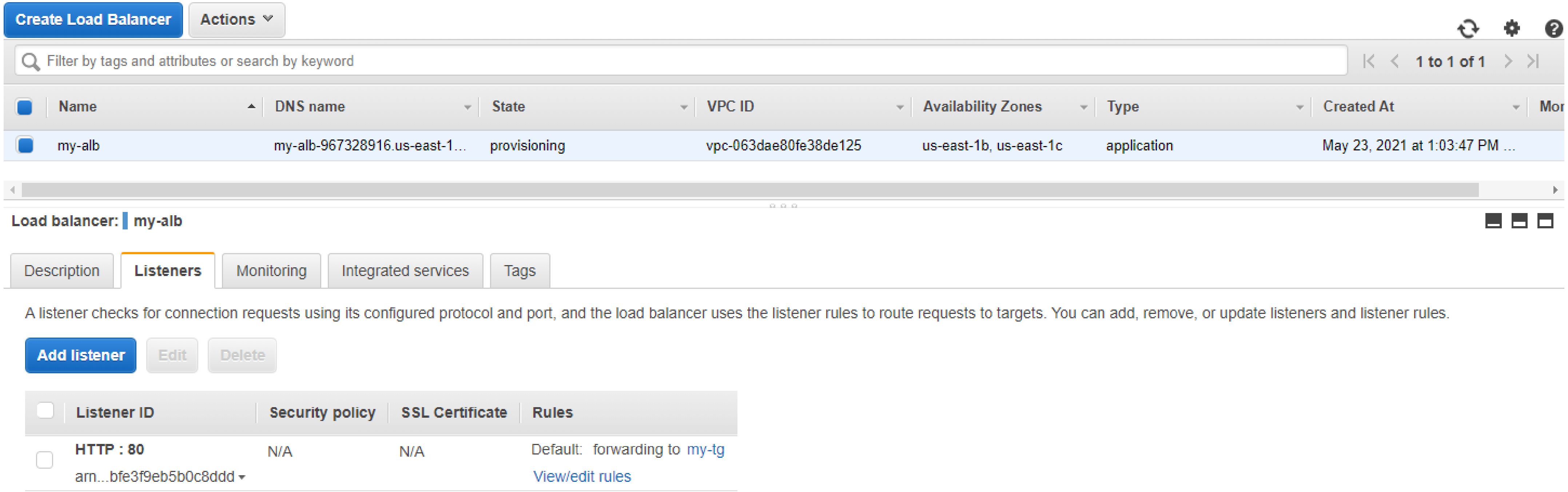
A load balancer serves as the single point of contact for clients. The load balancer distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones. This increases the availability of your application. You add one or more listeners to your load balancer.



ALB Listener

A listener checks for connection requests from clients, using the **protocol** or **port** that you configure. ALB supports HTTP (80) and HTTPS (443). That means you can have 2 rules on ALB at most.

A **certificate** is attached to the listener. You must define a default certificate if using HTTPS. AWS and custom certificates are stored on Amazon Certificate Manager (ACM). AWS certificates are free!



Name	DNS name	State	VPC ID	Availability Zones	Type	Created At	Mor
my-alb	my-alb-967328916.us-east-1...	provisioning	vpc-063dae80fe38de125	us-east-1b, us-east-1c	application	May 23, 2021 at 1:03:47 PM ...	

