The Flash Guide to AWS

A succinct, comprehensive and practical companion text designed for students and young professionals getting started with AWS.

By: Sujal Thapa

Dedicated to my parents,

Sharmila Homagai and Raj Kumar Thapa

Preface

This document is intended for educational purposes. It is designed as a study companion guide to aid individuals getting started with AWS, especially those preparing for AWS Practitioner and Associate level examinations. While this guide provides a succinct yet comprehensive overview of key AWS concepts and services, it is not intended to replace primary sources of information such as the official AWS documentation.

The goal of this document is to simplify the learning process and make the daunting task of diving into the AWS ecosystem more manageable, especially for those who may not have a background in IT. It serves as a quick reference and a supplemental study aid, providing a structured summary of crucial topics.

Some of the images included in this document have been sourced from external web platforms. There was no intention to infringe on copyright; and have been utilized while adhering to Creative Commons licensing guidelines. These images were chosen because they were the best infographics available. The use of these images is solely to enhance the educational value of this document.

This guide covers various AWS services, categorized into sections and chapters for the ease and convenience of the reader. From compute and storage services to security and machine learning, it aims to provide a broad overview of the major AWS services and develop a certain level of theoretical sophistication in the reader.

Though exact sources have not been mentioned throughout the book, all information presented within it can be verified by searching through the official AWS Documentation for the service and/or service suite in question.

Table of Contents

Intro to AWS

Cloud Computing 1
The Big Three 5
AWS 7
Types of AWS Services 8

Storage in AWS

Background 14
S3 18
EBS 19
Types of EBS Volumes 20
EFS and Multiple Access 21
S3 File Gateway 24
AWS DataSync vs Storage Gateway 25
Data Transfer Times 27
Snowmobile Capacity 29
Snowball Edge vs Snowball 30
FSX for Lustre 32
FSX for Windows 33
Compliance vs Governance Object Lock 34
Legal Hold vs Retention Periods 35
S3 Bucket Policy 36
S3 Storage Class and Lifecycle Policies 40
S3 Glacier 43
Athena 45
Transfer Acceleration 48
Accidental Deletion 49

Compute Services in AWS

Background 51
EC2, ECS, Lambda 53
Types of EC2 Instances 56
Hibernation and Stop 60
Autoscaling Group and Spanning 62
Benefits of Overprovisioning EC2 Instances 64
Reserved Instances and On-Demand Capacity Reservation 65
Elastic Container Service (ECS) 68
Lambda 70
ENV Variables for Lambda Functions 71

Databases in AWS

Background 73
RDS 75
RDS Proxy 77
Aurora 80
Database Cloning 81
DynamoDB and DynamoDB Streams 82
OLTP and OLAP 84
IAM DB Authentication 85
Disaster Recovery 86
Primary Database and CNAME 88
Redshift and AWS Lake Formation 89

Networking in AWS

Background 91
CloudFront 98
Global Accelerator 99
Cross Origin Resource Sharing 100
Direct Connect 101
Subnets 102
Block Size of a VPC 104
NAT and Internet Gateways 106
NAT Gateway vs NAT Instance 108
Gateway VPC Endpoints 110
Elastic Load Balancers 111
Types of Elastic Load Balancer 113
Health Checks 115
Listener Rules and Redirecting 117
API Gateway 119
Route53 and Failover 121
Geoproximity vs Geolocation Routing 124
Origin Access Identity vs Origin Access Control 126
Security Groups vs NACL 128
NACL/SG Priority 131
AWS PrivateLink 132

Application Integration

Decoupling 134
SQS, SNS and EventBridge 136
SQS FIFO 139
SQS Duplication 140
Amazon EventBridge 141
Simple Email Service (SES) 143
Kinesis 145
Quicksight 146

Amazon AppFlow 147

Management and Governance in AWS

CloudWatch and Metrics List 149
CloudWatch Logs Streams 150
CloudTrail 151
Cost and Usage Report + Cost Explorer 152
AWS Config 154
AWS Organizations and Principal OrgID 155
AWS Backup 157
Systems Manager Patch Manager vs Run Command 159
CloudFormation and CreationPolicy 160

Security, Identity and Compliance in AWS

Background 162
AWS Secrets Manager 165
IAM and IAM Roles 166
IAM Roles using AD 170
Multi-factor Authentication 171
KMS and Multi-Region Keys 172
Certificate Authority (CA) 173
AWS CloudHSM 175
Service Control Policies (SCP) 176
AWS Shield 178
AWS WAF 179
AWS GuardDuty 180
AWS Network Firewall 181

Machine Learning in AWS

Machine Learning Cheat Sheet 183
Macie 186
Rekognition 187
Textract and Comprehend Medical 188

Conclusion and Multi-Cloud

A Case for Technological Diversity 190
Conclusion 198

INTRO
TO
AWS

Cloud Computing

In this section we will discuss what cloud computing is, how it came to be, and the varying methods in which we take advantage of it.

Cloud Computing: A Brief History

Computers have become an integral part of businesses environments around the world, and different businesses have resorted to different methods of acquiring computers and computer resources. However, computers are a relatively recent invention and their current ubiquity was obviously not always the case.

In the beginning, computing was seen mostly as a tool for large governments and universities, with private companies adopting computing technology mostly as a novelty, something that could aid their worker's productivity and give them a potential competitive advantage against less "modern" competitors.

Over time though, as the adoption of computers became more widespread, technology became not only a competitive advantage against other companies but a competitive necessity, an essential resource without which companies could simply not be as productive as the rest of the economy.

Thus, as organizations became more reliant on computer-based workflows, they built larger and larger computers often with very specialized networking, security, computing, software and hardware resources: the very first **data centers**.

These data centers were often located either near or exactly on the premises of corporations offices, occupying expensive land and were managed by an in-house team of IT technicians and electricians, both of which did not come cheap. And that is not to mention the cost of electricity accrued by the operation of the computers and the cooling required to maintain them.

Referred to as an **on-premises cloud environment**, data centers like the one mentioned above were brilliantly secure and performant, but were a huge drain on the business's financials and a pain to manage. While many businesses were willing to front the costs associated with these private clouds, there were many that sought other options: enter **Cloud Computing**.

Cloud Computing: What it is, Tradeoffs and Characteristics

In 1961, MIT Professor John McCarthy said in a now-famous speech that someday "Computing can be sold as a utility, like water and electricity", an idea that was realized with the advent of cloud computing platforms, platforms that allow us to lease IT infrastructure and software solutions from third-party providers, usually over the internet.

These platforms often utilize pay-as-you-go pricing models, enabling flexible and cost-effective access to computing resources and organizations that had so far been maintaining extensive arsenals of hardware and software resources gave those arsenals up for the convenience and cheapness of renting their infrastructure from companies like AWS, Azure, Salesforce and Google.

Though this period of mass transition from on-premises environments to cloud computing is seen mostly as a net-positive, there were some trade-offs to the cloud. Firstly, the ability of organizations to take advantage of the cloud was dependent on the internet, and therefore necessitated organizations to adopt and invest in high speed internet connections.

And secondly, since organizations went from owning to simply renting resources under cloud computing, they had to relinquish a great deal of the control over the underlying infrastructure and software in the process, especially when compared to the degree of control offered by on-prem cloud environments.

Variations of this Simpsons meme is extremely popular among Cloud engineers online

That second point actually was and continues to be a point of particular frustration for cloud engineers, who often have to obey illogical rules and protocols within AWS and other cloud platforms simply because that is the way the parent company has decided things should work.

Engineers and businesses like the comparatively greater control that on-prem environments offer, but the price tag associated with them is often enough to make them unattractive. Well, this has led to many organizations adopting a sort of third way, a middle-ground between the two: **Hybrid Cloud Environments**.

Hybrid cloud environments combine a standard data center with an outsourced cloud computing platform. For many organizations, the hybrid cloud is the perfect migration to the cloud. In a hybrid architecture, the organization can run its applications and systems in its local data center and offload part of the computing to the cloud. This provides an opportunity for the organization to leverage its investment in more traditional data centers while simultaneously being able to take advantage of modern cloud solutions.

Now, depending on what exactly is being rented/leased from the platform, cloud computing services are typically categorized into three main types:

1. **Infrastructure as a Service (IaaS):** Provides virtualized hardware resources for the operation of digital applications/products, such as virtual machines, storage, and networks.

2. **Software as a Service (SaaS):** Delivers software applications, often related to extremely specific use-cases, and often on a subscription basis.

3. **Platform as a Service (PaaS):** Provides both hardware and software tools, typically make the hosting of application development and/or testing more convenient and simpler.

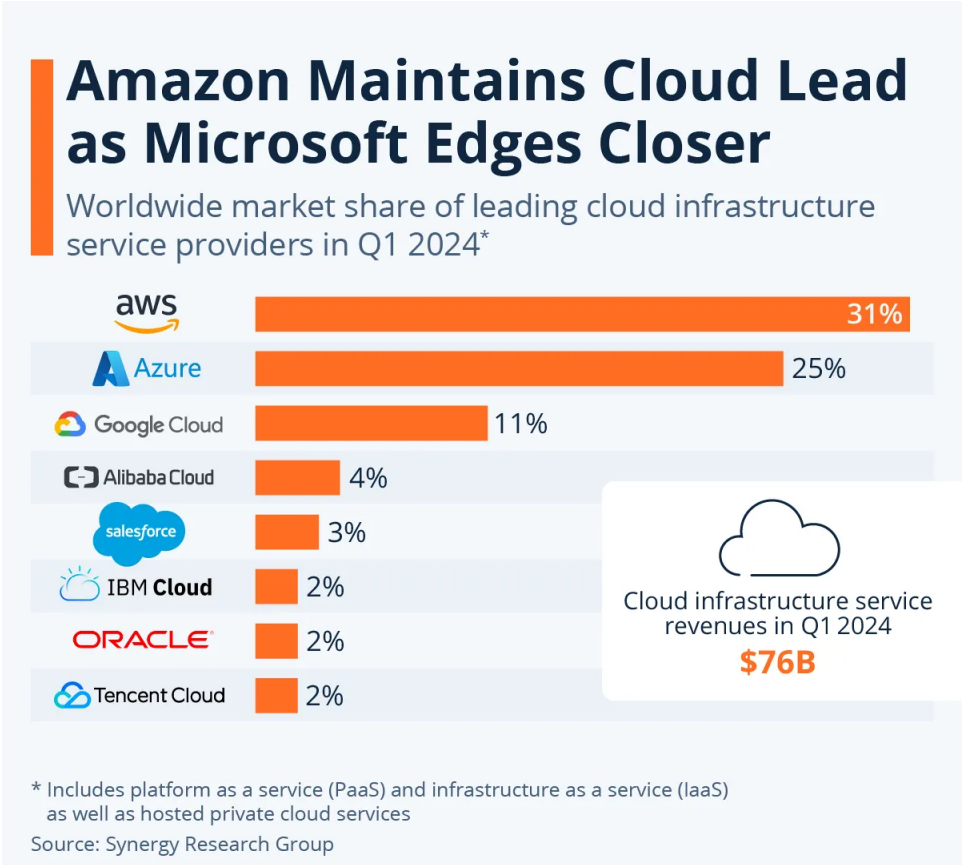
All three categories of cloud computing services, and their examples will be discussed in future sections of the book. The short descriptions above were written for the express purpose of making it clear to the reader that cloud computing and all of its constituent services amount to nothing more than renting agreements for hardware and/or software resources.

Cloud computing is therefore nothing complicated to understand, it is not quantum physics, it is not advanced mathematics and it is not rocket science, it is just **rent**. A fundamental practice that humanity has been performing for thousands upon thousands of years.

The Big Three

Though there are actually many cloud providers, the cloud computing market is mostly dominated by three large players, sometimes termed as the big three: **Amazon Web Services (AWS)**, **Microsoft Azure** and **Google Cloud Platform (GCP)**.

The big three services cumulatively control about 65% of the entire worldwide market for cloud computing with AWS alone accounting for a little more than 30% of the market all by itself at the time of writing. An infographic on the global cloud computing market and the largest players within it is provided below:



The bulk of this book deals with AWS, the cloud provider with the single-largest industry market share and the services indigenous to it but this does not mean that I endorse blindly relying on the service for business operations. In fact, I am highly against it.

A case for why companies should almost never rely on a single cloud provider and easy plug-and-play alternatives for all of the major AWS services, including offerings from both other Big 3 companies as well as other third-party providers will be considered and delved into the final portion of the book.

Readers are therefore to treat the concluding chapter of the book as a mandatory read lest they limit their range of knowledge of cloud computing to AWS, which though quite dominant in the industry, is only one of a myriad of cloud providers.

As such, the reader may progress through the book without worry of building a very Amazon-centric base of knowledge, as the concepts and type of services discussed throughout the book can be generalized to encompass the other cloud providers and it should be little hassle for someone comfortable with AWS to make the switch to say, Microsoft Azure or GCP.

AWS

Amazon Web Services is the largest and most widely adopted cloud provider in the world, used by everyone from the smallest of startups and hobbyists to well, the United States government and 90% of Fortune 500 companies.

Originally released to the public in July 2002, AWS (then called Amazon.com web services) started out as a service that allowed to incorporate information from Amazon.com into their own websites. The unexpectedly large positive response to the service from developers took Amazon by surprise and by fall 2003, ideas for expansion of Amazon.com web services into a platform for developers to rent database, storage and compute infrastructure had already been put in place.

These ideas would finally come into fruition starting from September 2006, with the release of **S3** and subsequently, **EC2** and **RDS**. The first generation of AWS Cloud Computing offerings. (All mentioned services will be discussed in detail in future sections)

It has since then grown into a behemoth of a platform with over 200 services (most of which are fully-featured and have built quite comprehensive ecosystems in and of themselves) and a massive global geographical presence, with infrastructure in all the seven continents serving 245 countries and territories. The dominance that AWS has over the cloud computing industry will be further explored in the next section.

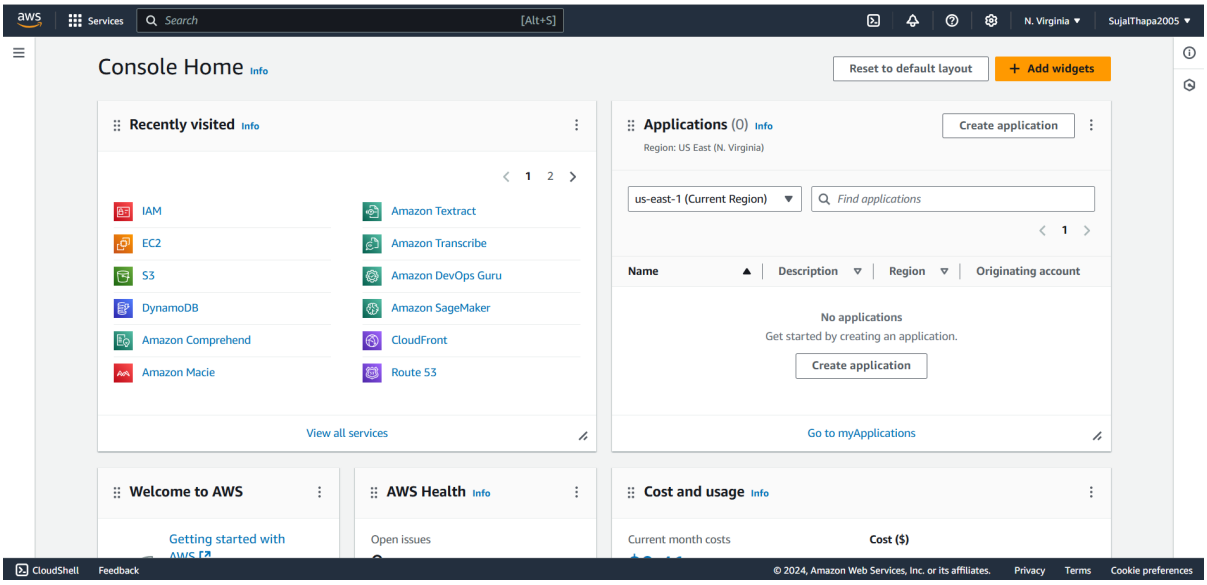
AWS is also obviously the primary subject of this book and honestly, diving into the ecosystem can be a psychologically daunting task, especially for those not from an IT or computing background. But it is precisely in order to make that task seem less daunting that this book has been written.

Connecting to AWS

Now there are three main interfaces by which the reader may utilize and administer resources in AWS: **The AWS Management Console**, **The Command-line interface** and **SDKs**. A more in-depth look at all three is given below:

The AWS Management Console

The main suggested method of interacting with AWS for people not of a technical persuasion, the AWS Management Console is a simple-to-use web interface provided to us by AWS in order to both create and manage AWS infrastructure and its related components. An image of how the Console Home page should look like for a relatively new AWS account is given below:



Having an account with AWS is a pre-requisite for logging in, a process that will yes, require entering the reader's credit card details. This may sound scary, especially for readers from the third world but do not be afraid, Amazon charges you a minimal amount and the information can be removed from your account

once it is created. (I personally know many brilliant people who have refused to make an AWS account simply because they will not enter their credit card information digitally).

As the reader progresses through the book, it would be prudent for them to log in to console, search for, and play around with any and all services they find interesting. That is after all, the ultimate aim of the book, to make the reader comfortable understanding the cloud and to get them started building real solutions using AWS.

AWS Command Line Interface

The preferred method of handling AWS resources for people familiar with the Linux operating system, the AWS Command Line Interface (or CLI) is a text-based user interface (like Command prompt, Windows Powershell etc) enables us to manage infrastructure via Linux commands and JavaScript Object Notation (JSON) scripting.

It is more efficient than using the console but also requires administrators with a wider set of knowledge and training, and is ideal organizations that already aware of what exactly is needed and how to configure properly.

With the CLI, the administrator is working in an environment with much less guardrails and guidance, so ensure that a certain base level of expertise has been acquired before engaging with it.

AWS Software Development Kit

A method for modifying and provisioning AWS resources programmatically, the AWS Software Development Kit (or SDK) allows us to build applications that leverage AWS resources for its functioning.

With support for over a dozen major programming languages like Python and Java, the SDK is going to be the major method of interacting with AWS for the readers who plan on being software engineers or developers. Utilizing SDKs properly however, will obviously require familiarity with an established programming language alongside knowledge of AWS, its resources and its 200+ services.

While all three of the mentioned interface types operate using radically different method from one another, it is important to remember that they are simply different ways of achieving the same thing: **The creation and management of AWS resources**. Therefore, having at least a basic understanding of what those resources are, and the services they relate to must be treated as a pre-requisite to operating the mentioned interfaces.

Once that has been achieved, the reader may find that there is actually very little difference between say, using AWS through the CLI or using AWS through the console, as the soft-skills and knowledge required to do both are quite similar, it is merely a matter of preference which interface the reader prefers.

Types of AWS Services

With over 200 different AWS Services, it might seem intimidating to start learning AWS. After all, where to start?

To make this task seem less insurmountable, the services have been divided into the following broad categories:

Compute Services

Compute services provide the processing power required to run various applications and manage servers. These services are crucial for tasks that need high computational power, scalability, and flexibility. They enable businesses to run their applications without needing to maintain physical hardware, allowing for more efficient resource management and cost-effectiveness. Compute services also support a variety of architectures and environments, ensuring compatibility with different types of workloads. Example services in this category include:

- Amazon EC2 (Elastic Compute Cloud)
- AWS Lambda
- Amazon ECS (Elastic Container Service)

Storage Services

Storage services offer scalable and durable storage solutions to meet the needs of data-intensive applications and workloads. These services provide secure and reliable storage for data of all types and sizes, including structured and unstructured data. They ensure data availability and integrity, supporting various use cases such as backup and recovery, data archiving, and content distribution. Storage services also facilitate efficient data management and retrieval, enabling

users to access their data from anywhere at any time. Example services in this category include:

- Amazon S3 (Simple Storage Service)
- Amazon EBS (Elastic Block Store)
- Amazon EFS (Elastic File System)
- AWS Glacier

Database Services

Database services provide managed database solutions for storing and managing data. These services include support for relational, NoSQL, and in-memory databases, catering to a wide range of applications and use cases. Database services offer high availability, scalability, and security, ensuring optimal performance for database operations. They also simplify database management tasks, such as backups, patching, and monitoring, allowing users to focus on their application logic rather than database administration. Example services in this category include:

- Amazon RDS (Relational Database Service)
- Amazon DynamoDB
- Amazon Aurora
- Amazon Redshift

Networking and Content Delivery Services

Networking and content delivery services enhance the performance, security, and availability of applications. These services enable efficient communication between different components of an application and optimize the delivery of content to users worldwide. They also provide tools for managing network traffic, securing data in transit, and ensuring high availability through global distribution and load balancing. Example services in this category include:

- Amazon VPC (Virtual Private Cloud)

- Amazon CloudFront
- AWS Direct Connect
- AWS Elastic Load Balancing (ELB)

Security, Identity, and Compliance Services

Security, identity, and compliance services help protect AWS resources and data. These services provide tools for managing access to resources, encrypting data, and ensuring compliance with regulatory requirements. They enable users to implement security best practices, monitor security events, and respond to security incidents, ensuring the integrity and confidentiality of their data. Example services in this category include:

- AWS IAM (Identity and Access Management)
- AWS KMS (Key Management Service)
- AWS Shield
- AWS WAF (Web Application Firewall)

Management and Governance Services

Management and governance services provide tools for monitoring, managing, and optimizing AWS environments. These services help users gain visibility into their resources, automate management tasks, and ensure compliance with organizational policies. They support efficient resource allocation, cost management, and performance optimization, enabling users to maintain control over their AWS environments. Example services in this category include:

- AWS CloudFormation
- AWS CloudTrail
- AWS Config
- AWS CloudWatch

Machine Learning Services

Machine learning services provide tools and frameworks for building, training, and deploying machine learning models. These services enable users to leverage advanced machine learning techniques for various applications, such as predictive analytics, natural language processing, and image recognition. Machine learning services simplify the process of developing and deploying models, allowing users to focus on creating innovative solutions without needing extensive machine learning expertise. Example services in this category include:

- Amazon SageMaker
- Amazon Comprehend
- Amazon Rekognition

Analytics Services

Analytics services provide tools for analyzing and processing large volumes of data. These services support various data analytics tasks, such as data warehousing, real-time analytics, and big data processing. They enable users to gain insights from their data, make data-driven decisions, and improve business outcomes. Analytics services also offer scalable and cost-effective solutions for managing and analyzing data, ensuring optimal performance and reliability. Example services in this category include:

- Amazon EMR (Elastic MapReduce)
- Amazon Kinesis
- AWS Glue
- Amazon QuickSight

Application Integration Services

Application integration services facilitate communication and data exchange between different applications and systems. These services enable users to build complex workflows, automate processes, and integrate various applications

seamlessly. They support various integration patterns, such as messaging, event-driven architecture, and service orchestration, ensuring reliable and efficient application integration. Example services in this category include:

- Amazon SQS (Simple Queue Service)
- Amazon SNS (Simple Notification Service)
- AWS EventBridge

We will explore each of the mentioned type of AWS service, either dedicating an entire chapter to them or pairing them with a complementary category of service. Each chapter is comprised of sections, with each section being devoted to a service or set of services that we have deemed to be important.

Some chapters also includes a background section, designed to provide readers with the necessary technical concepts, history and background knowledge required to understand the contents of the chapter and the services described within them.

The Background sections are a mandatory read for any and all readers from a non-technical field and/or for people not familiar with Computer Science fundamentals. More learned readers may however choose to either read or skip the section as they see fit.

STORAGE
IN
AWS

Storage: Background

In this section we will discuss theoretical concepts related to the storage of data, the three major types of cloud storage, and their characteristics. A concise list of the topics to be discussed is given below:

- Memory, Storage, and how do they differ from one another.
- On-prem storage, its advantages and disadvantages.
- Cloud storage, and its three different types.
- Object Storage and its use cases
- Block Storage and its use cases
- File Storage and its use cases

If the reader believes themselves to already be familiar with the listed concepts, then they can feel free to skip this section and move on to the rest of the chapter.

Memory vs Storage

As technology has become more ubiquitous in modern society, it must come as little surprise that technical terms have also found their way into the modern vocabulary. **Memory** and **Storage** are two such terms, and are often used interchangeably by people to refer to a computer's capacity to hold data. However, there do exist contrasts between the two which I find important to delve into.

Memory, sometimes also referred to as RAM, is used to refer to the hardware component(s) responsible for holding data temporarily, data typically related to the instructions that are currently being executed or operated by the computer, a task typically done by the CPU. It is quite fast, and enables quick data processing but **volatile** in nature, meaning that all data within it is lost when the device is turned off.

Storage on the other hand, is a broad term encompassing all hardware components and devices used to hold data for longer periods of time, data

typically related to the OS, installed applications and personal files like documents, photos and videos. Though slower than memory, storage is non-volatile in nature and can typically handle much larger quantities of data than memory can.

Basically, while memory serves as the computer's short-term workspace, enabling rapid access to data for active tasks, storage acts as the long-term repository, preserving data permanently.

On-prem Storage and RAID

Now, there are two major ways in which large organizations and enterprises acquire and organize their storage infrastructure: **On-premises**, **Cloud** and **Hybrid**. Since On-prem is the oldest kid on the block, it would be helpful for us to learn about it before transitioning towards Cloud and Hybrid storage.

On-Premises storage environments have all their storage systems, devices and related infrastructure physically located within or near their facilities, providing direct oversight and security control over them, and minimal redundancy to boot. It has remained extremely popular among legacy organizations and organizations with high data security needs, compliance requirements, or specialized performance criteria.

Also, On-prem environments often utilize **RAID (Redundant Array of Independent Disks)** setups for the organization and management of their hardware devices, but since RAID and its multitude configuration types are a complex topic, beyond the scope of a book dedicated to Cloud computing, we will not be discussing it here. Though the topic might be worth looking into for the interested.

Cloud Storage

Cloud storage environments involve the renting of storage systems, devices and related infrastructure, usually from a cloud provider like AWS or Microsoft Azure. These rented storage systems can then be connected to and taken advantage of either through the internet or though dedicated private channels.

Unlike with traditional storage, cloud storage can scale itself according to the capacity needs, and enterprises also only have to pay for the storage capacity that they use and not a dime more, allowing them to avoid the problems of under or over-provisioning of hardware resources associated with traditional storage. These factors, combined with the variety of cloud storage types and the convenience of not having to manage any hardware on their own, make cloud storage an appealing choice for businesses with fluctuating storage needs or those aiming to avoid the high upfront costs associated with on-premises infrastructure.

Note however that it is not necessary for organizations to chose one or the other, in fact many organizations often adopt environments that utilize both on-prem and cloud storage solutions in varying capacities, usually keeping frequently accessed or low-latency access data near their facilities; An approach termed as hybrid storage.

Now there are three primary types of cloud storage: **Object Storage**, **Block Storage**, and **File Storage**. All three types, their characteristics and use cases are discussed in detail below.

Object Storage

Object Storage (as the name may suggest) organizes data by breaking it into pieces called objects, with each object containing the data itself, metadata related to it, and a unique ID. It is highly scalable and well-suited for unstructured data like images, videos, and backups.

Particularly designed for scenarios where quick access to large volumes of data is required, it is ideal for content repositories, backup, and archiving. All the objects are stored in a non-hierarchical manner in a single “folder”, often referred to as the object bucket. Additionally, the metadata associated with the objects allow for the objects to be searched through using say, SQL Queries. It should therefore, not

come as a surprise to many that Object Storage is often used to power big data environments such as data lakes and data warehouses.

Block Storage

Block storage is a type of cloud storage that resembles a hard drive in traditional computers. It places data into blocks and then stores those blocks as separate pieces with each block having a unique identifier. A hierarchical storage method, it follows the nested file and folder structure similar to desktop file systems, placing the blocks of data wherever most prudent or efficient capacity wise.

File Storage

File storage is the traditional file systems we run on our desktop machines, just now being run on the cloud i.e. now being run in someone else's data center. It is mostly used to provide shared file systems for a network of devices running the same Operating System. Examples of this include NTFS-based volumes for Windows systems and NFS-based volumes for Linux/UNIX systems.

They are often adopted due to their ability to be mounted and directly accessed by numerous computers and agents at the same time.

S3

Amazon Simple Storage Service (also called Amazon S3) is a scalable, high-speed, web-based cloud storage service designed to store and retrieve any amount of data at any time. It uses an object storage architecture, where data is managed as objects rather than file hierarchies. The industry standard object storage service, S3 streamlines the process of storing files on the cloud, while providing easy methods of accessing and managing it.

Each object in S3 is stored in a bucket and consists of data, metadata, and a unique identifier. S3's key features include high (99.999999%) durability, availability, security, and performance. It's widely used for backup and recovery, content distribution, static website hosting, etc, making it a fundamental service for many cloud-based applications.

S3 has different storage classes or categories to choose from based on the performance, availability called S3 Standard, S3 Infrequent Access, S3 Intelligent Tiering, etc each optimized for different use cases based on access frequency and cost requirements. These will be discussed in future sections.

It is also the **cheapest** of the three major forms of storage in AWS, the others being: Block Storage using EBS (Elastic Block Store), which is used in databases and File Storage using EFS. Both types of cloud storage have already been discussed in the background, though their representative services will be discussed in later sections. Usually, when talking about the cost per Gigabyte(GB) for the three major storage services in AWS, they can be organized as follows:

EBS ⇒ Cheap **EFS** ⇒ Cheaper **S3** ⇒ Cheapest

EBS and Fast Snapshot Restore

Amazon Elastic Block Store (EBS) is a high-performance block storage service designed for use with EC2 instances. It provides persistent block-level storage volumes that can be attached to EC2 instances, allowing you to store data and access it from any instance. EBS volumes are durable and highly available, offering features such as snapshots for data backup and replication for data protection.

Additionally, EBS Volumes can be resized without any downtime, allowing us to increase/decrease storage capacity or adjust performance characteristics as needed on the fly. However, do note that all EBS volumes are tied to a single availability zone (AZ), so point-in-time copies of the EBS volumes (called Snapshots) must be created and attached/restored to a different AZ or Region. This is usually done to enhance data durability and recovery options.

It is also not uncommon to encrypt the data in EBS Volumes at rest, which can be done using encryption keys provided by the AWS Key Management Service (KMS), granting the EBS volumes with an additional layer of security. KMS and related concepts will be discussed in detail in a later chapter.

Note:

EBS also has a feature called **Fast Snapshot Restore** which can be used to clone data from an EC2 instance store into new EBS volumes with minimal time, reducing the amount of required to recover or create new volumes.

