High Availability & Scalability

Scalability & High Availability

- **Scalability** means that an application/system can handle greater loads by adapting.
- There are two kinds of scalability:
  - **Vertical Scalability**
  - **Horizontal Scalability** (= elasticity)
- **Scalability is linked but different from High Availability**
- Let's deep dive into the distinction, using a call center as an example

Vertical Scalability

- Vertically scalability means increasing the size of the instance.
- For example, your application runs on a **t2.micro**.
- Scaling that application vertically means running it on a **t2.large**.
- Vertical scalability is very common for **non-distributed systems**, such as a **database**.
- **RDS**, **ElastiCache** are services that can scale vertically.
- There's usually a **limit** to how much you can vertically scale (hardware limit).

Horizontal Scalability

- Horizontal Scalability means increasing the number of instances/systems for your application.
- Horizontal scaling implies **distributed systems**.
- This is very common for **web applications / modern applications**.
- It's easy to horizontally scale thanks to cloud offerings such as **Amazon EC2**.

High Availability

- High Availability usually goes hand in hand with **horizontal scaling**.
- High availability means running your application/system in at least **2 data centers** (== **Availability Zones**).
- The goal of high availability is to **survive a data center loss**.
- High availability can be **passive** (e.g. **RDS Multi-AZ**).
- High availability can be **active** (e.g. **horizontal scaling**).

High Availability & Scalability For EC2

- **Vertical Scaling**: Increase instance size (== scale up / down)
  - From: `t2.nano` – 0.5 GB of RAM, 1 vCPU
  - To: `u-12tb1.metal` – 12.3 TB of RAM, 448 vCPUs
- **Horizontal Scaling**: Increase number of instances (== scale out / in)
  - Auto Scaling Group
  - Load Balancer
- **High Availability**: Run instances for the same application across multiple Availability Zones (multi AZ)
  - Auto Scaling Group multi AZ
  - Load Balancer multi AZ

What is load balancing?

- **Load Balancers** are servers that forward traffic to multiple servers (e.g., EC2 instances) downstream.

How it works (diagram explanation):

Users send requests to an **Elastic Load Balancer** (ELB).
The ELB distributes the incoming traffic across multiple **EC2 instances**, ensuring no single instance is overwhelmed.
Each user request may be routed to a different EC2 instance, based on factors like load, availability, and routing algorithm.

Why use a load balancer?

- Spread load across multiple downstream instances
- Expose a single point of access (**DNS**) to your application
- Seamlessly handle failures of downstream instances
- Perform regular **health checks** on your instances
- Provide **SSL termination (HTTPS)** for your websites
- Enforce **stickiness** with cookies
- Enable **high availability** across zones
- Separate **public traffic** from **private traffic**

Why use an Elastic Load Balancer?

- An Elastic Load Balancer is a **managed load balancer**:
    - AWS guarantees that it will be working.
    - AWS takes care of upgrades, maintenance, and high availability.
    - AWS provides only a few configuration knobs.
- It costs less to set up your own load balancer, but it will require a lot more effort on your end.
- It is integrated with many AWS offerings/services:
    - EC2, EC2 Auto Scaling Groups, Amazon ECS
    - AWS Certificate Manager (ACM), CloudWatch
    - Route 53, AWS WAF, AWS Global Accelerator

Health Checks

- Health Checks are crucial for Load Balancers.
- They enable the load balancer to know if instances it forwards traffic to are available to reply to requests.
- The health check is done on a **port** and a **route** (e.g., `/health` is common).
- If the response is not **200 (OK)**, then the instance is considered **unhealthy**.

Types of load balancer on AWS

- AWS has **4 kinds of managed Load Balancers**:
    - **Classic Load Balancer (CLB)** (v1 - old generation, 2009)
        - Supports: HTTP, HTTPS, TCP, SSL (secure TCP)
    - **Application Load Balancer (ALB)** (v2 - new generation, 2016)
        - Supports: HTTP, HTTPS, WebSocket
    - **Network Load Balancer (NLB)** (v2 - new generation, 2017)
        - Supports: TCP, TLS (secure TCP), UDP

    - **Gateway Load Balancer (GWLB)** (2020)
        - Operates at layer 3 (Network layer) – IP Protocol

- It is recommended to use the newer generation load balancers as they provide more features.
- Some load balancers can be set up as **internal** (private) or **external** (public) ELBs.

Load Balancer Security Groups

How traffic flows:

- **Users** connect to the **Load Balancer** over **HTTPS/HTTP** from anywhere.
- The **Load Balancer** forwards requests to **EC2 instances**, but the traffic is **restricted** to come only from the Load Balancer.

