AWS Global Infrastructure

* AWS Regions
* AWS Availability Zones
* AWS Data Centers
* AWS Edge Locations / Points of Presence
* https://infrastructure.aws/

AWS Regions:

* AWS has Regions all around the world
* Names can be us-east-1, eu-west-3…
* A region is a cluster of data centers
* Most AWS services are region-scoped.

How to choose an AWS Region?

* Compliance with data governance and legal requirements: data never leaves a region without your explicit permission.
* Proximity to customers: reduced latency
* Available services within a Region: new services and new features aren’t available in every Region
* Pricing: pricing varies region to region and is transparent in the service pricing page

AWS Availability Zones:

* Each region has many availability zones (usually 3, min is 3, max is 6). Example:
  + ap-southeast-2a
  + ap-southeast-2b
  + ap-southeast-2c
* Each availability zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity.
* They’re separate from each other, so that they’re isolated from disasters.
* They’re connected with high bandwidth, ultra-low latency networking.

AWS Points of Presence (Edge Locations)

* Amazon has 400+ Points of Presence (400+ Edge Locations & 10+ Regional Caches) in 90+ cities across 40+ countries
* Content is delivered to end users with lower latency

Tour of the AWS Console:

**AWS has Global Services:**

• Identity and Access Management (IAM)

• Route 53 (DNS service)

• CloudFront (Content Delivery Network)

• WAF (Web Application Firewall)

**Most AWS services are Region-scoped:**

• Amazon EC2 (Infrastructure as a Service)

• Elastic Beanstalk (Platform as a Service)

• Lambda (Function as a Service)

• Rekognition (Software as a Service)

AWS Identity and Access Management (AWS IAM)

**IAM: Users & Groups**

• IAM = Identity and Access Management, Global service

• Root account created by default, shouldn’t be used or shared

• Users are people within your organization, and can be grouped

• Groups only contain users, not other groups

• Users don’t have to belong to a group, and user can belong to multiple groups

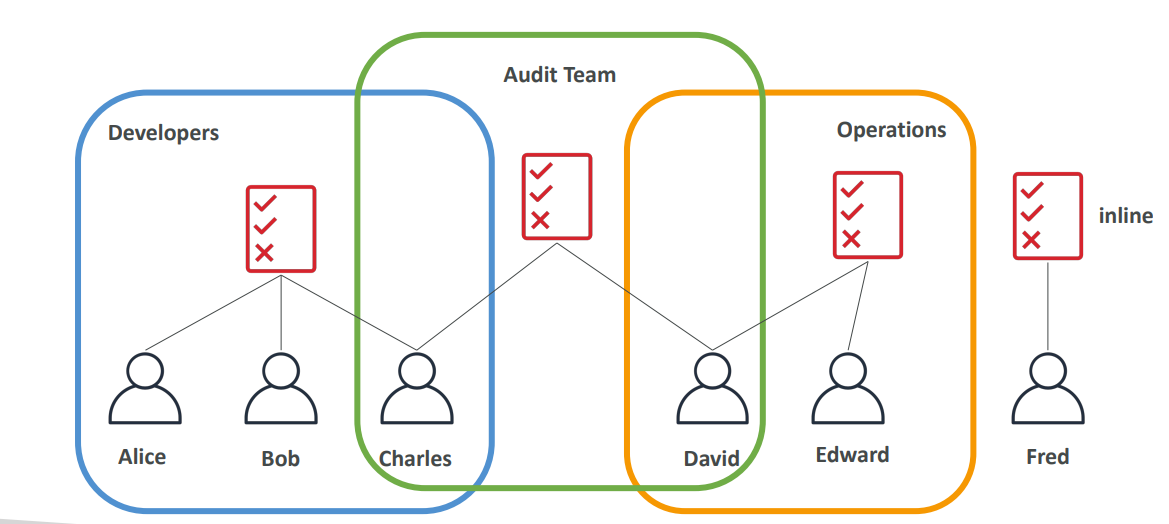
**IAM: Permissions**

• Users or Groups can be assigned JSON documents called policies

• These policies define the permissions of the users

• In AWS you apply the least privilege principle: don’t give more permissions than a user needs.

**IAM Policies inheritance**



**IAM Policy structure:**



**Consists of**

• Version: policy language version, always include “2012 -10 - 17”

• Id: an identifier for the policy (optional)

• Statement: one or more individual statements (required)

**Statements consists of**

• Sid: an identifier for the statement (optional)

• Effect: whether the statement allows or denies access (Allow, Deny)

• Principal: account/user/role to which this policy applied to

• Action: list of actions this policy allows or denies

• Resource: list of resources to which the actions applied to

• Condition: conditions for when this policy is in effect (optional)

**IAM – Password Policy:**

* Strong passwords = higher security for your account. In AWS, you can setup a password policy:
  + Set a minimum password length
  + Require specific character types:
  + including uppercase letters
  + lowercase letters
  + numbers
  + non-alphanumeric characters
* Allow all IAM users to change their own passwords
* Require users to change their password after some time (password expiration)
* Prevent password re-use

**Multi Factor Authentication – MFA:**

• Users have access to your account and can possibly change configurations or delete resources in your AWS account

• You want to protect your Root Accounts and IAM users

• MFA = password you know + security device you own.

• Main benefit of MFA: if a password is stolen or hacked, the account is not compromised.

**How can users access AWS ?**

* To access AWS, you have three options:
  + AWS Management Console (protected by password + MFA)
  + AWS Command Line Interface (CLI): protected by access keys
  + AWS Software Developer Kit (SDK) - for code: protected by access keys
* Access Keys are generated through the AWS Console, Users manage their own access keys
  + Access Keys are secret, just like a password. Don’t share them
  + Access Key ID ~= username
  + Secret Access Key ~= password.

**What’s the AWS CLI?**

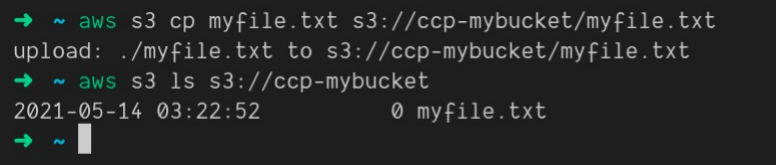
• A tool that enables you to interact with AWS services using commands in your command-line shell

• Direct access to the public APIs of AWS services

• You can develop scripts to manage your resources

• It’s open-source <https://github.com/aws/aws-cli>

• Alternative to using AWS Management Console.



**What’s the AWS SDK?**

AWS Software Development Kit (AWS SDK):

* Language-specific APIs (set of libraries)
* Enables you to access and manage AWS services programmatically
* Embedded within your application

Supports:

* SDKs (JavaScript, Python, PHP, .NET, Ruby, Java, Go, Node.js, C++)
* Mobile SDKs (Android, iOS, …)
* IoT Device SDKs (Embedded C, Arduino, …)

Example: AWS CLI is built on AWS SDK for Python.

**IAM Roles for Services:**

Some AWS service will need to perform actions on your behalf; to do so we will assign permissions to AWS services with IAM Roles

Common roles:

• EC2 Instance Roles

• Lambda Function Roles

• Roles for CloudFormation

**IAM Security Tools:**

IAM Credentials Report (account-level)**:**

A report that lists all your account's users and the status of their various credentials.

IAM Access Advisor (user-level):

* Access advisor shows the service permissions granted to a user and when those services were last accessed.
* You can use this information to revise your policies.

**IAM Guidelines & Best Practices:**

• Don’t use the root account except for AWS account setup

• One physical user = One AWS user

• Assign users to groups and assign permissions to groups

• Create a strong password policy

• Use and enforce the use of Multi Factor Authentication (MFA)

• Create and use Roles for giving permissions to AWS services

• Use Access Keys for Programmatic Access (CLI / SDK)

• Audit permissions of your account using IAM Credentials Report & IAM Access Advisor

• Never share IAM users & Access Keys

**IAM Section – Summary:**

• **Users:** mapped to a physical user, has a password for AWS Console

• **Groups:** contains users only

• **Policies**: JSON document that outlines permissions for users or groups

• **Roles:** for EC2 instances or AWS services

• **Security**: MFA + Password Policy

• **AWS CLI**: manage your AWS services using the command-line

• **AWS SDK**: manage your AWS services using a programming language

• **Access Keys**: access AWS using the CLI or SDK

• **Audit:** IAM Credential Reports & IAM Access Advisor.

**Amazon EC2 – Basics**

**Amazon EC2**

• EC2 is one of the most popular of AWS’ offering

• EC2 = Elastic Compute Cloud = Infrastructure as a Service

• It mainly consists in the capability of:

* Renting virtual machines (EC2)
* Storing data on virtual drives (EBS)
* Distributing load across machines (ELB)
* Scaling the services using an auto-scaling group (ASG)

• Knowing EC2 is fundamental to understand how the Cloud works.