Types of EBS Volumes

EBS offers several different volume types, each tailored to meet the performance and cost requirements of various use cases. The different EBS Volume types are as follows:

General Purpose SSD (sub-types: gp3, gp2): A balanced storage option suitable for a wide range of workloads. gp3 allows independent scaling of IOPS and throughput, while gp2 provides baseline performance with burst capabilities. Ideal for boot volumes, medium-sized databases, and general-purpose applications.

Provisioned IOPS (sub-types: io1, io2) is a storage option offered by AWS for applications requiring high-performance and consistent input/output operations per second (IOPS). It is designed for use with Amazon EBS (Elastic Block Store) and is ideal for I/O-intensive applications such as large databases, transactional systems, and other performance-sensitive workloads especially when compared to General Purpose SSD.

Throughput Optimized HDD (sub-types: st1): Low-cost, high-throughput storage optimized for large, sequential workloads. Ideal for big data processing, data warehousing, and log processing, where throughput is more important than IOPS.

Cold HDD (sub-types: sc1): The most cost-effective storage option, designed for infrequently accessed data. Best suited for archival storage, long-term backups, and datasets that require occasional access at a low cost.

EFS and Multiple Access

One of three major storage services offered by AWS, Amazon Elastic File System (EFS) provides us with a fully managed, serverless, scalable, POSIX-compatible file storage service on the cloud. As mentioned in the background, file storage systems follows the same structure and is operated in the same manner as many traditional desktop file systems.

EFS is highly available and durable, making it ideal for workloads that require shared access to file storage across multiple instances. The service is considered moderately priced, generally costing more than S3 but less than EBS (Elastic Block Store). It also has a Infrequent Access (EFS-IA) feature which functions similar to S3 Infrequent Access through its lifecycle policies with regards to IA are much more limited than its S3 counterpart.

Elastic File Systems can be easily mounted onto EC2 instances, AWS Containers, Lambda functions, or on-premise servers and are often done so in order to augment their default storage capabilities.

Finally, EFS offers two different performance and throughput modes for potential adopters:

Performance Modes:

- **General-Purpose** (Most widely used, particularly efficient for latency-sensitive use cases)
- **Max I/O** (For highly parallel applications which necessitate high throughput)

Differences between the two performance modes are discussed below:

Feature/Characteristic	**General Purpose Mode**	**Max I/O Mode**
Use Case	Ideal for latency-sensitive applications (e.g., web serving, content management)	Ideal for applications with highly parallelized workloads (e.g., big data, media processing)
Latency	Low latency	Higher latency compared to General Purpose
Throughput	Up to 7 GB/s with Bursting Throughput	Higher throughput (up to 12 GB/s) with Bursting Throughput
IOPS	Up to 500,000 IOPS	More than 500,000 IOPS
Parallelism	Suitable for moderate to high parallelism	Designed for very high parallelism
Performance Consistency	Consistent performance with low latency	Scales with load; latency can increase as load increases
Cost	Generally lower cost	May incur higher cost due to increased resource usage
Best For	File-based workloads that require consistent low latency	Applications that require high aggregate throughput and can tolerate higher latency

Throughput Modes:

- **Bursting Throughput (**Automatically scales the throughput based on file system size)
- **Provisioned Throughput (**Allows us to allocate throughput without being dependent on the file system size)

Differences between the two throughput modes are as follows:

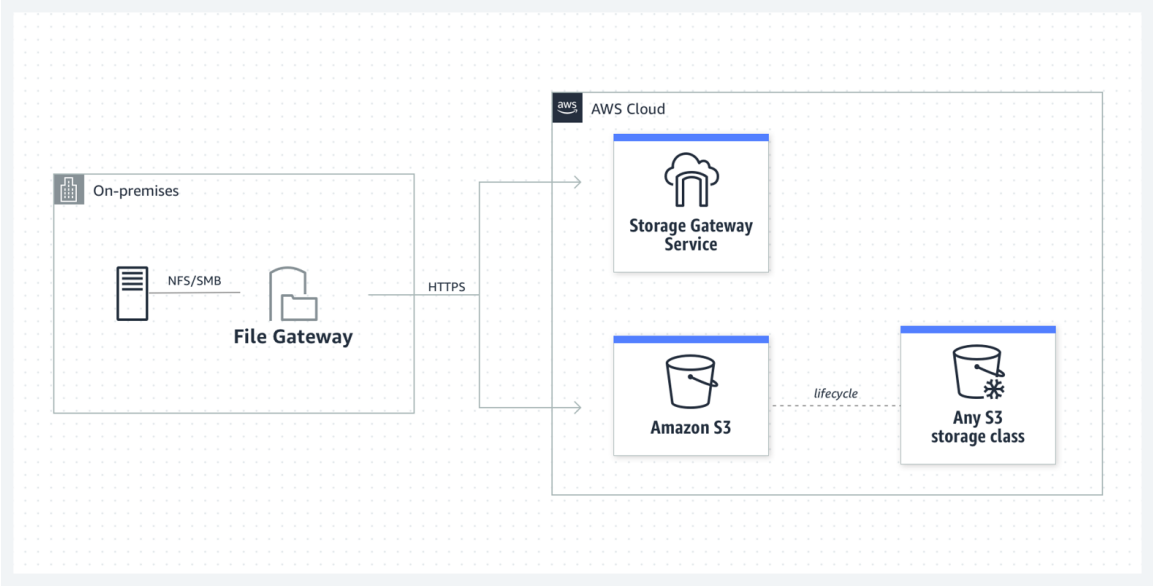
Feature/Characteristic	**Bursting Throughput Mode**	**Provisioned Throughput Mode**
Use Case	Suitable for most workloads with variable and unpredictable throughput needs	Ideal for applications requiring consistent, high levels of throughput, regardless of file system size
Throughput Allocation	Throughput scales with file system size, allowing bursts based on the amount of stored data	Throughput is provisioned independently of the file system size, ensuring a fixed throughput level
Throughput Range	Scales automatically with storage, up to 100 MB/s per TB (can burst up to 100 MB/s for file systems under 1 TB)	Customizable from 1 MB/s to 1024 MB/s, regardless of file system size
Bursting Capability	Supports bursting when workload needs exceed baseline throughput (depending on size and burst credits)	No bursting; throughput remains constant as provisioned
Performance Consistency	Variable throughput, can burst when needed but may be lower during sustained periods of heavy use	Consistent throughput, ideal for sustained high-performance needs
Cost	Cost-effective for variable workloads, as throughput is tied to file system size	May incur higher costs due to the fixed provisioned throughput, independent of storage usage
Best For	General-purpose workloads, including web serving, backups, and dev/test environments	High-performance workloads, like media processing, big data analytics, and machine learning

S3 File Gateway

Storage gateways are the services provided by AWS which enable hybrid cloud storage solutions, leveraging the best of both worlds, i.e. grant us the benefits of low-latency access associated with on-premise storage systems while retaining the virtually unlimited capacity of cloud storage storage solutions.

This is achieved by creating a cache of the cloud storage and storing it locally on premises. S3 File Gateway is one such storage gateway service, and provides a way to seamless integrate on-premise applications with Amazon S3, using a local cache to store data that would benefit the most from low-latency access while keeping the rest in S3. Asynchronous in nature, the service presents S3 buckets as NFS or SMB file shares, allowing applications to access objects stored in S3 using familiar file protocols.

A basic file gateway architecture is shown in the following diagram:



AWS DataSync vs Storage Gateway

AWS DataSync is a service provided by Amazon Web Services (AWS) that simplifies, accelerates, and automates the process of transferring data between on-premises storage systems and AWS services.

After the initial migration, AWS DataSync can be configured to maintain the consistency of data between on-premises storage and AWS storage services. In such cases, data synchronization between the two happens in either real or near real-time.

While both AWS DataSync and AWS Storage Gateway provide solutions for integrating on-premises storage environments with AWS cloud storage services, they serve different purposes and have distinct functionalities, including how they handle data synchronization. This is illustrated in the bulleted sub-sections below:

AWS DataSync

- **Purpose**: Primarily designed for high-speed, one-time data transfers and ongoing data synchronization between on-premises storage systems and AWS storage services.
- **Data Transfer Mechanism**: Utilizes optimized data transfer protocols to ensure fast and efficient transfer of large volumes of data.
- **Use Cases**: Ideal for scenarios requiring frequent updates or synchronization of data between on-premises and AWS environments, such as continuous data backup, real-time data processing, or maintaining consistent copies of data for disaster recovery purposes.

AWS Storage Gateway

- **Purpose**: Offers a hybrid storage solution that enables on-premises applications to seamlessly access data stored in AWS cloud storage services with lower latency than it otherwise could.
- **Data Synchronization**: While AWS Storage Gateway supports data migration and synchronization capabilities, its primary focus is on providing on-demand access to data stored in AWS, rather than continuous synchronization.
- **Storage Protocols**: Supports various storage protocols, including NFS, SMB, and iSCSI, allowing existing on-premises applications to interact with AWS storage as if it were local storage.
- **Use Cases**: Commonly used for extending on-premises storage capacities, disaster recovery, data archiving, and enabling cloud-based applications to access on-premises data.

TL;DR

AWS Data Sync: Used for data migration and synchronization.

AWS Storage Gateways: Used for low-latency access to data and adding capacity to existing on-prem storage.

Data transfer times

Data transfer to AWS involves moving data from your on-premises data center or other cloud environments into AWS. This process can be critical for cloud migration, data backup, disaster recovery, and large-scale data analytics. AWS offers multiple methods for data transfer, each tailored to different volumes of data, speed requirements, and cost considerations.

(Note: The following task might be hard for some in senior positions to perform)

Consider yourself as an employee has been tasked to transfer 200 TB worth of data to the Cloud with a 100 Mbps Internet connection, while taking into consideration the associated transfer costs and time taken. Now, AWS provides us with a myriad of different methods to tackle such a situation, and though a full explanation of all methods is beyond the scope of this book, some of those solutions are briefly discussed in the subsections below.

Over the Internet / Site-to-Site VPN

A Site-to-Site VPN creates a secure, encrypted connection (i.e. a tunnel) over the public internet between your on-premises network and AWS. Using a Site-to-Site VPN allows you to securely transfer data in an immediate and hassle-free manner using an internet connection already available to you. However, this method can be time-consuming, especially when transferring large volumes of data, such as 200 TB, at 100 Mbps would take approximately 185 days.

TLDR;

- Immediate to setup.
- Will take 200(TB) * 1000(GB) * 1000(MB) * 8(Mb) / 100 Mbps = 16,000,000s ⇒ ~185 days.

Over Direct Connect (1 Gbps)

AWS Direct Connect provides a dedicated, high-speed network connection between your data center and AWS, bypassing the public internet for a more reliable and consistent transfer experience. Setting up Direct Connect can take over a month due to the physical installation and configuration required, but once established, it reduces the transfer time for 200 TB of data to around 18.5 days. This method is ideal for organizations needing fast, reliable transfers and ongoing large-scale data replication but comes with higher setup costs and complexity.

TLDR;

- Time taken for initial setup is quite long (Over a month).
- Will take 200(TB) * 1000(GB) * 8(Gb) / 1 Gbps = 1,600,000s ⇒ ~18.5 days.

Over Snowball

AWS Snowball is a physical data transport solution where AWS ships you storage devices to load your data, which are then returned to AWS for uploading. This method can handle large data volumes quickly, with 200 TB taking about a week using 2 to 3 Snowballs in parallel. Snowball is particularly useful when internet-based transfers are impractical due to bandwidth limitations. Additionally, it can be combined with AWS Database Migration Service (DMS) for seamless database migrations. It's a secure and efficient method for one-time, large-scale data migrations.

TLDR;

- Can perform tasks by utilizing multiple Snowballs in parallel.
- Time taken to complete the end-to-end transfer is about 1 week.
- Can be combined with DMS.

Snowmobile Capacity

[**Note:** This service is now retired, but this section has been retained from the first edition for the purposes of historical context.]

One of the more unhinged services made available to us, AWS Snowmobile is a secure data transport service designed for transferring extremely large amounts of data (up to 100 petabytes) to AWS. The largest of the Snow family of storage device and services, it is ideal for large-scale migrations such as data center shutdowns, archives, and content libraries, offering a fast and cost-effective way to transfer massive datasets to the cloud securely.

Designed with the purpose of transferring exabyte scale data from on-premise servers to the cloud, AWS Snowmobile is a ruggedized, tamper resistant shipping truck that is 45 feet long, 9.6 feet high, and 8 feet wide.

Fitted with 100 Petabytes worth of storage capacity, networking equipment, etc, the AWS Snowmobile is essentially a data center on wheels which can attach itself to your local network, absorb the necessary data and then deliver it physically to the desired AWS data center.

Snowball Edge vs Snowball

AWS Snowball is a data migration service that uses secure, ruggedized devices to transfer large amounts of data into and out of AWS. Created for the purpose of simplifying and accelerating data migration, and reducing the challenges posed by limited network bandwidth, and is particularly useful for quickly migrating Small to Medium sized enterprises to the cloud for the first time.

AWS Snowball Edge is an offering which extends the functionality of the basic AWS Snowball by adding compute capabilities. The AWS Snowball Edge machines come with built-in storage and compute power, enabling local data processing and analysis before transferring data to AWS with support for running EC2 instances and AWS Lambda functions, enabling complex workflows and custom applications to run on the device. These additional capacities transform the AWS Snowball Edge from a simple storage device to a versatile edge computing machine.

TLDR;

AWS Snowball Edge can run EC2 Instances and lambda functions, allowing for Edge Computing use cases which the normal Snowball cannot. You do pay a little extra for this privilege, however.

A comprehensive, tabular comparison of Snowball and Snowball Edge is presented in the next page. Feel free to utilize the chart when deciding which of the two offerings to utilize.

Feature	AWS Snowball	AWS Snowball Edge
Primary Purpose	Large-scale data migration	Data migration plus edge computing capabilities
Storage Capacity	50 to 80 TB	80TB HDD to 210TB NVMe
Compute Capabilities	No	Yes
Types of Devices	Single configuration	Snowball Edge Storage Optimized, Snowball Edge Compute Optimized, Snowball Edge with Tape Gateway
Local Data Processing	No	Yes
Encryption	256-bit encryption, AWS KMS managed	Same as Snowball
Physical Security	Tamper-resistant, ruggedized, GPS tracking	Same as Snowball
Networking	Standard network interfaces for data transfer	High-speed network interfaces for data transfer and local compute
Supported AWS Services	S3	S3, EC2, Lambda, Tape Gateway
Typical Use Cases	Data migration to AWS	Edge computing, data migration, remote analytics, and processing
Pricing	Relatively low per GB	Relatively higher per GB
Management	Managed via AWS Snow Family Management Console	Same as Snowball
Target Environment	Data centers, enterprises	24/7 Workloads, Remote locations

FSX for Lustre

Amazon FSx is a fully managed service that allows you to launch and run popular file systems in the cloud. It supports various file systems, each tailored for specific workloads, offering the flexibility to choose the right file system for your needs. FSx handles all the complexities of file system deployment, management, scaling, and optimization, allowing you to focus on your applications.

One of the supported file systems by Amazon FSx is Lustre, which is a an open-source, high-performance parallel file system designed for large-scale, data-intensive workloads. It is particularly well-known for its ability to handle massive amounts of data and provide high throughput and low-latency access to that data.

Also worth mentioning is that Lustre is a Linux-based file system, and hence is natively compatible with Linux. In fact, AWS even provides its own Lustre client for Linux instances on AWS, allowing them to more conveniently connect to and use FSx for Lustre.

This makes it especially useful when designing architecture for applications which have intense performance requirements, such as High-Performance Computing (HPC), which necessitate highly performant file systems. Therefore, FSx for Lustre is often the default choice when designing architectures in Linux and/or designing architectures for performance-intensive applications.

FSX for Windows

Another service belonging to the Amazon FSx family, Amazon FSx for Windows is a fully managed file storage service designed to seamlessly run Windows file systems in the cloud. Built on Windows Server, FSx for Windows provides highly reliable and scalable file storage that is accessible over the Server Message Block (SMB) protocol.

This service eliminates the complexity of managing and maintaining traditional file servers for the NFTS, FAT32 and FAT16 file systems used by the Windows Operating System alongside many other convenient features like automatic backups, continuous monitoring, and integration with other AWS services.

FSx for Windows is particularly suited for Windows-based applications that require shared file storage, such as Microsoft SQL Server, SAP, and user home directories. By leveraging this service, organizations can run their Windows file systems in a cloud environment with ease, benefitting from the robust infrastructure that AWS provides.

Additionally, Amazon FSx for Windows includes a File Gateway that operates similarly to the S3 File Gateway. This feature further enhances the service's capability by providing a seamless bridge between on-premises environments and the cloud, allowing for efficient data transfer and storage management.

Compliance vs Governance Object Lock

Object Lock is a feature of Amazon S3 that helps enforce compliance by preventing deletion or modification of objects for a specified retention period. It ensures that data remains immutable and protected against accidental or malicious deletion, supporting regulatory requirements and data governance policies.

The two types of Object Lock in Amazon S3 are:

1. **Retention Periods**: This mode sets a fixed retention period during which objects cannot be deleted or altered. It ensures data immutability for compliance with regulations like SEC Rule 17a-4(f) and the General Data Protection Regulation (GDPR).
2. **Legal Hold**: This mode allows you to place legal holds on objects, which prevents them from being deleted by any user until the hold is removed. It is typically used for preserving data relevant to legal or investigative proceedings.

Also, Object Lock has two modes namely:

1. **Compliance Mode**: In this mode, the data is immutable and cannot be altered or deleted by any user, including governing authorities. This ensures the highest level of data protection, often required for regulatory compliance.
2. **Governance Mode**: This mode allows certain users with special permissions, such as governing authorities, to modify or delete the data if necessary. It provides a balance between data protection and administrative flexibility.

Legal Hold vs Retention Periods

As mentioned before, the two types of Object Lock made available by Amazon S3 are Legal Holds and Retention Periods. Though they both serve the same purpose of protecting an object from being overwritten or removed, they differ from each other in several different key aspects which are discussed below:

Legal Hold allows for the indefinite protection of an object version without a predefined time frame. Once applied, it remains in effect until explicitly removed by a user with the **s3:PutObjectLegalHold** permission.

Retention Period, on the other hand, is a specific duration during which the object version is protected. After this period expires, the object can be deleted or modified according to the permissions set.

Feature	Legal Hold	Retention Period
Duration	Indefinite, until removed	Fixed, defined at the time of setting
Modification	Can be placed or removed by users with the appropriate permissions	Cannot be altered until the period expires
Purpose	Ensures indefinite protection until no longer needed	Ensures protection for a set time period
WORM Enforcement	Yes	Yes

Note: Both Object Lock methods are unified by the Write Once, Read Many (WORM) model, which ensures that data cannot be altered once written.

S3 Bucket Policy

AWS Policies are JSON-based documents that define permissions for what actions can be performed on AWS resources and under what conditions. They are a key element of AWS security model, allowing fine-grained control over access to various services and resources. Each policy consists of statements that specify which principals (IAM users or AWS services) can perform certain actions (e.g., `s3:GetObject`) on specific resources (e.g., an S3 bucket) under particular conditions.

There are two subcategories of AWS policies in particular, which are crucial for managing access control in AWS, though they both serve different purposes and scopes:

1. **S3 Bucket Policies:** These are attached directly to S3 buckets and define permissions for all objects within the bucket. Bucket policies are written in JSON and can specify access permissions based on IP address, AWS account, or other conditions.
2. **IAM Policies:** These are attached to IAM identities (users, groups, roles) and specify what actions they can perform on AWS resources and are also written in JSON but are more granular, allowing fine-tuned control over specific AWS services and resources. They define permissions based on actions, resources, and conditions.

Basically, S3 bucket policies control access to S3 buckets and their contents based on conditions like IP address or AWS account, while IAM policies control access related to actions, resources, and identities within an AWS account. A detailed example of a S3 bucket policy is presented in the next page, with IAM policies receiving a similar treatment in later sections.