Security Group configuration:

Load Balancer Security Group:

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	Allow HTTP from anywhere
HTTPS	TCP	443	0.0.0.0/0	Allow HTTPS from anywhere

Application Security Group (for EC2 instances):

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	sg-054b5ff5ea02f2b6e (Load Balancer SG)	Allow traffic only from Load Balancer

Summary:

- The Load Balancer accepts traffic from the internet.
- The EC2 instances accept traffic **only from the Load Balancer** to increase security.

Classic Load Balancers (v1)

- Supports **TCP** (Layer 4), **HTTP** and **HTTPS** (Layer 7).
- Health checks are **TCP** or **HTTP** based.
- Fixed hostname format: `xxx.region.elb.amazonaws.com`.

How it works (diagram explanation):

- The **Client** connects to the **Classic Load Balancer (CLB)** using a **listener**.
- The **CLB** forwards the traffic internally to the **EC2 instance**.

Application Load Balancer (v2)

- Application Load Balancers operate at **Layer 7** (HTTP).
- Load balancing to multiple HTTP applications across machines (**target groups**).
- Load balancing to multiple applications on the same machine (e.g., **containers**).
- Support for **HTTP/2** and **WebSocket**.

- Support for **redirects** (e.g., from HTTP to HTTPS).

Application Load Balancer (v2) - Advanced Features

- Routing tables to different **target groups**:
  - Based on **path** in URL
    e.g. `example.com/users` and `example.com/posts`
  - Based on **hostname** in URL
    e.g. `one.example.com` and `other.example.com`
  - Based on **query string** or **headers**
    e.g. `example.com/users?id=123&order=false`
- ALBs are a great fit for **microservices** and **container-based applications** (e.g., Docker, Amazon ECS).
- ALB supports **port mapping** to redirect to a dynamic port in ECS.
- Compared to CLB, you'd need multiple Classic Load Balancers per application, while ALB can handle it all in one.

Application Load Balancer (v2) - HTTP Based Traffic

Example of path-based routing:

- The **Application Load Balancer (v2)** receives HTTP requests from the public (e.g. `www`).
- It uses **routing rules** based on the path of the URL to forward traffic to different **target groups**.

Routing logic:

- Requests to `/user` are routed to the **Target Group for Users application**
- Requests to `/search` are routed to the **Target Group for Search application**

Each target group:

- Contains one or more **EC2 instances**
- Has its own **Health Check** configuration to ensure only healthy instances receive traffic

Application Load Balancer (v2) – Target Groups

- **EC2 instances** (can be managed by an Auto Scaling Group) – HTTP
- **ECS tasks** (managed by ECS itself) – HTTP
- **Lambda functions** – HTTP request is translated into a JSON event
- **IP Addresses** – must be **private IPs**

Additional Notes

- ALB can **route to multiple target groups**
- **Health checks** are configured at the **target group level**

Application Load Balancer (v2) – Query Strings/Parameters Routing

- ALB can route traffic based on **query string parameters**.
- This allows fine-grained routing logic, for example based on device type or user preferences.

Example:

- A request to `?Platform=Mobile` is routed to **Target Group 1**, which is hosted on **AWS EC2**.

- A request to `?Platform=Desktop` is routed to **Target Group 2**, which is hosted **on-premises** using **private IP routing**.

Use case:

This feature is useful for:

- Serving different versions of an app (e.g., mobile vs desktop)
- Hybrid architectures mixing AWS and on-prem systems

Application Load Balancer (v2) – Good to Know

- **Fixed hostname** format: `xxx.region.elb.amazonaws.com`
- The **application servers do not see** the client IP directly:
  - The **true client IP** is forwarded in the `X-Forwarded-For` HTTP header
  - You can also retrieve:
    - The original **port** via `X-Forwarded-Port`
    - The original **protocol** via `X-Forwarded-Proto`

Example:

- The **Client IP** (e.g., `12.34.56.78`) connects to the **Load Balancer**
- The Load Balancer forwards the request to the **EC2 instance** using its **private IP**
- The original client IP is preserved using headers, because the connection is **terminated at the Load Balancer**

Network Load Balancer (v2)

- Operates at **Layer 4** (Transport Layer)
- Capable of forwarding **TCP** and **UDP** traffic to your instances
- Designed for:
  - **Handling millions** of requests per second
  - **Ultra-low latency** scenarios

Key features:

- Each NLB has **one static IP per Availability Zone**
- Can assign an **Elastic IP** (useful for IP whitelisting)

Notes:

- NLBs are used when **extreme performance** or **TCP/UDP** protocols are required
- **Not included** in the AWS Free Tier

Network Load Balancer (v2) – TCP (Layer 4) Based Traffic

Example:

- The **External Network Load Balancer (v2)** routes traffic based on **TCP rules**.
- Traffic from **WWW** (the client) is forwarded to the correct **target group** using **TCP or HTTP** protocols.