Load balancer: my-alb

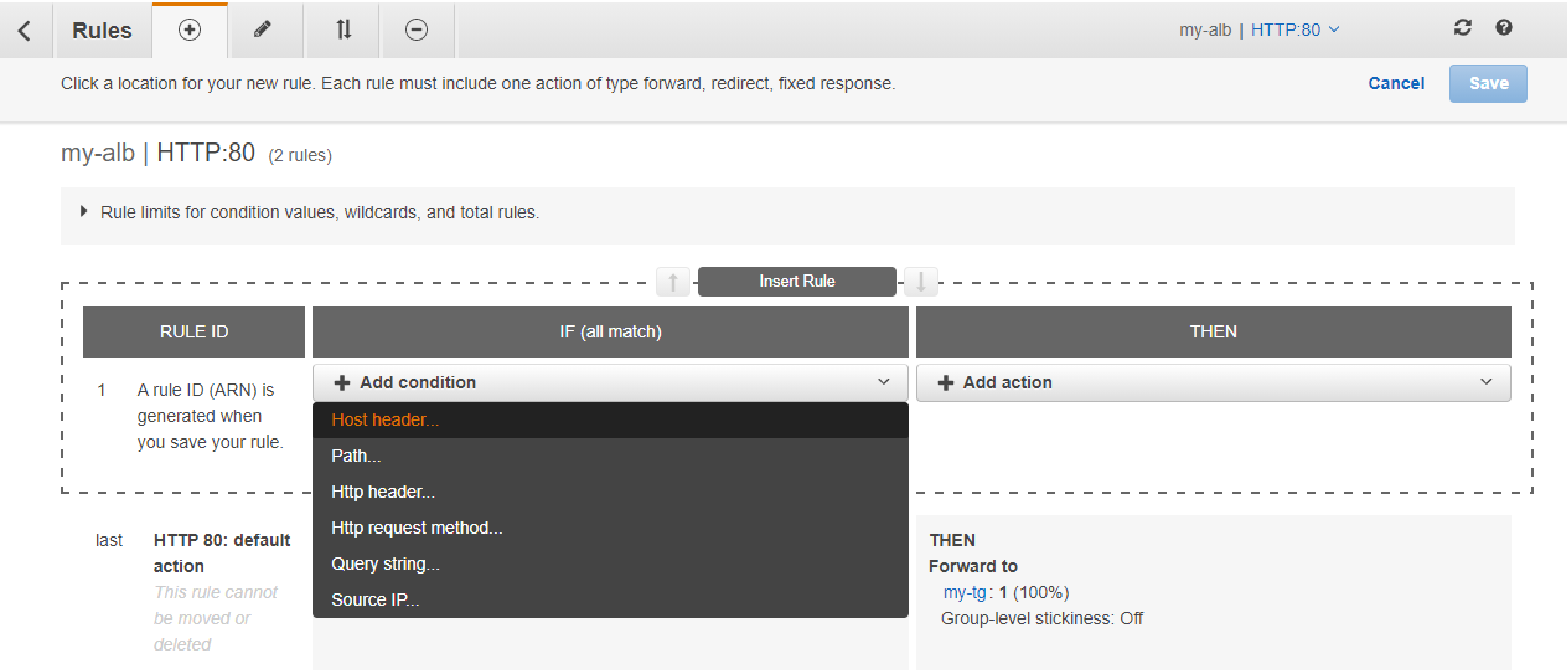
A listener checks for connection requests using its configured protocol and port, and the load balancer uses the listener rules to route requests to targets. You can add, remove, or update listeners and listener rules.

Listener ID	Security policy	SSL Certificate	Rules
HTTP : 80 arn...bfe3f9eb5b0c8ddd	N/A	N/A	Default: forwarding to my-tg View/edit rules

ALB Listener Rules

The **rules** that you define for a listener determine how the load balancer routes requests to its **registered targets**.

Each rule consists of a **priority, actions, conditions**. If you try to deploy a new rule with a rule ID that already exists, it fails.



Click a location for your new rule. Each rule must include one action of type forward, redirect, fixed response.

my-alb | HTTP:80 (2 rules)

Rule limits for condition values, wildcards, and total rules.

RULE ID	IF (all match)	THEN
1 A rule ID (ARN) is generated when you save your rule.	Add condition	Add action
last **HTTP 80: default action** *This rule cannot be moved or deleted*		**THEN** **Forward to** my-tg: **1** (100%) Group-level stickiness: Off

Running multiple apps behind a single ALB

You can run multiple apps behind an ALB. For that, you need to write listener rules and create a TG for each app. For example:

Request	Rule	Target Group
example.com/app1	Path based, if the request includes “/app1” in the path, route to the “TG1”.	TG1 – Target Group that groups resources (such as EC2 instances, ECS tasks, Lambda instances) that run the “app1”.
example.com/app2	Path based, if the request includes “/app2” in the path, route to the “TG2”.	TG2 – Target Group that groups resources (such as EC2 instances, ECS tasks, Lambda instances) that run the “app2”.

Not only the request path, you can define rules based on other parameters such as headers, query strings.

ALB Listener Rules

When the conditions for a rule are met, then its actions are performed. You must define a default rule for each listener, and you can optionally define additional rules. There could be many apps behind an ALB. You can have the following rules:

- If the request path contains “hr”, then forward them to the microservice for the HR module.
- If the request path contains “finance”, then forward requests to the target group that has the Finance app.

Not only the request path, you can also define rules based on query strings, headers, source IP, HTTP methods. And route requests to different target groups (or apps) based on the rules.