S3 Bucket Policy Example and Breakdown:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::example-bucket/*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": "203.0.113.0/24"
                }
            }
        }
    ]
}
```

Note: If you are not someone from a technical background, do not be scared by the curly braces, the JSON code you see above is extremely simple to understand and write.

Short for JavaScript Object Notation, JSON code is structured as a series of keys, (some of which may be nested within one another) with every key having a unique value associated with it. Let us examine the code one line at a time, to better understand this structure and the purpose of the code.

"Version": "2012-10-17"

This line specifies that the 2012-10-17 version of the AWS Policy Language is being used.

"Statement": [ ]

The core of our policy, this line defines the statement key, which acts as a container for any number of different individual permissions (often times also referred to as “Statements”). Consider each individual statement to be like a rule that dictates who can do what with your resources on AWS.

"Effect": "Allow"

The “Effect” field determines whether an action is allowed or denied. In our case, the specified value is "Allow", meaning that the policy grants permission to perform the mentioned action. If the specified value were to be "Deny" instead, then it would explicitly block access to the desired action.

"Principal": “*”

The “Principal” element specifies the users, accounts and/or services which the policy applies to. In our case, the asterisk (*), also known as the wildcard character, is set as the element value. This makes the policy applicable to all users, accounts and/or services (basically everyone). You could replace this with a specific AWS account or user if you wanted to limit access to said agent.

"Action": “s3:GetObject”

Used to specify the operations or API calls that are to be allowed or denied. Each AWS service has its own associated set of actions that describe the tasks that can be performed with said service. For example, the above statement deals with whether the policy allows the associated user to retrieve objects from the S3 Object Storage service.

"Resource": “arn:aws:s3:::example-bucket/*”

A self-explanatory field, the “Resource” element specifies the AWS resources (like an S3 bucket, EC2 instance, lambda function, etc) that the policy applies to. It allows us to ensure that the policy only applies to the objects we deem necessary, with the AWS resources being identified by their ARN (Amazon Resource Name).

For example, in our case, the policy applies to all objects within the S3 Bucket named `example-bucket` as specified by the ARN (*arn:aws:s3:::example-bucket/**)

"Condition": { }

An optional field, the “Condition” element lets us specify the circumstances in which the policy would be effective, allowing us to define additional constraints that must be met for the policy to apply. Mentioned conditions are written as a series of key-value pairs, for example in the mentioned snippet, the “IpAddress” condition key is used to limit the effects of the policy to requests coming from the very specific IP range of `192.0.2.0/24`, adding an extra layer of security to the S3 Buckets.

TLDR;

In our example policy, the following fields are used to:

Statement: Define the individual permissions.

- **Effect**: Specify that the purpose of the policy is to grant permission/"Allow" to perform specific actions.
- **Principal**: Make the policy apply universally to all users/accounts, through use of the wildcard character `"*"`.
- **Action**: `s3:GetObject` allows users to retrieve objects from the S3 bucket.
- **Resource**: Specifies the S3 bucket (`example-bucket`) and all objects inside (`/*`).
    - **Condition**: Limits access based on IP address. In this case, only users coming from the IP range `203.0.113.0/24` are allowed access.

Therefore, the purpose of the policy is to grant read-only access to all objects in the `example-bucket` , provided the users are accessing it from a specific IP range.

S3 Storage Classes and Lifecycle Policies

As mentioned before, S3 has a range of storage classes designed for different balances of cost, durability, availability and performance with prospective users being able to choose one based on their intended use cases. There are six major different storage classes, all of which are discussed below:

1. **S3 Standard**: For frequently accessed data. It offers high availability, low latency, and durability of 99.999999999% (11 nines). This storage class is best suited for general-purpose storage.

2. **S3 Standard-IA (Infrequent Access)**: For data that is accessed less frequently but needs fast retrieval when required. Lower storage cost but retrieval fees apply.

3. **S3 Intelligent-Tiering**: Automatically moves data between S3 Standard and S3 Infrequent Access as it sees fit using proprietary algorithms to determine which of the two tiers an object may be kept in. So objects that have a history of being more scarcely accessed would be automatically moved from S3 Standard to S3 Infrequent Access, without any extra retrieval charges. Most useful in cases where some files are accessed frequently while other files are rarely accessed in an unpredictable pattern.

4. **S3 One Zone-IA**: Similar to Standard-IA but data is stored in a single Availability Zone, making it cheaper albeit less available. This storage class is most suitable for non-critical, reproducible data.

5. **S3 Glacier**: For archival data where retrieval times of minutes to hours are acceptable. It's extremely cost-effective for long-term storage.

6. **S3 Glacier Deep Archive**: The lowest-cost storage class, designed for data that is accessed quite rarely and has the largest retrieval times of up to 12 hours.

Lifecycle Policies

With so many different storage classes available to us, there might be times when we would like to sort the objects into different buckets based on the time that has passed since the object's creation for the purposes of cost optimization, data retention, etc. Lifecycle policies provide us with this luxury.

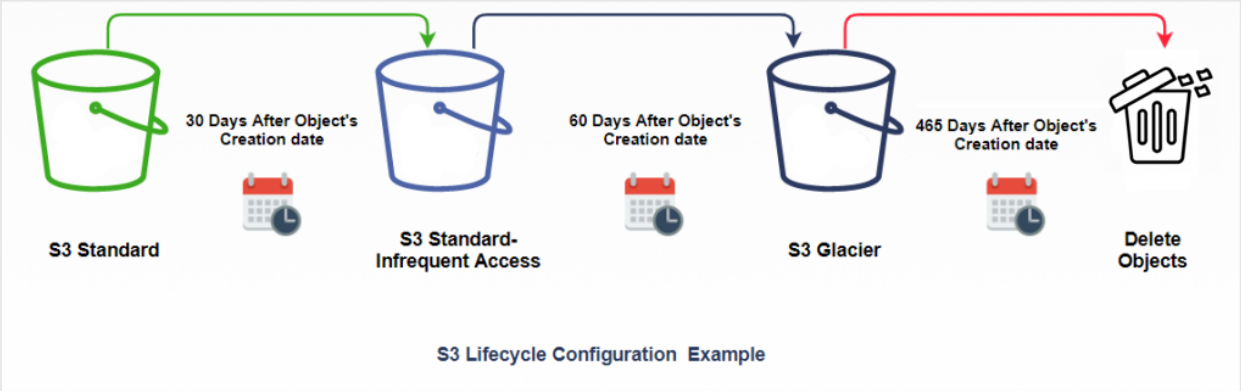
Lifecycle policies in AWS are a set of rules that allow us to manage resources (such as objects in S3 Buckets) and automate certain actions that are to be performed on the data, such as transitioning data to different storage classes or deleting them after a certain period of time.

Let us consider for example, that you are the administrator for a company that uses S3 to store all of its cash transaction logs. Said transaction logs are frequently accessed for the first month after their creation, and are sporadically accessed for the month after that. Though the transaction logs are rarely accessed after that, government regulations mandate transaction logs to be stored up till 465 days after the fact.

Now, in order to meet the restrictions laid out above, we can configure a lifecycle policy for the S3 Bucket where transaction logs are:

1. By default created and stored in S3 Standard
2. Moved from S3 Standard to S3 Infrequent Access 30 days after the object creation date

3. Moved from S3 Infrequent Access to S3 Glacier 60 days after the object creation date.
4. Retained in S3 Glacier until 465 days after the object creation date, after which it is promptly deleted.



Thus, through the use of S3 Lifecycle policies we were able to easily automate the process of managing our S3 object data based on the time that has passed since the object creation date in a hassle-free manner.

S3 Glacier

Of the different storage tiers introduced in the previous pages, there is one in particular that perhaps warrants a closer inspection namely, the Amazon S3 Glacier storage classes.

The Amazon S3 Glacier storage classes are designed to be the most cost-effective storage solutions in the cloud providing a good compromise between performance, price and flexibility. Best suited for the purposes of data-archiving, S3 Glacier has also become a sort-of industry standard for organizations that wish to retain large amounts of rarely accessed, legacy data for the long term mostly due to its relative cheapness, virtually unlimited scalability and 99.99999999999% (11 nines) data durability guarantee.

There are three major S3 Glacier storage classes, each with its own unique set of strengths and drawbacks that must be taken into consideration when choosing between them. Said major S3 Glacier storage classes are as follows:

1. **Amazon S3 Glacier Instant Retrieval:** Built specifically for cases where the stored data may need to be accessed within a more immediate timeframe, S3 Glacier Instant Retrieval provides the lowest-cost storage option with millisecond-level retrieval times, making it suitable for workloads that require fast, frequent access to their stored data (examples include: medical archives, genomics data, etc) offering a good middle ground for users who need a combination of cost-effectiveness and fast access.

2. **Amazon S3 Glacier Flexible Retrieval:** Sometimes simply referred to as S3 Glacier, the S3 Glacier Flexible Retrieval class is designed for situations where

occasional, large-scale data retrievals are needed at a minimal cost alongside slightly more flexible retrieval options, with retrieval times that ranging between as little as 1-5 minutes and up to 12 hours.

For example, bulk retrievals, which is the standard flexible retrieval option can be executed for free within 5-12 hours, making it well-suited for backup and disaster recovery scenarios whilst for cases that require more swift access to the data, the service also offers the option of expedited retrieval, which can retrieve the data in a matter of minutes albeit at an additional cost. Though the retrieval speed is lower than the one provided by Instant Retrieval, the trade-off may be worth it for many organizations due to the reduced cost of storage, which can add up to significant sums quite easily, especially when dealing with large quantities of data.

3. **Amazon S3 Glacier Deep Archive:** The lowest cost storage class, S3 Glacier Deep Archive is designed for data that are to be stored for the long-term, with the expectation of being accessed infrequently. However, the low cost comes at the trade-off of retrieval time, which can take up to 12 hours. This makes it ideal for use cases where immediate access to the data is not a priority but long-term durability and cost efficiency are crucial such as compliance archives, legal records or digital media preservation.

TLDR;

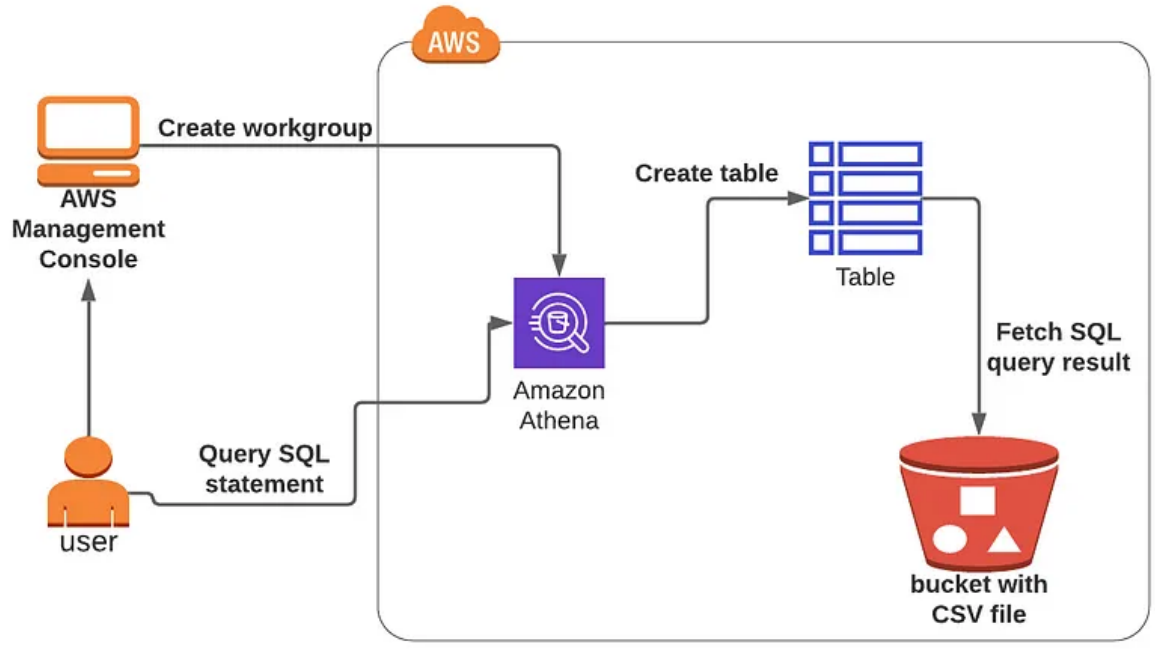
S3 Glacier is a storage service tier specializing in data archival, and offers three different storage classes each offering data access at varying retrieval speeds and price points, often trading one for the other. With the Instant retrieval class for example, being slightly more expensive but providing millisecond-level retrieval speeds, as opposed to the Deep Archive class, which has the lowest storage costs available, but data retrieval can take up to 12 hours.

Athena

Named after the Greek goddess of wisdom, Amazon Athena is a serverless, interactive query service that allows you to analyze data stored in different locations and formats as a single dataset using standard SQL, mostly sourcing the data from Amazon S3 and supports a wide variety of file formats such as CSV, JSON, ORC and Apache Paraquet right out of the box thereby eliminating the need for complex ETL processes and infrastructure setup, enabling the user to start querying their data immediately.

Though not necessarily low latency in nature, Athena usually returns the results of queries in a matter of seconds, making it rather quick to use which alongside its relative cost-effectiveness has led to its widespread integration and use with business intelligence tools such as Microsoft PowerBI and Amazon Quicksight (A service we will learn more about later). In fact, Amazon even provides its own Microsoft Power BI Connector for Amazon Athena which can be used to analyze data present in Amazon Athena from the Microsoft PowerBI desktop application directly (at least on Windows machines).

Also worth mentioning is that workloads that utilize Apache Spark can also be run using Amazon Athena making it extremely easy to run analytics applications without the need to provision or manage resources and infrastructure, due to the serverless nature of Amazon Athena. In fact, since Athena is a serverless offering, when using it to run queries on the desired S3 buckets, the user usually only has to pay for the queries they run, and nothing else, making it a cost-effective solution for ad-hoc data analysis and reporting.



Amazon AthenaSQL Architecture (Credit: Whizlabs)

Amazon Athena works through something called workgroups, where users can control and manage their different query statements as well as the resources required to run said queries. Once the workgroup is created, the user can input SQL queries into Athena through the AWS Management Console or Command Line Interface (CLI) as shown in the diagram above. Athena, a query service, interacts with data stored in Amazon S3, running SQL statements directly on the files without requiring data movement.

Next, a table is created in Athena which serves as a metadata layer over the data in the S3 bucket. The actual data, in this case, is stored as CSV files in S3, and Athena reads the table structure to know how to query it. Finally, after the query is executed, Athena fetches the results from the CSV file in the S3 bucket and presents them to the user in the form of SQL query results, which can be retrieved through the console/ CLI.

Similar architecture and methods are used when running Apache Spark applications using Amazon Athena.

TLDR;

Amazon Athena can be used to run queries on-demand on an S3 bucket with support for a wide variety of file formats such as CSV, JSON, Apache Paraquet, etc. The queries run by Amazon Athena usually return results within a matter of seconds and the service only charges for the queries and not the associated resources required to run them.

The service also supports running analytics applications that utilize Apache Spark workloads.

Transfer Acceleration

Amazon S3 Transfer Acceleration speeds up file transfers to and from Amazon S3 by using AWS's globally distributed edge locations. It leverages Amazon CloudFront's network of edge locations to route data optimally, reducing latency and improving upload and download speeds, especially for users far from the S3 bucket's region. This service is ideal for transferring large files or gathering data sets under the domain of one roof/service quickly and efficiently.

This is especially useful in cases where data stored across various points globally needs to be centrally gathered and performed operations on. However, while it can significantly improve data transfer speeds, it comes with an additional charge compared to regular S3 transfers. Therefore, it’s crucial to evaluate whether the performance improvement is worth the cost for specific use cases. For example, organizations dealing with global users or heavy media files will likely benefit, but for smaller, local transfers, the cost may outweigh the performance gains.

It is also worth noting that once transfer acceleration is enabled for a S3 bucket, it receives an accelerated endpoint with a `s3-accelerate` domain name (for example, `https://<bucketname>.s3-accelerate.amazonaws.com`). In order to benefit from the accelerated data transfers, one has to use the new accelerated endpoint as AWS will not automatically optimize transfers performed using the default standard S3 endpoint.

TLDR;

Aggregate the data from all these global sites as quickly as possible in a single Amazon S3 bucket ⇒ S3 Transfer Acceleration

Accidental Deletion

AWS has made it extremely easy to create, configure, monitor and manage your S3 objects be it through the AWS Management Console, CLI or the S3 API. However, this ease-of-use may also backfire as the barriers preventing the deletion objects are rather low by default. This can, if not managed properly lead to the accidental deletion of the data stored using Amazon S3.

As such, to prevent to accidental deletion of your data in AWS from occurring, it may be desirable to add extra layers of security to protect the data from accidental deletion. The most common way in which this can be achieved is through the combined use of **Versioning** and **Multi-Factor Authentication (MFA)**.

Versioning keeps multiple versions of an object in an S3 bucket, allowing you to recover previous versions if an object is mistakenly deleted or overwritten. It is a built-in feature of all object storage methods, though this does result in greater expenses on the user end as Amazon charges us for the total storage used, which can balloon into large amounts quite easily when storing all the different historical versions of the objects in an S3 bucket.

MFA Delete, short for Multi-Factor Authentication Delete, adds an extra layer of security by requiring additional authentication for delete operations, ensuring that only authorized users can permanently remove data, resulting in the addition of a very human level of safeguarding for your critical data against accidental or unauthorized deletion.

COMPUTE SERVICES IN AWS

Compute: Background

In this section we will discuss fundamental concepts related to compute resources, their nature and their evolution throughout the years. Below is a brief overview of the topics to be covered:

- Compute services, and the hardware resources associated with it
- Virtualization, its purpose and financial impact
- Serverless, its advantages and disadvantages

If the reader believes themselves to already be familiar with the listed concepts, then they can feel free to skip this section and move on to the rest of the chapter.

Compute and related infrastructure

The resources that provide the computer system with processing power and are responsible for the running of applications are called compute resources. Whether it be on-prem or on the cloud, compute resources refer to three components:

- CPU (Central Processing Unit): Often called the "brain" of the computer, the CPU performs calculations and processes instructions required by applications. It handles all logical and arithmetic operations, controlling how quickly and efficiently tasks are executed.
- RAM (Random Access Memory): Already discussed in the storage chapter, RAM is the temporary holding space for information related to the programs currently being run on the computer.
- GPU (Graphics Processing Unit): Used to handle complex parallel-computing tasks like matrix and vector operations, heavily utilized in specific use cases such as gaming, cryptographic and AI operations.

Therefore, when we rent compute resources from a cloud provider what we are really renting is the CPU and the RAM, with us generally being charged based on

the sophistication of the CPU Chip and the associated memory capacity.

Virtualization

Now, a layman might assume that separate physical servers would be allocated for every prospective renter but that is actually not the way cloud providers facilitate us with compute resources. In reality, cloud providers utilize **virtualization**, a technology that divides the underlying compute hardware into abstract partitions, and creating a **Virtual Machine (VM)** based off it, a virtual computer based on the partitioned resources.

Each VM has its own OS and computing resources, and operates independently of the other VMs, even if they were abstracted from the same underlying hardware enabling multiple users to share share a single physical server while maintaining the illusion of individual dedicated servers. Indeed, virtualization makes the leasing of compute resources a much more profitable venture for the cloud providers than it otherwise would be, allowing cloud providers to maximize the mileage they get out of their physical servers.

Serverless

Now, virtual machines have been around since the 1970s but there has been a major development in the field of managing compute and cloud computing in the last 10-15 years or so: **Serverless computing**.

Serverless allows us to run applications without having to manage any of the infrastructure ourselves, with the cloud providers handling the allocation of resources required by our workload. This allows us to focus only on the development of the application and not worry about operating the servers that it runs on. However, there are benefits to having control over the underlying infrastructure, it allows us to be in complete control of the security and configuration of the resources, a level of control taken away from us in the serverless faustian bargain.

EC2, ECS and Lambda

Compute services in the cloud are used for the provision of processing power/compute capacity required to run a workload. So, the virtual machines that are going to be running your applications on the cloud for example.

AWS has three main compute services, namely: **EC2**, **ECS** and **Lambda**. These are discussed further below:

EC2 (Elastic Compute Cloud): For the purposes of this book, the definition of a Virtual Machine (VM) given in the background, i.e. any software-based emulation of a physical computer that is being run on hardware infrastructure owned by the cloud provider (in our case, AWS), should suffice. VMs allow us to create and manage an isolated and portable computer with its own OS (Operating System) on the servers owned by someone other than ourselves, a “Virtual” machine if you will.

EC2 is the AWS service that allows us to rent virtual machines in the cloud, known as instances, offering a wide range of flexibility. These instances can be optimized for various use cases, including memory, compute, or storage. Said EC2 instances are highly customizable, allowing users to configure CPU, memory, storage, and networking capacity according to their needs. They also come with security features like security groups and IAM roles for access control.

Mentioned features as well as the different types of EC2 instances will be discussed in future sections.

ECS (Elastic Container Service): Containers are the "hot new thing" that's gaining popularity throughout the IT world due to their cost-effective computations. As monolithic applications continue to be replaced by microservice applications, it's only a matter of time before container usage becomes more prevalent than instance usage.

Docker, the most widely used containerization technology, defines containers as "a standard unit of software that packages up code and all its dependencies such that the application can run quickly and reliably, even after being transferred from one computing environment to another." Put simply, containers are standardized, lightweight, and secure pre-packaged solutions that run efficiently no matter what situation they're thrown into. Also worth mentioning is that these containers are usually, though not exclusively, created using Docker and its related packages.

ECS is a managed container orchestration service designed for running containerized applications. While it supports and is mostly used to run Docker containers, though it has made an active effort not to be limited to just docker containers, with the service accommodating all containers built to the Open Container Initiative (OCI) image format. Users can manage ECS clusters themselves on EC2 instances or opt for AWS Fargate, a serverless option that abstracts away infrastructure management. With Fargate, users focus solely on deploying and managing containers, without dealing with server provisioning.

A more detailed exploration of Elastic Container Service, and its related services and configurations will be discussed in future sections.

Lambda: Functions are one of the most basic yet powerful concepts in programming, and can be understood as self-contained pieces of code designed to accomplish a certain task. In order to calculate the sum of two numbers for example, a programmer could create a self-contained piece of code called *sumCalc* that would take two numbers, let us call them *num1* and *num2* as arguments, usually represented as: `sumCalc(num1, num2)`

Lambda is a serverless compute service that allows users to run function code such as the one mentioned above without managing servers. Users upload their code, and Lambda handles scaling and infrastructure management automatically. When a Lambda function is invoked, users receive an ARN (Amazon Resource Name) that uniquely identifies the function. Lambda is commonly used with API Gateway to create serverless REST APIs, enabling infinitely scalable endpoints. Lambda prioritizes simplicity, offering a streamlined development experience without the burden of managing infrastructure.

A more detailed exploration of Lambda and its associated configurations will be discussed in future sections.

Types of EC2 Instances

Because EC2 Instances are used to power a wide variety of workloads with similarly varying resource needs, Amazon provides us with different types of EC2 instances. A table showcasing all the different EC2 instance types is given below:

	General Purpose		Compute Optimized	Memory Optimized		Accelerated Computing	Storage Optimized		
Type	t2	m5	c5	r4	x1e	p3	h1	i3	d2
Description	Burstable, good for changing workloads	Balanced, good for consistent workloads	High ratio of compute to memory	Good for in-memory databases	Good for full in-memory applications	Good for graphics processing and other GPU uses	HDD backed, balance of compute and memory	SDD backed, balance of compute and memory	Highest disk ratio
Mnemonic	**t** is for **tiny** or **turbo**	**m** is for **main** or happy **medium**	**c** is for **compute**	**r** is for **RAM**	**x** is for **xtreme**	**p** is for **pictures**	**h** is for **HDD**	**i** is for **IOPS**	**d** is for **dense**

It might seem intimidating at first, but the designation names are there to help you, and allow you to make the best decision in regards to the type of EC2 instance that is best suited for your needs.

Understanding the different EC2 Instance Types

Amazon EC2 instances power a wide variety of workloads, each with unique resource needs. To accommodate these differences, Amazon provides several types of EC2 instances, each designed to optimize for a specific workload type. At first glance, the various EC2 instance designations might seem intimidating, but their naming conventions are designed to help you make the right choice for your specific requirements. Here's an overview of the key instance types:

General Purpose Instances

General Purpose instances provide a balanced mix of compute, memory, and networking resources, making them versatile for a variety of workloads. These instances are suitable for applications such as web servers, small databases, and development environments.

Example Types:

- **T2**: Burstable instances ideal for applications with low to moderate baseline performance needs that can benefit from burst capability. The mnemonic "t" stands for tiny or turbo, reflecting the burstable performance nature.
- **M5**: Balanced instances for consistent workloads, offering a stable balance between compute, memory, and network resources. The "m" is for main or medium.

Memory Optimized Instances

These instances are designed to deliver fast performance for workloads that process large data sets in memory. They typically have a higher amount of RAM compared to other instance types, making them suitable for memory-intensive applications like in-memory databases, real-time big data analytics, and high-performance computing.

Example Types:

- **R4**: Designed for applications needing a high memory-to-CPU ratio, such as in-memory databases and real-time analytics. The "r" in R4 stands for RAM.
- **X1e**: Optimized for extremely large memory needs, often used for full in-memory applications like SAP HANA. The "x" represents xtreme memory (slightly childish I know, but it is what it is).

Compute Optimized Instances

These instances are optimized for compute-bound applications that require high-performance processors. They typically have a high ratio of vCPUs to

RAM/Memory, making them suitable for applications that require intensive computational processing such as gaming servers, scientific modeling, batch processing, and media transcoding.

Example Type:

- **C5**: These instances provide a high ratio of compute to memory. The "c" stands for compute, making them ideal for tasks that need powerful processors but do not require a lot of memory.

Storage Optimized Instances

These instances are designed to deliver high storage performance for workloads that require high I/O performance. They come with local instance storage optimized for high-speed, low-latency access, making them suitable for applications that require frequent and fast access to large data sets, such as NoSQL databases, distributed file systems, data warehousing, and data processing.

Example Types:

- **H1**: These instances use HDD-backed storage and offer a balance between compute and storage performance. The "h" stands for HDD.
- **I3**: SSD-backed instances, ideal for applications needing high IOPS, such as transactional databases. The "i" stands for IOPS.
- **D2**: Offers the highest disk throughput with dense storage for applications like data warehousing. The "d" stands for dense, representing the storage capacity.

Accelerated Computing Instances

Accelerated Computing instances are designed for applications that benefit from hardware accelerators (such as GPUs). These instances are perfect for

computationally expensive applications, such as machine learning, graphics rendering, and gaming.

Example Type:

- **P3**: Ideal for GPU-heavy workloads like graphics processing or deep learning. The "p" stands for pictures, reflecting its role in image and graphics processing.

Choosing the Right Instance

Now, with so many different EC2 instance types one might think that the process of selecting an EC2 instance might be tedious and confusing, but it is actually quite simple once the workload requirements have been decided upon. Are you running a memory-intensive in-memory database? An **R4** instance might be ideal. Need to process large datasets stored locally? Consider a **D2** or **I3** instance. By understanding the key characteristics of each EC2 instance type, you can optimize performance and cost efficiency for your applications. Not to mention, in cases where workload requirements are yet to be identified one can simply start off with a general purpose instance with the intention of shifting to another instance type in the future.

Hibernation and Stop

Sometimes it might not be technically and/or economically prudent for us to have an EC2 instance running 24/7, and we might instead wish to halt an instance at a specific moment in time, with the intention of resuming it in the future as we see fit. AWS provides us with two methods of halting EC2 instances: **Hibernation** and **Stop**. Both methods are discussed in greater detail below:

Stopping an EC2 instance

What we must understand before we differentiate between Hibernation and Stopping an EC2 instance is that all EC2 instances have an EBS (Elastic Block Store) volume attached to them upon creation, with the specifics of the EBS volume being decided by the administrator who performed the creation operation in the first place. Said EBS volumes serve as the default secondary storage device for the created EC2 instances with the files, directories and programs utilized by the EC2 instance stored in it.

Stopping an EC2 instance means temporarily shutting down the virtual machine while preserving its configuration and data on the attached EBS volume. When an instance is stopped, its CPU and RAM are no longer running, but its storage and settings remain intact. The instance goes through a normal shutdown process when stopped, similar to when we turn off our personal computers with all running processes and applications terminated.

Once the instance stops, the account will not be charged for the EC2 compute resources (CPU, RAM) though the account will still be charged for the EBS volumes and any Elastic IP Addresses attached to the instance, that is, until the instance is restarted of course.

Hibernating an EC2 instance

Hibernation is similar to stopping an EC2 instance though it may be best understood as the pausing and resuming of EC2 instances rather than the stopping and restarting of them. When we hibernate an EC2 instance, we cease its operation just like with the stopping of an EC2 instance with the key difference being that the data stored in the primary memory/RAM is retained when an EC2 instance is hibernated, which is not the case with the stopping of an EC2 instance.