**EC2 sizing & configuration options:**

* Operating System (OS): Linux, Windows or Mac OS
* How much compute power & cores **(CPU)**
* How much random-access memory **(RAM)**
* How much storage space:

Network-attached **(EBS & EFS)**

hardware **(EC2 Instance Store)**

* Network card: speed of the card, Public IP address
* Firewall rules: security group
* Bootstrap script (configure at first launch): EC2 User Data.

**EC2 User Data:**

• It is possible to bootstrap our instances using an EC2 User data script.

• bootstrapping means launching commands when a machine starts

• That script is only run once at the instance first start

EC2 user data is used to automate boot tasks such as:

* + Installing updates
  + Installing software
  + Downloading common files from the internet
  + Anything you can think of

• The EC2 User Data Script runs with the root user

**EC2 Instance Types – Overview:**

• You can use different types of EC2 instances that are optimized for

different use cases (https://aws.amazon.com/ec2/instance-types/)

• AWS has the following naming convention:



• m: instance class

• 5: generation (AWS improves them over time)

• 2xlarge: size within the instance class

**EC2 Instance Types – General Purpose**

• Great for a diversity of workloads such as web servers or code repositories

* + Balance between:
  + Compute • Memory
  + Networking

• In the course, we will be using the t2.micro which is a General Purpose EC2 instance

**EC2 Instance Types – Memory Optimized**

• Fast performance for workloads that process large data sets in memory

• Use cases:

* + High performance, relational/non-relational databases
  + Distributed web scale cache stores
  + In-memory databases optimized for BI (business intelligence)
  + Applications performing real-time processing of big unstructured data.

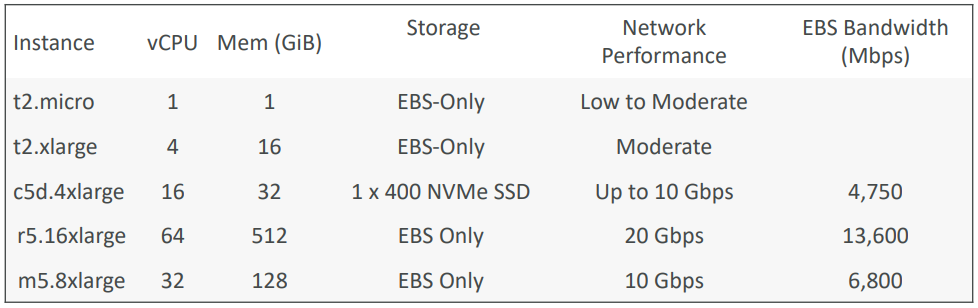
**EC2 Instance Types – Storage Optimized**

• Great for storage-intensive tasks that require high, sequential read and write access to large data sets on local storage

• Use cases:

* + High frequency online transaction processing (OLTP) systems
  + Relational & NoSQL databases
  + Cache for in-memory databases (for example, Redis)
  + Data warehousing applications
  + Distributed file systems

**EC2 Instance Types: example**



**Introduction to Security Groups**

• Security Groups are the fundamental of network security in AWS

• They control how traffic is allowed into or out of our EC2 Instances.

• Security groups only contain allow rules

• Security groups rules can reference by IP or by security group

• Security groups are acting as a “firewall” on EC2 instances

• They regulate:

* + Access to Ports • Authorized IP ranges – IPv4 and IPv6
  + Control of inbound network (from other to the instance)
  + Control of outbound network (from the instance to other)

**Good to know:**

• Can be attached to multiple instances

• Locked down to a region / VPC combination

• Does live “outside” the EC2 – if traffic is blocked the EC2 instance won’t see it

• It’s good to maintain one separate security group for SSH access

• If your application is not accessible (time out), then it’s a security group issue

• If your application gives a “connection refused “error, then it’s an application

error or it’s not launched

• All inbound traffic is blocked by default

• All outbound traffic is authorized by default.

**EC2 Instances Purchasing Options:**

• On-Demand Instances – short workload, predictable pricing, pay by second

• Reserved (1 & 3 years)

• Reserved Instances – long workloads

• Convertible Reserved Instances – long workloads with flexible instances

• Savings Plans (1 & 3 years) –commitment to an amount of usage, long workload

• Spot Instances – short workloads, cheap, can lose instances (less reliable)

• Dedicated Hosts – book an entire physical server, control instance placement

• Dedicated Instances – no other customers will share your hardware

• Capacity Reservations – reserve capacity in a specific AZ for any duration.

**EC2 On Demand:**

• Pay for what you use:

* + Linux or Windows - billing per second, after the first minute
  + All other operating systems - billing per hour

• Has the highest cost but no upfront payment

• No long-term commitment

• Recommended for short-term and un-interrupted workloads, where you can't predict how the application will behave.

**EC2 Reserved Instances:**

* Up to 72% discount compared to On-demand
* You reserve a specific instance attribute (Instance Type, Region, Tenancy, OS)
* Reservation Period – 1 year (+discount) or 3 years (+++discount)
* Payment Options – No Upfront (+), Partial Upfront (++), All Upfront (+++)
* Reserved Instance’s Scope – Regional or Zonal (reserve capacity in an AZ)
* Recommended for steady-state usage applications (think database)
* You can buy and sell in the Reserved Instance Marketplace

**Convertible Reserved Instance**

• Can change the EC2 instance type, instance family, OS, scope and tenancy

• Up to 66% discount.

**EC2 Savings Plans:**

Get a discount based on long-term usage (up to 72% - same as RIs)

• Commit to a certain type of usage ($10/hour for 1 or 3 years)

• Usage beyond EC2 Savings Plans is billed at the On-Demand price

• Locked to a specific instance family & AWS region (e.g., M5 in us-east-1)

• Flexible across:

* + Instance Size (e.g., m5.xlarge, m5.2xlarge)
  + OS (e.g., Linux, Windows)
  + Tenancy (Host, Dedicated, Default)

**EC2 Spot Instances**

Can get a discount of up to 90% compared to On-demand

• Instances that you can “lose” at any point of time if your max price is less than the current spot price

• The MOST cost-efficient instances in AWS

• Useful for workloads that are resilient to failure

* + Batch jobs
  + Data analysis
  + Image processing
  + Any distributed workloads
  + Workloads with a flexible start and end time

• Not suitable for critical jobs or databases

**EC2 Dedicated Hosts**

• A physical server with EC2 instance capacity fully dedicated to your use

• Allows you address compliance requirements and use your existing server- bound software licenses (per-socket, per-core, pe—VM software licenses)

• Purchasing Options:

* + On-demand – pay per second for active Dedicated Host
  + Reserved - 1 or 3 years (No Upfront, Partial Upfront, All Upfront)

• The most expensive option

• Useful for software that have complicated licensing model (BYOL – Bring Your Own License)

• Or for companies that have strong regulatory or compliance needs

**EC2 Dedicated Instances**

• Instances run on hardware that is dedicated to you

• May share hardware with other instances in same account

• No control over instance placement (can move hardware after Stop / Start).

**EC2 Capacity Reservations**

• Reserve On-Demand instances capacity in a specific AZ for any duration

• You always have access to EC2 capacity when you need it

• No time commitment (create/cancel anytime), no billing discounts

• Combine with Regional Reserved Instances and Savings Plans to benefit from billing discounts

• You’re charged at On-Demand rate whether you run instances or not

• Suitable for short-term, uninterrupted workloads that needs to be in a specific AZ.

**Which purchasing option is right for me?**

• On demand: coming and staying in resort whenever we like, we pay the full price

• Reserved: like planning ahead and if we plan to stay for a long time, we may get a good discount.

• Savings Plans: pay a certain amount per hour for certain period and stay in any room type (e.g.,

King, Suite, Sea View, …)

• Spot instances: the hotel allows people to bid for the empty rooms and the highest bidder keeps the

rooms. You can get kicked out at any time

• Dedicated Hosts: We book an entire building of the resort

• Capacity Reservations: you book a room for a period with full price even you don’t stay in it.

**EC2 Spot Instance Requests**

• Can get a discount of up to 90% compared to On-demand

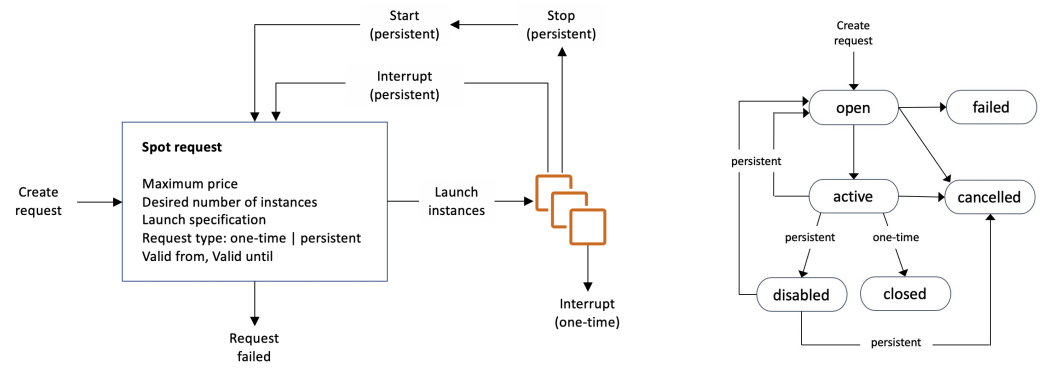
• Define max spot price and get the instance while current spot price < max

* + The hourly spot price varies based on offer and capacity
  + If the current spot price > your max price you can choose to stop or terminate your instance with a 2 minutes grace period.