ALB Target Groups

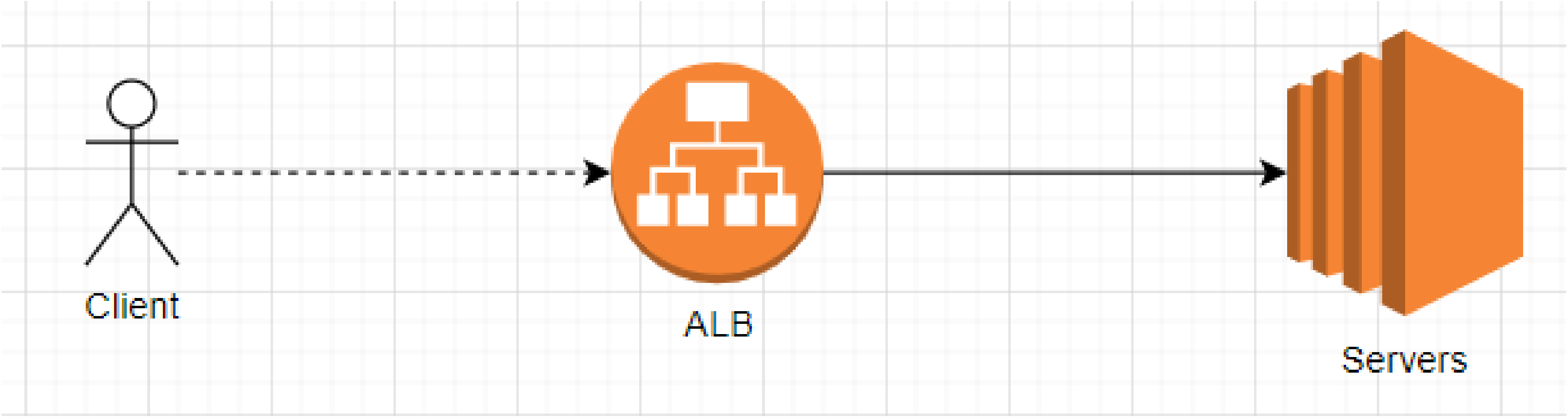
A Target Group (TG) is a multiple resources that is running the same app. It could be a fleet of EC2 servers running the same app based on the custom AMI. Resources in a TG are mostly created by Auto Scaling Groups (ASG) and the number of resources in the TG increases and decreases automatically based on the incoming traffic.

You can configure health checks on a per-target group basis. Health checks are performed on all targets registered to a target group.

Note: When you associate a resource with an ALB, you associate it with its target group. When creating an ALB, the target group can have no resources. But the important thing is you must specify the right target group type (IP, instance, lambda).

ALB Target Groups

Each target group routes requests to one or more registered targets, such as EC2 instances, **using the protocol and port number that you specify**. That means there are 2 connections on the way. One from the client to ALB and the connection terminates there. Then the other from the ALB to the registered target (servers) in the TG.



Network Load Balancer (NLB)

NLB

NLB is just like an ALB with some differences.

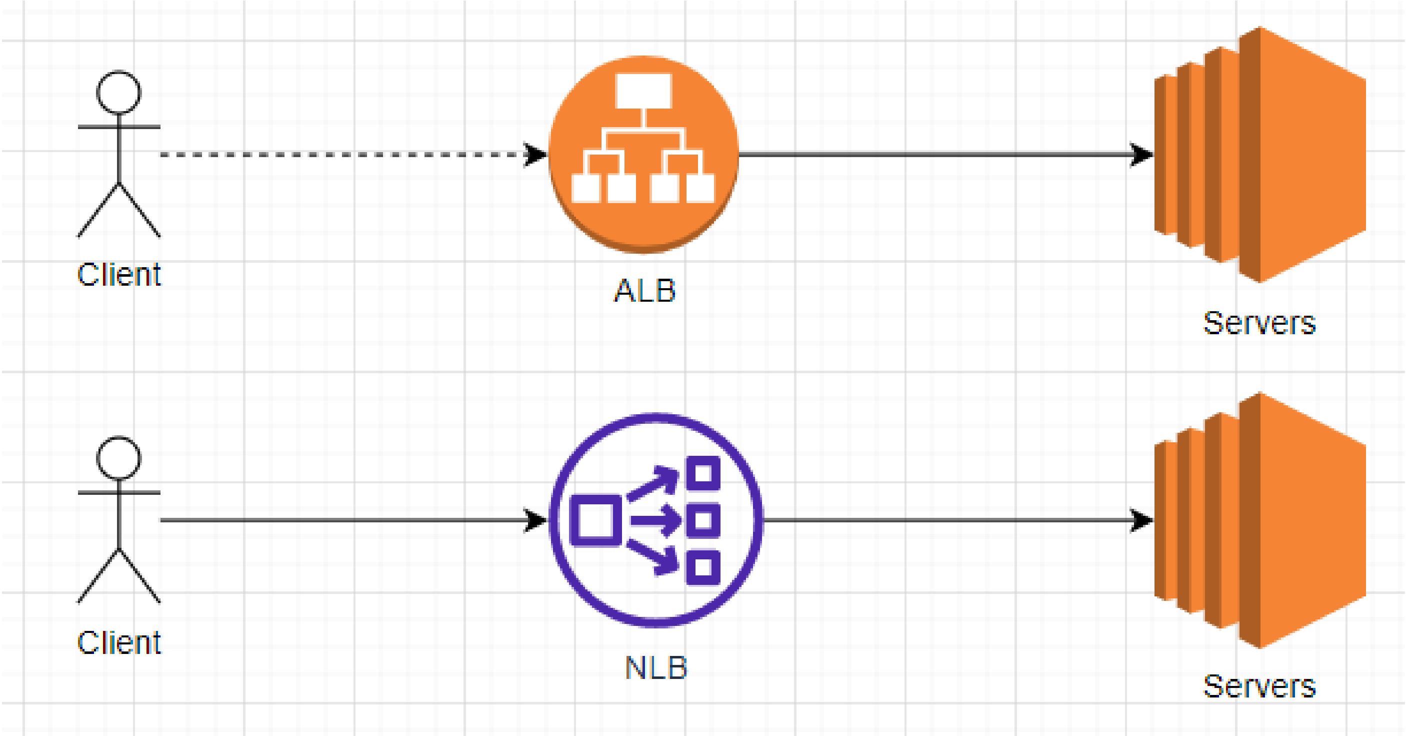
ALB features that NLB doesn’t support:

- Web sockets.
- Authentication with well-known identity providers such as Microsoft, Google, and Facebook. So, you can offload the authentication part from your applications.
- Can send a fixed response without a backend.

The difference between ALB and NLB

ALB	**NLB**
Operates at OSI Layer 7 (Application)	Operates at OSI Layer 4 (Transport)
Lower performance than NLB	High performance
IP address, ECS, EC2, and Lambda as a target	Only IP address as a target
Routing rules based on hostname, path, query string parameter, HTTP method, HTTP headers, source IP, or port number.	Routing is only based on port number.
The source IP is the ALB node IP. To get the client IP, enable preserve client IP and it will be in the header.	The client IP is preserved by default as the source IP.
HTTP and HTTPS	TCP and UDP
Great for web apps.	Great for streaming, near-real-time applications.
Has SG.	There is no SG.

- The client connection terminates at ALB whereas the client connection terminates at the server in NLB.
- The source IP in server is ALB where the source IP in server is Client IP in NLB. You can change this default behavior in NLB by deselecting preserve client IP.
- NLB doesn’t have SG. To whitelist NLB, you can use subnets in which the NLB nodes created.



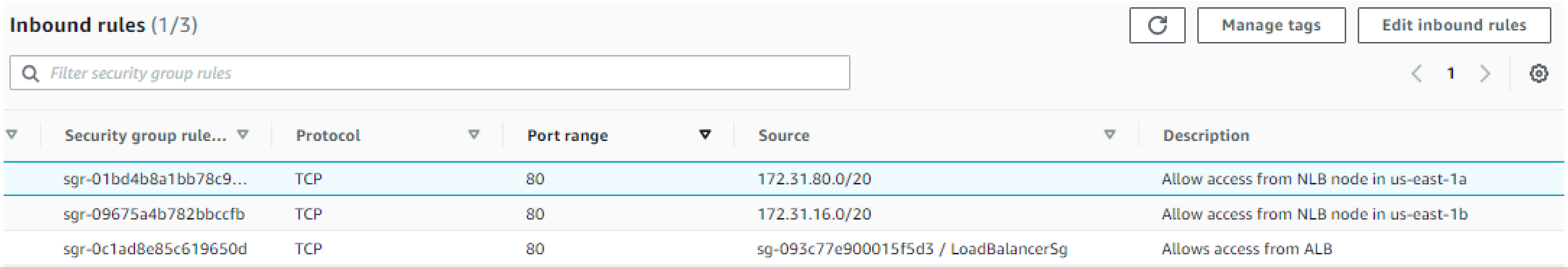
The instance is unhealthy in TG

There are 3 reasons.

The web app is not started – To test it, change the SG temporarily and add an inbound rule that allows access from the internet. Grab the public IP and see if the web app returns. You must create a custom index.html. Otherwise, the default Apache page returns 403 but the ALB expects 200. So, it is unhealthy. To fix it, you must SSH into your instance and do the steps manually. Or use the user data I provided. There could be some issues due to indent, escaping, unnecessary spaces, special characters, etc. So carefully check the command that you are copying.

The instance is unhealthy in TG

Web server’s SG – Make sure you allowed access in the web server’s SG on the right port 80 from the ALB’s SG or subnets in which the NLB nodes are created.



Inbound rules (1/3)

Security group rule...	Protocol	Port range	Source	Description
sgr-01bd4b8a1bb78c9...	TCP	80	172.31.80.0/20	Allow access from NLB node in us-east-1a
sgr-09675a4b782bbccfb	TCP	80	172.31.16.0/20	Allow access from NLB node in us-east-1b
sgr-0c1ad8e85c619650d	TCP	80	sg-093c77e900015f5d3 / LoadBalancerSg	Allows access from ALB

The instance is unhealthy in TG

Sometimes the target group doesn’t directly pick up the changes after you did the fix on the instance with SSH. In that case, deregister and register the instance again. When you **reregister** the EC2 instance, it picks up the new config and marks the instance healthy.

Routing algorithm

The routing algorithms choose the target in the target group.