This is particularly relevant in environments that are time-sensitive, have long startup times or must retain sticky sessions because it saves the administrator the effort of setting up the environment or applications all over again every time an EC2 instance has to be halted. Instead of having to rebuild the memory footprint from the ground up, hibernation allows applications to pick up exactly where they left off often reducing both the startup time taken by the Operating system as well as the frictions associated with setting up a work environment.

Also worth noting is that just like with the stopping of an EC2 instance, while the instance is in hibernation, you pay only for the EBS volumes and Elastic IP Addresses attached to it; there are no other hourly charges (just like any other stopped instance). However, unlike with stopping which is a function available to all EC2 instances, hibernation is a feature which the user must opt-in to when an EC2 instance is being created, as it is not possible to enable or disable hibernation for an instance after it has been launched.

TLDR;

Stopping: Halts the EC2 instance while retaining the data in the secondary storage; Equivalent to Shut Down / Turn Off options in personal computers.

Hibernation: Halts the EC2 instance while retaining both the data in the secondary storage as well as the data in the memory (RAM). Equivalent to Sleep / Hibernate options in personal computers.

Autoscaling Group and Spanning

Now, EC2 instances are a finite resource capable of handling a very finite amount of traffic and load, with the overburdening of EC2 instances causing performance degradation and bottlenecks at best, and leading to the whole application or program crashing at worst. Autoscaling groups allow us to avoid such scenarios, and ensure that our application can accommodate itself with the right amount of resources, no matter the traffic load.

An Auto Scaling Group is essentially the administrator's best friend when it comes to managing EC2 instances, automatically adjusting the number of instances running at any given time, based on criterion defined by the administrator. Whether your app is facing a sudden spike in traffic or things are slowing down during off-peak hours, an ASG can scale out or scale in to meet the demand, ensuring that your app stays both performant and cost-effective.

At its core, an ASG monitors your application's performance through metrics like CPU usage or network traffic (with a little help from CloudWatch, of course). When traffic increases, the ASG adds more EC2 instances to balance the load. When things quiet down, it reduces the number of instances, saving you money. This automatic scaling makes sure that your app is never under-provisioned or over-provisioned.

Another one of the really cool features of ASGs is the ability to perform **health checks** on your instances. If an instance starts acting up or fails, ASGs detect this and immediately replace it with a healthy one. This keeps your app running smoothly without you having to manually intervene. Plus, ASGs work hand in hand

with Elastic Load Balancing (ELB). As new instances are added, ELB distributes incoming traffic evenly across your instances. So, no single instance gets overwhelmed while others sit idle, ensuring that your app performs consistently no matter how many users are hitting it.

But what if you know in advance that you're going to need more resources, say for a big product launch or during holiday shopping peaks? ASGs also allow for **scheduled scaling**, meaning you can set specific times for when additional instances should be added or removed. This proactive approach is especially useful for companies and organizations that have already identified time-periods when the product may be overloaded.

Also worth noting is that ASGs can be fine-tuned with CloudWatch alarms and custom scaling policies. Let's say you want your ASG to add more instances if CPU utilization hits 80%. Or maybe you want it to scale back once CPU usage drops below 50%. These kinds of scaling policies let you control exactly how and when your instances are scaled based on real-time data.

Finally, ASGs can span across multiple Availability Zones (AZs) and is an easy and convenient way for us to ensure high availability. If one AZ is experiencing issues for example, your ASG might automatically distribute the load to instances in other AZs. This cross-zone scaling provides the instances comprising the ASG with resilience against potential failures in a single geographic area.

TLDR;

Auto Scaling Groups (ASGs) automatically adjust the number of EC2 instances based on real-time demand, ensuring your app stays performant without overspending. They monitor health, distribute traffic evenly through Elastic Load Balancing, and can even schedule scaling for predictable traffic spikes. Plus, they span multiple Availability Zones to keep your app highly available.

Benefits of Overprovisioning EC2 instances

Consider a scenario where the administrator is designing an ASG for a system which calls for a minimum of 2 instances operating at all times, then common sense would tell us to create an ASG with a minimum of 2 instances. However, in the real world, you will often find that administrators go a step further and have at least 3 instances running at all times.

This might seem odd for the uninitiated, after all, is not the entire point of autoscaling for us to be saving costs by using only the required amount of resources? So then why would we wish to provision more EC2 instances than absolutely necessary?

This is because if an Availability Zone (AZ) outage occurs, your Auto Scaling Group (ASG) will spin up a new instance in an unaffected AZ to compensate for the lost one. However, this new instance doesn't come online immediately, so for a short while, you'll be running with only 1 active instance—leaving your application below the minimum threshold required to operate.

Therefore, over-provisioning by just one instance can be the difference between maintaining your app's availability and leaving it vulnerable during an AZ outage making it a simple, proactive approach that buys you time during unexpected events, and ensures that your application stays resilient and continues to readily meet user demand.

Reserved Instances and On-Demand Capacity Reservation

When we use EC2 instances, what we are essentially doing is renting computer infrastructure from AWS. The default renting agreement, i.e. On-Demand EC2 instances provides us with EC2 instances on-the-go, provisioning us with resources according to the load being put on their computers. However, this is not the only renting agreement available to us, and if we are already aware of the compute capacity that is required for our workload then AWS provides us with two other methods of renting EC2 instances: **Reserved Instances** and **On-Demand Capacity Reservation**.

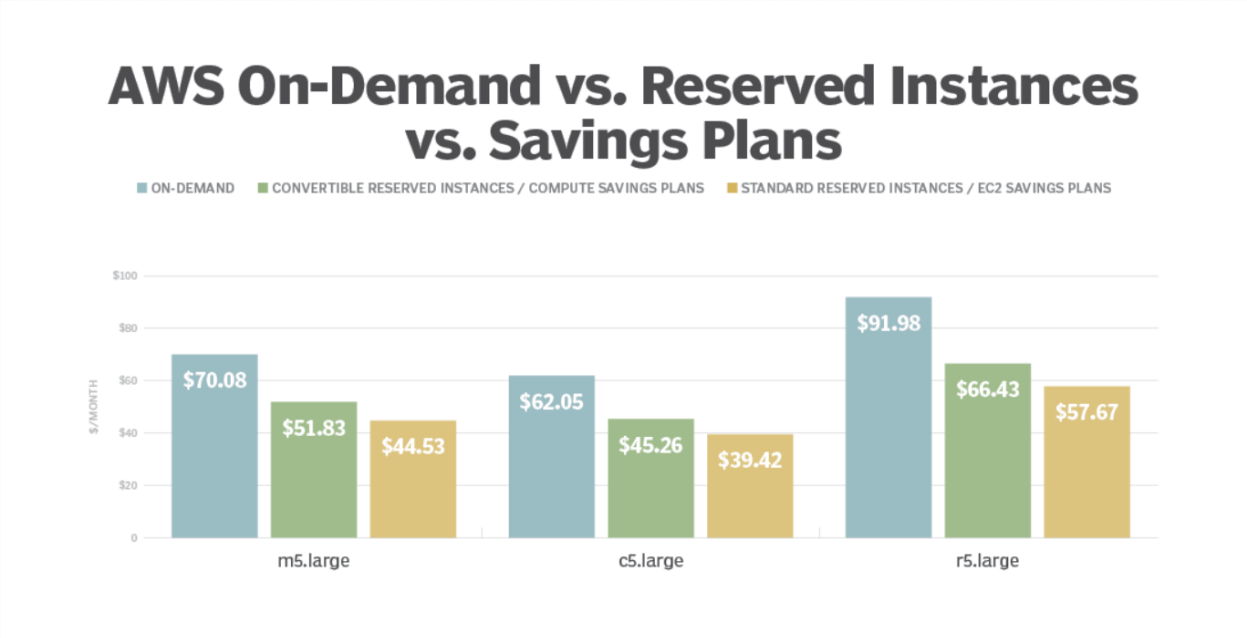
Leveraging these pricing models allow us to get a fixed number of EC2 instances of a specific hardware configuration potentially for a fraction of the price usually extracted when utilizing On-Demand EC2 instances.

Reserved Instances (RIs) in AWS are a cost-efficient way to secure compute capacity for long-term, predictable workloads. By committing to a one- or three-year term, RIs offer significant discounts (up to 75%) compared to On-Demand instance pricing. This pricing model is especially useful for workloads that have steady, consistent demand, such as databases, backend services, or enterprise applications. When you know your infrastructure needs in advance, RIs allow you to plan ahead and reduce overall cloud costs without sacrificing performance or availability.

RIs come in two types: **Standard** and **Convertible**. Standard RIs offer the deepest discounts but require a fixed commitment to a specific instance type, operating system, and tenancy. In contrast, Convertible RIs provide more flexibility by

allowing changes to the instance family, OS, or tenancy during the reservation period, albeit with slightly lower savings. Additionally, users can choose different payment options (All Upfront, Partial Upfront, or No Upfront) to balance between immediate cost savings and cash flow management.

While Reserved Instances can drastically reduce costs, it is crucial to remember that they do come with certain caveats which must be taken into consideration. If the workload pattern changes and the user does not fully utilize the reserved capacity for example, they are still obligated by contract to pay for the reservation. Therefore, careful planning is essential while utilizing Reserved Instances to avoid any unnecessary financial commitments.



Source: TechTarget and AWS Pricing Calculator

On-Demand Capacity Reservations offer AWS users the flexibility to reserve EC2 capacity for any fixed duration of time that is comfortable to them, without the need to specify exact instance types or Availability Zones (AZs). This approach provides much greater control over resource management, particularly for workloads requiring precise instance placement and configuration.

Unlike Reserved Instances (RIs), which necessitate a fixed one-year or three-year commitment, On-Demand Capacity Reservations are more suitable for tasks that have known capacity requirements but a shorter duration. If a workload demands reserved capacity for a temporary period, these reservations allow you to secure resources without long-term contracts, making them an efficient alternative to RIs. However, do note that On-Demand Capacity Reservation is simply a guarantee of compute capacity, and does not offer any reduction in cost when compared to normal On-Demand pricing, unlike reserved instances.

TLDR;

AWS provides us with two methods of reserving compute capacity: **Reserved Instances** and **On-Demand Capacity Reservation**.

Reserved instances: Provide the most savings but the contract must be for a 1 or 3 year duration.

On-Demand Capacity Reservation: Provide capacity without any long-term contracts, making them better for more temporary workloads though they are slightly pricier than Reserved instances.

Elastic Container Service (ECS)

Another one of the three aforementioned major compute options made available by AWS, **Elastic Container Service (ECS)** is Amazon's fully managed container orchestration service, designed to simplify the deployment and management of Docker containers. Since we've already discussed containers and Docker in previous sections, I will spare the reader from a repeat explanation.

ECS allows you to run containerized applications in two main ways: on a scalable cluster of EC2 instances, where you control the underlying infrastructure, or using **AWS Fargate**, a serverless compute option that abstracts away infrastructure management, allowing administrators/developers to focus solely on the containers themselves without having to worry about the hardware it is running on.

Especially useful for administrators that wish to avoid the operational overhead of managing the container's compute infrastructure (such as provisioning and scaling of EC2 instances), AWS Fargate is a relatively painless method of operating containerized workloads. However, this does mean relinquishing a great deal of control over the infrastructure used over to AWS. However, Fargate is neither universally desirable, nor universally adoptable, as organizations and workloads that desire a certain amount of control over the infrastructure or need specialized configurations, will always find running ECS on EC2 Instances to be the more appropriate option.

Now, while ECS is an extremely powerful tool for container orchestration, many organizations prefer to use Kubernetes, an open-source container orchestration tool and platform. Kubernetes offers more flexibility and extensibility, with a rich ecosystem of plugins and integrations which has led to the platform becoming somewhat of an industry standard. To meet this demand for Kubernetes

workloads, AWS also started offering **Elastic Kubernetes Service (EKS)**, a fully managed Kubernetes service that allows you to run Kubernetes clusters on AWS without the complexity of managing the Kubernetes control plane yourself.

With EKS, much like ECS, you can run Kubernetes workloads on either EC2 instances or AWS Fargate, providing flexibility based on your operational needs. When working with a team or organization which is already experienced with Kubernetes or when working with workloads that require Kubernetes-specific features and integrations, using EKS over ECS might be the right choice.

TLDR;

AWS provides two main services for managing containers: **Elastic Container Service (ECS)** and **Elastic Kubernetes Service (EKS)**.

Both services simplify the deploying and managing of Docker containers, with the option to run workloads on EC2 instances (which offers more control and flexibility) or AWS Fargate.

An extension on the ECS and EKS offerings, **AWS Fargate** hands over the responsibility of managing the infrastructure required to operate the containers, and is ideal for teams avoiding the operational overhead associated with such tasks.

Lambda

The final one of the aforementioned major compute services, AWS Lambda is a FaaS (Functions as a Service) offering provided by AWS, and allows us to run small self-contained pieces of code without having to provision and manage hardware infrastructure yourself. Usually marketed as both infinitely scalable and incredibly cheap, Lambda is particularly useful for minute tasks that need to be run on-demand in an irregular fashion, where renting a server or EC2 instance 24/7 may not be desirable.

It is also worth noting that whilst AWS Lambda is a serverless offering, we do have to specify the memory capacity allocated for every invoked lambda function (usually chosen during the creation stage of the lambda function). By default, all Lambda functions have 128 MB of memory allocated to them though we can increase that value up to 10 GB.

Though we usually deliver our function code to Lambda through packaged zip files, Lambda is also capable of running containerized images, which we can either upload directly or pull from **Amazon ECR (Elastic Container Registry)**, an AWS service designed to store and manage container images. The ability to run containerized images is especially useful in deployments which require external dependencies to be installed on the system (For example, imagine running python code in a system where python was not even installed to begin with).

ENV variables for Lambda functions

All modern computer operating systems utilize a set of key-value pairs called **Environment Variables** to store configuration data that applications may require access to during their runtime, allowing developers to set certain settings and values without embedding them directly into the code. This is especially useful when managing sensitive information such as database credentials, API keys, or authentication tokens whereby instead of hard-coding these details into your application, we can simply store them as environment variables that your application can reference during execution.

Though we do not have control over the hardware infrastructure using which Amazon in its infinite wisdom (*sarcastic*) decides our Lambda function will be run, Amazon does provide us with the ability to set the environment variables which might be referenced by our function.

Do remember however that when we create or update Lambda functions that use environment variables, AWS Lambda encrypts them using the AWS Key Management Service (often shortened to KMS, will be discussed deeply in later sections) with the intention of decrypting them and making said values available to the Lambda code whenever the function is invoked.

DATABASES
IN
AWS

Databases: Background

In this section, we will discuss what databases are and their types. The topics to be discussed in this chapter are as follows:

- Introduction to databases
- SQL, its advantages and disadvantages
- NoSQL, its advantages and disadvantages

If the reader believes themselves to already be familiar with the listed concepts, then they can feel free to skip this section and move on to the rest of the chapter.

Introduction to Databases

Databases are the main method of storing and organizing volumes of related data in modern organizations, serving as the backbone of data management by providing structured environments for storing, retrieving, and updating information. Whether maintaining employee records, managing transactions in an e-commerce system, or supporting real-time data analytics, databases are essential for organized data storage and retrieval.

Today, databases come in a variety of forms, and are often tailored for specific needs. From traditional relational databases (SQL) designed to handle structured data with strict relationships, to flexible NoSQL databases, which excel at handling unstructured or semi-structured data in real-time applications.

SQL

Short for Structured Query Language, SQL databases organize data into tables that define relationships between different types of data. They offer a structured tabular format, providing a high level of consistency and reliability for applications where data integrity is crucial, such as finance and healthcare. It is ideal for data

that benefits from adhering to the ACID (Atomicity, Conistency, Isolation and Durability) set of properties.

Having consistent schema makes it easy to run queries on the stored data, which is where SQL gets its name from. All SQL databases regardless of their differences store data in tables that can be queried according to similar commands and principles.

However, the structured format offered by SQL databases mean that the type of database is ill suited for the handling of unstructured data, data that cannot easily be organized into a rigid database schema.

NoSQL

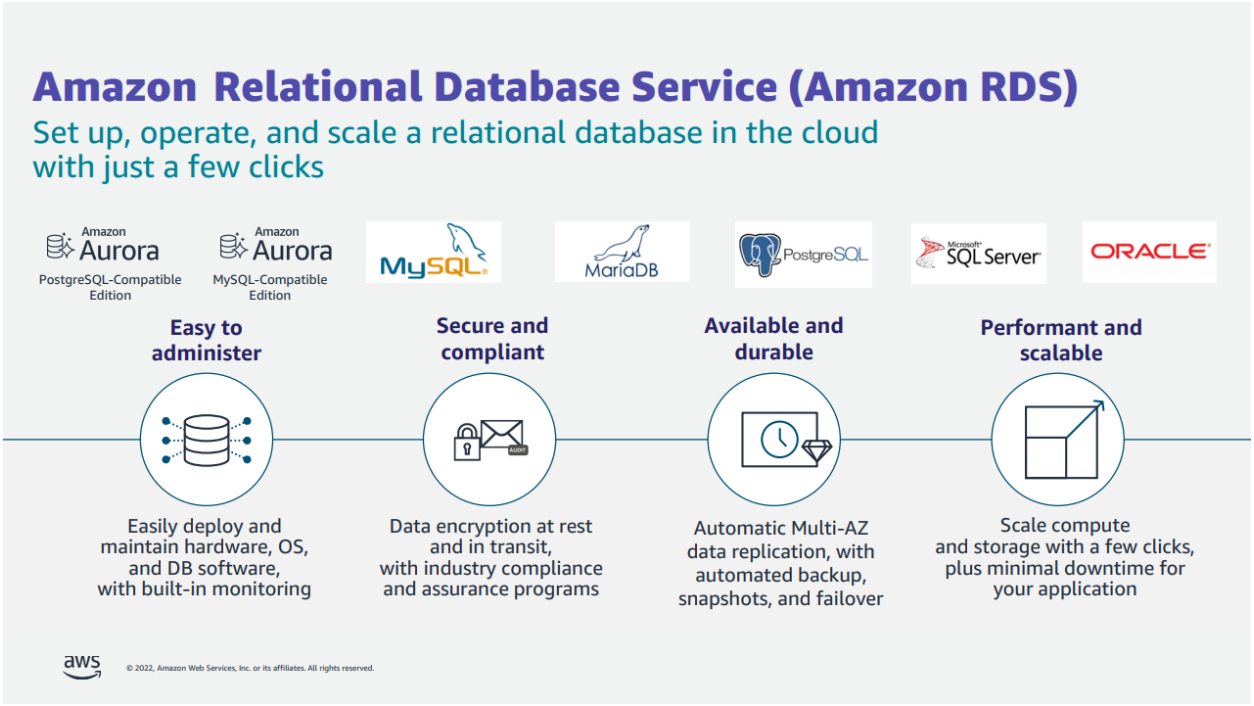
Short for Not Only SQL, NoSQL databases operate without a defined, rigid schema, the defining property of SQL databases. They store data in non-tabular formats, prioritizing flexibility and scalability, making them ideal for handling large volumes of unstructured databases.

Now, NoSQL databases come in many different flavors, and are often fundamentally different from one another. A full list of all the different types of NoSQL documents is beyond the scope of this book but some of the major ones include: **Document databases**, **Key-value databases** and **Graph databases**.

While this variety means NoSQL databases can handle a wider range and variety of data, it also means that there is no standard querying model/language that works across different types of NoSQL databases, with integration between different types of NoSQL databases often being a subject of particular hassle.

Relation Database Service (RDS)

Managing and maintaining the infrastructure required to run SQL servers can often be a troublesome task (to say the least). Amazon Relational Database Service (RDS) is a **managed SQL database service** provided by Amazon Web Services (AWS) which makes this task easier, allowing you to set up and operate SQL servers using AWS infrastructure, abstracting away a great deal of the headaches associated with having a relational database. Many organizations in fact, prefer managed database services like RDS over self-managed databases because of how hassle-free they make a lot of the database-adjacent tasks such as data migration, backup, recovery and patching.



Source: AWS

Boasting support for a whole host of database engines (listed above), Amazon RDS is considered an industry standard for storing and organizing data in the cloud using databases for reasons mentioned in the illustration. Additionally, though this fact is probably clear to the reader by this point, it is widespread enough of a misunderstanding that I find it necessary to clearly re-iterate: **Amazon RDS itself is not a database; It is a service used to manage relational databases.**

The main allure of RDS and its popularity in my personal opinion is that once we adopt and get used to the features of managed services such as Automatic scaling, encryption, Read Replicas, Multi-AZ databases, proxies, etc (Mentioned features will be discussed in later sections), the convenience factor is so great that it becomes hard to decouple entirely and go back to being self-managed.

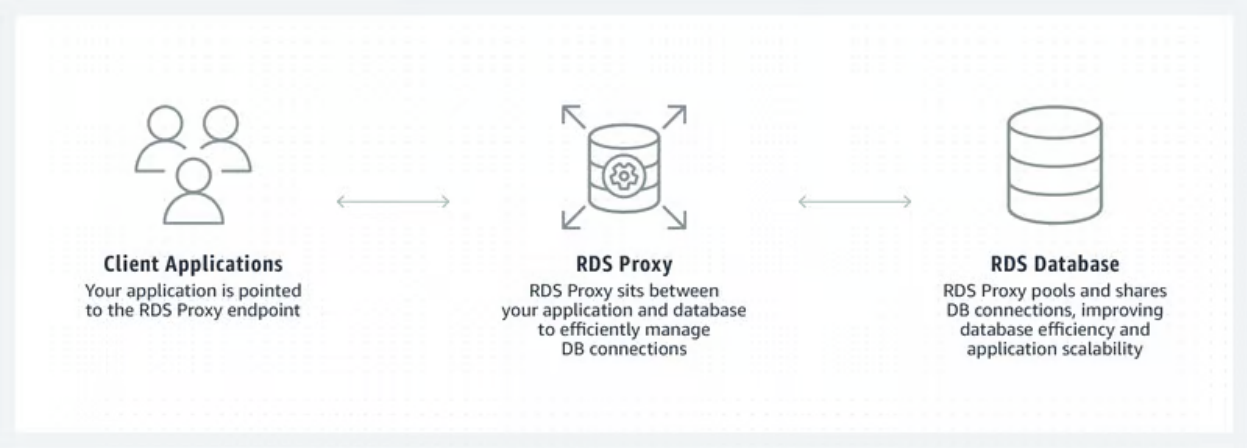
RDS Proxy and Read Replicas

A database proxy is best understood as a middle-man that resides between an application and a database, receiving all the requests sent by the application before forwarding it to the desired database. Now, why is it exactly that we need this middle-man? What possible benefits could there be after all, to adding a extra hop between the client and the database? Well, there are a few, such as:

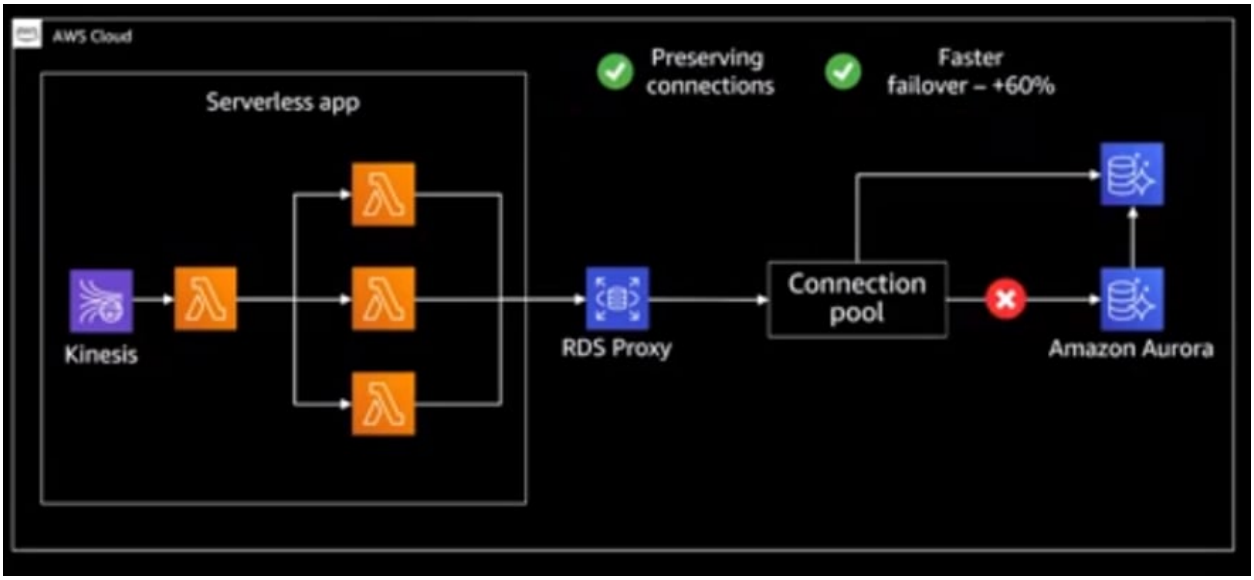
- **Connection Pooling**: Without a proxy, every time an application makes a request, it might need to establish a new connection to the database. This is resource-intensive, especially with high traffic. The proxy manages a pool of connections, reusing them efficiently, reducing the time and resources spent on opening and closing connections.
- **Load Balancing**: If multiple database instances exist (e.g., for replication or sharding), the proxy can distribute traffic evenly across them. This reduces the risk of overloading any single instance and ensures a smoother application performance.
- **Failover and High Availability**: In the event that a database instance goes down, the proxy can detect this and reroute traffic to a healthy instance automatically, without requiring changes on the application side. This ensures continuous availability.
- **Security**: The proxy can centralize access controls, authentication, and encryption, acting as a gatekeeper to protect the database from direct exposure. This adds an extra layer of security while streamlining how applications interact with the database.

RDS Proxy is an AWS service that provides us with a fully managed, highly available database proxy just like the one discussed above, acting as an

intermediary between our application (usually also being hosted on an EC2 instance) and Amazon RDS database instances, enhancing their relationship with all the benefits mentioned above.



The ability of the RDS Proxy to pool connections is particularly useful for application developers and cloud architects as it allows us to preserve connections and/or failover to an alternative database instance during cases where the connections between the database and the application are interrupted due to, for example, too many agents trying to connect at the same time, as shown in the architecture diagram below:



RDS Read Replicas

RDS Read Replicas are duplicates of your primary Amazon RDS database that are specifically designed to handle read-only queries. They play a key role in scaling database performance, and are particularly useful in scenarios where an application has a read-heavy workload but doesn't require comparably as many write operations such as for example, querying tools and programs.

Though they do cost extra, many organizations do not mind paying the cost due to the myriad of benefits associated with having Read Replicas such as:

1. **Offloading Read Traffic**: By creating one or more Read Replicas, organizations can direct read-only queries (like SELECT statements) to these replicas instead of burdening the primary database. This reduces the load on the primary instance, allowing it to focus more on write operations (like INSERTs, UPDATEs, or DELETEs).
2. **Horizontal Scaling**: Read Replicas enable horizontal scaling of read operations. If the number of read requests increases as traffic grows, organizations can create additional replicas to handle the increased load without affecting the performance of the primary database.
3. **High Availability for Reads**: In the event of a failure or downtime of the primary database, the Read Replicas can continue serving read requests, ensuring continuous availability for read operations.
4. **Geographic Distribution**: Read Replicas can be deployed in different regions to reduce latency for users in various geographic locations. This allows users to access data from the closest replica, improving read query response times.

Aurora

Considered as a part of the RDS extended family of offerings, Amazon Aurora is a proprietary relational database engine built by Amazon to take full advantage of AWS's vast cloud infrastructure. Aurora was designed from the ground up to leverage the distributed, scalable, and fault-tolerant capabilities of the cloud. While it is a proprietary database engine, it is compatible with MySQL and PostgreSQL at the relational database layer, meaning that applications written for those databases can run on Aurora with little to no modification.

With features like Aurora Global Database allowing for cross-region replication and Multi-AZ deployments, Aurora can efficiently power modern, scalable applications. However, the privilege of having Amazon manage your servers for you does come at an extra cost, thereby making the service more expensive sometimes, especially in comparison to RDS. This cost is however worth it as Aurora is considered to be superior to RDS, delivering higher performance and scalability than traditional database engines with Aurora providing upto five times the throughput of normal MySQL and upto three times the throughput of normal PostgreSQL databases.

Aurora is also better suited for the handling of large, distributed workloads due to its superior availability and performance, with built-in support for features such as automatic scaling of storage (upto 128TB) and automatic addition of read replicas (upto 15 separate instances). Aurora DB clusters are also highly fault tolerant by nature, where cluster volumes span multiple Availability Zones (AZs; will be discussed in further sections) with each Availability Zone containing a copy of the data, allowing said clusters to tolerate the failure of an Availability Zone without any major interruptions.

Database Cloning

A database clone is an exact copy of an existing database, created usually for the purpose of testing, development, troubleshooting, or scaling. It replicates the entire database, including its structure, data, indexes, and sometimes user privileges, depending on the system. In the context of RDS, Database cloning is a method that allows us to create a database clone of an RDS database instance, with all the same settings and configurations.

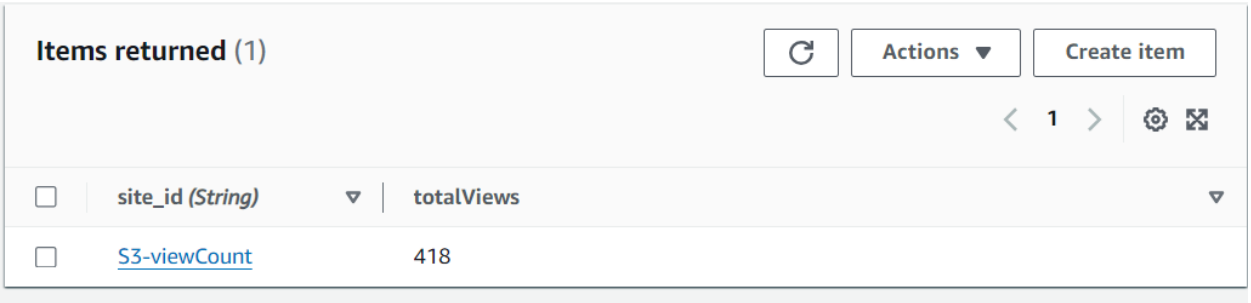
RDS database cloning is also particularly useful when creating **staging databases** (i.e. temporary databases created to serve an Extract, Transform and Load, aka ETL procedure; ETL procedures will be discussed in-depth later) on-demand in a streamlined, efficient and almost real-time manner without the overhead of traditional backup and restore processes.

However, it is worth remembering that database clones are (by deliberate design) isolated from their source databases, meaning any modifications made to the cloned database do not affect the original, and the clone can be queried and updated independently. This isolation is critical for scenarios where testing or development needs to mimic production data without risking changes to the live environment.

DynamoDB and DynamoDB Streams

As discussed in the start of this section, databases can broadly be categorized into two categories: SQL and NoSQL, with NoSQL offerings being famous for their flexibility and ability to store and organize structured as well as unstructured and semi-structured data, well DynamoDB is one such a database, offered to us by Amazon as a fully managed DBaaS, meaning that we will not have to handle tasks like hardware provisioning, setup, configuration, or scaling.

With support for both key-value and document data models, DynamoDB organizes data into tables, with each table consisting of items (analogous to rows) and attributes (akin to columns). It is especially great for the handling of large amounts of data with high availability, and is ideal for applications requiring consistent, low-latency data access without compromising on performance or scalability.



A simple example of a DynamoDB table with two key-value fields, site_id and totalViews

Querying in DynamoDB also works quite differently than with conventional SQL-based relational databases, using fields called Partition Keys and Sort Keys. Partition keys are the primary key of the DynamoDB table, uniquely identifying

each item in the table while sort keys are an extension of primary keys, allowing us to uniquely identify items with the same primary key. Sort keys can be thought of as a sub-primary key, as they are a composite key consisting of both a partition key and a sort key enabling us to differentiate between different items with the same partition key.

Additionally, DynamoDB also has multiple interesting additions that can enhance the capabilities and functionality of the service, of which there are two in particular which I feel deserve special attention: **DynamoDB Streams** and **DynamoDB Accelerator**.

DynamoDB Streams is a feature of DynamoDB that captures real-time data modifications in DynamoDB tables, enabling applications to respond promptly to updates. This feature facilitates use cases such as real-time analytics, cross-region replication. Note however, that DynamoDB streams is not enabled by default and instead must be manually enabled if it is to be taken advantage of.

DynamoDB Accelerator or DAX for short (as it is often commonly referred to), on the other hand is not a feature of DynamoDB but rather a separate service designed to improve the performance of DynamoDB tables by providing an in-memory caching layer specifically for said tables. This is particularly useful for read-heavy workloads as frequently requested data can be stored in the DAX cache and applications can fetch said data without having to repeatedly ping DynamoDB for it, greatly reducing the load on the DynamoDB tables.

OLTP and OLAP

Another manner of categorizing databases is through database processing systems (the methods used to create, manage and manipulate databases), of which there are two major ones: **OLTP** and **OLAP**.

OLTP, short for Online Transaction Processing, typically involves managing and processing more atomized transactional data in real-time. It is optimized for rapid and efficient querying and updating of individual data records. OLTP databases are typically used for day-to-day transactional operations such as order processing, financial transactions, and customer relationship management (CRM). AWS services that support OLTP include:

- **Amazon RDS (Relational Database Service)**: Provides managed database services for relational databases like MySQL, PostgreSQL, Oracle, and SQL Server, optimized for OLTP workloads.
- **Amazon DynamoDB**: A fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. It is ideal for applications requiring low-latency data access and high throughput, such as gaming and mobile apps.

OLAP, short for Online Analytical Processing, on the other hand, focuses on processing large volumes of data for complex analytical queries supporting operations such as aggregations, calculations, and data mining to derive insights and make informed business decisions. AWS services that support OLAP include:

- **Amazon Redshift**: A fully managed data warehouse service that allows you to run complex queries across petabytes of structured data. It has built-in integration with Business Intelligence and reporting tools for data visualization and analysis.

IAM DB Authentication

When dealing with databases, having systems put in place for the secure, reliable authentication of agents (such as a user or an application) which have access to the database are legitimate, and prevent unwanted actors from engaging with the database especially when it deals with sensitive information.

RDS instances therefore, utilize and support a whole suite of established traditional authentication methods such as Usernames and Passwords, SSL / TLS certificates, etc. However, AWS has its own set of services that facilitate easy and simple methods of authentication and identity management under their IAM umbrella (will be elaborated on in future sections) of services and are especially critical when connecting AWS resources with RDS databases.

EC2 instances for example, are not allowed permission to connect to RDS tables and databases by default, and we must both enable an option with RDS called IAM DB Authentication as well as assign an IAM role to the EC2 instance if we desire to circumvent the restriction.