• Other strategy: Spot Block

* + “block” spot instance during a specified time frame (1 to 6 hours) without interruptions
  + In rare situations, the instance may be reclaimed.

**How to terminate Spot Instances?**



* You can only cancel Spot Instance requests that are open, active, or disabled. Cancelling a Spot Request does not terminate instances You must first cancel a Spot Request, and then terminate the associated Spot Instances

**Spot Fleets**

• Spot Fleets = set of Spot Instances + (optional) On-Demand Instances

• The Spot Fleet will try to meet the target capacity with price constraints

* + Define possible launch pools: instance type (m5.large), OS, Availability Zone
  + Can have multiple launch pools, so that the fleet can choose
  + Spot Fleet stops launching instances when reaching capacity or max cost

• Strategies to allocate Spot Instances:

* + Lowest Price: from the pool with the lowest price (cost optimization, short workload)
  + diversified: distributed across all pools (great for availability, long workloads)
  + capacity Optimized: pool with the optimal capacity for the number of instances
  + price Capacity Optimized (recommended): pools with highest capacity available, then select the pool with the lowest price (best choice for most workloads)

• Spot Fleets allow us to automatically request Spot Instances with the lowest price.

**Private vs Public IP (IPv4) Fundamental Differences**

Public IP:

* + Public IP means the machine can be identified on the internet (WWW)
  + Must be unique across the whole web (not two machines can have the same public IP).
  + Can be geo-located easily

Private IP:

* + Private IP means the machine can only be identified on a private network only
  + The IP must be unique across the private network
  + BUT two different private networks (two companies) can have the same IPs.
  + Machines connect to WWW using a NAT + internet gateway (a proxy)
  + Only a specified range of IPs can be used as private IP

Elastic IPs:

* + When you stop and then start an EC2 instance, it can change its public IP.
  + If you need to have a fixed public IP for your instance, you need an Elastic IP
  + An Elastic IP is a public IPv4 IP you own as long as you don’t delete it
  + You can attach it to one instance at a time
  + With an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account.
  + You can only have 5 Elastic IP in your account (you can ask AWS to increase

that).

Overall, try to avoid using Elastic IP:

* + They often reflect poor architectural decisions
  + Instead, use a random public IP and register a DNS name to it
  + Or, as we’ll see later, use a Load Balancer and don’t use a public IP.

**Placement Groups:**

• Sometimes you want control over the EC2 Instance placement strategy

• That strategy can be defined using placement groups

• When you create a placement group, you specify one of the following strategies for the group:

* + Cluster—clusters instances into a low-latency group in a single Availability Zone
  + Spread—spreads instances across underlying hardware (max 7 instances per group per AZ)
  + Partition—spreads instances across many different partitions (which rely on different sets of racks) within an AZ. Scales to 100s of EC2 instances per group (Hadoop, Cassandra, Kafka)

**Placement Groups Cluster:**

• Pros: Great network (10 Gbps bandwidth between instances with Enhanced

Networking enabled - recommended)

• Cons: If the AZ fails, all instances fails at the same time

• Use case:

* + Big Data job that needs to complete fast
  + Application that needs extremely low latency and high network throughput

**Placement Groups Spread:**

• Pros:

* + Can span across Availability Zones (AZ)
  + Reduced risk is simultaneous failure
  + EC2 Instances are on different physical hardware

• Cons:

* + Limited to 7 instances per AZ per placement group

• Use case:

* + Application that needs to maximize high availability
  + Critical Applications where each instance must be isolated from failure from each other

**Placements Groups Partition:**

• Up to 7 partitions per AZ

• Can span across multiple AZs in the same region

• Up to 100s of EC2 instances

• The instances in a partition do not share racks with the instances in the other partitions

• A partition failure can affect many EC2 but won’t affect other partitions

• EC2 instances get access to the partition information as metadata

• Use cases: HDFS, HBase, Cassandra, Kafka

**Elastic Network Interfaces:**

• Logical component in a VPC that represents a virtual network card

• The ENI can have the following attributes:

* + Primary private IPv4, one or more secondary IPv4
  + One Elastic IP (IPv4) per private IPv4
  + One Public IPv4
  + One or more security groups
  + A MAC address

• You can create ENI independently and attach them on the fly (move them) on EC2 instances for failover

• Bound to a specific availability zone (AZ)

**EC2 Hibernate**

Introducing EC2 Hibernate:

* The in-memory (RAM) state is preserved
* The instance boot is much faster! (the OS is not stopped / restarted)
* Under the hood: the RAM state is written to a file in the root EBS volume
* The root EBS volume must be encrypted

Use cases:

* + Long-running processing
  + Saving the RAM state
  + Services that take time to initialize

**EC2 Hibernate – Good to know**

• Supported Instance Families – C3, C4, C5, I3, M3, M4, R3, R4, T2, T3, …

• Instance RAM Size – must be less than 150 GB.

• Instance Size – not supported for bare metal instances.

• AMI – Amazon Linux 2, Linux AMI, Ubuntu, RHEL, CentOS & Windows…

• Root Volume – must be EBS, encrypted, not instance store, and large.

• Available for On-Demand, Reserved and Spot Instances.

• An instance can NOT be hibernated more than 60 days.

**Amazon EC2 – Instance Storage**

**What’s an EBS Volume?**

• An EBS (Elastic Block Store) Volume is a network drive you can attach to your instances while they run

• It allows your instances to persist data, even after their termination

• They can only be mounted to one instance at a time (at the CCP level)

• They are bound to a specific availability zone

• Analogy: Think of them as a “network USB stick”

• Free tier: 30 GB of free EBS storage of type General Purpose (SSD) or Magnetic per month

**EBS Volume**

• It’s a network drive (i.e. not a physical drive)

* + It uses the network to communicate the instance, which means there might be a bit of latency
  + It can be detached from an EC2 instance and attached to another one quickly

• It’s locked to an Availability Zone (AZ)

* + An EBS Volume in us-east-1a cannot be attached to us-east-1b
  + To move a volume across, you first need to snapshot it

• Have a provisioned capacity (size in GBs, and IOPS)

* + You get billed for all the provisioned capacity
  + You can increase the capacity of the drive over time

**EBS – Delete on Termination attribute**

• Controls the EBS behaviour when an EC2 instance terminates

* + By default, the root EBS volume is deleted (attribute enabled)
  + By default, any other attached EBS volume is not deleted (attribute disabled)

• This can be controlled by the AWS console / AWS CLI

• Use case: preserve root volume when instance is terminated

**EBS Snapshots:**

• Make a backup (snapshot) of your EBS volume at a point in time

• Not necessary to detach volume to do snapshot, but recommended

• Can copy snapshots across AZ or Region.

**EBS Snapshots Features**

• EBS Snapshot Archive

* + Move a Snapshot to an” archive tier” that is 75% cheaper • Takes within 24 to 72 hours for restoring the archive

• Recycle Bin for EBS Snapshots

* + Setup rules to retain deleted snapshots so you can recover them after an accidental deletion
  + Specify retention (from 1 day to 1 year)

• Fast Snapshot Restore (FSR)

* + Force full initialization of snapshot to have no latency on the first use ($$$)

**AMI Overview**

• AMI = Amazon Machine Image

• AMI are a customization of an EC2 instance

* + You add your own software, configuration, operating system, monitoring…
  + Faster boot / configuration time because all your software is pre-packaged

• AMI are built for a specific region (and can be copied across regions)

• You can launch EC2 instances from:

* + A Public AMI: AWS provided
  + Your own AMI: you make and maintain them yourself
  + An AWS Marketplace AMI: an AMI someone else made (and potentially sells)

**AMI Process (from an EC2 instance)**

• Start an EC2 instance and customize it

• Stop the instance (for data integrity)

• Build an AMI – this will also create EBS snapshots

• Launch instances from other AMIs

**EC2 Instance Store:**

* EBS volumes are network drives with good but “limited” performance
* If you need a high-performance hardware disk, use EC2 Instance Store
* Better I/O performance
* EC2 Instance Store lose their storage if they’re stopped (ephemeral)
* Good for buffer / cache / scratch data / temporary content
* Risk of data loss if hardware fails
* Backups and Replication are your responsibility

**EBS Volume Types**

EBS Volumes come in 6 types

* + **gp2 / gp3 (SSD):** General purpose SSD volume that balances price and performance for a wide variety of workloads
  + **io1 / io2 Block Express (SSD):** Highest-performance SSD volume for mission-critical low-latency or high-throughput workloads
  + **st1 (HDD):** Low-cost HDD volume designed for frequently accessed, throughput- intensive workloads
  + **sc1 (HDD):** Lowest cost HDD volume designed for less frequently accessed workloads

• EBS Volumes are characterized in Size | Throughput | IOPS (I/O Ops Per Sec)

• When in doubt always consult the AWS documentation – it’s good!