Round-robin (default) load balancing is one of the simplest methods for distributing client requests across a group of servers. Going down the list of servers in the group, the round-robin load balancer forwards a client request to each server in turn. When it reaches the end of the list, the load balancer loops back and goes down the list again.

Least outstanding requests is an algorithm that choses which instance receives the next request by selecting the instance that, at that moment, has the lowest number of outstanding (pending, unfinished) requests.

Pricing

Both ALB and NLB charge $15 in a month for up time. Plus, the number of connections, rules evaluated (10 rules are free), data processed cost.

It costs $6 when there are:

- 25 new connections per second
- 3,000 active connections per minute
- 1 GB data is processed per hour (each request has 300KB)
- 1,000 rules evaluated per second

There is no additional charge for an ASG. You only pay for the resources it created.

Auto Scaling Group

Scaling factors

- Tune one server based on the following 5 factors. For example, if your app is High-performance computing (HPC) app, then you would need more CPU than memory, network, GPU, and storage.
  - CPU,
  - Memory,
  - Network,
  - GPU,
  - Storage
- Once you tuned the server, you want to plan the capacity or the entire fleet of servers. Whether you want to rent those servers from the cloud provider or just have your own servers. How many servers you want etc.

Auto Scaling benefits

- **Better fault tolerance** - Auto Scaling can detect when a resource is unhealthy, terminate it, and launch a resource to replace it. You can also configure resources to use multiple AZs. If one AZ becomes unavailable, Auto Scaling can launch resources in another one to compensate.
- **Better availability** - Auto Scaling helps ensure that your application always has the right amount of capacity (**desired number** of resources) to handle the current traffic demand.
- **Better cost management** - Auto Scaling can dynamically increase and decrease capacity as needed. Because you pay for the resources you use.

Scaling process

Let’s say you have an autoscaling policy that keeps CPU utilization around 50%. Then

1. If it is more than 50%, it will scale out (add a new instance)
2. If it is less than 50%, it will scale in (remove an instance)

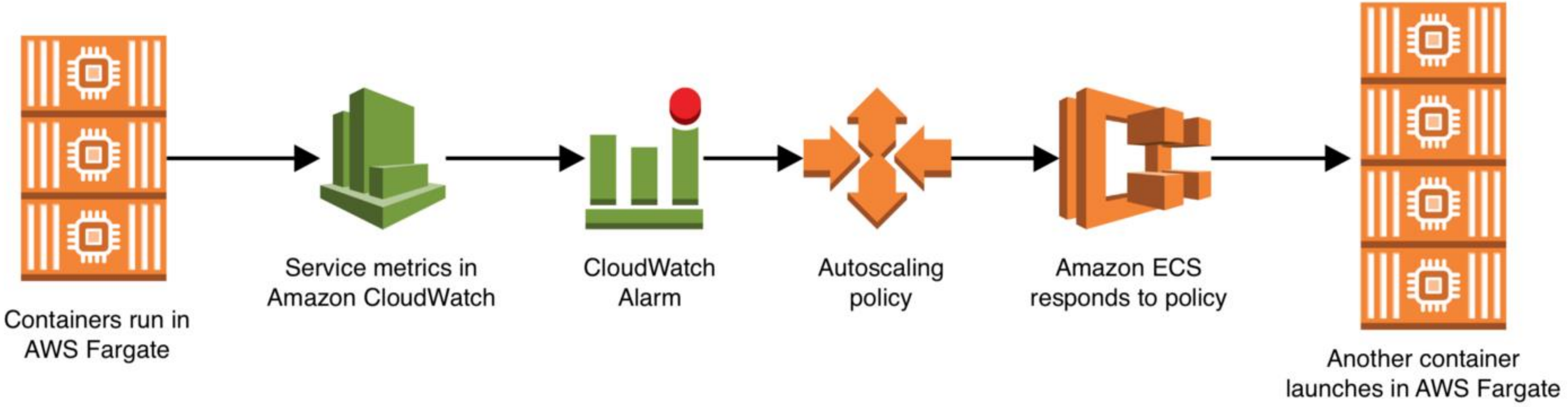
Steps to scale where “x” is the current number of resources running at the that time, “y” is the desired number:

1. x = 4 and y = 4 => Initial state
2. less than 50% (scale in) => It has been less than 50% for a while, so it triggers next scaling in activity.
3. x = 4 and y = 3 => Autoscaling decreased the desired number of instances by one. But current number of running instances is 4. Any activities has not happened yet.
4. Scaling activity occurs and it removes an instance => Since there is a difference between the current number and desired number, scaling activity starts. The current number follows the desired number.
5. x = 3 and y = 3 => Scaling has been done successfully, now it waits until next alarm is triggered.