Disaster Recovery

Databases are an integral component of any modern business, and their failure can often be quite devastating and disrupting to its operations. Planning for disasters and setting up contingencies for database related disasters is hence considered as an important task by many companies, particularly those related to database recovery.

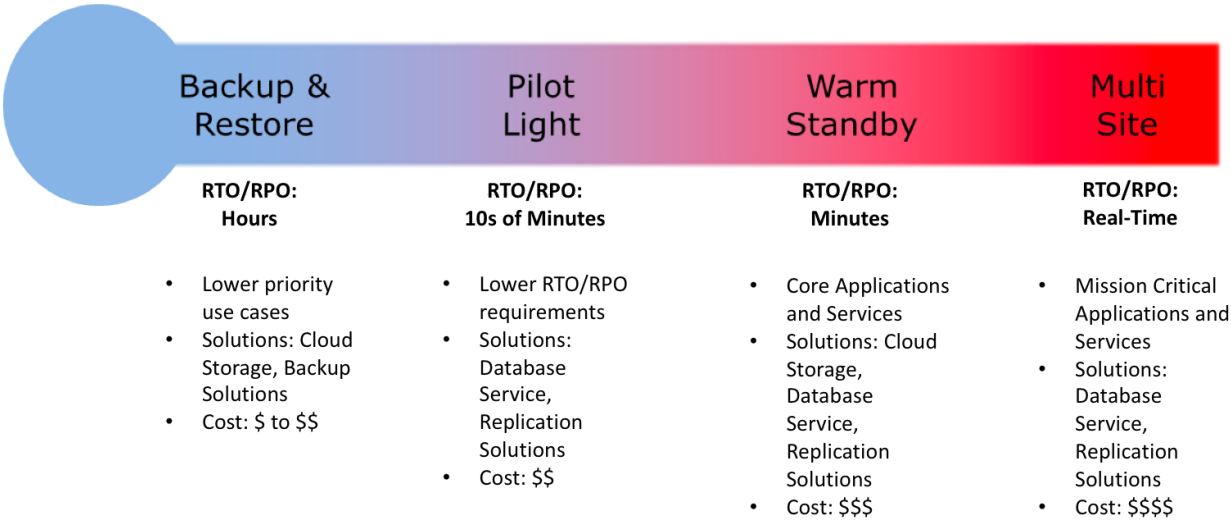
There exist however, a myriad of different approaches and practices for disaster recovery which are measured using the metrics **Recovery Point Objective (RPO)** and **Recovery Time Objective (RTO)**.

RPO defines the maximum allowable data loss in terms of time; for instance, if a business has an RPO of 30 minutes, it means that, in the event of a disaster, they can only tolerate losing data from the last 30 minutes.

RTO on the other hand, specifies the maximum time allowed for system recovery after a failure. For example, if a company has an RTO of 2 hours, it indicates that they must restore their database and resume normal operations within that timeframe following an outage.

As such, a low RTO/RPO method of disaster recovery is acceptable when downtime is not a dealbreaker, and the business can handle some amount of lost data whilst a high RTO/TPO method of disaster recovery is more suited for databases that must be maximally available and cannot handle the loss of data.

An in-depth discussion of all the disaster recovery methods is beyond the scope of this book, and hence I will be relying on a diagram made available by AWS. Consult the diagram for a brief overview of all the disaster recovery methods and their respective RTO/RPOs:



I encourage the readers to research the different disaster recovery methods listed above on their own, searching the internet for their accompanying architecture diagrams as well if desired.

Primary Database and CNAME

Cloud architects and engineers often like to have multiple instances of the same database running in multiple Availability Zones (or AZs; discussed in the next section) in order to maintain high availability and low-latency, among a whole host of other reasons. When spinning off RDS databases into a multi-AZ setup however, we must designate one of the database instances as the primary or main database instance.

This so-called primary database instance is also the holder of something called a Canonical name, or CNAME for short. A CNAME is a type of DNS record that allows one domain name to act as an alias for another, essentially mapping an alias name to a true or canonical domain name. This means that when an application connects to the CNAME, it will resolve to the actual endpoint of the database instance, ensuring smooth communication.

In the event that the primary database fails, the Canonical Name (CNAME) record in the Domain Name System (DNS) is automatically updated to redirect database requests to the standby instance, a process that is considered essential for seamless failover. By updating the CNAME, the system ensures that applications continue to connect to the standby instance without interruption. This process minimizes downtime and enhances the overall reliability and resilience of your database services, allowing for high availability even in the event of a failure.

Redshift and AWS Lake Formation

Finally, though technically a cloud storage solution, I do find it necessary to at least mention data warehouses and data lakes in the storage section as they are often utilized to store and manage large volumes of data (usually data that is miscellaneous in nature). A purpose that keen readers may recall was mentioned as the defining characteristic of storage in the background section of the chapter.

Now, data warehouses are designed to assist with business intelligence and data analytics tasks and usually is responsible for storing data, performing simple quantitative analyses on the data and visualizing data. Amazon's data warehousing solution is called **Redshift** (discussed already in the OLAP section), and focuses on operating more so as a business and data analysis tool than as a service for the storage of data. Redshift is built around a cluster of computing nodes which sit on top of Redshift's data store allowing us to prepare, perform simple arithmetic on and visualize quantitative data at the petabyte-scale.

Data lakes on the other hand, allow an organization to store virtually any type of data at an almost unlimited scale. They are used when storing data pertaining to a wide range of services and belonging to a diverse set of formats (such as databases, BI tools and multimedia files) all in a single location. They are highly adaptable and can change with ease to meet new requirements. **AWS Lake Formation** is Amazon's managed service for the creation and deployment of data lakes, and is usually utilized when organizations using AWS require a repository to hold vast amounts of both processed and unprocessed data from a myriad of unrelated sources, which companies sometimes have to do, in order to comply with regulatory standards and policies.

NETWORKING
IN
AWS

Networking: Background

In this section we will discuss the fundamentals of networking, how AWS global network is organized and the components it is comprised of. A concise list of the topics is given below:

- Brief History of The Internet
- OSI Model
- IP Addresses and DNS, what they are and how they work
- AWS Global infrastructure, Availability Zones and Local Zones
- Caching and CDN, what they are and how they work

If the reader believes themselves to already be familiar with the listed technologies, then they can feel free to skip this section and move on to the rest of the chapter.

Brief History of The Internet

We would not be talking about cloud computing if it were not for the internet, this monumental invention of humans that allows a diverse range of computers quite geographically dispersed to communicate with each other. It is after all, the chief way in which most businesses relying on AWS access their rented resources and the chief way in which your customers will access AWS resources if you're running a public-facing operation. As such, I find it necessary to provide a brief history (albeit a very diluted one) on how the internet was created.

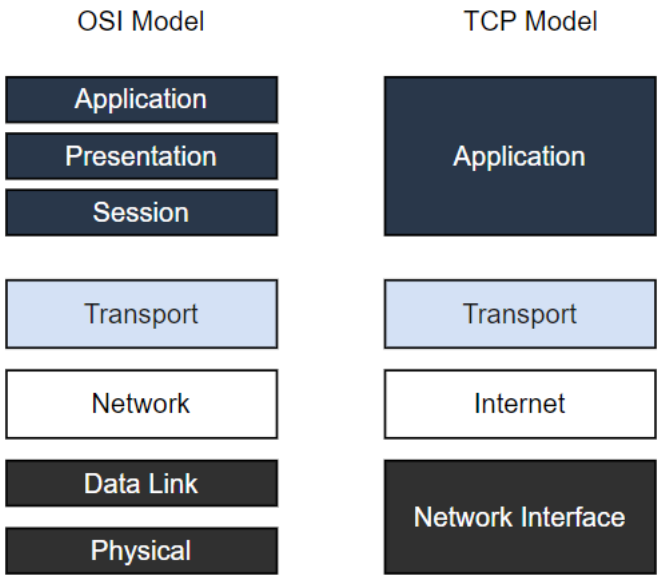
Before the advent of the internet, the largest networks of connected computers were all private networks, they were networks owned either by government agencies, university campuses and large corporate houses which were all using separate pieces of technology to organize and operate their networks. And since these networks did not utilize standardized ways of operating, communicating across networks was very difficult. A computer in a network owned by a university

campus therefore, could not talk and share information with a computer owned by a large corporate house, or even another university.

The United States government has an agency called the Defense Advanced Research Projects Agency (DARPA), which is responsible for performing R&D operations for the American military. In 1969, DARPA started a project that would later become known as ARPANET, a project to unify various private networks and allow their constituent computers to talk with one another. The success of ARPANET led to others trying the same and in 1973, a proposal to connect these disparately connected networks into a single, large network through an **internetworking protocol** was put forward: The internet was born.

OSI and TCP models of the internet

The internet today has evolved works through not just one but through many internetworking protocols layered on top of one another. There are two major models of these interlayered protocols taught in universities, **The OSI Model** and **The TCP Model**.



Now, every layer in that model has its own purpose in the sending and receiving of data across the internet, and every layer has its own set of protocols and rules for doing so. Now, of the two models, The OSI model is mostly an academic model, used to teach students about networking, while the TCP model is the one actually used in the real world, the one that reflects what businesses are using as I write this very sentence.

As such, the TCP model, short for the Transmission Control Protocol model, is the one that this section will elaborate on and explain, but the reader may feel free to look into the OSI model if they find the topic to be interesting. The four layers of the TCP model are as follows:

Layer 1, Network Interface: The network interface layer, also known as the data link layer, handles the physical infrastructure that lets computers communicate with one another over the internet. This covers ethernet cables, wireless networks, network interface cards, and so on. The network interface layer also includes the technical infrastructure, such as the code that converts digital data into transmittable signals, that makes network connection possible. Some major layer 1 protocols are: **Ethernet**, **Wifi** and **ARP**.

Layer 2, Internet: The internet layer, also known as the network layer, controls the flow and routing of traffic to ensure data is sent speedily and accurately. This layer is also responsible for reassembling the data packet at its destination. If there's lots of internet traffic, the internet layer may take a little longer to send a file, but there will be a smaller chance of an error corrupting that file. **Internet Protocol (IP)** is for example, a layer 2 protocol.

Layer 3, Transport Layer: The transport layer provides a reliable data connection between two communicating devices. It's like sending an insured package: the transport layer divides the data in packets, acknowledges the packets it has received from the sender, and ensures that the recipient acknowledges the packets it receives. **TCP** and **UDP** are two common layer 3 protocols.

Layer 4, Application Layer: The application layer is the group of applications that let the user access the network. For most of us, that means email, messaging apps, and cloud storage programs. This is what the end-user sees and interacts with when sending and receiving data. **HTTPS** and **FTP** are two very common layer 4 protocols.

IP Addresses and DNS

Now, IP addresses are like the ID assigned to every computer that is part of the internet, it has two types, IPv4 and IPv6 and looks something like this: `192.168.1.1` (IPv4 addresses at least). IP addresses also help us locate a device geographically when we are either sending or receiving packets to them, and every website, API or service on the internet that the reader has ever used is in part powered by them. These numerical addresses however, are quite hard to remember and asking people to remember them every time they wish to visit a website might prove difficult, enter DNS.

A **Domain Name** is a unique name that identifies a website on the internet, imagine nytimes.com or google.com. We map these domain names to IP Addresses like the ones above, allowing us to navigate to the devices possessing the address without having to remember their complex numerical designation.

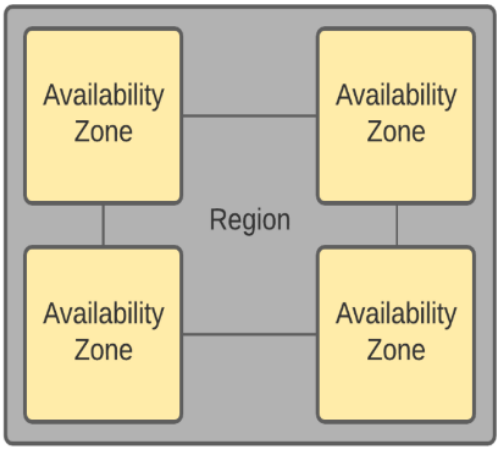
DNS (Domain Name System) then is like the phonebook of the internet, its a giant registry of domain names and the IP addresses that they point to. Whenever we go to a webpage, say Amazon.com, our browser consults the DNS to find out which IP Address it must navigate and send requests to.

AWS Global Infrastructure

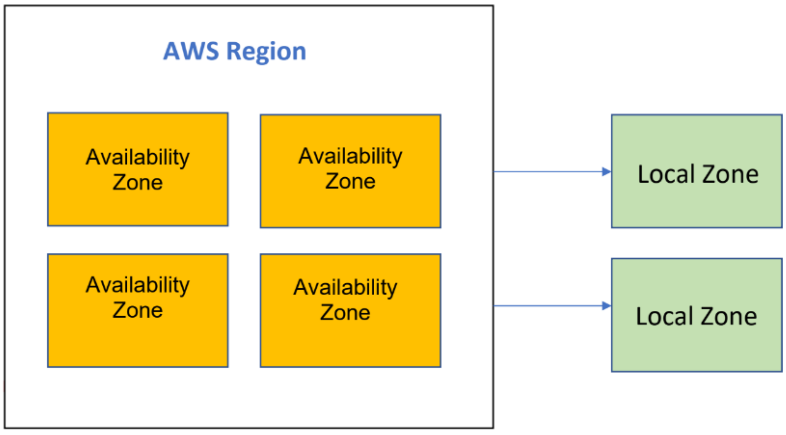
We've mentioned before how AWS operates infrastructure in over 245 countries, with presence in every one of the seven continents. This infrastructure is organized into **Regions**, **Availability Zones**, **Local Zones** and **Edge Locations**.

Regions in AWS represent data centers clustered around a large geographic area i.e. a continent or a part of a continent. Example, us-east-1 represents a group of

data centers clustered around the eastern coast of the United States. An Availability Zone (AZ) on the other hand, is simply one of the data centers within it. This is succinctly illustrated in the diagram below:

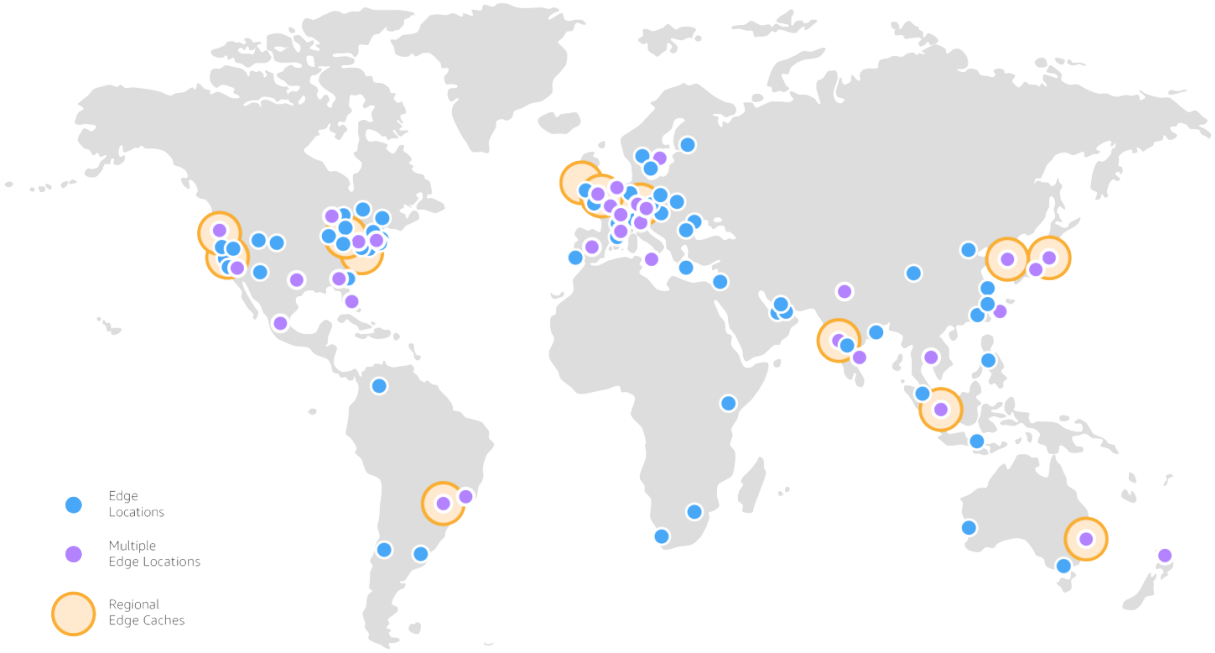


There may however, be times where a data center is needed outside the boundaries of these regions, due to a large cluster of users being concentrated in areas at a significant distance from the region, which is usually the case for large cities. In order to address such situations, AWS created the Local Zone. The local zone is effectively, a data center closer to the user. Organizations place systems that require low latency in these local zones. A diagram of a local zone is given below:



Finally, Edge Locations are the small pieces of infrastructure used to power CloudFront, Amazon's Content Delivery Network. They mostly serve the purpose of improving internet performance and website security. CDNs and CloudFront are both discussed liberally throughout the chapter.

At the time of writing, AWS infrastructure spans across 34 Geographical regions, 108 Availability Zones, 34 Local Zones and 600+ Edge locations. A map of the regions and edge locations is given below:



Caching and CDN

Caching is nothing but the process of storing frequently accessed data in a location that can be retrieved quickly, minimizing the time and resources it takes to serve that data to a user. Its like putting the car keys in a drawer right by the door.

We use the keys frequently, and it saves time to have them in a convenient, easy-to-reach spot rather than searching the entire house for them each time we need to go out. In web terms, this means data that users often request (like images,

website code, or video content) is temporarily stored closer to them, resulting in less loading times.

Now, if we were to create a full-blown distributed network of these caches, and connect them both to AZs and to each other, what we would get is a **Content Delivery Network (CDN)**, a system designed to speed up the delivery of critical and/or frequently requested data to the user.

Application Programming Interfaces (APIs)

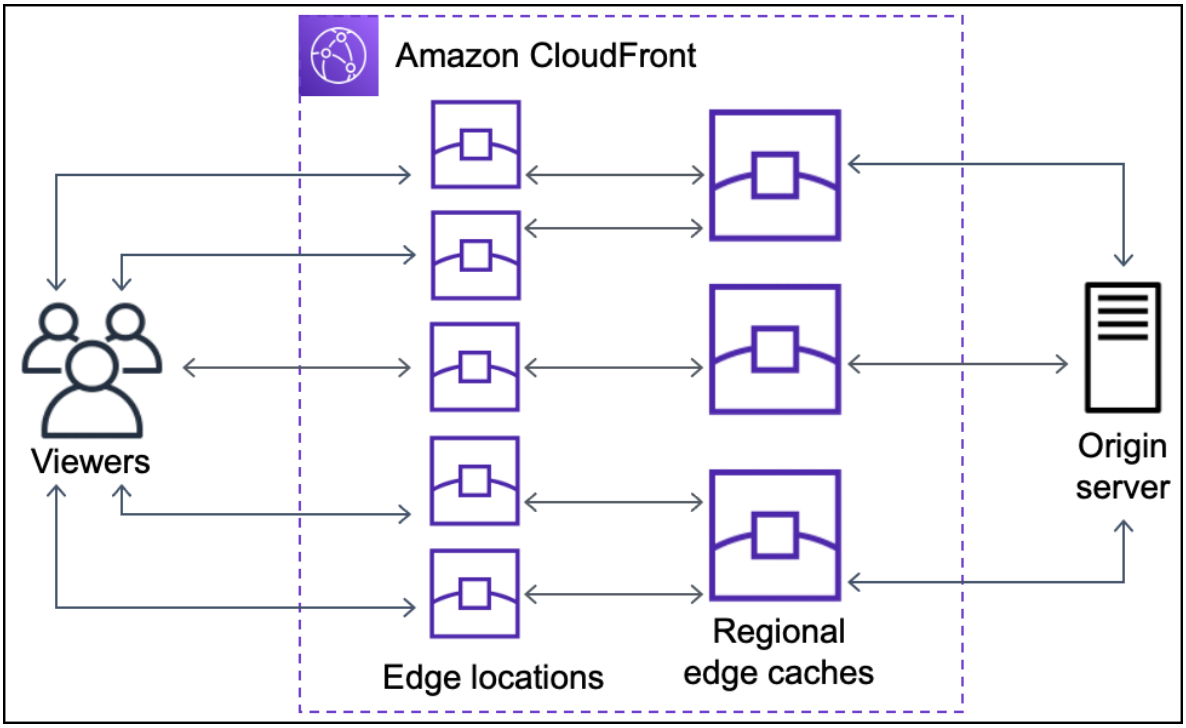
An unplanned addition to this section, APIs (Application Programming Interfaces) are sets of rules that allow different software systems to communicate. They are genuinely the cornerstone of many internet applications, and dedicating a paragraph to the concept in this section was thus believed to be necessary.

APIs, in simple words, enable one system to request or send data to another without needing to know how the other system works internally. They handle the creation of a standard channel for communication between two systems on our behalf, allowing two systems to talk with each other in extremely convenient ways.

AWS has its own set of APIs, with the python Boto3 API in particular being a popular tool in many cloud engineer’s arsenals. Also many of the individual services like S3 and EC2 also have their own APIs, allowing developers to more easily automate tasks and build integrations.

CloudFront

AWS CloudFront is a fast and highly secure content delivery network (CDN) service that accelerates the delivery of your websites, APIs, video content, and other web assets. It improves the performance of your applications by caching content at edge locations around the world, reducing latency and providing a better user experience.



Architecture diagram for inner working of CloudFront

CloudFront improves performance and reduces latency for both static content stored in a S3 bucket as well as dynamic content stored behind say, multiple Application Load balancers and EC2 instances. (i.e. Cloudfront distributions can have multiple services as their origins.) Also worth noting is that all Cloudfront distributions offer DDoS protection by default due to them having AWS Shield Advanced installed on them out-of-the-box.

Global Accelerator

Applications hosted using AWS often have to serve information to users and clients quite far from their geographical data center locations, resulting often in considerable latency and less-than-desirable data transfer speeds.

AWS however, as mentioned before, has an entire global network of infrastructure at its disposal. AWS Global Accelerator is a service designed to take advantage of said infrastructure in order to reduce latency and improve data transfer speeds. It also provides the applications using it with features such as automatic failover, helping it ensuring higher availability and reliability alongside faster loading and operating times.

This makes it an excellent fit for non-HTTP use cases such as gaming (UDP), IoT, and Voice over IP, where low latency and fast data transfer are critical. In fact, there are many use cases where the utilization of Global Accelerator is basically a pre-requisite such as the updating of financial boards and various IoT applications.

That does not however mean that the service is only meant for non-HTTP applications, and can enhance the connections for HTTP applications by providing a more consistent and reliable experience as well.

Cross Origin Resource Sharing

A major source of headache for engineers around the world, Cross-Origin Resource Sharing (or CORS for short) is a security feature in web browsers that controls how web pages can request resources from a different domain. It ensures safe interactions between web pages and resources across different origins, protecting against **cross-site scripting (XSS) attacks**.

As a general rule of thumb, enabling CORS is basically essential for public-facing services like Amazon S3 and API Gateway. For S3 in particular, it allows web applications to directly request and interact with S3 buckets from different domains, useful for functionalities like file uploads. While for API Gateway, CORS settings enable APIs to be accessed from web applications on different domains. This integration allows client-side applications to connect to serverless backend services securely and efficiently.

Also worth noting is that Amazon CloudFront also leverages CORS through its `Access-Control-Allow-Origin` header. This header specifies which origins are permitted to access resources on a CloudFront distribution. By configuring the `Access-Control-Allow-Origin` header, we can control which domains can make cross-origin requests to your CloudFront-distributed content, ensuring secure and efficient delivery of static and dynamic content across different domains.

Direct Connect

AWS Direct Connect, sometimes also referred to as simply DX, is a network service that establishes a dedicated, private, wired connection between on-premises infrastructure and AWS services, in fact AWS claims it to be the shortest path to the AWS resources we rent from them, providing us with a high-speed, low-latency connection that helps improve performance and allow us to receive a more consistent network experience with our cloud infrastructure compared to standard internet connections.

Direct Connect is ideal for applications requiring stable and high-throughput data transfer, such as large-scale data migrations, hybrid cloud architectures, and real-time data processing. Since the service allows us to connect to AWS resources through a wired connection, it essentially bypasses the public internet and therefore essentially eliminates the security threats and interruption risks associated with web-based traffic further enhancing the reliability and connectivity to AWS resources compared to standard connections.

Finally, it is also worth noting that because DX is based on dedicated private connections, it does not compete with the organization’s bandwidth allowing us to negate any internet-speed restrictions for example.

Subnets

As discussed before, VPCs (Virtual Private Clouds), i.e. the virtual boxes into which we place our AWS resources are an integral component of any cloud environment/architecture that is even remotely complex.

Well, to take the box analogy one step further, when placing a vast array of things into boxes, is it not so much easier and neater if we were to group all the similar items together? Such as putting all the toys in one compartment and all the books in another. Makes things a whole lot easier right? Subnets in AWS are similar to those compartments except for VPCs. They allow us to logically separate different types of resources within your VPC.

So one could for example, have a VPC containing both the web server EC2 instances as well as say, RDS database instances. Subnets allow us to segment portions of a VPC allowing us to logically isolate our EC2 instances from the RDS database instances. They allow us to allocate IP addresses to resources (like EC2 instances) and enforce network access controls through Network Access Control Lists (ACLs) and Security Groups (Will be discussed in later chapters) therefore playing a crucial role in organizing and securing your AWS resources within a VPC environment.

Let us illustrate the logical segmentation performed by the VPC using the aforementioned scenario as an example, utilizing CIDR blocks:

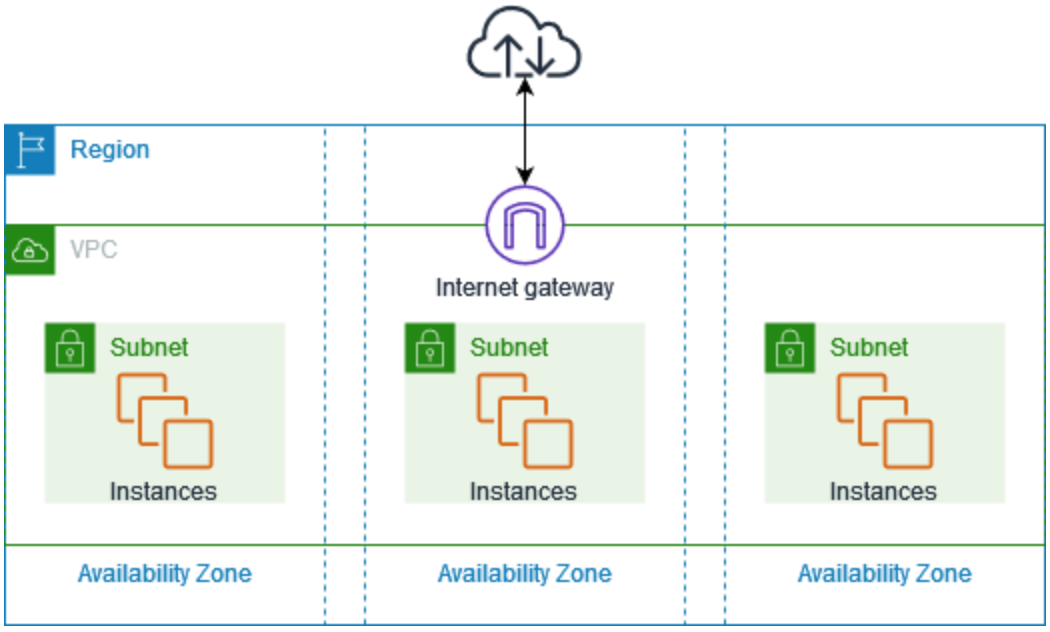
Imagine the hypothetical VPC CIDR block to be of the range: `0.0.0.0/16`

- This gives us a total of 65,536 IP addresses (from `10.0.0.0` to `10.0.255.255`)

Then, we can design the Subnet to be as follows:

- **Public Subnet for Web Servers**
  - **CIDR Block**: `10.0.1.0/24`
  - **IP Address Range**: `10.0.1.1` to `10.0.1.254`
  - **Purpose**: This subnet will host the EC2 instances that run the web servers, allowing them to be accessible from the internet.
- **Private Subnet for RDS Database Instances**
  - **CIDR Block**: `10.0.2.0/24`
  - **IP Address Range**: `10.0.2.1` to `10.0.2.254`
  - **Purpose**: This subnet will host the RDS instances. It is private, meaning the RDS instances are not directly accessible from the internet.

Note however that subnets are afflicted by a restriction which are not applicable to VPCs, as each subnet created by us is mapped to a single Availability Zone while VPCs have the capability to span across multiple availability zones.



Architecture of a VPC with 3 Subnets (Source: AWS)

Block Size of a VPC

In the context of VPCs, block size refers to the range of IP addresses defined by CIDR notation that can be allocated for subnets i.e. the range of addresses that can be used to define subnets. AWS VPC allows different block sizes, typically ranging from `/16` (65,536 IP addresses) to `/28` (16 IP addresses). The 10.0.0.0/16 CIDR block notation for example, theoretically gives us access to 65,536 addresses (from 10.0.0.0 to 10.0.255.255). However, not all of these are usable as AWS has something called reserved addresses, which is used for the operation of the subnet and are not made available to us. These IP Addresses, and their uses are as follows:

- **.0**: Network address, used to identify the network device
- **.1**: Reserved for the VPC router
- **.2**: Reserved for DNS
- **.3**: Reserved for future use
- **.255**: Broadcast address, the last address within the IP address range, it is used to communicate with other broadcasting devices

Thus, while a `/16` subnet may offer 65,536 addresses, only 65,531 of them are actually usable. Here are some examples of usable IP addresses within the 10.0.0.0/16 CIDR block:

- **First Usable Address**: **10.0.0.1**
- **10.0.1.1**
- **10.0.128.5**
- **Last Usable Address**: **10.0.255.254**

Though we have been using the `/16` subnet so far, it is obviously not the only CIDR block size available, a list of the most commonly used CIDR Block sizes and their number of IP Addresses:

CIDR Notation	Total IP Addresses	Usable IP Addresses	Suitable For
/8	16,777,216	16,777,214	Very Large Applications
/16	65,536	65,534	Large Applications
/20	4,096	4,094	Medium to Large Applications
/24	256	254	Small to Medium Applications
/25	128	126	Small to Medium Applications
/26	64	62	Small to Medium Applications
/27	32	30	Small Applications
/28	16	14	Very Small Applications
/29	8	6	Very Small Applications
/30	4	2	Smallest Applications

Note that block sizes with a larger amount of available IP addresses have a premium associated with them and as such choosing the right block size is as much an exercise in cost minimization as it is in IP Address or device management.

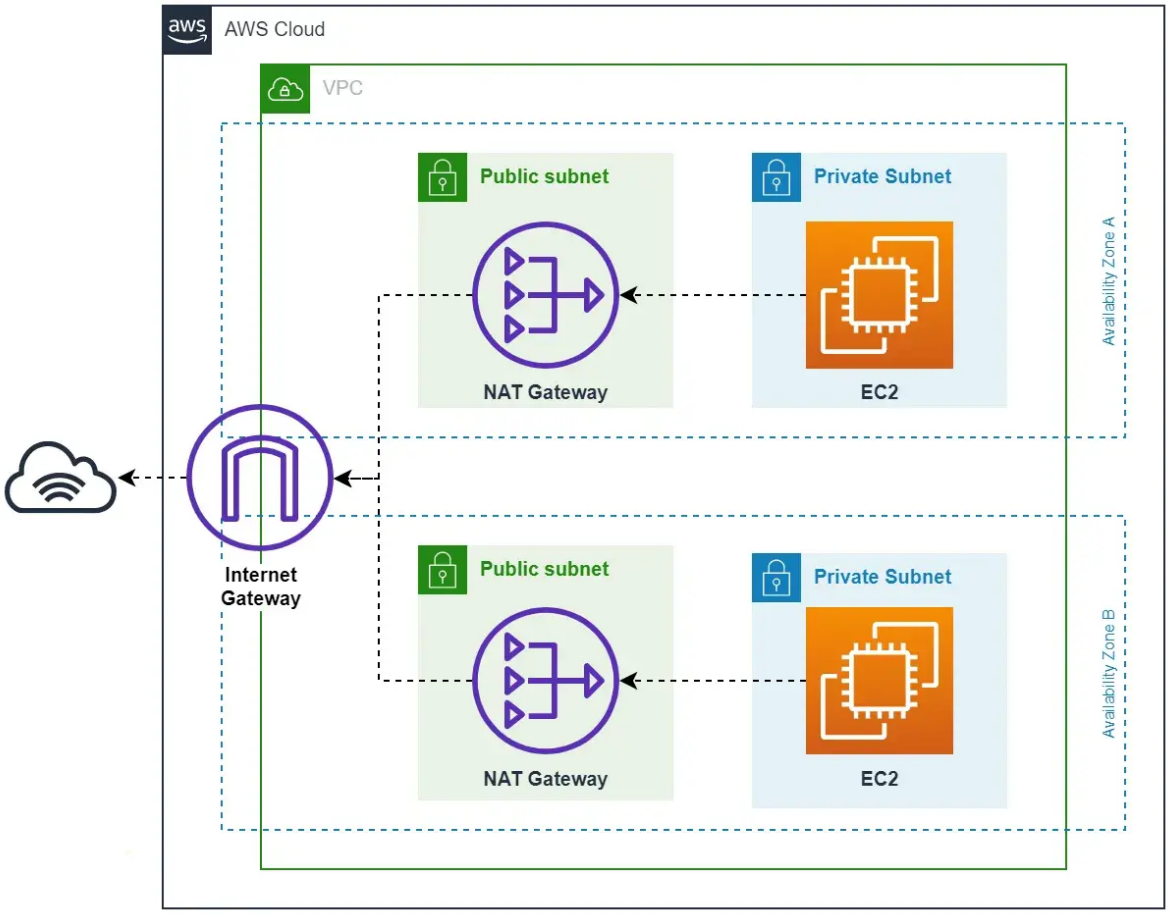
NAT and Internet Gateways

Network Address Translation or NAT for short, allows instances in a private subnet to access the internet while preventing the internet from initiating connections to those instances. This is crucial for maintaining security, as it helps shield private resources from external threats. In modern AWS environments Network Address Translation (NAT) services are provided by something called NAT Gateways. NAT gateways are used so that our AWS resources such as EC2 or RDS instances that are occupying a private subnet can connect to external services i.e. services outside the VPC, but said external services cannot initiate a connection with the instances inside the private subnet, allowing our web servers and databases to scour the internet or fetch software updates for example, without the risk of malicious actors accessing them.

When we create a NAT gateway, we must specify one of the following connectivity types for it:

- **Public** – The default connectivity type, instances in private subnets can connect to the internet through a public NAT gateway, but cannot receive unsolicited inbound connections from the internet. You create a public NAT gateway in a public subnet and must associate an elastic IP address with the NAT gateway at creation. You route traffic from the NAT gateway to the internet gateway for the VPC. Alternatively, you can use a public NAT gateway to connect to other VPCs or your on-premises network. In this case, you route traffic from the NAT gateway through a transit gateway or a virtual private gateway.
- **Private** – Instances in private subnets can connect to other VPCs or your on-premises network through a private NAT gateway. You can route traffic from the NAT gateway through a transit gateway or a virtual private gateway. You cannot associate an elastic IP address with a private NAT gateway.

Internet gateways on the other hand, are a more general purpose VPC component, allowing the resources within a VPC to have full access to the internet, and scour it in a manner which is more similar to how we browse it ourselves, allowing both inbound and outbound connections to be established. Internet Gateway therefore serves as both a bridge for outbound traffic from your VPC to the internet whilst providing a path for inbound traffic as well.



Architecture showing NAT Gateways connected to an Internet Gateway

Note in the above diagram how the NAT Gateways are visualized as an additional layer between the EC2 instances and Internet gateway, this is because the presence of an internet gateway is necessary for the functioning of a NAT Gateway though the opposite is never true.

NAT Gateway vs NAT instance

NAT Gateways and NAT instances both serve the same purpose of providing Network Address Translation capabilities within an AWS ecosystem. However, they possess characteristics that make them quite different from each other. These characteristics and how they make both offerings suitable for slightly different use cases is explored below:

A NAT Instance is an EC2 instance configured to perform NAT functionality. The burden of installing and managing said NAT software, its scaling, and perform maintenance on it however falls entirely upon the cloud administrator.

A NAT Gateway on the other hand, is a fully managed service provided by AWS, the cloud administrator therefore does not need to manage the underlying infrastructure with AWS handling the scaling and maintenance of the NAT infrastructure.

NAT Gateways are a newer offering compared to NAT instances, and treated by AWS as successor to NAT instances, a much older services. NAT gateways are, as such, kind of configured for modern environments out-of-the-box, something not true for NAT instances which make them a sort-of legacy offering by default.

Therefore, it is worth noting that when dealing with NAT, AWS encourages the use of NAT Gateways in almost all possible cases, especially for newer projects. In fact, AWS suggests that if feasible, cloud administrators managing legacy projects replace all older NAT instances with NAT gateways and might be planning on making NAT instances defunct soon (though it has not done it so far, at least at the time of writing).

A more detailed comparison of the two is provided below:

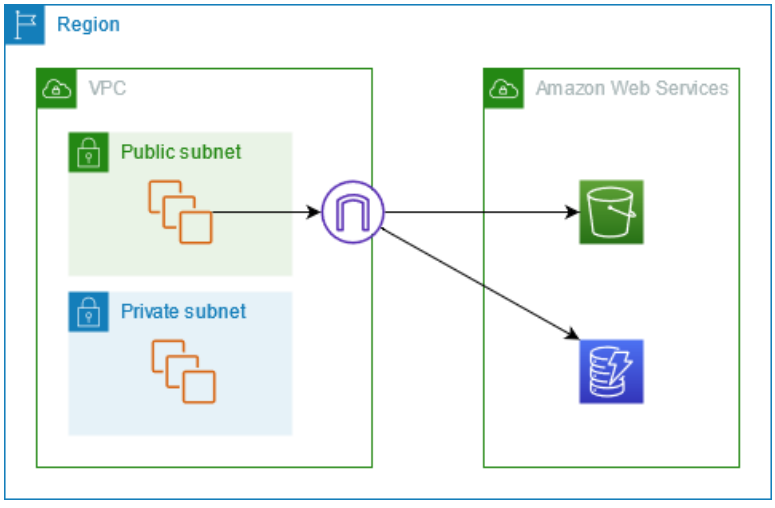
Feature	NAT Instance	NAT Gateway
Management	User-managed (you control the EC2 instance)	AWS-managed (fully managed service)
Cost	Generally cheaper (pay for EC2 instance)	More expensive (pay per hour + data processed)
Scalability	Manual (need to manage scaling)	Automatic (scales based on usage)
Availability	You need to set up redundancy and failover	Highly available by default
Performance	Limited by instance type and size	Generally better, with higher bandwidth
Maintenance	User responsible for updates and patches	No maintenance required
Use Cases	Custom configurations or special requirements	Standard use cases with no special requirements

In summary, if you're looking for ease of management, scalability, and reliability, a NAT Gateway is the way to go. If you have specific customization needs or are operating on a tight budget, a NAT Instance might be more suitable.

Gateway VPC Endpoint

Now, so far we have discussed the use of internet gateways and NAT instances, however these require us to access AWS services being hosted in the same data centers where the NAT infrastructure is through a connection external to AWS itself. Setting up and maintaining the networks using these connections therefore may be victims of needlessly redundant network hopping and operating costs.

Gateway VPC Endpoints aim to alleviate this problem somewhat by allowing us to access the AWS services of **S3** and **DynamoDB** directly from our VPC without connectivity to the internet, no NAT services required. They have become quite popular among cloud architects in recent years, probably also because of the fact that Gateway VPC Endpoints are **free** and have no additional charge associated with them, no matter the amount of inbound and outbound traffic passing through them.



Architecture diagram showcasing Gateway VPC Endpoints

VPC endpoints and how they work will be discussed in future sections.

For now, remember that VPC Gateway Endpoints differ from normal VPC Endpoints because they do not use AWS PrivateLink, unlike the other type of VPC endpoints.

Elastic Load Balancers

Now, when making complex architectures we may often find ourselves operating multiple versions of a resource to meet various scaling and operational needs. When one EC2 instance is not sufficient to satisfy all the requests for example, we often spin up and add more EC2 instances to the VPC using Auto-Scaling Groups as discussed before.

Now, in the above scenario, how do we ensure that all the EC2 instances are having the network traffic distributed properly among them, and that certain instances are being overburdened with work while others lay relatively dormant? The answer is **Load Balancers**.

Load balancers are systems that distribute network or application traffic across multiple servers, ensuring that no single server is overwhelmed with too much traffic, which can lead to performance degradation or outages.

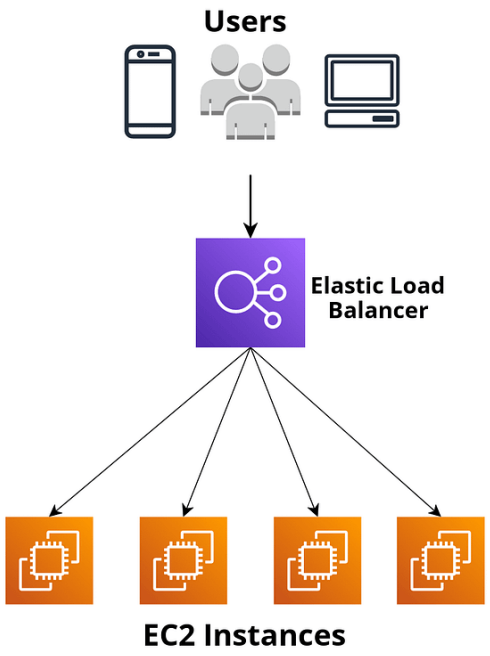
Load Balancers serve several critical purposes:

- **Traffic Distribution**: They balance incoming requests to prevent any single server from becoming a bottleneck.
- **High Availability**: Load balancers enhance the availability of applications by routing traffic to healthy instances. If one server goes down, the load balancer reroutes requests to other operational servers.
- **Scalability**: As demand increases, load balancers make it easier to add more servers, ensuring that applications can handle varying traffic loads.
- **Performance Optimization**: They can help optimize response times by directing requests to the most capable or least loaded servers.

In summary, load balancers allow us to combat **Single points of failure**, **Traffic spikes** and **Resource management**.

Elastic Load balancers (ELB) is the package provided to us by AWS that allows us to implement load balancers, distributing incoming application traffic across multiple AWS resources, such as EC2 instances, containers and IP addresses, enhancing the availability and fault tolerance of your applications by balancing the load and ensuring traffic is directed to healthy instances as mentioned above.