• Only gp2/gp3 and io1/io2 Block Express can be used as boot volumes.

**EBS Volume Types Use cases**

General Purpose SSD

• Cost effective storage, low-latency

• System boot volumes, Virtual desktops, Development and test environments

• 1 GiB - 16 TiB

• gp3:

* Baseline of 3,000 IOPS and throughput of 125 MiB/s
* Can increase IOPS up to 16,000 and throughput up to 1000 MiB/s independently

• gp2:

* + Small gp2 volumes can burst IOPS to 3,000
  + Size of the volume and IOPS are linked, max IOPS is 16,000
  + 3 IOPS per GB, means at 5,334 GB we are at the max IOPS

**EBS Volume Types Use cases:**

Provisioned IOPS (PIOPS) SSD

• Critical business applications with sustained IOPS performance

• Or applications that need more than 16,000 IOPS

• Great for databases workloads (sensitive to storage perf and consistency)

• io1 (4 GiB - 16 TiB):

* + Max PIOPS: 64,000 for Nitro EC2 instances & 32,000 for other
  + Can increase PIOPS independently from storage size

• io2 Block Express (4 GiB – 64 TiB):

* + Sub-millisecond latency
  + Max PIOPS: 256,000 with an IOPS:GiB ratio of 1,000:1
  + Supports EBS Multi-attach

**EBS Volume Types Use cases:**

Hard Disk Drives (HDD)

• Cannot be a boot volume

• 125 GiB to 16 TiB

• Throughput Optimized HDD (st1)

* + Big Data, Data Warehouses, Log Processing
  + Max throughput 500 MiB/s – max IOPS 500

• Cold HDD (sc1):

* + For data that is infrequently accessed
  + Scenarios where lowest cost is important • Max throughput 250 MiB/s – max IOPS 250

**EBS Multi-Attach – io1/io2 family**

• Attach the same EBS volume to multiple EC2 instances in the same AZ

• Each instance has full read & write permissions to the high-performance volume

• Use case:

• Achieve higher application availability in clustered Linux applications (ex: Teradata)

• Applications must manage concurrent write operations

• Up to 16 EC2 Instances at a time

• Must use a file system that’s cluster-aware (not XFS, EXT4, etc…)

**EBS Encryption**

• When you create an encrypted EBS volume, you get the following:

* + Data at rest is encrypted inside the volume
  + All the data in flight moving between the instance and the volume is encrypted
  + All snapshots are encrypted
  + All volumes created from the snapshot

• Encryption and decryption are handled transparently (you have nothing to do)

• Encryption has a minimal impact on latency

• EBS Encryption leverages keys from KMS (AES-256)

• Copying an unencrypted snapshot allows encryption

• Snapshots of encrypted volumes are encrypted

**Encryption: encrypt an unencrypted EBS volume**

• Create an EBS snapshot of the volume

• Encrypt the EBS snapshot ( using copy )

• Create new ebs volume from the snapshot ( the volume will also be

encrypted )

• Now you can attach the encrypted volume to the original instance

**Amazon EFS – Elastic File System**

• Managed NFS (network file system) that can be mounted on many EC2

• EFS works with EC2 instances in multi-AZ

• Highly available, scalable, expensive (3x gp2), pay per use

**Amazon EFS – Elastic File System**

• Use cases: content management, web serving, data sharing, Wordpress

• Uses NFSv4.1 protocol

• Uses security group to control access to EFS

• Compatible with Linux based AMI (not Windows)

• Encryption at rest using KMS

• POSIX file system (~Linux) that has a standard file API

• File system scales automatically, pay-per-use, no capacity planning!

**EFS – Performance & Storage Classes**

• EFS Scale

* + 1000s of concurrent NFS clients, 10 GB+ /s throughput
  + Grow to Petabyte-scale network file system, automatically

• Performance Mode (set at EFS creation time)

* + General Purpose (default) – latency-sensitive use cases (web server, CMS, etc…)
  + Max I/O – higher latency, throughput, highly parallel (big data, media processing)

• Throughput Mode

* + Bursting – 1 TB = 50MiB/s + burst of up to 100MiB/s
  + Provisioned – set your throughput regardless of storage size, ex: 1 GiB/s for 1 TB storage
  + Elastic – automatically scales throughput up or down based on your workloads

Up to 3GiB/s for reads and 1GiB/s for writes

Used for unpredictable workloads

**EFS – Storage Classes**

• Storage Tiers (lifecycle management feature – move file after N days)

* + Standard: for frequently accessed files
  + Infrequent access (EFS-IA): cost to retrieve files, lower price to store.
  + Archive: rarely accessed data (few times each year), 50% cheaper
  + Implement lifecycle policies to move files between storage tiers

• Availability and durability

* + Standard: Multi-AZ, great for prod
  + One Zone: One AZ, great for dev, backup enabled by default, compatible with IA (EFS One Zone-IA)

• Over 90% in cost savings

|  |  |  |
| --- | --- | --- |
| **Feature** | **EFS (Elastic File System)** | **EBS (Elastic Block Store)** |
| **Storage Type** | Network File System (NFS) | Block Storage |
| **Use Case** | Shared storage for multiple EC2 instances | Persistent storage for a single EC2 instance |
| **Performance** | Scales automatically, but higher latency | Low-latency, high-performance storage |
| **Throughput** | Scalable throughput, based on usage | Provisioned throughput with IOPS options |
| **Latency** | ~5–10 ms (higher due to network overhead) | ~1 ms (low-latency, local to instance) |
| **Mounting** | Can be mounted on multiple EC2 instances | Attached to one EC2 instance at a time |
| **Scalability** | Automatically scales (no capacity planning) | Must be manually resized (up to 64 TiB per volume) |
| **Availability** | Multi-AZ, highly available | Limited to a single Availability Zone (unless using snapshots) |
| **Durability** | Designed for 99.999999999% (11 nines) durability | 99.999% durability (Snapshots for backup) |
| **Pricing** | Pay per GB stored & access charges | Pay per GB provisioned & IOPS provisioned |
| **Backup & Snapshots** | AWS Backup integration, no native snapshots | Supports automated & manual snapshots |
| **Common Use Cases** | Shared file storage, web servers, container storage, big data workloads | Databases, boot volumes, high-performance applications |

**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 to High Availability

**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, Elastic Cache 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 the cloud offerings such as Amazon EC2.

**High Availability**

High Availability first building in New York • High Availability usually goes hand in

hand with horizontal scaling

• High availability means running your application / system in at least 2 datacenters (== Availability Zones)

• The goal of high availability is to survive a data center loss

• The high availability can be passive (for RDS Multi AZ for example)

• The high availability can be active (for horizontal scaling)

**High Availability & Scalability For EC2**

• Vertical Scaling: Increase instance size (= scale up / down)

* + From: t2-nano - 0.5G of RAM, 1 vCPU To: u-12tb1.metal – 12.3 TB of RAM, 448 vCPUs

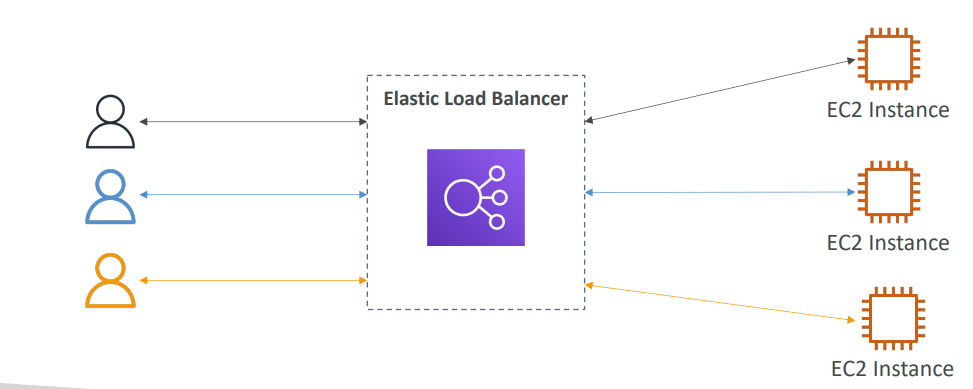
• Horizontal Scaling: Increase number of instances (= scale out / in)

* + Auto Scaling Group
  + Load Balancer

• High Availability: Run instances for the same application across multi-AZ

* + Auto Scaling Group multi-AZ
  + Load Balancer multi-AZ

**What is load balancing?**

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

**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

• Do regular health checks to your instances

• Provide SSL termination (HTTPS) for your websites

• Enforce stickiness with cookies

• 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, high availability
  + AWS provides only a few configurations knobs

• It costs less to setup your own load balancer but it will be 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 (/health is common)

• If the response is not 200 (OK), then the instance is unhealthy

**Types of load balancer on AWS**

• AWS has 4 kinds of managed Load Balancers

• Classic Load Balancer (v1 - old generation) – 2009 – CLB

* + HTTP, HTTPS, TCP, SSL (secure TCP)