Routing Example:

- Requests using TCP with specific rules are routed to:
  - **Target Group for Users application** (TCP)

- **Target Group for Search application** (HTTP)

Notes:

- Each target group performs **health checks** on its instances.
- NLB supports **Layer 4 routing**, forwarding raw TCP and UDP traffic with high performance and low latency.

Network Load Balancer – Target Groups

Supported target types:

- **EC2 instances**
- **IP Addresses** – must be **private IPs**
- **Application Load Balancer** (ALB)

Health Checks:

- Health checks support the **TCP**, **HTTP**, and **HTTPS** protocols

Example target groups:

- Target Group (EC2 Instances): `i-1234567890abcdef0`, ...
- Target Group (IP Addresses): `192.168.1.118`, `10.0.4.21`
- Target Group (Application Load Balancer)

Gateway Load Balancer

- Deploy, scale, and manage a fleet of 3rd party network virtual appliances in AWS.
- Example use cases include:
  - Firewalls
  - Intrusion Detection and Prevention Systems (IDPS)
  - Deep Packet Inspection Systems
  - Payload manipulation
- Operates at **Layer 3** (Network Layer), handling **IP packets**.
- Combines the following functions:
  - **Transparent Network Gateway**: a single entry/exit point for all traffic.
  - **Load Balancer**: distributes traffic across your virtual appliances.
- Uses the **GENEVE** protocol on **port 6081**.

Architecture Explanation

Traffic from users is routed through a **Gateway Load Balancer**, based on route table entries (e.g., 172.16.0.0/16).
The load balancer forwards the traffic to a **target group** consisting of **3rd party security virtual appliances**.
After inspection or manipulation, traffic continues to the final **application destination**.

Gateway Load Balancer – Target Groups

- **Target types supported:**
  - **EC2 Instances**
  - **IP Addresses** (must be **private IPs**)

Explanation

A Gateway Load Balancer can route traffic to a target group composed of:

1. **EC2 instances**
   Each target is identified by its instance ID (e.g., `i-1234567890abcdef0`).
2. **Private IP addresses**
   Targets are specified using their private IPs (e.g., `192.168.1.118`, `10.0.4.21`).
   These can point to appliances running outside of AWS, such as in on-premises environments or other VPCs (via AWS Transit Gateway or VPC peering).

This flexibility allows integration of virtual appliances deployed in various locations while keeping the traffic flow managed through a centralized load balancer.

Sticky Sessions (Session Affinity)

- It is possible to implement **stickiness** so that the same client is always redirected to the same instance behind a load balancer.
- Supported by:
  - **Classic Load Balancer (CLB)**
  - **Application Load Balancer (ALB)**
  - **Network Load Balancer (NLB)**
- For both **CLB** and **ALB**, a **cookie** is used to maintain stickiness.
  This cookie has an expiration time that you can configure.
- **Use case:**
  Ensure that a user doesn't lose session data across multiple requests.
- **Important consideration:**
  Enabling stickiness may cause **load imbalance** across backend EC2 instances, as some instances may receive more traffic than others.

Diagram Explanation

- Each client (Client 1, 2, 3) connects to the load balancer.
- Due to stickiness, the load balancer always routes the same client to the same backend EC2 instance.
- For example:
  - Client 1 is always sent to a specific EC2 instance.
  - Client 2 and Client 3 are routed to their own designated instances.

Sticky Sessions – Cookie Names

Application-based Cookies

Custom Cookie

- Generated by the **target**.
- Can include any custom attributes required by the application.
- Cookie name must be specified individually for each target group.
- **Do not use** the following reserved names:
  - `AWSALB`

    - `AWSALBAPP`
    - `AWSALBTG`

Application Cookie

- Generated by the **load balancer**.
- Cookie name is: `AWSALBAPP`

Duration-based Cookies

- Cookie is generated by the **load balancer**.
- Cookie name:
    - `AWSALB` for **Application Load Balancer (ALB)**
    - `AWSELB` for **Classic Load Balancer (CLB)**

Cross-Zone Load Balancing

With Cross-Zone Load Balancing (enabled)

- Each **load balancer node** distributes traffic **evenly across all targets** in **all Availability Zones (AZs)**.
- This results in uniform traffic distribution, regardless of how many targets exist in each AZ.