Though there are different types of Elastic Load Balancers (will be discussed in the next section), they all occupy similar positions in AWS network architecture, usually serving as a middleman between user agents and a swathe of compute or database instances as shown below:



Simple architecture diagram showcasing an Elastic Load Balancer

Finally, it is worth noting that Elastic Load Balancers, just like subnets within a VPC, can only be connected to Availability Zones in a single region.

Types of Elastic Load Balancers

As mentioned in the last section, there are different types of Elastic Load Balancers, each of which have different characteristics and serve different use cases. These are as follows:

Application Load Balancer (ALB):

Application Load Balancers operate at the application layer, the seventh layer of the OSI model. This makes it particularly suited for dealing with HTTP and HTTPS traffic, two prominent application layer protocols. Its primary strength lies in advanced routing capabilities, which include host-based and path-based routing. This functionality allows ALBs to intelligently direct requests to different backend services based on the content of the request.

Additionally, they provide features like sticky sessions and request tracing, enhancing the user experience by maintaining session continuity which combined with their intelligent routing capabilities have led to the widespread adoption of ALBs in dynamic web applications, RESTful APIs, microservice architectures and containerized applications; all systems and services that require and/or benefit from having fine-grained routing logic.

Network Load Balancer (NLB):

In contrast, Network Load Balancers operate at the transport layer, the fourth layer of the OSI model and is optimized for handling traffic utilizing the TCP and UDP protocols. NLBs are designed to handle millions of requests per second while maintaining ultra-low latency, making them an excellent choice for applications that demand high performance, such as gaming servers, VoIP applications, and real-time data processing.

NLBs can also operate in scenarios where static IP addresses are required, providing a more stable endpoint for applications. Furthermore, NLBs also increase a systems capacity to handling sudden spikes in traffic, and are thus often utilized in high-throughput environments.

Gateway Load Balancer (GLB):

The Gateway Load Balancer is a newer addition to AWS's suite of load balancers, operating primarily at the third layer of the OSI model. Its primary purpose is to facilitate the deployment and scaling of third-party virtual appliances such as firewalls, intrusion detection systems, and other network security devices.

A more structured, tabular comparison of the three different types of load balancers is given below:

Feature	**Application Load Balancer (ALB)**	**Network Load Balancer (NLB)**	**Gateway Load Balancer (GLB)**
OSI Layer	Layer 7 (Application Layer)	Layer 4 (Transport Layer)	Layer 3 (Network Layer)
Protocol Support	HTTP, HTTPS, WebSocket, HTTP/2	TCP, UDP	Primarily IP traffic (integrates with other services)
Routing Capability	Advanced routing (host-based, path-based)	Simple routing based on IP and port	Distributes traffic to virtual appliances
Performance	Optimized for complex application traffic	High performance, low latency, millions of requests	Scalable for virtual appliances, but latency depends on appliances
Use Case	Microservices, dynamic web applications	Gaming, VoIP, real-time data processing	Security appliance integration, network monitoring
Health Checks	Application-level health checks	Transport-level health checks	Monitors appliance health, not application health
Sticky Sessions	Supports sticky sessions	Does not support sticky sessions	Not applicable

Health Checks

We mentioned Health Checks in the last section, but what are they exactly and what purpose do they serve? Well that is what we will discuss in this section.

Health Checks as the name suggests, are processes that monitor the status and performance of our AWS resources and components to determine whether or not they are functioning correctly, i.e. whether or not they are “Healthy”. As mentioned in the last section, they are an integral part of load balancers and are often used to assess whether backend assets (such as EC2 instances) are capable of handling traffic. They are quite widely used and help us better our systems by providing:

- **High Availability**: Health checks help maintain high availability by ensuring that traffic is only routed to instances that are operational. If an instance fails a health check, it is marked as unhealthy, and the load balancer stops sending traffic to it.
- **Fault Tolerance**: By identifying unhealthy instances quickly, health checks facilitate swift recovery and failover, allowing the system to redirect traffic to healthy instances and minimize downtime.
- **Performance Monitoring**: Health checks can provide insights into the performance of application components, allowing teams to detect and address issues proactively.

Of the three types of load balancers, Gateway Load Balancers do not support health checks leaving us with the other two, namely: Application Load Balancers and Network Load Balancers.

Application Load Balancers support HTTP health checks and are the best at it, allowing them to check specific API URLs and endpoints, directing the requests only to the paths that it has determined to be capable of performing work successfully.

Network Load Balancers on the other hand, primarily handle health checks related to TCP/UDP traffic, and are especially useful for operations requiring low latency and high throughput, such as real-time gaming or IoT applications.

Note however that Network Load Balancers are also capable of performing HTTP health checks, though NLB health checks are simpler in nature, typically only checking if a specific port is open for connection, and not performing anything beyond that making them extremely suitable for simple services that do not require complex health verification.

Finally, an often overlooked aspect of health checks in AWS are their integrations with other features such as Auto Scaling. Not many know this but we can actually use AWS Auto Scaling to automatically add or remove instances based on health check status.

For example, if an instance is marked as unhealthy, Auto Scaling can launch a new instance to replace it, automating maintenance and ensuring continuous availability. This is usually done by utilizing a sub-feature of Auto Scaling in AWS called Auto Scaling actions. I shall spare the reader with a more in-depth explanation of the sub-feature as I judge it to be beyond the scope of both this section and this book in general.

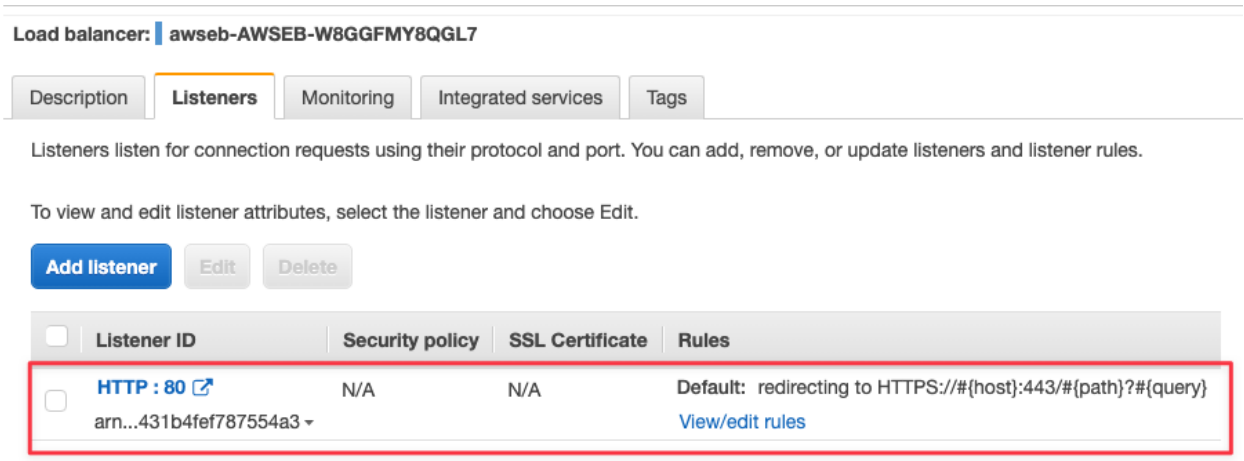
Listener Rules and Redirecting

As mentioned in the previous sections, Application Load Balancers (ALBs) have the ability to perform complex routing decisions, directing incoming traffic to appropriate backend services. Well, ALBs are able to make said decisions through the use of **Listeners** and **Listener Rules**.

Listeners are the processes that check for connection requests using the port and protocol we specify. Said listeners have a set of **rules**, called listener rules defined within them that determine how the load balancer should route requests to its registered **targets**. Both Listener Rules and their targets have to be defined by the cloud administrators when an Application Load Balancer (ALB) is being being created.

Listen Rules and ALBs allow us to perform a great many functions in conjunction beyond just simple traffic routing. One such function is the ability of ALBs to redirect HTTP requests to HTTPS, ensuring that all traffic is transmitted with the extra layer of security provided by HTTPS. Another such function is request manipulation, that is, the ability of Listener Rules to be able to modify the contents of a request before they reach backend services by, for example, adding headers to them. Request manipulation can prove especially handy when dealing with, say, CORS restrictions (a major source of headache for many people around the world).

An example of a Listener that checks for connections requests using HTTP on port 80, and redirects said request to HTTPS is shown below:



A listener rule for redirecting from HTTP to HTTPs (Source: Vanta)

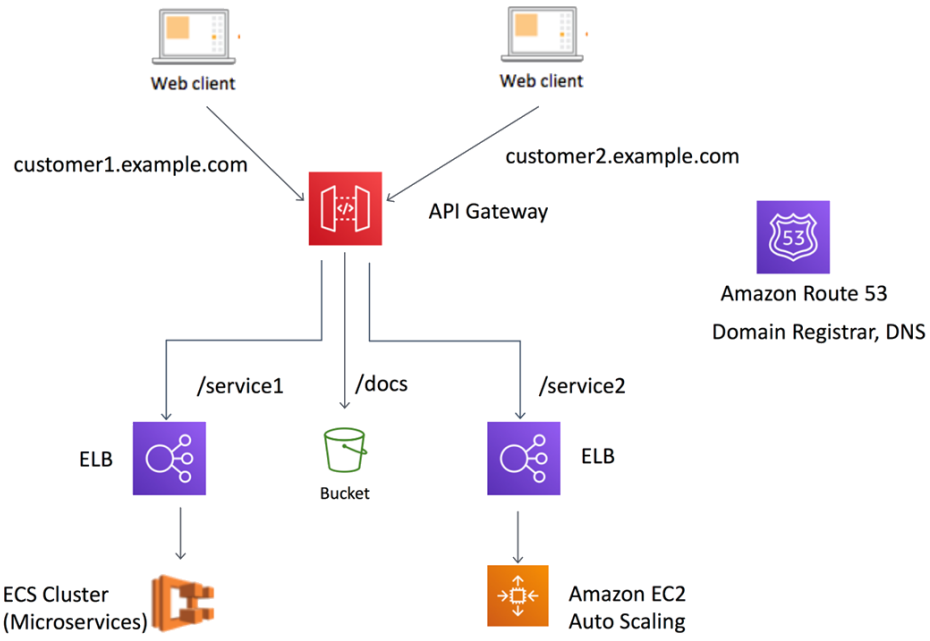
TLDR;

Listener Rules are the conditions set on an ALB that determines how requests should be handled. Used to redirect HTTP to HTTPS; Can also add headers to requests.

API Gateway

APIs are a widespread part of the modern technical world, serving as the middlemen powering communications between many software applications using the internet. In order to allow organizations to more easily create, publish, maintain and monitor APIs at scale, Amazon has a fully-managed service called API Gateway. It handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, authorization and access control, throttling, monitoring, and API version management thereby reducing a great deal of the hassle associated with APIs.

An architecture diagram of an organization that uses an API Gateway as an entry point to multiple software applications is shown below:



Source: AWS

Also worth mentioning is that the service has no minimum fees or startup costs and that we only have to pay for the amount of times the API was called and the amount of data transferred out. In fact, the mentioned advantages of API Gateway have led to a great deal of organizations moving their legacy API workloads to Amazon API Gateway, which often operate on the company’s domain name and corresponding certificate.

(Optional: Only necessary reading for AWS administrators)

To design the API Gateway URL with the company's domain name and corresponding certificate, the company needs to do the following:

1. Create a Regional API Gateway endpoint: This will allow the company to create an endpoint that is specific to a region.
2. Associate the API Gateway endpoint with the company's domain name: This will allow the company to use its own domain name for the API Gateway URL.
3. Import the public certificate associated with the company's domain name into AWS Certificate Manager (ACM) in the same Region: This will allow the company to use HTTPS for secure communication with its APIs.
4. Attach the certificate to the API Gateway endpoint: This will allow the company to use the certificate for securing the API Gateway URL.
5. Configure Route 53 to route traffic to the API Gateway endpoint: This will allow the company to use Route 53 to route traffic to the API Gateway URL using the company's

Route53 and Failover

Amazon Route 53 is a scalable Domain Name System (DNS) web service that translates domain names into IP addresses and routes users to endpoints, turning a user-friendly URL like `www.example.com` into a numeric IP address like `192.0.2.1`, usually representing an EC2 instance, a load balancer, or another type of endpoint or AWS resource.

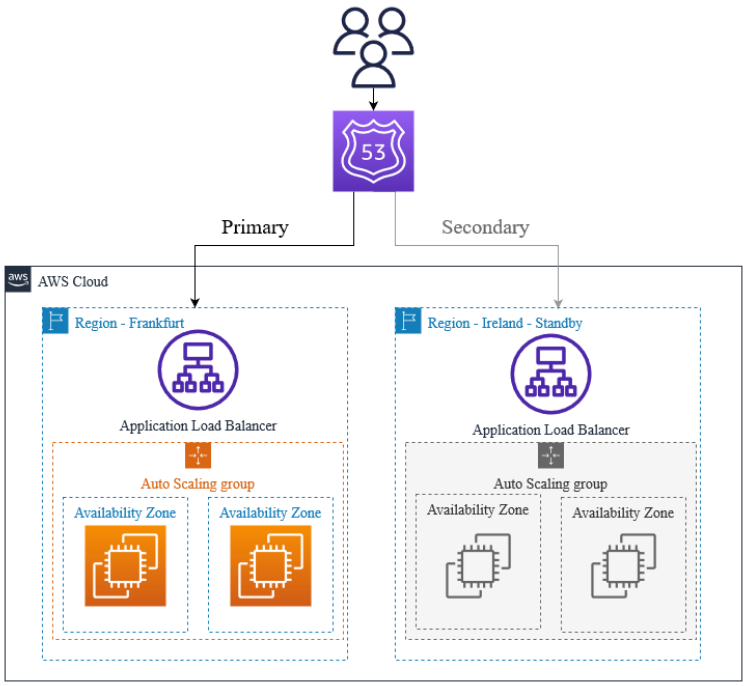
The service also offers a myriad of routing policies, capable of directing traffic to the necessary IP address based on specific criteria and conditions. A brief summary of the different routing policies is given below:

- **Simple Routing**: This is the most straightforward option, where Route 53 returns a single resource record set for a domain name. It's useful when you want to direct traffic to a single endpoint.
- **Failover Routing**: Failover routing allows us to configure primary and secondary endpoints and switch between them. If the primary endpoint becomes unhealthy (as determined by health checks), Route 53 automatically routes traffic to the secondary endpoint, ensuring minimal downtime.
- **Latency-Based Routing**: This policy routes traffic to the endpoint that provides the lowest latency for the user, enhancing performance by directing users to the nearest or fastest server.
- **Geolocation Routing**: This option lets you route traffic based on the geographic location of the user. You can specify different endpoints for users in different regions, which can be helpful for compliance or to provide localized content.
- **Geoproximity Routing**: Similar to geolocation routing, this policy allows you to route traffic based on the user's location and the location of your resources. You can also specify bias to favor certain endpoints.

- **Weighted Routing**: This policy allows you to distribute traffic across multiple endpoints based on assigned weights. For example, you might route 70% of traffic to one endpoint and 30% to another, useful for gradual rollouts or A/B testing.
- **Multi-Value Answer Routing**: This option allows Route 53 to return multiple IP addresses in response to a single DNS query. Clients can then choose which address to connect to, which can enhance redundancy and load balancing.

Now, while a more in-depth summary of all the different routing policies is beyond the scope of this book, we will be taking a closer look at Failover Routing in this section, with Geoproximity and Geolocation routing being the subjects of the next section.

Failover routing, simply put, allows us to direct traffic to one particular path when a resource (designated as the primary resource) is healthy and to direct it through another path when it is not. The resource that Amazon Route53 redirects to when the primary resource is unavailable is also called the secondary resource.



Failover Routing with Primary and Secondary Resource **(Source: StormIT)**

Failover routing furthermore, has two different configuration modes, each with its own modus operandi: **Active-Active Failover** and **Active-Passive Failover**

Active-Active failover in Route 53 allows traffic to be distributed across multiple endpoints, all of which are capable of handling requests simultaneously. This setup improves availability and responsiveness by distributing traffic among healthy endpoints.

Active-Passive failover, on the other hand, uses a standby endpoint that only becomes active when the primary endpoint fails. This configuration ensures minimal downtime by automatically redirecting traffic to the standby endpoint when the primary endpoint becomes unavailable.

Therefore, we use the Active-Active failover configuration when we want all of your resources to be available the majority of the time while the Active-Passive failover configuration is to be used when we wish that the secondary resource serve us mostly in a reserve capacity, being utilized only in cases where they system is facing significant disturbances.

TLDR;

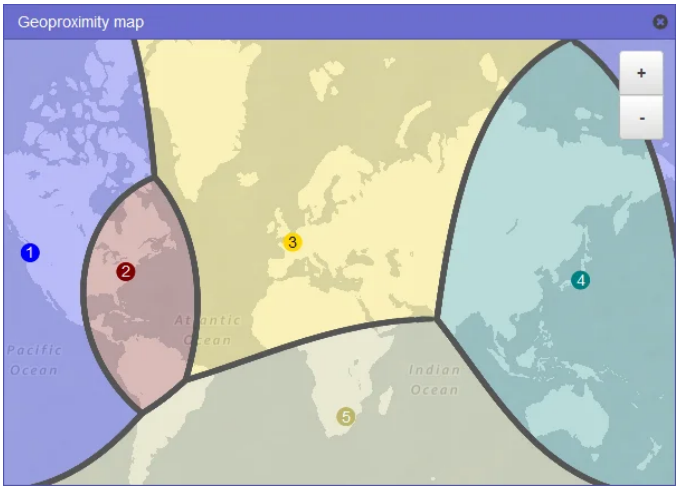
Route 53 is a DNS web service with support for failover policies that enable robust fault tolerance and high availability for applications, helping ensure a seamless user experience even during moments of endpoint failure.

Geoproximity vs Geolocation Routing

Both Geoproximity and Geolocation routing policies allow us to direct traffic based on geographical presence, but they differ in many important ways. However, a more deep dive into both routing policies is necessary to understand those differences.

Geoproximity Routing Policy

A Geoproximity routing policy is used to route traffic based on the location of our organization's resources, and shift the flow of traffic using a geographical paradigm more suitable to us.This policy allows us to direct users to different servers, even if those servers might be further away, using something called a *bias*.



Geoproximity map dividing the world based on four AWS Regions (numbered 1 through 4) and a location in Johannesburg, South Africa that is specified by latitude and longitude (5).

A bias allows us to impose and assign a certain set of resources as the de-facto traffic route for requests from a well-defined geographical boundary.

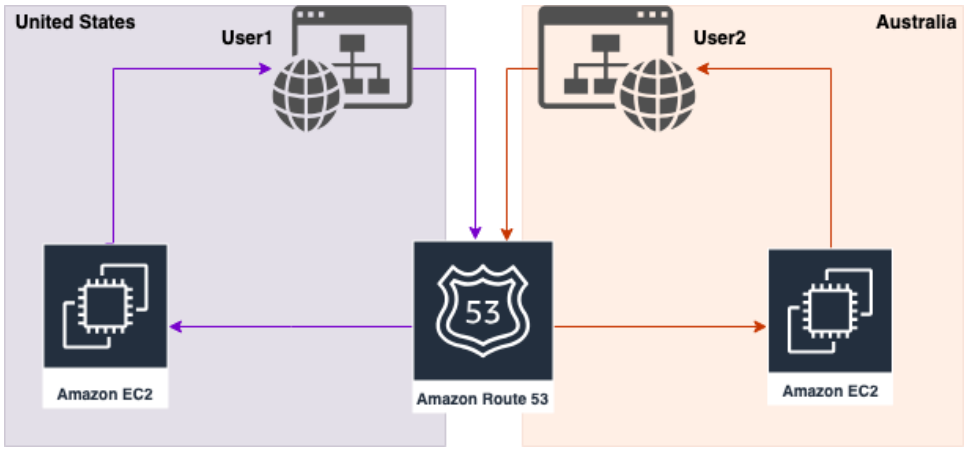
This is clearly illustrated in the diagram above where the world is divided into 5 areas, and any request originating within said area will be directed to its associated resource.

Geolocation Routing Policy

A Geolocation routing policy on the other hand, is used when when we wish to route traffic based on the location of our users. After all, many businesses these days have users all over the world, and it is natural for them to want to serve the appropriate content to those users as fast as possible. A geolocation routing policy allows you to allocate the resources that serve your traffic based on the location that users' DNS queries originate from.

Additionally, with geolocation routing, we can also localize the served content and restrict the distribution of specific content to only those locations where we are allowed to distribute, granting us the benefits of localization alongside the ability to adhere and comply with local rules and regulations.

For example, say, a streaming service like Netflix could have the international rights to an American movie and thus wants to prominently display it on its website in Australia, but not have the domestic rights to the same movie, and would have to serve something else in its stead for its users in America.



Example architecture for a Geolocation based application

Origin Access Identity and Origin Access Control

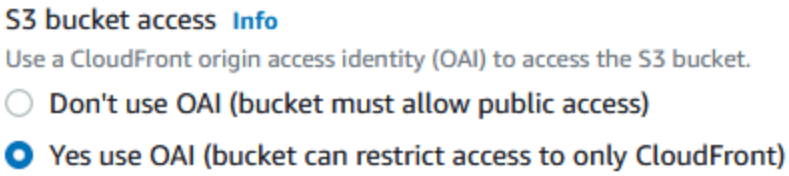
When using S3 with CloudFront, a setup often used when serving static webpages and application files, having an additional layer of security would not hurt. Well, AWS provides us with two methods of restricting access to an Amazon S3 distribution: Origin Access Identity (OAI) and Origin Access Control (OAC).

Origin Access Identity (OAI): A feature of the CloudFront service, it allows us to serve the content of a S3 Bucket without granting public access to the bucket. This restricts any unauthorized user from accessing the S3 files through its direct URL and forces them to use the CloudFront distribution URL instead.

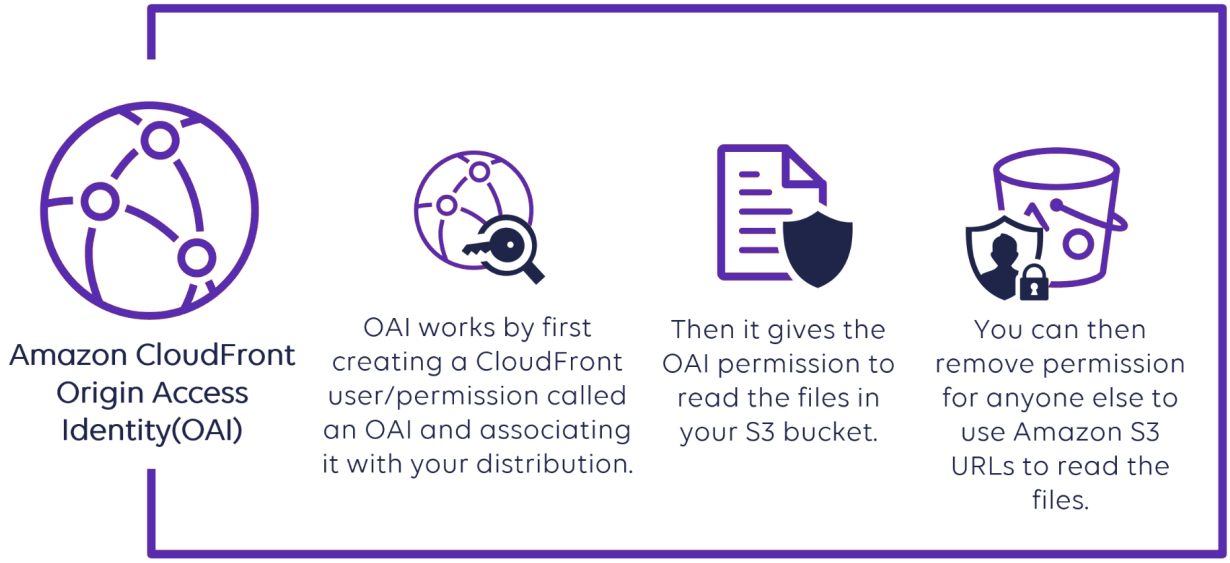
Origin Access Control (OAC): A relatively recent addition to the AWS feature lineup, it performs the same function as OAI, that of permitting access to a select group of S3 buckets, but it uses IAM principals to authenticate with the S3 bucket origin. As mentioned before, IAM is Amazon's proprietary access control and management service suite and AWS considers it a best practice to use it for the security of AWS resources.

However, OAC, unlike OAI is not exclusive to CloudFront, and can also be applied to other types of endpoints such as custom origins or origins hosted on EC2 instances. This makes it a much wider offering than OAI, and its reliance on IAM also makes it a much better integrated part of the AWS ecosystem (IAM and its features will be covered in more depth in future sections).

Refocusing back to OAI, it is worth remembering that OAI is not enabled by default and that it must be configured in the S3 bucket access settings, as shown in the image below:



Additionally, a brief description of how Origin Access Identity works is given in the infographic below:



Security Groups vs NACL

Once we have set up our Virtual Private Cloud (VPC) environments with EC2 instances or other resources within them, ensuring their security and putting guardrails in place such that undesirable users do not access our VPCs automatically becomes a crucial task. AWS provides us with two ways of achieving this: **Network Access Control Lists (NACLs)** and **Security Groups**. Short descriptions of them are given below:

NACLs (Network Access Control Lists): These operate at the subnet level and control traffic for all the instances within that subnet. Unlike Security Groups, NACLs are stateless, meaning that rules need to be explicitly stated for both inbound and outbound traffic separately. Each NACL has numbered rules, and traffic is evaluated against these rules in ascending order, starting from the lowest number. The first rule that matches the traffic is applied, and the rest are ignored. This allows for both "allow" and "deny" rules, providing more flexibility for granular control over traffic.

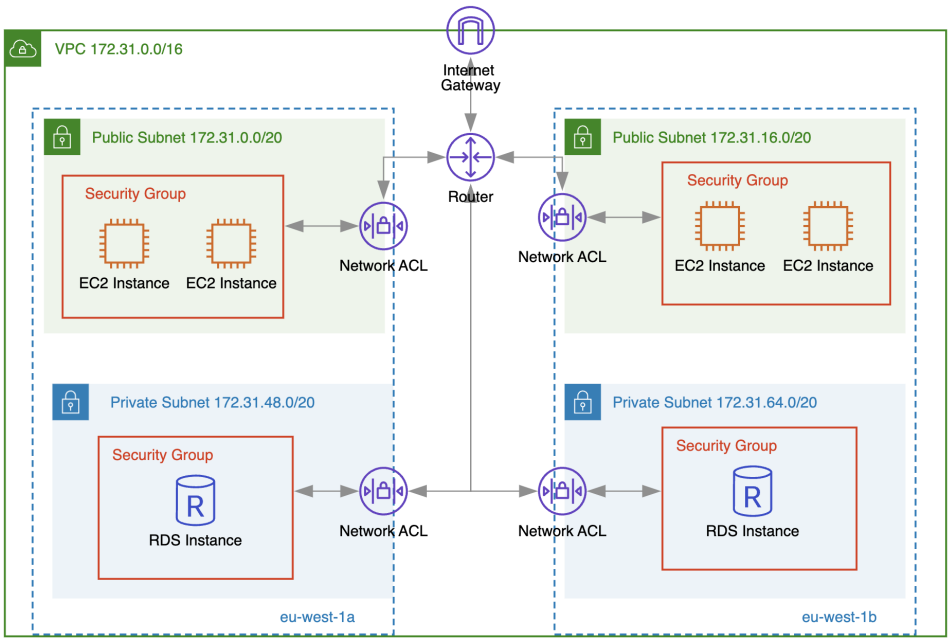
By default, NACLs have a rule that denies all inbound and outbound traffic unless otherwise specified. This makes NACLs useful for applying broad security controls to an entire subnet, as opposed to individual instances. However, their stateless nature means that we need to create separate rules for incoming and outgoing traffic, which can lead to more complex configurations compared to Security Groups. NACLs are often used in scenarios where subnet-level protection is needed, such as when managing public-facing resources that need additional security.

Security Groups: A more granular, EC2 instance focused offering, these act as firewalls at the instance level, controlling the traffic going in and out of individual

EC2 instances. Also, Security Groups are stateful, meaning that if we create a rule allowing an inbound request, the outbound response is automatically allowed, and vice versa. This simplifies configuration compared to NACLs, where each direction must be handled separately. Security Groups do not have an ordered rule set; instead, all rules are evaluated equally, and any traffic that matches an "allow" rule is permitted.

By default, Security Groups has a rule that denies all inbound traffic and allows all outbound traffic. However, they only support "allow" rules, meaning you cannot explicitly block traffic using a Security Group. This limitation is offset by their granular control over instance-level traffic. Security Groups are ideal for managing access to specific EC2 instances for example, allowing us to define precise security policies and protocols.

Furthermore, such that it becomes easier for the reader to visualize Network Access Control Lists (NACLs) and Security Groups, an architecture diagram showcasing the placement and bounds of both are given below:



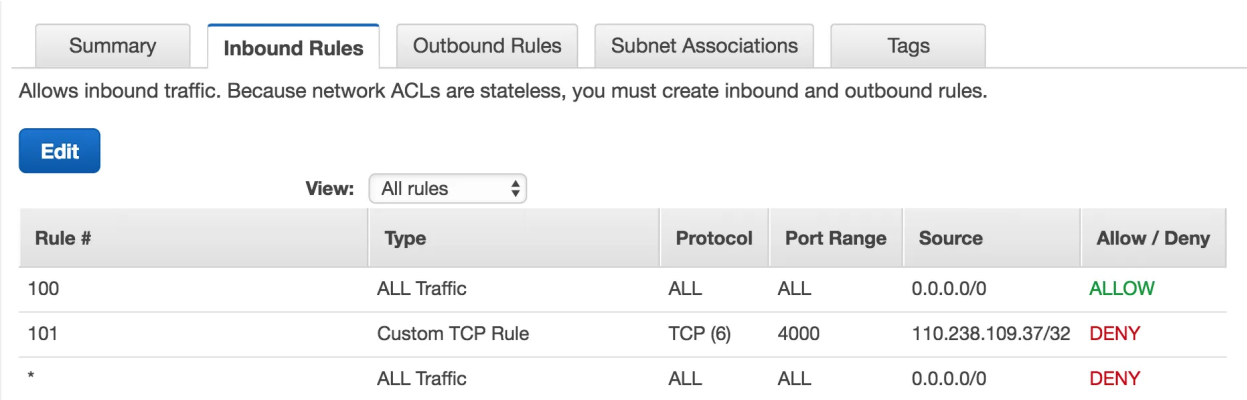
Source: CloudViz

As the descriptions might have already betrayed, though they serve the same purpose, Security Groups and NACLs operate at different levels and have quite a few contrasting characteristics. These characteristics have been tabulated for the ease of the reader below:

Security Group	Network Access Control List
Acts as a firewall for associated Amazon EC2 instances and other specific resources	Acts as a firewall for associated subnets; Ambivalent to resources within the subnet
Controls both inbound and outbound traffic at the instance level	Controls both inbound and outbound traffic at the subnet level
You can secure your VPC instances using only security groups	Network ACLs are an additional layer of defense
Supports allow rules only	Supports both allow and deny rules
Stateful i.e. Return traffic is automatically allowed, regardless of any rules)	Stateless i.e. Return traffic must be explicitly allowed by rules
Evaluates all rules before deciding whether to allow traffic	Evaluates rules in number order when deciding whether to allow traffic, starting with the lowest numbered rule
Applies only to the instance that is associated to it	Applies to all instances in the subnet it is associated with
Has separate rules for inbound and outbound traffic	Has separate rules for inbound and outbound traffic
A newly created security group denies all inbound traffic by default	A newly created NACL denies all inbound traffic by default
A newly created security group has an outbound rule that allows all outbound traffic by default	A newly created NACL denies all outbound traffic by default
Instances associated with a security group can't talk to each other unless you add rules allowing it	Each subnet in your VPC must be associated with a network ACL. If none is associated, the default NACL is selected.
Security groups are associated with network interfaces	You can associate a network ACL with multiple subnets; however, a subnet can be associated with only one network ACL at a time

NACL/SG Rule priority

Both NACLs and Security Groups have been discussed in the last section and how they rely on a sequence of rules. In order to better understand how these rules are organized and evaluated, an example set of NACL inbound rules are given below:



Summary | **Inbound Rules** | Outbound Rules | Subnet Associations | Tags

Allows inbound traffic. Because network ACLs are stateless, you must create inbound and outbound rules.

Edit

View: All rules

Rule #	Type	Protocol	Port Range	Source	Allow / Deny
100	ALL Traffic	ALL	ALL	0.0.0.0/0	ALLOW
101	Custom TCP Rule	TCP (6)	4000	110.238.109.37/32	DENY
*	ALL Traffic	ALL	ALL	0.0.0.0/0	DENY

Source: TutorialsDojo

Lower the rule number, the higher priority it is treated with. The “*” rule in particular being the lowest priority and highest number rule. For example, the “*” rule as shown in the image is the default and lowest rule of the NACL. Additionally, higher priority rules will overpower lower priority ones. In the list of rules present in the above example, rule “100“ will be called first, then “101“, then “*“. So, because all traffic goes through rule 100 first, any and all inbound requests will be passed through regardless of protocol or port range.

A similar format is followed for the organization and evaluation of NACL Outbound rules and Security Groups as well, and therefore I shall refrain from regurgitating the same set of information for the aforementioned categories.

AWS PrivateLink

It was mentioned in a previous section that there are two different types of VPC endpoints: **Gateway VPC Endpoints** and **Interface VPC Endpoints**. Since we have already spent a section discussing Gateway VPC Endpoints, I find it to be worth spending a section discussing the other type of VPC endpoint too.

Now, the main differentiator that we can use to distinguish between Interface VPC Endpoints and Gateway VPC Endpoints are its use of AWS PrivateLink and the range of services covered by it. Let us start with the second one because it is in my opinion, the more important one. Unlike Gateway VPC Endpoints which are limited to the S3 and DynamoDB services, Interface VPC Endpoints allow us to create a private and secure connection with any publicly available AWS service (well technically, any service that supports PrivateLink, which is almost all of them) like EC2, ECS, RDS and all the others that we have discussed in prior sections.

Secondly, Interface VPC Endpoints create these secure connections by utilizing private IP addresses via a service called AWS PrivateLink which allows the service to act as a bridge between VPCs, AWS services and on-premises networks without ever having to expose the infrastructure and traffic to the public internet.

Therefore, whenever we wish to connect one AWS service to another, say EC2 to SNS (Will be discussed in later sections), then utilizing interface VPC endpoints is usually the most convenient way of doing so, especially when privacy and not exposing the resources to the public internet is a major concern.

APPLICATION INTEGRATION
IN
AWS

Decoupling

Decoupling in the context of cloud architecture usually means designing the different parts of an application such that they can function independently of each other. Decoupling is so ubiquitous in the modern tech environment that its actually hard even to imagine an example of a software architecture with different parts of the application tightly knit and dependent on one another.

Lets say for example that we were designing a pedometer application which counts the number of steps each day and then displays it on a calendar even if the device is offline. Then, it might benefit us to package the database and front-end application together within the same distribution (.apk, .exe file) such that these two parts of the application are shipped together and run together, without any middlemen. A mock UI of said app is given below:

This however means that the application and the database code are coupled together, and that if an issue were to happen with say, the database, then the front-end application would go down as well and vice versa since separation between the two parts is non-existent.

Now, it is not hard to see how this could be a problem in more modern software paradigms which rely on quite complex architectures with many different moving parts. After all, risking the operation of the entire application every time a small component within it faces issues.

Decoupling is extremely important within an AWS environment in particular, as we often rely on a wide array of offerings, often belonging to different category of services and serving unique purposes and during such situations it becomes almost a necessity to ensure that issues in one service does not lead to the failure of the entire application.

This also allows us to manage and scale AWS services independently of one another, enhancing the flexibility as well as the reliability of our architecture. Now, there are three major offerings provided by AWS that enable us to efficiently decouple our AWS environment. These are as follows: **Simple Queue Service (SQS)**, **Simple Notification Service (SNS)** and **EventBridge**. All three will be discussed in the next section.

SQS, SNS and EventBridge

Let us take a closer look at the three aforementioned services used for the purposes of decoupling:

Simple Queue Service (SQS): SQS is designed for reliable, one-to-one asynchronous communication, where messages are held temporarily in a queue until they are processed. It is ideal for distributed systems that require message decoupling. SQS excels in situations where we need a temporary message holding pool and ordered message processing, especially when the consumer may not be available to process messages immediately. It is primarily used in cases where we want guaranteed message delivery whilst being able to tolerate a little flexibility in processing speed.

Simple Notification Service (SNS): SNS on the other hand, is a fully managed pub/sub (publish/subscribe) service allowing a one-to-many communication model. It is suited for scenarios where a single message needs to be sent to multiple subscribers in parallel, such as in fan-out patterns. It supports high throughput and can handle many subscribers. SNS is often used when we wish to decouple different parts of our system by broadcasting a message to various services at once.

Amazon EventBridge: A relatively new service, EventBridge is used when we wish to connect AWS services or integrate with third-party SaaS applications in a scalable manner. While it also supports one-to-many communication similar to SNS, it has more limitations than SNS in terms of throughput and distribution, but excels in the ability to conveniently integrate AWS services with both external and internal SaaS products.

An infographic comparing and contrasting the three different services is given below:

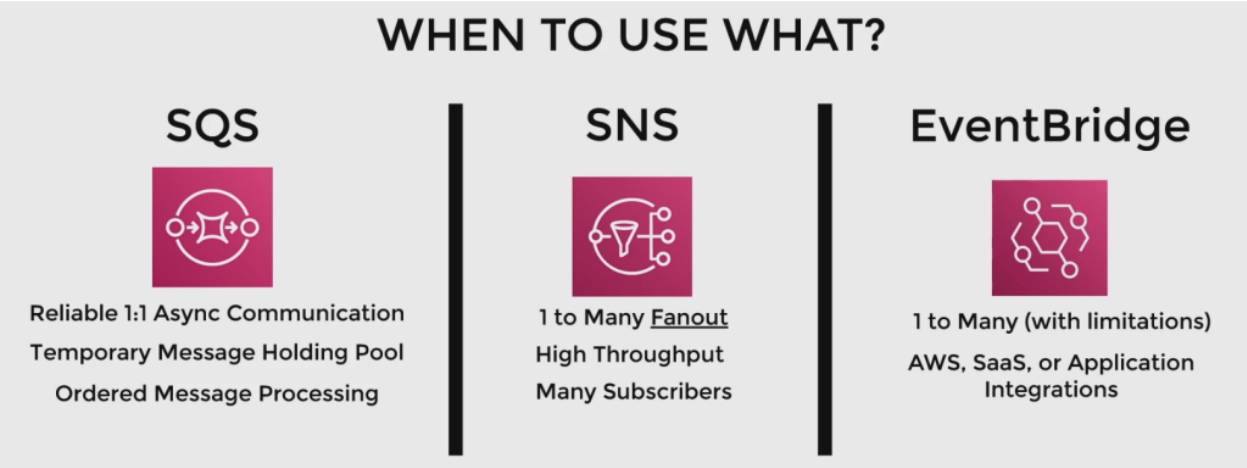


Image credit: Be a Better Dev; (Do check out his YouTube Channel)

SQS Size-based Scaling and other features

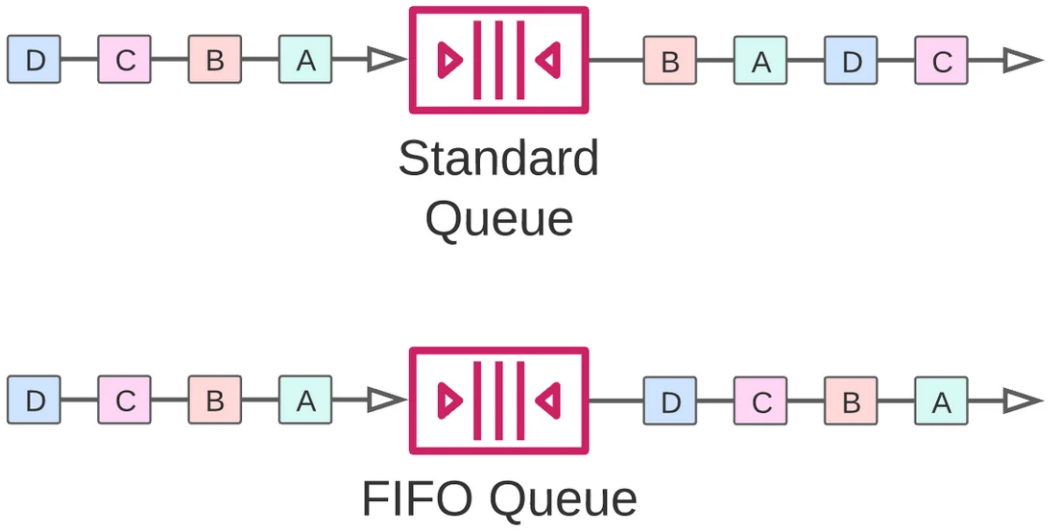
Though we gave a brief description of SQS in the last section, the service actually has many interesting sub-features and use cases which makes it a much more versatile service than was described in the last section. Though a full exploration of all the sub-features and use cases is beyond the scope of this book, there are two that I would like to point out: **SQS based scaling**, and **SQS-triggered functions**. A concise description of both is given below:

SQS based scaling: There might be cases where we wish to scale the resources within an Autoscaling Group based on the amount of orders or requests being received. This can easily be done using SQS, as AWS allows us to scale Autoscaling Groups based on the size (i.e. number of processes) of the queue, increasing or decreasing compute capacity depending on the number of orders in a queue, for example.

SQS-triggered functions: SQS can also act as an event source for AWS Lambda functions. When messages arrive in the queue, they can trigger Lambda functions to process these events, making SQS an effective mechanism for building event-driven, serverless workflows.

SQS FIFO

Amazon SQS FIFO (First-In-First-Out) queues are designed to provide strict message ordering and ensure exactly-once processing. This means that messages are processed precisely in the sequence they were sent and are never duplicated, making FIFO queues particularly suitable for scenarios where message order is critical. This is in contrast to SQS Standard, where orders are processed at random and/or in an order not decided upon or controlled by us. An image comparing how the two SQS Queue types process requests differently is given below:



Source: FourTheorem