• Application Load Balancer (v2 - new generation) – 2016 – ALB

* + HTTP, HTTPS, WebSocket

• Network Load Balancer (v2 - new generation) – 2017 – NLB

TCP, TLS (secure TCP), UDP

• Gateway Load Balancer – 2020 – GWLB

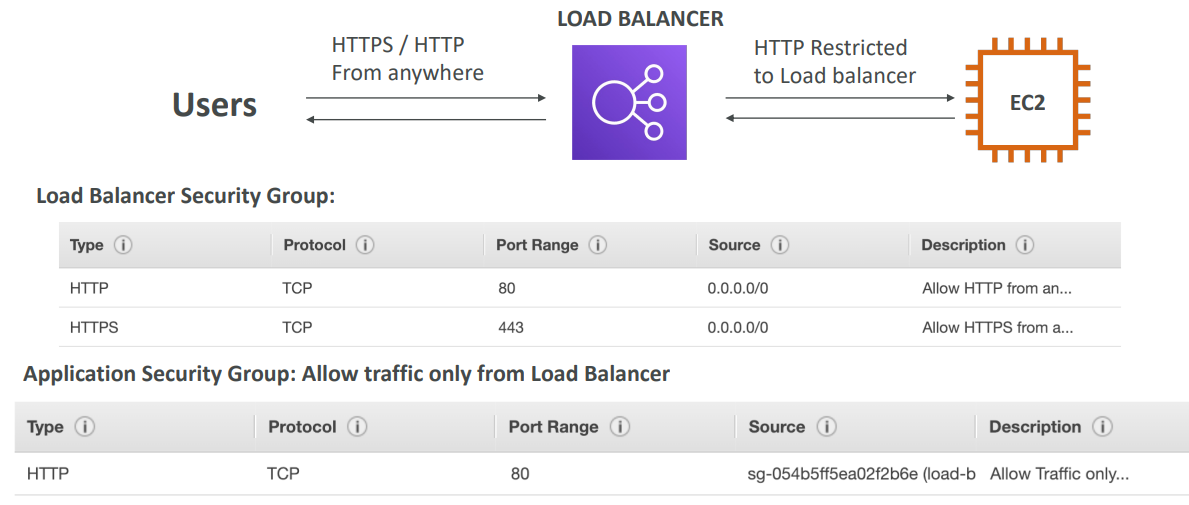
* + Operates at layer 3 (Network layer) – IP Protocol

• Overall, it is recommended to use the newer generation load balancers as they

provide more features

• Some load balancers can be setup as internal (private) or external (public) ELBs.

**Load Balancer Security Groups**

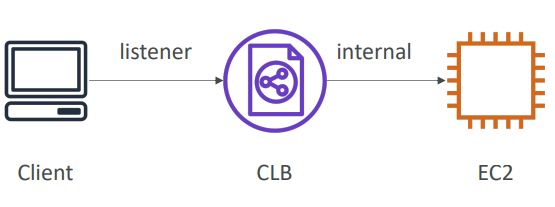


**Classic Load Balancers (v1)**

• Supports TCP (Layer 4), HTTP & HTTPS (Layer 7)

• Health checks are TCP or HTTP based

• Fixed hostname XXX.region.elb.amazonaws.com



**Application Load Balancer (v2)**

• Application load balancers is Layer 7 (HTTP)

• Load balancing to multiple HTTP applications across machines (target groups)

• Load balancing to multiple applications on the same machine (ex: containers)

• Support for HTTP/2 and WebSocket

• Support redirects (from HTTP to HTTPS for example)

**Application Load Balancer (v2)**

• Routing tables to different target groups:

* + Routing based on path in URL (example.com/users & example.com/posts)
  + Routing based on hostname in URL (one.example.com & other.example.com)
  + Routing based on Query String, Headers

(example.com/users?id=123&order=false)

• ALB are a great fit for micro services & container-based application (example: Docker & Amazon ECS)

• Has a port mapping feature to redirect to a dynamic port in ECS

• In comparison, we’d need multiple Classic Load Balancer per application.

**Application Load Balancer (v2) Target Groups**

• EC2 instances (can be managed by an Auto Scaling Group) – HTTP

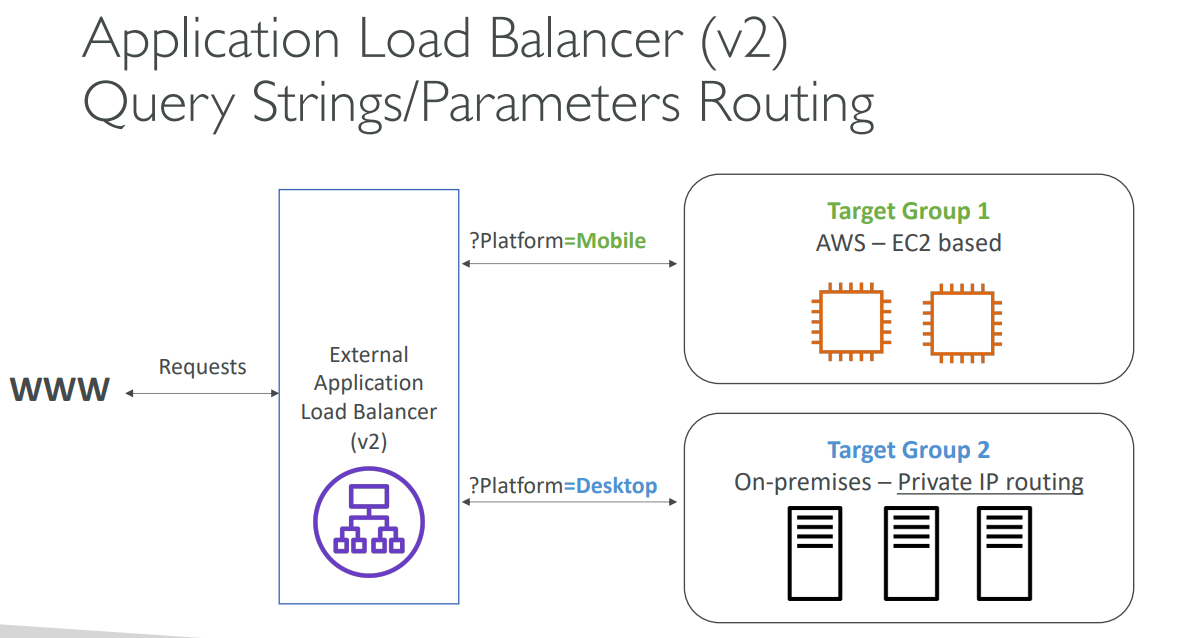
• ECS tasks (managed by ECS itself) – HTTP

• Lambda functions – HTTP request is translated into a JSON event

• IP Addresses – must be private IPs

• ALB can route to multiple target groups

• Health checks are at the target group level



**Application Load Balancer (v2) Good to Know**

• Fixed hostname (XXX.region.elb.amazonaws.com)

• The application servers don’t see the IP of the client directly

• The true IP of the client is inserted in the header X-Forwarded-For

• We can also get Port (X-Forwarded-Port) and proto (X-Forwarded-Proto)

**Network Load Balancer (v2)**

• Network load balancers (Layer 4) allow to:

* + Forward TCP & UDP traffic to your instances
  + Handle millions of requests per seconds
  + Ultra-low latency

• NLB has one static IP per AZ, and supports assigning Elastic IP (helpful for whitelisting specific IP)

• NLB are used for extreme performance, TCP or UDP traffic

• Not included in the AWS free tier.

**Advantages of Sending Traffic First to NLB and Then to ALB**

Static IP Addresses for Whitelisting:

NLBs provide static IP addresses per Availability Zone (AZ) and support Elastic IPs. This is useful when you need to whitelist specific IP addresses (e.g., for third-party integrations or security policies). ALBs, on the other hand, use dynamic IPs, which can change over time. By placing an NLB in front, you can expose a fixed IP address to the internet or external systems. Ultra-Low Latency for Initial Connection:

NLBs operate at Layer 4 (Transport Layer) and are optimized for ultra-low latency. This makes them ideal for handling the initial connection from the client. Once the traffic reaches the ALB, the ALB can handle Layer 7 (Application Layer) tasks like content-based routing, SSL termination, and advanced request processing.