Auto Scaling

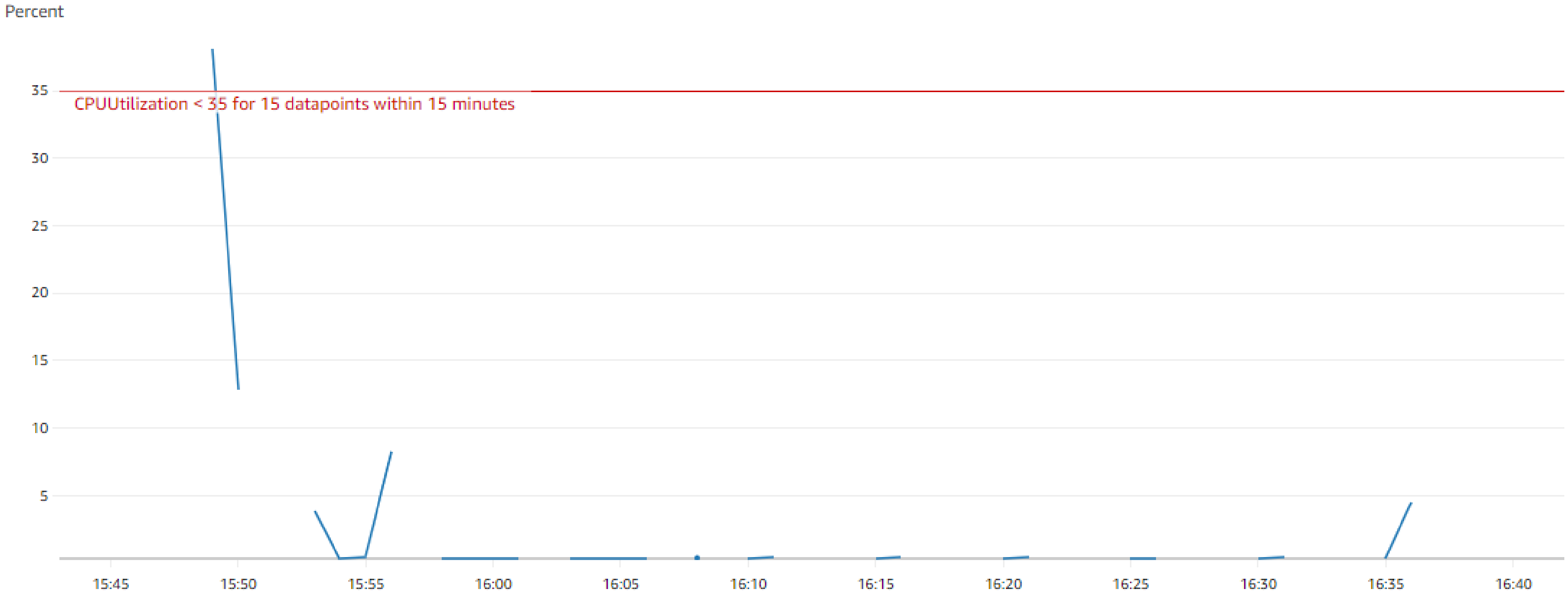
Application Auto Scaling automatically scales scalable resources such as EC2, Elastic Container Service (ECS), Lambda, Aurora replicas (RDS), DynamoDB, and more.

It is nothing but changes the desired number of resources running. The desired number is an ideal number of resources to meet the demand. If the current number of instances is 4 and desired number is 3, it will remove 1 instance.


Containers run in AWS Fargate
Service metrics in Amazon CloudWatch
CloudWatch Alarm
Autoscaling policy
Amazon ECS responds to policy
Another container launches in AWS Fargate