Example:

- 100 total requests are distributed as:
    - 10 requests to each of the 10 EC2 instances (spread across AZ1 and AZ2)

Without Cross-Zone Load Balancing (disabled)

- Each **load balancer node** distributes traffic **only to targets in its own AZ**.
- This can lead to **unbalanced traffic** if the number of targets differs between AZs.

Example:

- 50 requests go to each AZ's load balancer node.
- In AZ1, 50 requests are distributed among 2 instances (25 each).
- In AZ2, 50 requests are spread over 8 instances (6.25 each).

Key Takeaway

- **Enabling Cross-Zone Load Balancing** helps achieve better load distribution across all targets.
- **Disabling** it may cause **load imbalance** if target group distribution is uneven across AZs.

Cross-Zone Load Balancing

Application Load Balancer (ALB)

- **Enabled by default**
- Can be **disabled at the Target Group level**
- **No charges** for inter-AZ data transfer

Network Load Balancer (NLB) & Gateway Load Balancer (GWLB)

- **Disabled by default**
- **Charges apply** for inter-AZ data transfer **if enabled**

Classic Load Balancer (CLB)

- **Disabled by default**

- **No charges** for inter-AZ data transfer **if enabled**

SSL/TLS – Basics

- An **SSL Certificate** allows traffic between your clients and your load balancer to be **encrypted in transit** (in-flight encryption).
- **SSL** = Secure Sockets Layer
  Used to encrypt connections.
- **TLS** = Transport Layer Security
  Newer version of SSL.
- **TLS certificates are mainly used** nowadays, but the term **SSL** is still commonly used to refer to them.

Certificate Authorities (CA)

- Public SSL certificates are issued by **Certificate Authorities**, such as:
  - Comodo
  - Symantec
  - GoDaddy
  - GlobalSign
  - Digicert
  - Letsencrypt
  - etc.

Expiration

- SSL certificates have an **expiration date** (defined by the user) and **must be renewed**.

Load Balancer – SSL Certificates

- The **load balancer** uses an **X.509 certificate** (SSL/TLS server certificate) to encrypt connections.
- You can manage certificates using **ACM (AWS Certificate Manager)**.
- Alternatively, you can create and upload your own certificates manually.

HTTPS Listener Configuration

- You must **specify a default certificate**.
- You can **add an optional list of certificates** to support multiple domains.
- **Clients can use SNI (Server Name Indication)** to specify the hostname they are trying to reach.
- You can define a **security policy** to support older versions of SSL/TLS (for legacy clients).

Traffic Flow Example

- Users connect to the Load Balancer over **HTTPS** (encrypted).
- The Load Balancer forwards traffic to EC2 instances over **HTTP** (within a private VPC).

SSL – Server Name Indication (SNI)

- **SNI** allows multiple **SSL certificates** to be loaded onto a single web server or load balancer.
- This is useful to **serve multiple domains** using a single listener.
- It's a **newer protocol**, and the **client must indicate the hostname** it wants to reach during the **initial SSL handshake**.
- Based on the hostname, the server selects the **correct SSL certificate**, or falls back to the **default certificate**.

How it works:

- A client requests `www.mycorp.com`.
- The ALB uses SNI to choose the correct SSL certificate (e.g., for `www.mycorp.com`).
- Traffic is then routed to the correct **target group** associated with that hostname.

Notes:

- **Supported by**:
  - **Application Load Balancer (ALB)**
  - **Network Load Balancer (NLB)**
  - **CloudFront**
- **Not supported by**:
  - **Classic Load Balancer (CLB)**

Elastic Load Balancers – SSL Certificates

Classic Load Balancer (CLB – v1)

- Supports **only one SSL certificate**.
- To serve **multiple hostnames with different SSL certificates**, you must **use multiple CLBs**.

Application Load Balancer (ALB – v2)

- Supports **multiple listeners** with **multiple SSL certificates**.
- Uses **Server Name Indication (SNI)** to match the hostname and select the correct certificate.

Network Load Balancer (NLB – v2)

- Also supports **multiple listeners** with **multiple SSL certificates**.
- Uses **SNI** just like ALB to determine the correct certificate based on hostname.

Connection Draining

Feature Naming

- **Classic Load Balancer (CLB):** Connection Draining
- **Application Load Balancer (ALB) & Network Load Balancer (NLB):** Deregistration Delay

Description

- Allows **in-flight requests** to complete before an EC2 instance is deregistered or marked unhealthy.
- While draining, the load balancer **stops sending new requests** to that EC2 instance.
- Time range: **1 to 3600 seconds** (default: **300 seconds**).
- Can be **disabled** by setting the value to `0`.
- Recommended to set a **lower value** if your requests are short-lived.