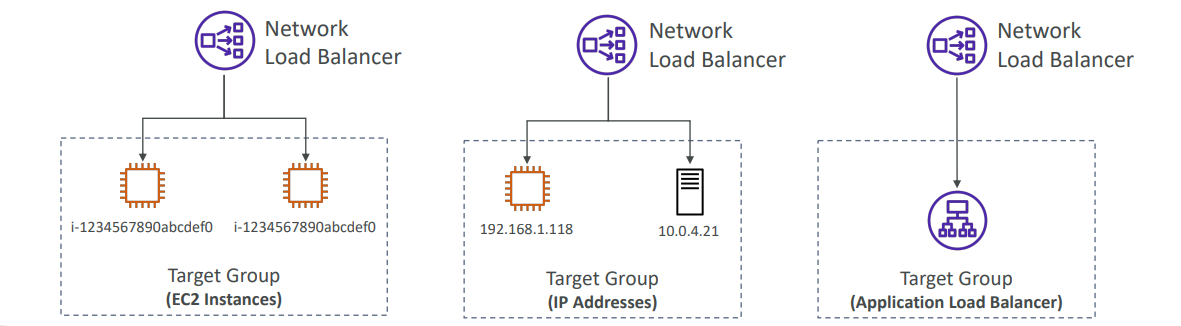
**Network Load Balancer –Target Groups**

• EC2 instances

• IP Addresses – must be private IPs

• Application Load Balancer

• Health Checks support the TCP, HTTP and HTTPS Protocols



**Gateway Load Balancer**

• Deploy, scale, and manage a fleet of 3rd party network virtual appliances in AWS

* + Example: Firewalls, Intrusion Detection and Prevention Systems, Deep Packet Inspection Systems, payload manipulation, …

• Operates at Layer 3 (Network Layer) – IP Packets

• Combines the following functions:

* + Transparent Network Gateway – single entry/exit for all traffic
  + Load Balancer – distributes traffic to your virtual appliances

• Uses the GENEVE protocol on port 6081

**Gateway Load Balancer –Target Groups**

• EC2 instances

• IP Addresses – must be private Ips

**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

• This works for Classic Load Balancer, Application Load Balancer, and Network Load Balancer

• For both CLB & ALB, the “cookie” used for stickiness has an expiration date you control

• Use case: make sure the user doesn’t lose his session data

• Enabling stickiness may bring imbalance to the load over the backend EC2 instances.

**Sticky Sessions – Cookie**

Sticky sessions (also known as session persistence) ensure that a user is consistently routed to the same backend server for a defined period. This is crucial for stateful applications where session data needs to be maintained on a specific target. There are two primary types of sticky sessions based on cookie handling: application-based cookies and duration-based cookies.

**1.Application-Based Cookies**

Application-based sticky sessions rely on cookies managed by the application or target instances rather than the load balancer. This approach provides flexibility since the application can set custom attributes in the cookie.

**Key Characteristics:**

* Custom Cookie: The application or target instance generates the cookie rather than the load balancer.
* Custom Attributes: The cookie can store additional data required by the application.
* Target-Specific Cookie Naming: Each target group requires a unique cookie name to be specified.

**How It Works:**

The application generates a custom cookie when a client makes a request.

The cookie is sent back to the client.

On subsequent requests, the client includes this cookie.

The load balancer checks the cookie and routes the request to the same target.

**2. Application Cookie (AWSALBAPP)**

This is a specialized application-based cookie generated by the Application Load Balancer (ALB) rather than the application itself.

**Key Characteristics:**

Managed by ALB: The ALB creates and manages the AWSALBAPP cookie.

Standardized Cookie Name: The name is always AWSALBAPP.

Ensures Session Stickiness: The cookie is used to consistently route requests from the same client to the same target.

**How It Works:**

The ALB sets the AWSALBAPP cookie for the first request.

The client includes this cookie in subsequent requests.

The ALB directs future requests from that client to the same target.

**3. Duration-Based Cookies**

Duration-based sticky sessions use cookies that expire after a configured duration, ensuring that clients remain connected to the same backend server for that period. These cookies are managed by the load balancer.

**Key Characteristics:**

Cookie is Generated by the Load Balancer

Fixed Duration: Configurable duration after which the session expires.

Standardized Cookie Names:

Application Load Balancer (ALB): Uses the cookie name AWSALB.

Classic Load Balancer (CLB): Uses the cookie name AWSELB.

**How It Works:**

The load balancer assigns a AWSALB or AWSELB cookie to a client upon first request.

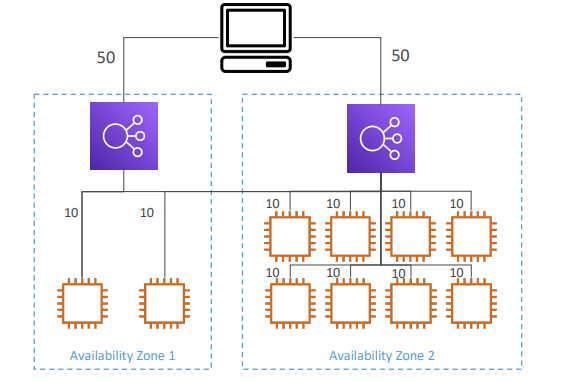
The client includes this cookie in future requests.

The load balancer ensures all subsequent requests with this cookie are sent to the same backend server.

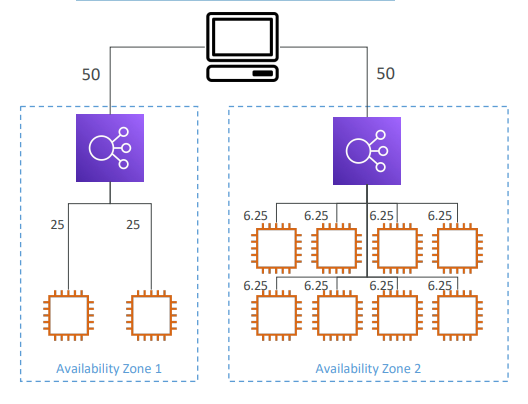
Once the cookie expires, a new target may be selected for future requests.

**Cross-Zone Load Balancing**

With Cross Zone Load Balancing: each load balancer instance distributes evenly across all registered instances in all AZ.



Without Cross Zone Load Balancing: Requests are distributed in the instances of the node of the Elastic Load Balancer



**Cross-Zone Load Balancing**

• Application Load Balancer

* + Enabled by default (can be disabled at the Target Group level)
  + No charges for inter AZ data

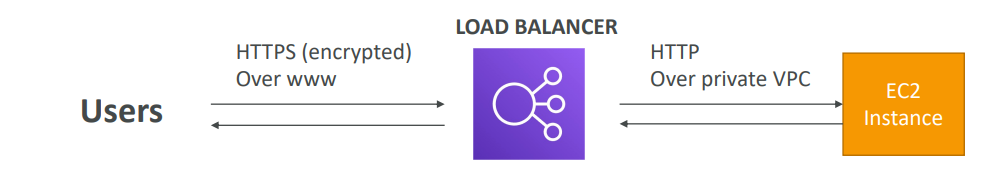
• Network Load Balancer & Gateway Load Balancer

* + Disabled by default
  + You pay charges ($) for inter AZ data if enabled

• Classic Load Balancer

* + Disabled by default
  + No charges for inter AZ data if enabled.

**Load Balancer - SSL Certificates**



• The load balancer uses an X.509 certificate (SSL/TLS server certificate)

• You can manage certificates using ACM (AWS Certificate Manager)

• You can create upload your own certificates alternatively

• HTTPS listener:

* + You must specify a default certificate
  + You can add an optional list of certs to support multiple domains
  + Clients can use SNI (Server Name Indication) to specify the hostname they reach
  + Ability to specify a security policy to support older versions of SSL / TLS (legacy clients)

**SSL – Server Name Indication (SNI)**

• SNI solves the problem of loading multiple SSL certificates onto one web server (to serve multiple websites)

• It’s a “newer” protocol, and requires the client to indicate the hostname of the target server in the initial SSL handshake

• The server will then find the correct certificate, or return the default one

Note:

* + Only works for ALB & NLB (newer generation), CloudFront
  + Does not work for CLB (older gen)

**Elastic Load Balancers – SSL Certificates**

• Classic Load Balancer (v1)

* + Support only one SSL certificate
  + Must use multiple CLB for multiple hostname with multiple SSL certificates

• Application Load Balancer (v2)

* + Supports multiple listeners with multiple SSL certificates
  + Uses Server Name Indication (SNI) to make it work

• Network Load Balancer (v2)

* + Supports multiple listeners with multiple SSL certificates
  + Uses Server Name Indication (SNI) to make it work

**Connection Draining / Deregistration Delay in AWS**

When an EC2 instance behind a load balancer is being de-registered (removed from the target group) or is marked as unhealthy, AWS provides a mechanism to ensure that ongoing requests are gracefully completed before shutting down the instance. This mechanism is called Connection Draining in Classic Load Balancer (CLB) and Deregistration Delay in Application Load Balancer (ALB) and Network Load Balancer (NLB).

• Feature naming

* + Connection Draining – for CLB
  + Deregistration Delay – for ALB & NLB

• Time to complete “in-flight requests” while the instance is de-registering or unhealthy

• Stops sending new requests to the EC2 instance which is de-registering

• Between 1 to 3600 seconds (default: 300 seconds)

• Can be disabled (set value to 0)

• Set to a low value if your requests are short.