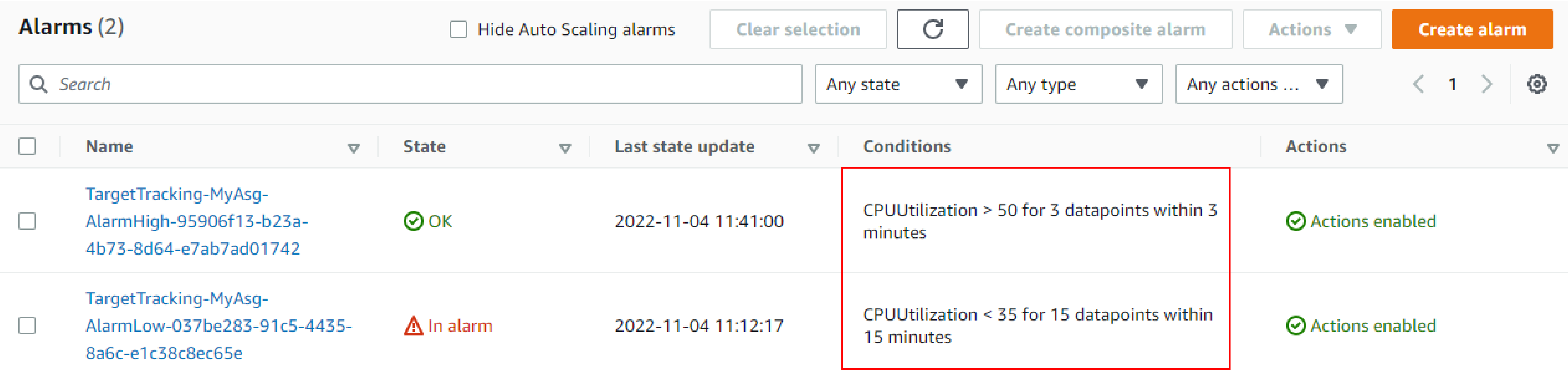
CloudWatch Alarm for AutoScaling

It scales in (terminates) EC2 instances from the target group automatically when CPU utilization is less than 35% for 15 minutes.



CloudWatch Alarm for AutoScaling

By default, scaling out activity takes effect in 3 minutes to handle the surge of transactions. Scaling out activity waits longer for 15 minutes to make sure the demand went back to normal.



Alarms (2)

□	Name	State	Last state update	Conditions	Actions
□	TargetTracking-MyAsg-AlarmHigh-95906f13-b23a-4b73-8d64-e7ab7ad01742	✓ OK	2022-11-04 11:41:00	CPUUtilization > 50 for 3 datapoints within 3 minutes	✓ Actions enabled
□	TargetTracking-MyAsg-AlarmLow-037be283-91c5-4435-8a6c-e1c38c8ec65e	⚠ In alarm	2022-11-04 11:12:17	CPUUtilization < 35 for 15 datapoints within 15 minutes	✓ Actions enabled

EC2 Auto Scaling components

Groups - Your EC2 instances are organized into groups so that they can be treated as a logical unit for the purposes of scaling and management.

Configuration templates - Your group uses a launch (configuration) template where you can specify information such as the AMI ID, instance type, key pair, security groups, and block device mapping for your instances.

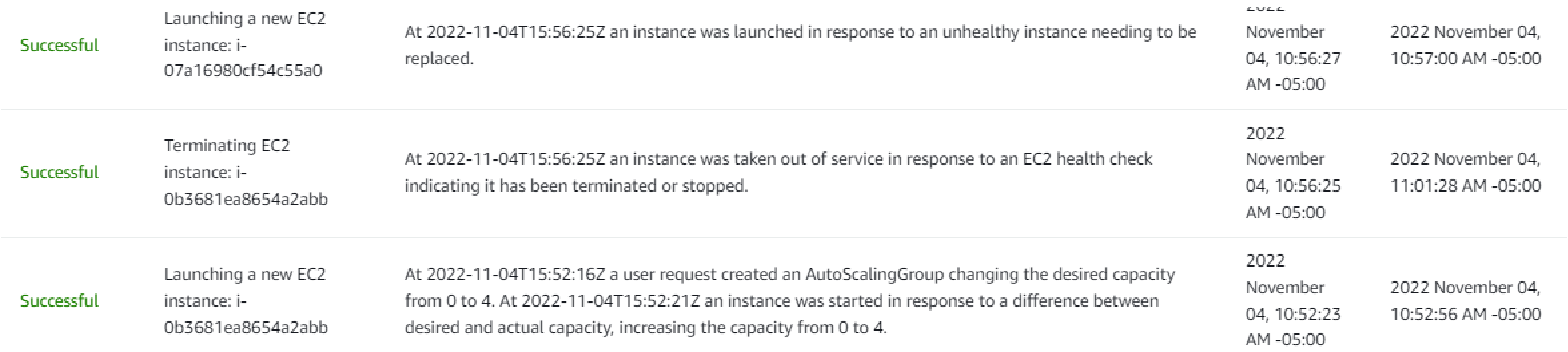
Scaling options - Amazon EC2 Auto Scaling provides several ways for you to scale your Auto Scaling groups. For example, you can configure a group to scale based on the occurrence of specified conditions (dynamic scaling) or on a schedule.

Launch template and ASG

- Launch template contains
  - AMI
  - instance type
  - IAM profile
  - User Data – Startup script. It will be executed when the server starts.
  - SG and so on.
- ASG contains
  - Networking (AZs)
  - Scaling policies
  - Load Balancer
  - Health check configuration