Use Case

- Ensures a **graceful shutdown** of EC2 instances by waiting for existing connections to finish.
- Prevents disruption to users during deployments or scaling events.

Visual Explanation

- When an instance is in **DRAINING** state:
  - It continues serving **existing connections**.
  - No **new connections** are routed to it.

    - New traffic is routed to other **healthy instances**.

What’s an Auto Scaling Group?

- In real life, the load on your websites and applications can change.
- In the cloud, you can create and terminate servers very quickly.

The goal of an Auto Scaling Group (ASG) is to:

- Scale out (add EC2 instances) to match an increased load.
- Scale in (remove EC2 instances) to match a decreased load.
- Ensure a minimum and a maximum number of EC2 instances are always running.
- Automatically register new instances with a load balancer.
- Re-create an EC2 instance if a previous one is terminated (e.g., if unhealthy).
- ASGs are free (you only pay for the underlying EC2 instances).

Auto Scaling Group in AWS

An Auto Scaling Group (ASG) in AWS manages a group of EC2 instances and automatically adjusts their number based on demand.

Key Concepts:

- **Minimum Capacity**: The minimum number of EC2 instances that should always be running.
- **Desired Capacity**: The ideal number of EC2 instances to handle the current load.
- **Maximum Capacity**: The upper limit of EC2 instances that can be created.
- **Scale Out as Needed**: When the load increases, new EC2 instances are launched up to the maximum capacity.

Auto Scaling Group Attributes

A Launch Template

(Older "Launch Configurations" are deprecated)

Includes the following settings:

- AMI + Instance Type
- EC2 User Data
- EBS Volumes
- Security Groups
- SSH Key Pair
- IAM Roles for your EC2 Instances
- Network + Subnets Information
- Load Balancer Information

Additional Attributes

- Min Size / Max Size / Initial Capacity
- Scaling Policies

Auto Scaling – CloudWatch Alarms & Scaling

- It is possible to scale an Auto Scaling Group (ASG) based on CloudWatch alarms.
- An alarm monitors a metric (such as **Average CPU**, or a **custom metric**).
- Metrics such as **Average CPU** are computed across all instances in the ASG.

Based on the alarm:

- We can create **scale-out policies** (increase the number of instances).
- We can create **scale-in policies** (decrease the number of instances).

CloudWatch alarms can trigger scaling actions automatically based on these metrics.

Auto Scaling Groups – Scaling Policies

Dynamic Scaling

Target Tracking Scaling

- Simple to set up
- Example: I want the average ASG CPU to stay at around 40%

Simple / Step Scaling

- When a CloudWatch alarm is triggered (e.g., CPU > 70%), then add 2 instances
- When a CloudWatch alarm is triggered (e.g., CPU < 30%), then remove 1 instance

Scheduled Scaling

- Anticipate scaling based on known usage patterns
- Example: Increase the minimum capacity to 10 at 5 PM on Fridays

Auto Scaling Groups – Scaling Policies

Predictive Scaling

- **Predictive scaling**: continuously forecast load and schedule scaling actions ahead of time.

How it works:

1. **Analyze historical load**: monitor past traffic patterns.
2. **Generate forecast**: predict future usage based on historical data.
3. **Schedule scaling actions**: automatically plan capacity changes to match predicted demand.

> *The system adjusts capacity before traffic increases, improving responsiveness and efficiency.*

Good Metrics to Scale On

When configuring an Auto Scaling Group (ASG), choosing the right metric is essential to ensure optimal responsiveness and cost-efficiency.

Recommended Metrics:

- **CPUUtilization**
  Measures the **average CPU usage** across your EC2 instances.
  Useful for CPU-bound applications.

- **RequestCountPerTarget**
  Ensures that the **number of requests per instance** stays stable.
  Ideal for web applications behind a load balancer.

- **Average Network In / Out**
  Useful if your application is **network bound**, such as video streaming or file transfer systems.

- **Custom Metrics**
  You can define and push **custom metrics** to **Amazon CloudWatch** (e.g., memory usage, queue depth).
  These allow scaling based on **application-specific** behavior.

Auto Scaling Groups – Scaling Cooldowns

- After a **scaling activity** (scale out or scale in), the Auto Scaling Group enters a **cooldown period**.
- **Default cooldown duration:** `300 seconds` (5 minutes).
- During this period, the ASG **does not launch or terminate** additional instances.
- Purpose: allows **metrics to stabilize** before triggering another scaling event.

Best Practice

- Use a **preconfigured AMI** (Amazon Machine Image) that contains all necessary software and configuration.
- This ensures that new instances:
  - **Start serving requests faster**
  - **Reduce the cooldown period**, improving responsiveness.