**What’s an Auto Scaling Group?**

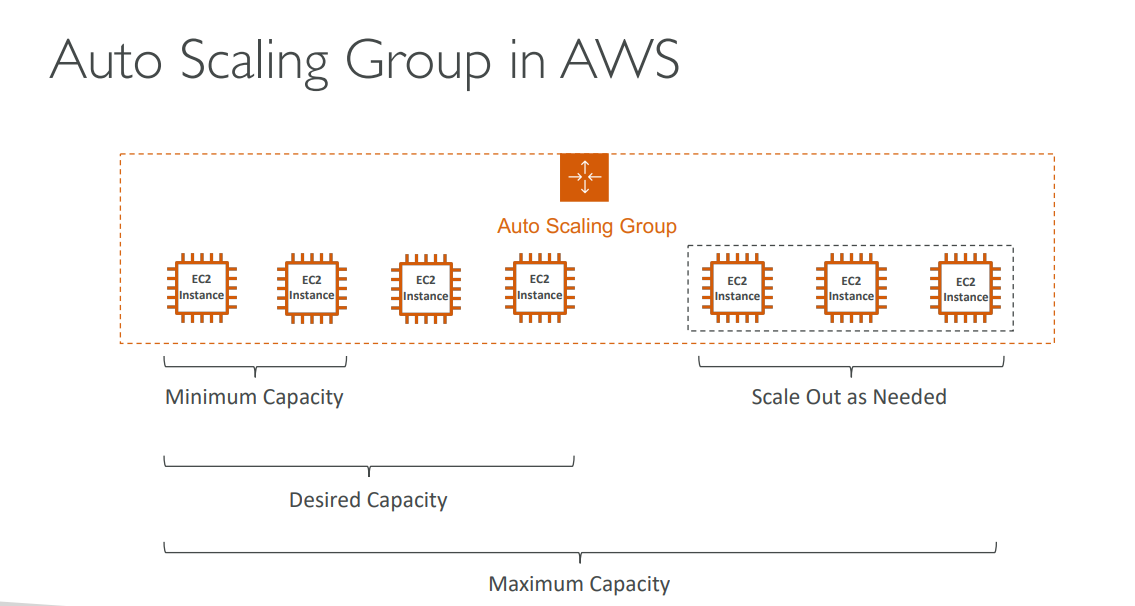
• In real-life, the load on your websites and application can change

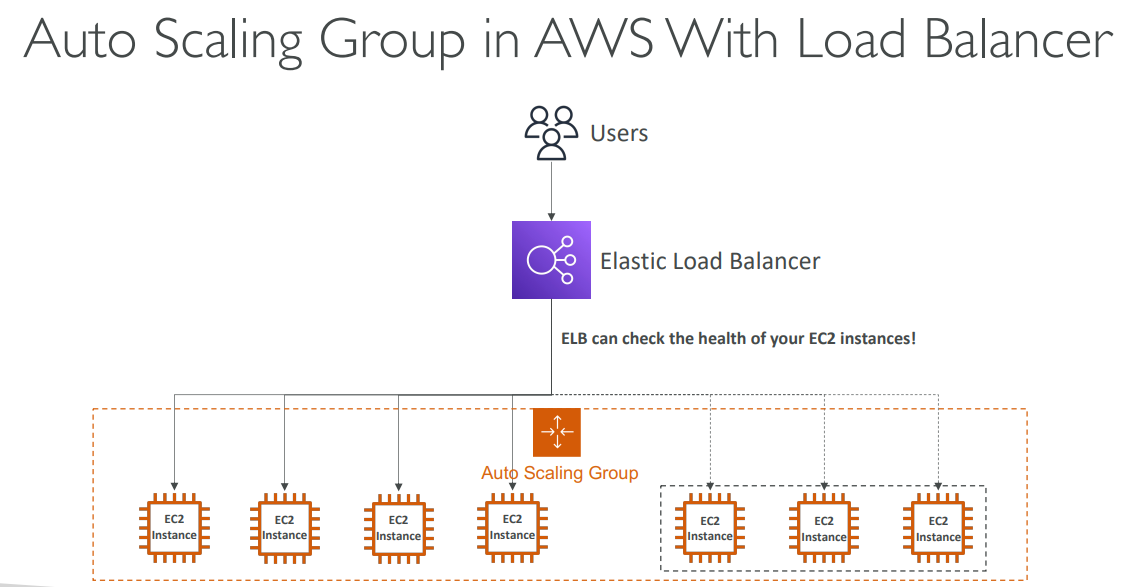
• In the cloud, you can create and get rid of servers very quickly

• 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 we have a minimum and a maximum number of EC2 instances running
  + Automatically register new instances to a load balancer
  + Re-create an EC2 instance in case a previous one is terminated (ex: if unhealthy)

• ASG are free (you only pay for the underlying EC2 instances)





**Auto Scaling Group Attributes**

• A Launch Template (older “Launch Configurations” are deprecated)

* + 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

• Min Size / Max Size / Initial Capacity

• Scaling Policies

**Auto Scaling - CloudWatch Alarms & Scaling**

• It is possible to scale an 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 for the overall ASG instances

• 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).

**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 (example CPU > 70%), then add 2 units
  + When a CloudWatch alarm is triggered (example CPU < 30%), then remove 1

• Scheduled Scaling

* + Anticipate a scaling based on known usage patterns
  + Example: increase the min capacity to 10 at 5 pm on Fridays

**Auto Scaling Groups – Scaling Policies**

• Predictive scaling: continuously forecast load and schedule scaling ahead

**Good metrics to scale on**

• CPUUtilization: Average CPU utilization across your instances

• RequestCountPerTarget: to make sure the number of requests per EC2 instances is stable

• Average Network In / Out (if you’re application is network bound)

• Any custom metric (that you push using CloudWatch).

**Auto Scaling Groups - Scaling Cooldowns**

• After a scaling activity happens, you are in the cooldown period (default 300 seconds)

• During the cooldown period, the ASG will not launch or terminate additional instances (to allow for metrics to stabilize)

• Advice: Use a ready-to-use AMI to reduce configuration time in order to be serving request fasters and reduce the cooldown period.

**RDS, Aurora & ElastiCache**

**Amazon RDS Overview.**

• RDS stands for Relational Database Service

• It’s a managed DB service for DB use SQL as a query language.

• It allows you to create databases in the cloud that are managed by AWS

* + Postgres
  + MySQL
  + MariaDB
  + Oracle
  + Microsoft SQL Server
  + IBM DB2
  + Aurora (AWS Proprietary database).

**Advantage over using RDS versus deploying DB on EC2.**

• RDS is a managed service:

* + Automated provisioning, OS patching
  + Continuous backups and restore to specific timestamp (Point in Time Restore)!
  + Monitoring dashboards
  + Read replicas for improved read performance
  + Multi AZ setup for DR (Disaster Recovery)
  + Maintenance windows for upgrades
  + Scaling capability (vertical and horizontal)
  + Storage backed by EBS

• BUT you can’t SSH into your instances

**RDS– Storage Auto Scaling**

• Helps you increase storage on your RDS DB instance dynamically

• When RDS detects you are running out of free database storage, it scales automatically

• Avoid manually scaling your database storage

• You have to set Maximum Storage Threshold (maximum limit for DB storage)

• Automatically modify storage if:

* + Free storage is less than 10% of allocated storage
  + Low-storage lasts at least 5 minutes
  + 6 hours have passed since last modification

• Useful for applications with unpredictable workloads

• Supports all RDS database engines

**RDS Read Replicas for read scalability**

• Up to 15 Read Replicas

• Within AZ, Cross AZ or Cross Region

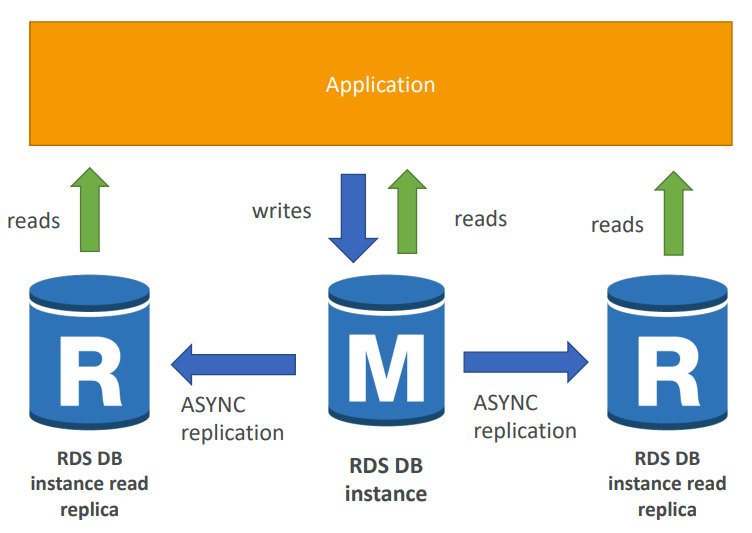
• Replication is ASYNC, so reads are eventually consistent

• Replicas can be promoted to their own DB

• Applications must update the connection string to leverage read replicas.