EC2 Auto Scaling

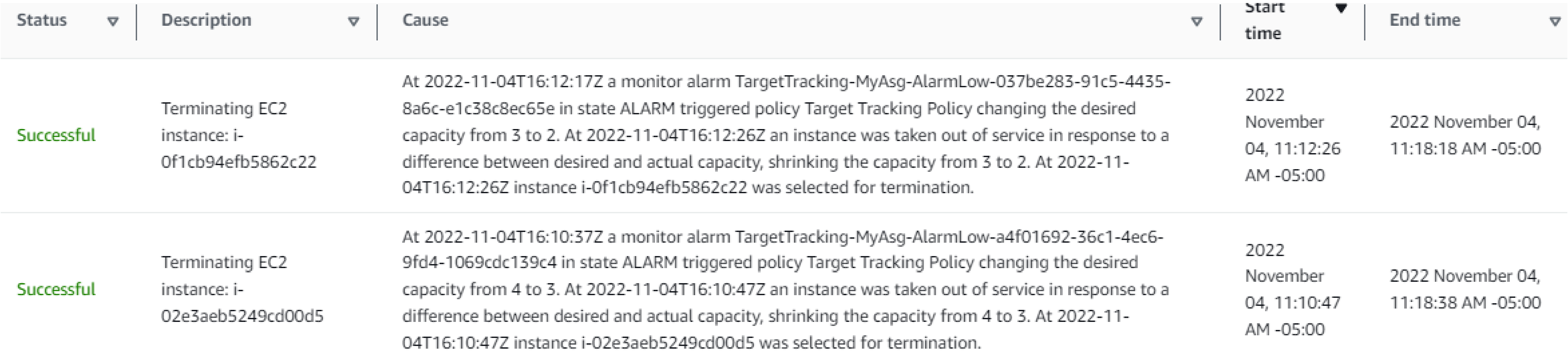
The server was terminated intentionally to mimic the server went down. After that, ASG automatically detects if the server is terminated and brings a new server. It added a new instance automatically.



Successful	Launching a new EC2 instance: i-07a16980cf54c55a0	At 2022-11-04T15:56:25Z an instance was launched in response to an unhealthy instance needing to be replaced.	2022 November 04, 10:56:27 AM -05:00	2022 November 04, 10:57:00 AM -05:00
Successful	Terminating EC2 instance: i-0b3681ea8654a2abb	At 2022-11-04T15:56:25Z an instance was taken out of service in response to an EC2 health check indicating it has been terminated or stopped.	2022 November 04, 10:56:25 AM -05:00	2022 November 04, 11:01:28 AM -05:00
Successful	Launching a new EC2 instance: i-0b3681ea8654a2abb	At 2022-11-04T15:52:16Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 4. At 2022-11-04T15:52:21Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 4.	2022 November 04, 10:52:23 AM -05:00	2022 November 04, 10:52:56 AM -05:00

EC2 Auto Scaling

ASG automatically adjusts the number of desired instances based on CW alarms. In this case, CPU utilization was low, so it changed the desired number of instances from 4 to 3 and from 3 to 2 until it hits the minimum number of instances.



Status	Description	Cause	Start time	End time
Successful	Terminating EC2 instance: i-0f1cb94efb5862c22	At 2022-11-04T16:12:17Z a monitor alarm TargetTracking-MyAsg-AlarmLow-037be283-91c5-4435-8a6c-e1c38c8ec65e in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 3 to 2. At 2022-11-04T16:12:26Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2022-11-04T16:12:26Z instance i-0f1cb94efb5862c22 was selected for termination.	2022 November 04, 11:12:26 AM -05:00	2022 November 04, 11:18:18 AM -05:00
Successful	Terminating EC2 instance: i-02e3aeb5249cd00d5	At 2022-11-04T16:10:37Z a monitor alarm TargetTracking-MyAsg-AlarmLow-a4f01692-36c1-4ec6-9fd4-1069cdc139c4 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 4 to 3. At 2022-11-04T16:10:47Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 4 to 3. At 2022-11-04T16:10:47Z instance i-02e3aeb5249cd00d5 was selected for termination.	2022 November 04, 11:10:47 AM -05:00	2022 November 04, 11:18:38 AM -05:00

Scaling policies

Application Auto Scaling allows you to automatically scale your scalable resources according to conditions that you define:

1. **Target tracking scaling** - Scale a resource based on a target value for a specific CloudWatch metric. Like the thermostat at your home, it maintains the metric at that threshold.
2. **Step scaling** – With step scaling, you can set complex scaling policies based on custom metrics. It allows you to set custom rules.
3. **Scheduled scaling** - Scale a resource based on the date and time. For example, more instances from 9 am to 5 pm.
4. **Predictive scaling** - Uses machine learning to analyze each resource's historical workload and regularly forecasts the future load for the next two days.

Scale in algorithm

You can define a termination policy in case of scaling in. You can learn more about these kind of advanced concepts in the official documentation.

- By default, it selects the AZ with at least two instances, then terminates the instance that was launched from the oldest launch template.
- If the instances were launched from the same launch template configuration, Amazon EC2 Auto Scaling selects the instance that is closest to the next billing hour and terminates it.

The ALB knows a number of current sessions. So, it waits until all requests are completed and terminate instances gracefully.

Your app should be **stateless**. Because instances are terminated automatically. If you store data (state) in it, you will lose that. For that reason, store those states in separate shared storage such as EFS, S3, DynamoDB, Redis, etc. Storing data in such services is what stateless is.