• In AWS there’s a network cost when data goes from one AZ to another

• For RDS Read Replicas within the same region, you don’t pay that fee



**RDS Read Replicas – Use Cases**

• You have a production database that is taking on normal load

• You want to run a reporting application to run some analytics

• You create a Read Replica to run the new workload there

• The production application is unaffected

• Read replicas are used for SELECT (=read) only kind of statements (not INSERT, UPDATE, DELETE).

**RDS Multi AZ (Disaster Recovery)**

• SYNC replication

• One DNS name – automatic app failover to standby

• Increase availability

• Failover in case of loss of AZ, loss of network, instance or storage failure

• No manual intervention in apps

• Not used for scaling

• Note: The Read Replicas be setup as Multi AZ for Disaster Recovery (DR).

**RDS – From Single-AZ to Multi-AZ**

• Zero downtime operation (no need to stop the DB)

• Just click on “modify” for the database

• The following happens internally:

• A snapshot is taken

• A new DB is restored from the snapshot in a new AZ

• Synchronization is established between the two databases.

**RDS Custom**

• Managed Oracle and Microsoft SQL Server Database with OS and database customization

• RDS: Automates setup, operation, and scaling of database in AWS

• Custom: access to the underlying database and OS so you can

* + Configure settings
  + Install patches
  + Enable native features
  + Access the underlying EC2 Instance using SSH or SSM Session Manager

• De-activate Automation Mode to perform your customization, better to take a DB snapshot before

• RDS vs. RDS Custom

* + RDS: entire database and the OS to be managed by AWS
  + RDS Custom: full admin access to the underlying OS and the database

**Amazon Aurora**

• Aurora is a proprietary technology from AWS (not open sourced)

• Postgres and MySQL are both supported as Aurora DB (that means your drivers will work as if Aurora was a Postgres or MySQL database)

• Aurora is “AWS cloud optimized” and claims 5x performance improvement over MySQL on RDS, over 3x the performance of Postgres on RDS

• Aurora storage automatically grows in increments of 10GB, up to 128 TB.

• Aurora can have up to 15 replicas and the replication process is faster than MySQL (sub 10 ms replica lag)

• Failover in Aurora is instantaneous. It’s HA (High Availability) native.

• Aurora costs more than RDS (20% more) – but is more efficient.

**Aurora High Availability and Read Scaling**

• 6 copies of your data across 3 AZ:

* + 4 copies out of 6 needed for writes
  + 3 copies out of 6 need for reads
  + Self-healing with peer-to-peer replication
  + Storage is striped across 100s of volumes

• One Aurora Instance takes writes (master)

• Automated failover for master in less than 30 seconds

• Master + up to 15 Aurora Read Replicas serve reads

• Support for Cross Region Replication

**Features of Aurora**

• Automatic fail-over

• Backup and Recovery

• Isolation and security

• Industry compliance

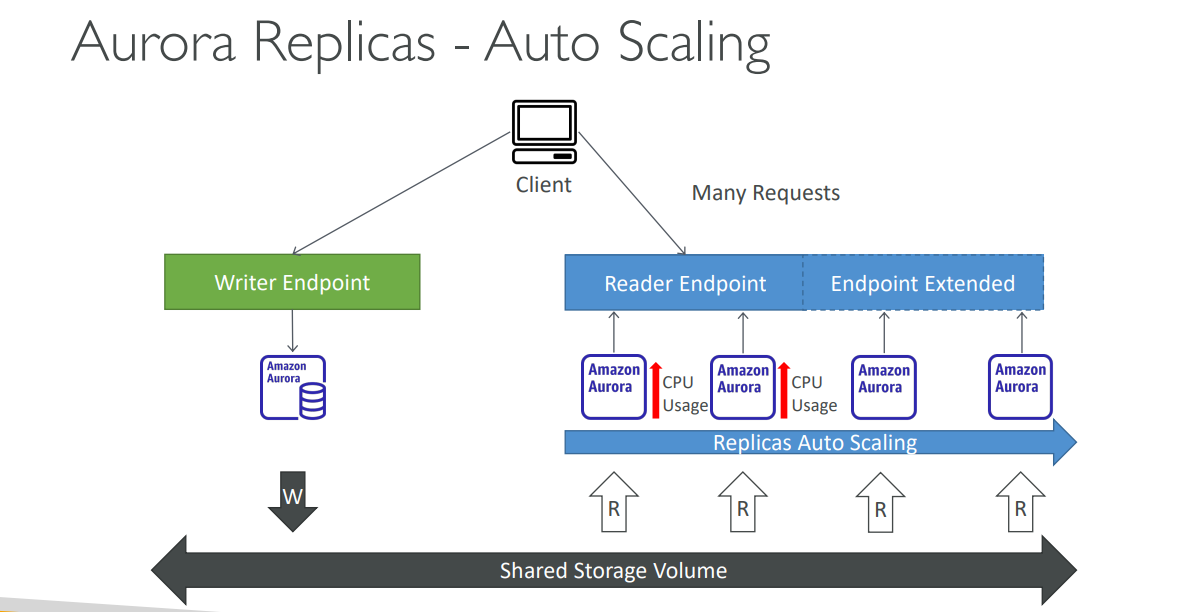
• Push-button scaling

• Automated Patching with Zero Downtime

• Advanced Monitoring

• Routine Maintenance

• Backtrack: restore data at any point of time without using backups.

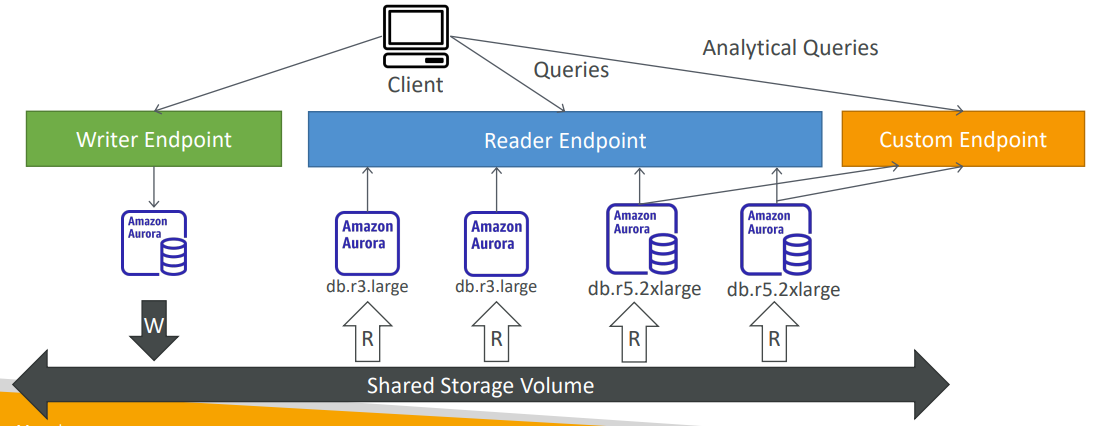


**Aurora – Custom Endpoints**

• Define a subset of Aurora Instances as a Custom Endpoint

• Example: Run analytical queries on specific replicas

• The Reader Endpoint is generally not used after defining Custom Endpoints



**Aurora Serverless**

• Automated database instantiation and auto - scaling based on actual usage

• Good for infrequent, intermittent or unpredictable workloads

• No capacity planning needed

• Pay per second, can be more cost-effective.

**Global Aurora**

• Aurora Cross Region Read Replicas:

* + Useful for disaster recovery
  + Simple to put in place

• Aurora Global Database (recommended):

* + 1 Primary Region (read / write)
  + Up to 5 secondary (read-only) regions, replication lag is less than 1 second
  + Up to 16 Read Replicas per secondary region
  + Helps for decreasing latency
  + Promoting another region (for disaster recovery) has an RTO of < 1 minute
  + Typical cross-region replication takes less than 1 second

**Aurora Machine Learning**

• Enables you to add ML-based predictions to your applications via SQL

• Simple, optimized, and secure integration between Aurora and AWS ML services

• Supported services

* + Amazon SageMaker (use with any ML model)
  + Amazon Comprehend (for sentiment analysis)

• You don’t need to have ML experience

• Use cases: fraud detection, ads targeting, sentiment analysis, product recommendations.

**RDS Backups**

• Automated backups:

* + Daily full backup of the database (during the backup window)
  + Transaction logs are backed-up by RDS every 5 minutes
  + => ability to restore to any point in time (from oldest backup to 5 minutes ago)
  + 1 to 35 days of retention, set 0 to disable automated backups

• Manual DB Snapshots

* + Manually triggered by the user
  + Retention of backup for as long as you want

• Trick: in a stopped RDS database, you will still pay for storage. If you plan on stopping it for a long time, you should snapshot & restore instead.

**RDS & Aurora Restore options**

• Restoring a RDS / Aurora backup or a snapshot creates a new database

• Restoring MySQL RDS database from S3

* + Create a backup of your on-premises database
  + Store it on Amazon S3 (object storage)
  + Restore the backup file onto a new RDS instance running MySQL

• Restoring MySQL Aurora cluster from S3

* + Create a backup of your on-premises database using Percona XtraBackup
  + Store the backup file on Amazon S3
  + Restore the backup file onto a new Aurora cluster running MySQL