By maintaining this strict ordering, SQS FIFO queues are ideal for use cases such as financial transactions, e-commerce order processing, or complex workflow management systems—where processing messages out of order or more than once could lead to errors, inconsistencies, or significant operational risks.

SQS Duplication

Sometimes when utilizing SQS it is possible to face the problem of messages being repeatedly processed, a phenomena termed as **SQS Duplication**. The two major reasons why SQS duplication happens are as follows: **Visibility** and **Deletion**.

Let us deal with visibility first. Say that there is an SQS queue and two instances that consume from it. If both instances consume from the same SQS queue at the same time, then the same item in the queue will be processed twice. In order to prevent this, SQS provides us with an option called the visibility timeout, the time period during which a message is invisible to other services which consume from the same SQS queue. If however, the consumer doesn't delete the message within this timeout (perhaps due to a processing delay or failure), the message becomes visible again, leading to its reprocessing by the same or another consumer.

The second reason why SQS duplication may occur be due to a failure in deleting items in the SQS queue once it has been used. Note that SQS does not automatically delete the message. Instead, the consumer must explicitly delete the message using the `DeleteMessage` action after it has been successfully processed.

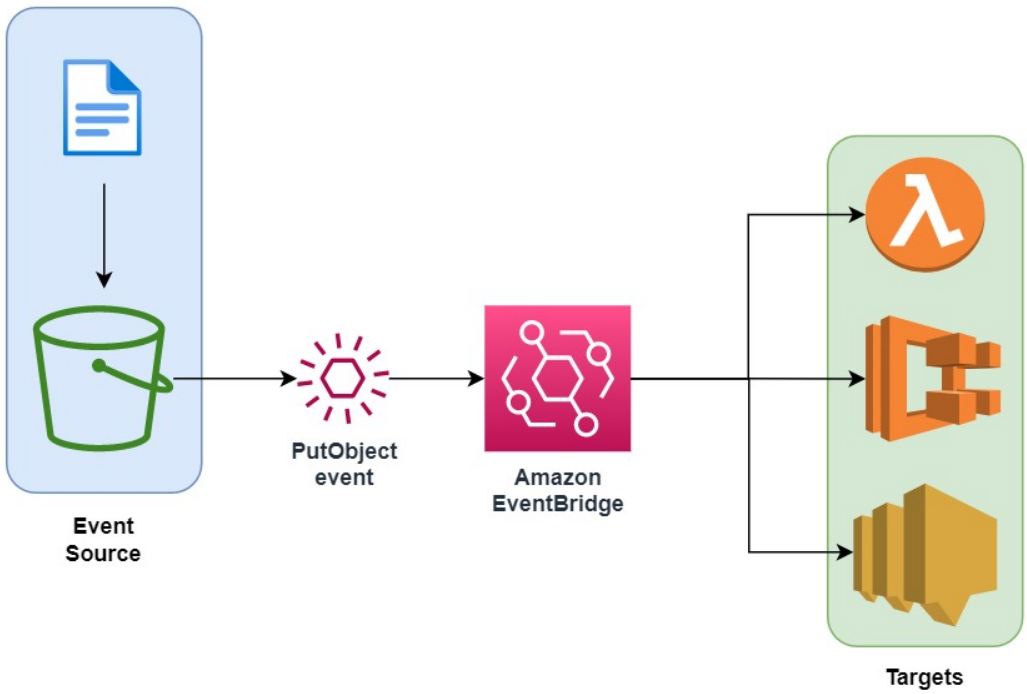
To avoid this duplication problem however is relatively easy and can be easily mitigated by:

- Setting the visibility timeout appropriately, most likely using the `ChangeMessageVisibilty` API call, allowing enough time for message processing.
- Ensuring that the services which consume the messages from the SQS queue successfully delete a message from the queue after reading it.

Amazon EventBridge

One of three aforementioned services used for the decoupling of architectures, **Amazon EventBridge (formerly called Amazon CloudWatch Events)** is a serverless event bus that makes it easy to connect applications together. It uses data from your own applications, integrated software as a service (SaaS) applications, and AWS services.

This simplifies the process of building event-driven architectures by decoupling event producers from event consumers. This allows producers and consumers to be scaled, updated, and deployed independently. Loose coupling improves developer agility in addition to application resiliency.



Source: AWS

Further, something worth noting is that we can use Amazon EventBridge (Amazon CloudWatch Events) to run Amazon ECS tasks when certain AWS events occur. This can allow us to easily integrate AWS events with tasks defined in AWS Lambda, ECS and other targeted serves.

An architecture diagram showing how EventBridge rules can be used to run an Amazon ECS task whenever a file is uploaded to a certain Amazon S3 bucket using the Amazon S3 PUT operation has been provided for the easy understanding of the reader.

Simple Email Service (SES)

Businesses since the dawn of the internet age have become increasingly reliant on emails for outreach to customers and other stakeholders. Now this has led to a great deal of services that manage the formatting, sending and monitoring of emails often called **Email Communication APIs** and broadly categorized under the more broader term of **CPaaS (Communication Platforms as a Service)**.

Simple Email Service or just SES for short, is one such Email Communication API made available and managed by AWS. SES allows us to send, process and receive emails just like most other communication APIs but its ability to take advantage of AWS vast hardware infrastructure and easy integration with other AWS services has led to multiple unique use cases and standout features, some of which are listed below:

- **Event-Driven Email Sending:** SES can be integrated with Amazon EventBridge to trigger emails based on events in your application. This is useful for automating responses to specific actions like order status changes, user sign-ups, or system alerts. For example, when a user's subscription is about to expire, SES can send an automated reminder email driven by a scheduled event or condition.

- **Transactional Email Optimization:** SES enables us to personalize transactional emails (like order confirmations or password resets) at scale by integrating with Lambda and other AWS services for dynamic content generation. For example, SES can be combined with Lambda to customize an email with personalized data such as user-specific discounts or usage reports.

- **Personalized Bulk Emails:** SES can send bulk emails while personalizing each email's content. This is especially useful when sending tailored marketing campaigns where each recipient receives customized content. For example, using configuration sets, we can track individual user interactions (opens, clicks) in a bulk email campaign and take specific actions for users who haven't engaged, such as follow-up reminders.

- **Email Receiving and Processing:** Once again, SES is not just for sending emails—it can also receive and process incoming emails. We can also configure SES to automatically trigger workflows in response to received emails, such as forwarding, archiving, or triggering Lambda functions. For example, if we are running a customer support service, SES can route incoming customer inquiries to specific teams based on say, specific keywords in the email content or automatically trigger follow-up workflows.

These capabilities make the service something beyond just simple email delivery and make SES a highly adaptable tool for any business that need to manage email communications effectively especially if its already familiar with leveraging the other AWS services.

Infographic on Amazon SES; Source: AWS

Kinesis

The **Kinesis** family provides us with Amazon's set of offerings related to real-time data streaming, allowing us to collect, process, and analyze data continuously at any scale. It's ideal for applications that require real-time analytics, such as monitoring logs, processing IoT data, and running machine learning models. The Kinesis services enable us to ingest large streams of data and process it with quite low latency, providing immediate insights and actions. There are four main services within the Kinesis family, which are shown below:

Amazon Kinesis Data Streams (KDS): Used to build custom pipelines and applications that process or analyze streaming data, it provides us with the ability to continuously capture gigabytes of data per second from hundreds of sources like website clickstreams for example.

Amazon Kinesis Firehose: Responsible for simplifying the process of loading streaming data into data lakes, warehouses and analytics services into AWS data stores at a **near real-time** pace, it can automatically scale to match the throughput of our data streams. However, it should be mentioned that Kinesis Firehose is a more limited service compared to KDS in terms of supported destinations as Firehose mostly being used to send data to S3, Redshift or Elasticsearch.

Amazon Kinesis Video Streams: Similar to KDS but used specifically for securely streaming media streams (like video or audio) from connected devices to AWS data sources.

Amazon Kinesis Data Analytics: Sometimes also referred to as Amazon Managed Service for Apache Flink, it processes data streams in real time with SQL or Apache Flink, two popular languages for data analysis and processing.

Quicksight

Amazon QuickSight is a cloud-powered business intelligence (BI) service and reporting solution that allows us to easily create and publish interactive dashboards and visualizations, as well as share them with IAM Users and Groups. These visualizations can take the form of charts, graphs, maps, and many more.

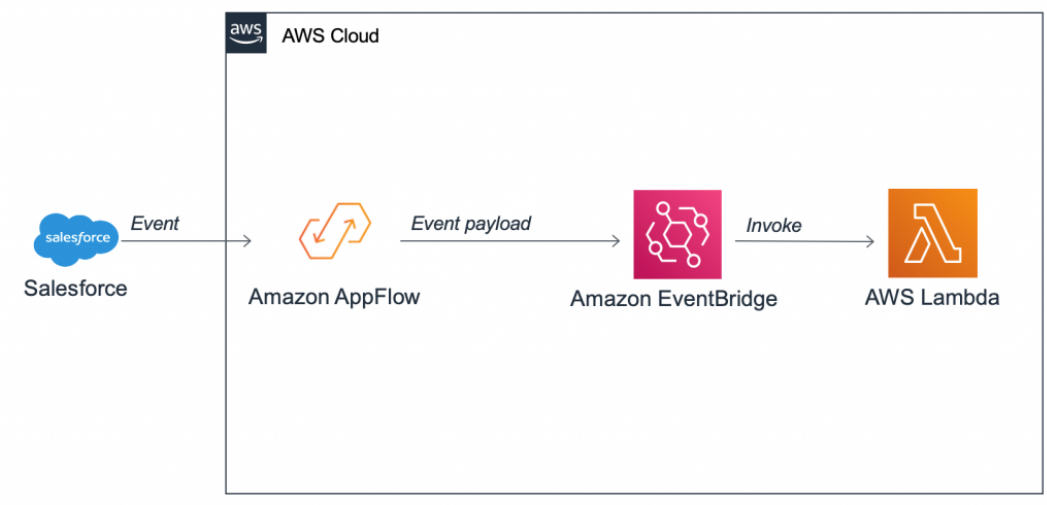
QuickSight's integration with AWS services ensures seamless data connectivity and scalability, making it ideal for organizations leveraging data lakes to efficiently derive actionable business intelligence, providing insights into large datasets stored in the data lake.

The purpose of QuickSight is to democratize access to data analytics by enabling users of all skill levels, from data scientists to business managers, to create insightful reports and dashboards without relying heavily on IT teams. Its development was driven by the need for businesses to gain real-time insights from their data without the complex infrastructure management and costs associated with traditional BI tools. As data volumes grew, services like QuickSight were created to simplify the process of connecting, analyzing, and visualizing large datasets in a scalable and cost-effective manner within the AWS ecosystem.

Amazon AppFlow

As businesses digitized, they sought for ways to use software technology to streamline various processes and increase their efficiency at certain tasks. This is usually where **Software as a Service (SaaS)** products like Salesforce, Google Analytics and ServiceNow enter the picture, providing businesses with CRM (Customer Relationship Management), HR management and other enterprise solutions. In response to the widespread adoption of SaaS solutions, Amazon launched **AppFlow**, a fully managed integration service for the secure transfer of data between AWS services and SaaS applications like the ones mentioned before. It simplifies data transfer processes, enabling us to automate and control data flows without needing to write custom code or create data pipelines from scratch.

It is often used in conjunction with EventBridge, a service that we have already established as a helpful tool when working with third-party software. An example architecture showcasing how a Salesforce event can be used as the invocation source for a Lambda function is given below:



MANAGEMENT AND GOVERNANCE
IN
AWS

CloudWatch and Metrics list

Amazon CloudWatch is a monitoring and observability service that provides insights into AWS resources and applications using easily readable dashboards. It collects and tracks metrics, logs, and events, allowing us to monitor performance, detect anomalies, and set alarms for operational issues. CloudWatch creates its dashboards and performs its operations based on something called Metrics, numerical data points that represent the performance of our systems.

Said metrics help us track key performance indicators (KPIs) such as CPU usage, memory utilization, and request counts, enabling us to analyze and optimize the performance of our applications and infrastructure. CloudWatch has three metrics setup by default and another five custom metrics which can be enabled as per our requirement, both of which are worth remembering and noting, especially for cloud administrators.

The three default metrics in CloudWatch are as follows:

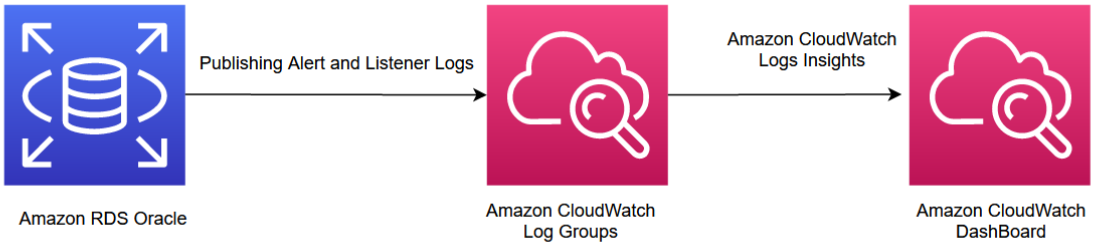
- **CPU Utilization of an EC2 instance**
- **Disk Reads activity of an EC2 instance**
- **Network packets out of an EC2 instance**

While the five custom metrics readily available for setup are as follows:

- **Memory utilization**
- **Disk swap utilization**
- **Disk space utilization**
- **Page file utilization**
- **Log collection**

CloudWatch Logs Streams

As mentioned before, CloudWatch can be used to generate and aggregate logs in a central location using a sub-offering from CloudWatch called CloudWatch Logs, which allows us to organize logs pertaining to all of your systems, applications, and AWS resources all in a single, highly scalable service. A diagram of how CloudWatch logs can be integrated with existing AWS infrastructure is illustrated below using the example of RDS databases:



Amazon CloudWatch Logs Streams is an extension of CloudWatch Logs, and can be seen as Kinesis for logs, allowing sequences of log events from the same source to be streamed in **near real-time** into other services such as S3 or Elasticsearch (Amazon OpenSearch), allowing us to stay up to date on the happenings of our resources quite conveniently, a particularly useful tool for cloud administrators who can often mitigate and solve infrastructure disasters immediately. In fact, we can use the CloudWatch API alongside CloudWatch Log streams to better manage our AWS environment, creating a lambda function to say, re-establish the connection between our RDS and EC2 instance, trigerring it whenever either faces an issue and has to restart.

CloudTrail

Since AWS environments are so large and complicated, it is often helpful for us to be able to track the actions that are undertaken by users, services and agents in relation to each other. AWS CloudTrail provides us with the ability to do just that, recording said actions as events within it. CloudTrail is always automatically enabled on every AWS account on the point of creation and helps us perform **operational and risk auditing, governance and compliance** operations on our AWS account.

The above description might make it sound like CloudTrail serves many of the same functions as CloudWatch but such beliefs are not to be heeded. This is because CloudTrail is best understood as a tracking tool, allowing us to keep a breadcrumb trail of all the minute actions that were performed within our AWS environment while AWS CloudWatch is mainly a monitoring tool, used to observe the current status of various AWS resources.

TLDR;

CloudTrail ⇒ Tracking

CloudWatch ⇒ Monitoring

Cost and Usage Report + Cost Explorer

Now, we have discussed a whole host of services and how they can be used to spin up complex architectures and create a variety of novel applications but it is also worth remembering that there is no such thing as a free lunch, and it is not very hard to imagine companies utilizing AWS racking up (sometimes unnecessarily) huge bills on cloud infrastructure. Well, AWS also graciously provides us with services for financial management, allowing us to better understand the costs associated with our infrastructure. In this section, we will be very briefly discussing two of those services: **Cost and Usage Report (CUR)** and **Cost Explorer**.

Both services are quite different in nature and serve both different use cases and potentially even different user bases. Cost and Usage Report provides us with the most comprehensive set of cost and usage data available for a user, organized as a detailed billing report. Said billing reports can be then published to a S3 bucket of our choosing.

Cost and Usage Report allows us to create reports that break down our costs by by the hour, day, or month, by product or product resource, or by tags that we define ourselves as a Comma-Separated Values (CSV) file, and can be viewed in any spreadsheet software of our choice.

AWS Cost Explorer on the other hand is a more technically limited service, in the sense that it cannot provide us with the same level of granular detail about the costs being incurred by the different resources used in AWS, has a 24 hour lag

when displaying monthly data, and cannot give us in-depth hourly data beyond the last 14 days, a capability that CUR can easily meet.

However, Cost Explorer is a more accessible service, relying on visualizations to present the cost and usage data in a more easily readable and understandable format usually in the form of graphs, charts and tables. Said visualizations can after all be understood by anyone with basic statistical literacy regardless of their technical familiarity with AWS, especially when forecasting and creating graphs of forecasted future costs, a feature made available to us by AWS Cost Explorer.

A table comparing the two services has been made available for the ease of the reader below:

	AWS Cost and Usage Report	**AWS Cost Explorer**
Description	A comprehensive, spreadsheet-like report of your billing data for a Payer AWS account	Highly visual cost graphs and tables showing a relatively high-level view of your costs and usage data for a specific Payer account in AWS
Unique functionality	Details historical cost and usage data and sends it to an Amazon S3 bucket for further analysis and longer retention	Not only offers historical records but also creates forecasts and savings recommendations
Data fields	Multiple line items. Also supports Cost Categories and Cost Allocation Tags	Up to 18 filters and groupings
Format	CSV and Parquet	CSV
Cost data duration	Hourly, daily, and monthly	Hourly (up to 14 days), daily, and monthly
Pricing	Free, but standard Amazon S3 charges apply	Free, although querying cost and usage data via the Cost Explorer API costs $0.01 per paginated request

AWS Config

As an AWS environment grows larger in size and starts integrating more and more components and services within itself, managing the different settings and configurations of all the different resources does become quite a bit of a hassle. In order to make said tasks easier, Amazon launched the AWS Config service.

AWS Config (short for AWS Configurations) is a service that allows us to assess, audit and evaluate the configurations (and configuration changes) of different AWS resources within our account, presenting the information to us in a relatively consistent format.

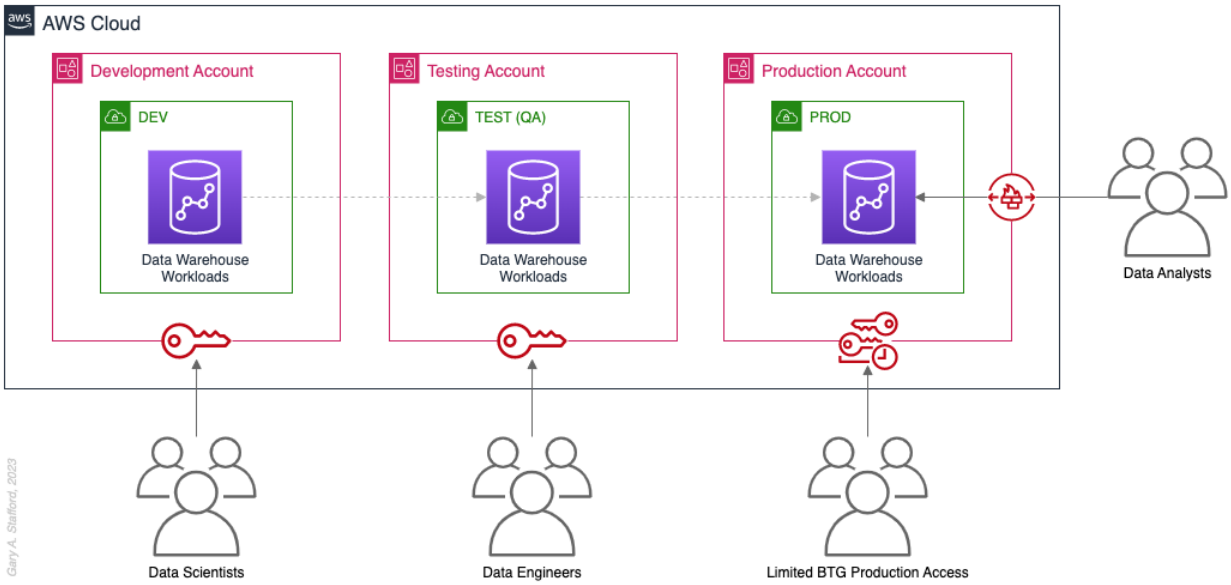
It also allows us to govern over our resources in other ways by providing us with ways to define rules to for example, detect improperly tagged resources or check if a certain setting has been enabled. In fact, the rules defined by us, (called AWS Config custom rules) are often used as invocation sources for Lambda functions, an example of which will be provided in a later section.

Also, AWS Config is automatically utilized by AWS Control Tower for its functioning, a service used when managing an AWS environment that has resources spread across multiple accounts. AWS Config Resources automatically created by AWS Control tower can be easily identified using the `aws-control-tower` tag alongside the `managed-by-control-tower` value. Certain sub-features of the AWS Config service also aid in maintaining the security of AWS resources and will be touched upon in a future section.

AWS Organizations and PrincipalOrg Id

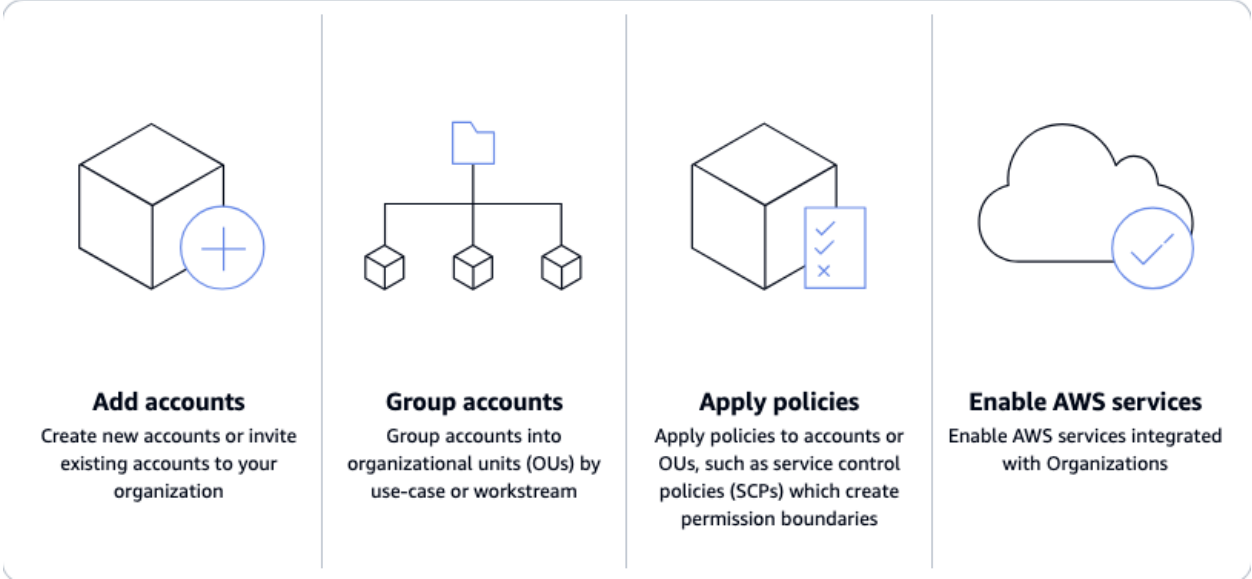
Now, multi-account AWS environments were mentioned previously once before but an explanation as to what they were and how they work was never explained. Multi-account AWS environments are, simply put, architectures that utilize multiple accounts to manage resources, users and workloads within an organization. This is often done for security and categorization reasons, allowing us to insulate workloads from one another and prevent unwanted users poking around in places where they do not belong, causing undesirable consequences.

An example of an environment which utilizes three different accounts for three different workloads, using separate accounts for development, testing and production, is given below:



Source: AWS

In order to more easily create and manage accounts that utilize a shared set of resources, AWS created a service called **AWS Organizations**. It works by grouping accounts into something called Organizational Units (or OUs), to which policies can be assigned/added to. These policies define the OUs permission boundary, i.e. which services, instances and other general AWS resources that they have access to, dictating what the accounts can and cannot do. An infographic presenting the mentioned capabilities of AWS Organizations is given below:



Source: AWS

Finally, OUs can help give some much needed structure to complex architectures and help conveniently locate, isolate and differentiate similar sets of resources from one another. This is because resources being governed by an OU can be quite easily identified by the presence of the Principal Org ID, whose value is used to uniquely identify and distinguish OUs from one another. In contrast, resources that do not belong to any OU have no such tag, making them easy to identify as well.

AWS Backup

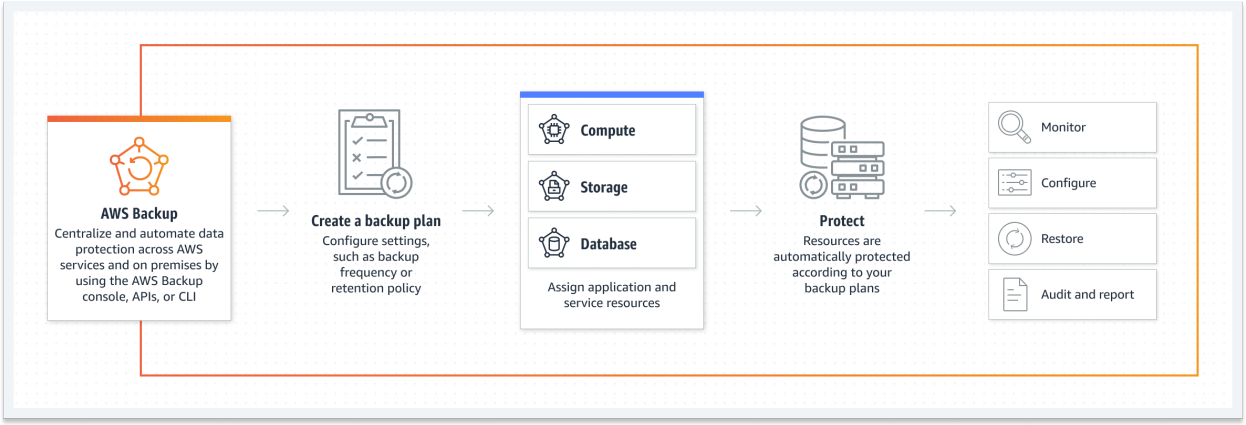
No business is completely immune from the loss of information. Breakdown of hardware, malware, hacking, unwanted intrusion and accidental deletion are just some of the ways in which data relating to business operations may be lost. Having backups of said data therefore may be a desirable thing for many businesses and is in fact, a necessity by law in many countries.

In the early days of cloud computing, companies had to manually script backup jobs, store data offsite, and frequently monitor backup operations, tasks that are prone to error. As cloud usage grew, these processes became an increasingly significant (and also often increasingly tedious) task. It was in this environment that AWS first launched **AWS Backup** in January 2019, offering an easy-to-use fully managed backup service that centralizes and automates the backup of data stored within a group of select AWS services which are listed below:

- **Amazon EC2** / **Amazon EBS**
- **Amazon S3**
- **Amazon RDS** (including all DB engines like **Aurora** and **DynamoDB**)
- **Amazon DocumentDB** / **Amazon Neptune**
- **Amazon EFS** / **Amazon FSx** (including Lustre and Windows File Server)
- **AWS Storage Gateway** (Volume Gateway)
- **Amazon Redshift**
- **Amazon Timestream**
- **AWS CloudFormation**
- **VMWare Cloud on AWS**

Note that AWS Backup is limited to the above set of services, and data stored in any other AWS offering must either be manually backed up or backed up using a third party tool.

AWS Backup also allows us to create data backup plans for the mentioned services, define backup policies for the data within them, automate backup scheduling, manage backup retention periods and store said backups across both multiple AWS regions and multiple AWS accounts. An infographic on how AWS Backup works is given below:



Source: AWS

The service definitely simplifies a myriad of data protection tasks and helps companies ensure the resilience of their data, meet regulatory compliances and perform recovery tasks without the need for custom scripting and/or manual intervention and has led to AWS Backup becoming the somewhat default option for backing up data related to AWS-specific resources. Implementations of the **Backup and Restore** method of disaster recovery mentioned in the Databases section for example, are often based on the AWS Backup service.

Systems Manager Patch Manager vs Run Command

AWS Systems Manager Run Command allows you to remotely execute commands on multiple EC2 instances, on-premises servers, and virtual machines (VMs) at scale. It simplifies operational tasks by eliminating the need for manual SSH or RDP access, making it ideal for automating software installations, configuration changes, and administrative tasks across your infrastructure.

Patch Manager, also part of AWS Systems Manager, automates the process of patching operating systems and applications across your instances. It ensures that your systems remain up-to-date with the latest security patches and updates, thereby enhancing security and compliance while reducing manual effort in managing patch deployments. Together, these tools streamline management tasks and improve operational efficiency within your AWS environment.

TLDR;

Systems Manager Patch Manager is used to perform OS level patches on the EC2 instances.

On the other hand, System manager run command is used to run third party commands and software updates and the like.

CloudFormation and Creation Policy

We have talked so far about the different types of resources and architectures in AWS, but we have not really delved deeply into the many novel ways in which our cloud infrastructure can be created and managed. Two ways in which resources can be created, configured and managed within AWS are the AWS Management Console and AWS CLI, both of which have been discussed in an earlier background section. However, there is another method: **CloudFormation**.

CloudFormation is an **IaC(Infrastructure as Code)** service which allows us to automate the provisioning of AWS resources using code. This is done by writing a **CloudFormation Template**, a single JSON file that describes all the AWS resources that need to be created and the relations between them, non-technical readers may imagine this file as a sort of code version of the architecture diagrams that have been sprinkled throughout the book. Said templates can then be fed into CloudFormation, spinning up and configuring the described infrastructure on our behalf, saving us the hassle of individually creating and configuring AWS resources and figure out what's dependent on what, immensely simplifying the process of deploying and managing infrastructure on AWS.

CloudFormation also has an attribute called **CreationPolicy** which can be used to setup certain third party software and ensure certain components are running before the resources mentioned in the template are spun up. This is because the resources utilizing the CreationPolicy attribute may be forcefully restrained from reaching the ***"Completed"*** stage until AWS CloudFormation verifies that certain conditions such as a specified number of success signals or timeout period has been met.

SECURITY, IDENTITY, AND COMPLIANCE IN AWS

Security: Background

In this section, we will discuss the components and fundamental concepts related to infrastructure and software security. The topics to be discussed in this chapter are as follows:

- Importance of security
- Encryption and Decryption
- Credentials and automatic rotation
- DDoS attacks
- Firewalls, and their necessity

If the reader believes themselves to already be familiar with the listed concepts, then they can feel free to skip this section and move on to the rest of the chapter.

Importance of Security

There might be readers who find the concept of securing infrastructure that they do not own to be rather uneasy. After all its their infrastructure, and we are their customer, should not the responsibility of fighting against intrusion and protecting our data be part of the contract?

Well, yes and no. While cloud infrastructure and environments do provide with some basic level of security to the prospective user, this is quite limited and easy to circumvent for any malicious agent worth their salt. Therefore, more often than not, the responsibility of protecting our data and the infrastructure being rented falls upon our own shoulders.

Failure to do so can result in the loss of crucial business data and/or cause the whole system to crash and cease functioning.

Encryption and Decryption

Converting data into an unreadable format, such that even if a malicious agent should get hold of the data, they will not be able to make use of it is called **encryption**, while the process of obtaining the original data from the converted data is called **decryption**. Both processes are performed using a cryptographic token called a **key**, and an **encryption algorithm**, the sequence of operations performed on the data using the key as a variable in order to encrypt and/or decrypt it.

Credentials and Automatic Rotation

Effective credential management is vital to maintaining secure access to infrastructure. Credentials (such as usernames, passwords, API keys, and tokens) are often required to authenticate users or systems, but when mismanaged, they can become points of vulnerability.

Automatic rotation of credentials is a practice that enhances security by periodically updating credentials without manual intervention. Regular rotations help limit the potential exposure window if a credential is compromised. Automations and services that make credential rotation more convenient will be mentioned in later sections.

DDoS Attacks

A Distributed Denial of Service (DDoS) attack aims to overwhelm an infrastructure or service with excessive traffic from multiple sources, causing it to slow down or crash entirely. These attacks are often orchestrated by malicious entities looking to disrupt operations or exploit vulnerabilities for personal gain.

Cloud providers offer DDoS protection services, which can detect unusual traffic patterns, identify DDoS threats, and automatically scale resources to absorb the increased traffic. However, the best DDoS defense strategy combines provider-

based solutions with proper architectural planning, such as deploying services across multiple regions to distribute the load effectively.

Firewalls and Their Necessity

Firewalls act as a critical line of defense by monitoring and controlling incoming and outgoing network traffic based on predefined security rules. They are essential for enforcing boundaries within and outside your cloud infrastructure.

By setting up firewalls, organizations can block unauthorized access while permitting trusted communication, reducing exposure to external threats. Firewalls can operate at various levels, from network firewalls that control access to entire subnets to web application firewalls (WAFs) that specifically filter HTTP/HTTPS traffic for web applications.

AWS Secrets Manager

Accessing AWS services often involves the use of credentials, keys and other information (Usernames, passwords, etc) which need to be kept secure and protected from unauthorized access and distribution. In fact, AWS even has a name for such sensitive pieces of information: **Secrets** and a service called **AWS Secrets Manager** that helps us well, manage them.

To be more specific, AWS Secrets Manager is a service that helps securely store, manage, and retrieve **database credentials**, API keys, and other sensitive secrets. It allows **automatic rotation of credentials**, reducing the risk of exposure and ensuring robust security. By integrating seamlessly with AWS services such as RDS, Redshift, and Lambda, Secrets Manager simplifies managing access to sensitive information, enhancing the security of our AWS environment.

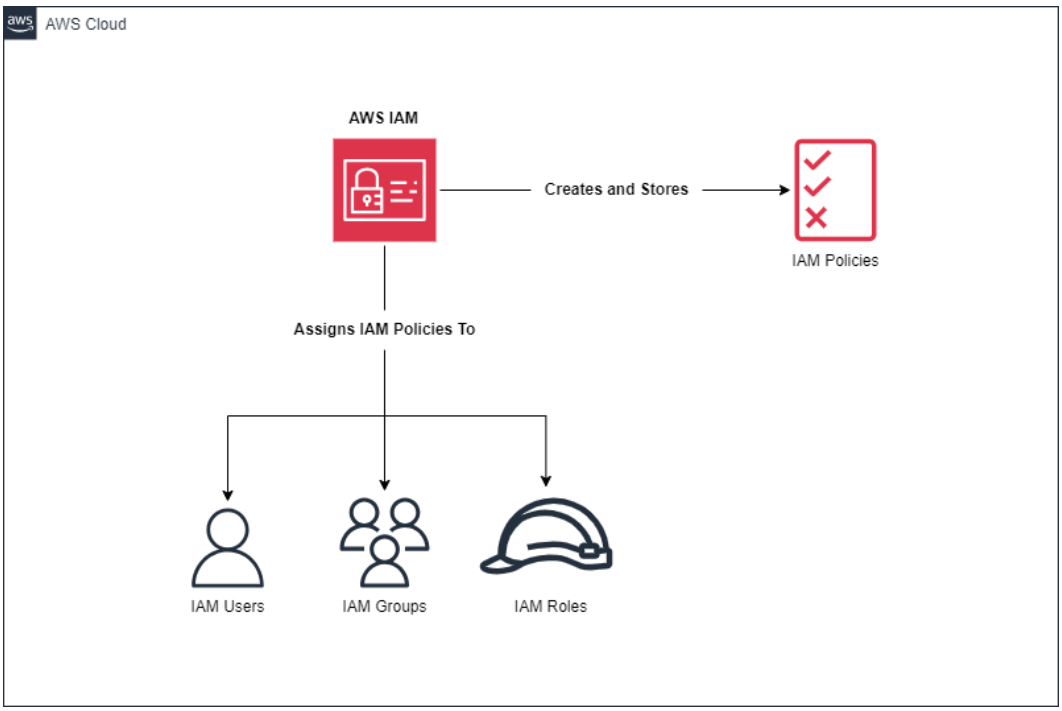
It also offers fine-grained access control using AWS Identity and Access Management (IAM), a cornerstone service for security in AWS (discussed in next section), ensuring that only authorized users or services can retrieve secrets. Additionally, Secrets Manager encrypts secrets using AWS KMS, providing an extra layer of protection.

Note however that AWS Secrets Manager is not the end-all be-all service for storing sensitive information and that certain other AWS services (some of which will be discussed later) may be better suited for specific types of secrets. When storing encryption keys for example, it might be better to use the AWS Key Management Service and when storing string parameters it might be better to use AWS Systems Manager Parameter Store.

IAM and IAM Roles

The principle of least privilege has already been discussed in the background section, but how exactly do we ensure that the people and teams using our AWS environment are made to follow it? How do we micro-manage permissions for individual users, groups and all the different agents that may be utilizing our AWS resources? The answer is **AWS Identity and Access Management**, or IAM for short.

AWS Identity and Access Management (IAM) is a key service that enhances security and access control within the Amazon Web Services ecosystem. It allows cloud administrators to manage users and groups, setting and assigning fine-grained permissions to them. IAM helps us ensure that only authorized users can perform specific actions, maintaining robust security and compliance.



The versatility of IAM, its wide-ranging use cases and multitude of features and sub-features make it a hard service to summarize properly. But most of what IAM does can be encapsulated as being responsible for **A)** Creating and storing IAM Policies and **B)** Assigning them to different identity types (IAM Users, Groups and Roles). Both factors have been illustrated succinctly in the diagram above.

Now, from that above statement we can craft the following questions, answering which will help us get a more complete understanding of IAM:

- What are IAM Policies?
- What is an identity type, and how do IAM Users, Groups and Roles differ from one another?

IAM Policies

An IAM policy is a document which decides the permissions (i.e. what it can and cannot do) of any AWS resource or IAM identity associated with it. Discussing all the different types of policies is beyond the scope of this book, limiting the discussion to identity-based policies as shown in the diagram.

Identity based policies are documents written in JSON that can assign and control the permissions associated with any IAM User, Group or Role it is associated with. An example of an identity-based IAM Policy that enabled any resource/service it is attached to the ability to access an S3 bucket named `iamBucket`:

```
{
   "Version": "2012-10-17",
   "Statement": {
       "Sid": "ListObjectsInBucket",
       "Effect": "Allow",
       "Action": "s3:ListBucket",
       "Resource": "arn:aws:s3:::iamBucket"
   }
}
```

Note however that it is not necessary for a cloud administrator to write JSON documents every time they wish to assign permissions related to a service. For most widely used tasks, AWS provides us with **Managed Policies**, which are JSON documents created and administered by AWS on our behalf. We can simply pick them up and attach them to the desired resource, bypassing the hurdle of having to write one ourselves.

IAM Identities

We have mentioned IAM identities a great deal, but what are they exactly? In simple words, an IAM Identity is any agent or collection of agents that can interact with resources and services in AWS. Said agent could be a human or it could be a computer, with different identity types being used accordingly.

IAM identities are yes, used to define permission boundaries (what an agent can or cannot do), but it is also used for easy identification, authentication and verification of the agent it relates to, helping us ensure that the agent is "legitimate". The three different identity types are very briefly explained below:

- **IAM Users**: An IAM User represents a single person, a single human being within our AWS environment. When we first create an AWS account for example, a single identity that has complete access to all AWS services and resources in the account is created. This identity is called the AWS account ***Root User***.

- **IAM Groups:** An IAM Group represents a collection of IAM Users, allowing us to easily organize groups of IAM Users together.

- **IAM Roles:** An IAM Role is a broader identity type, and can be assumed by any entity that needs it, be it human or computer. It is most widely used to bestow permissions upon specific applications and/or AWS Services. For example, if we wished for an EC2 instance to be able to access DynamoDB, we would assign an IAM Role to it and write a policy that grants access to DynamoDB before attaching it to said IAM Role.

Note for Beginners

People toying around with AWS through the Management Console may be regularly utilizing the Root User Email to login to the console, however, this is not accepted as a good practice due to various security vulnerabilities. Instead, it is considered better to create an IAM User, assign an AWS Managed Policy called `AdministratorAccess` to it, and login through the credentials of said IAM User credentials.

IAM Roles when using AD

Active Directory (AD) is a directory service developed by Microsoft for Windows domain networks. AD enables organizations to enforce security policies, manage access to resources, and maintain a hierarchical structure of the network in a centralized way. They perform the tasks of making sure each person is who they claim to be (Authentication), usually by checking the user ID and password they enter, and allow them to access only the data they're allowed to use (Authorization). The two methods available to us if we wish to use IAM roles alongside AD are as follows: **IAM Roles** and **AWS AD Connector**.

A keen reader might've noticed that both authentication and authorization are tasks usually performed using IAM in an AWS environment. Integrating Active Directory with IAM therefore makes logical sense, and AWS allows us to perform this using IAM Roles, usually done by first granting the roles with necessary permissions and then using a language called SAML (Security Assertion Markup Language) to create a relationship between our Active Directory service and desired IAM Roles.

Another way of using AD and AWS together is through the **AWS AD Connector,** a rather straightforward service that does exactly what it sounds like, it acts as a proxy between the AD service and AWS, connecting them without the need to replicate the AD infrastructure within our AWS environment.

Multi-Factor Authentication

Multi-factor Authentication (MFA) is a widely used method of authentication where users are required to provide two or more types of verification factors to gain access to a system, application or account. This is usually done in order to make it harder for unauthorized individuals to access sensitive resources, even if they have compromised the user's password.

Now, IAM, which is responsible for handling the signing in of accounts to the AWS Console and granting access to AWS services, provides us with out-of-the-box MFA integration as one its key features, allowing us to add an additional layer of security to our AWS account in the form of an One-Time-Password (OTP) from a mobile app, a hardware token, or an SMS.

In fact, not utilizing MFA when using IAM credentials as the primary method of accessing the AWS Management Console or CLI is generally frowned upon, with AWS designating Multi-Factor Authentication as an **IAM Best Practice**.

It is accepted as an IAM best practice for system administrators to utilize MFA (Multi-Factor Authentication) for more secure access to AWS Services.

KMS and Multi-Region Keys

We discussed secrets and AWS Secrets Manager in a previous section, but one of the types of secrets that the mentioned service is not suited for are encryption keys, the cryptographic signatures used to secure stored data.

AWS Key Management Service (KMS) is a vital service for managing cryptographic keys and controlling their use across a wide range of AWS services and applications. In AWS, said keys are usually utilized in order to encrypt/decrypt data either at rest, or during transmission from one service to another. KMS provides us with centralized management of encryption keys, enhancing data security both at rest and in transit. A service that is quite well-integrated into the AWS ecosystem, KMS ensures seamless encryption and decryption across a vast array of AWS services, facilitating strong security measures without adding operational complexity.

KMS supports several types of keys, including:

- **Customer Managed Keys (CMKs)**: Created and managed by the user, offering full control over permissions and lifecycle.
- **AWS Managed Keys**: Automatically created and managed by AWS services such as S3, RDS, and EBS, providing ease of use without manual key management.
- **AWS Owned Keys**: Managed entirely by AWS and used across multiple AWS accounts, offering a simplified, cost-effective solution for less sensitive data.

Also, when utilizing KMS for info stored across AWS regions, we must use the **Multi-Region KMS key**, a variant of the normal KMS key which is designed specifically for dealing with data that spans multiple regions and availability zones.

Certificate Authority (CA)

A **Certificate Authority (CA)** is a trusted entity that issues digital certificates used to verify the identity of individuals, organizations, and servers, either over the internet (i.e., public certificates like SSL/TLS) or within private networks (i.e., private certificates). These certificates are essential for establishing secure communication by encrypting data and confirming the authenticity of the communicating parties. There are two services in AWS primarily used when dealing the type of digital certificates mentioned above, **AWS Private Certificate Authority (AWS Private CA)** and **AWS Certificate Manager (ACM)**. Both services are discussed briefly below:

AWS Private Certificate Authority (AWS Private CA) is a proprietary service offered by AWS that helps manage and issue digital certificates within a private AWS environment. It plays a crucial role in securing communications between AWS resources, ensuring encrypted data transfer, and verifying the identities of the parties involved in internal transactions.

Now, while AWS Private CA primarily focuses on issuing and managing private certificates for AWS resources, they are not the dominant type of digital certificates, and we may face situations where might need to integrate external certificates (e.g., SSL/TLS certificates issued by third-party providers) with our pre-existing AWS resources.

AWS allows us to import these external certificates via **AWS Certificate Manager (ACM)** rather than AWS Private CA. However, these imported certificates have limitations. For example, while AWS-issued certificates can be automatically renewed, external certificates imported into AWS must be manually rotated, and requiring more effort to maintain in general.

There are ways though to make the process of handling external certificates in ACM slightly more convenient, for example AWS Config has a managed rule named `acm-certificate-expiration-check` which can automatically check for expiring certificates, reminding us to rotate the certificate manually whenever the expiration date of the certificate approaches near.

AWS CloudHSM

A **Hardware Security Module (HSM)** is a physical device that provides a secure environment for generating, storing, and managing cryptographic keys. Think of it as a hardware-based counterpart of AWS KMS, though it operates on a more isolated and secure level. HSMs safeguard sensitive information, such as private keys, by isolating them within tamper-resistant hardware, providing robust physical security.

By adding a physical component to the security model, HSMs ensure the integrity of cryptographic operations and help organizations meet stringent security and compliance standards, particularly important for highly regulated industries like finance, government, and healthcare, where data security is paramount.

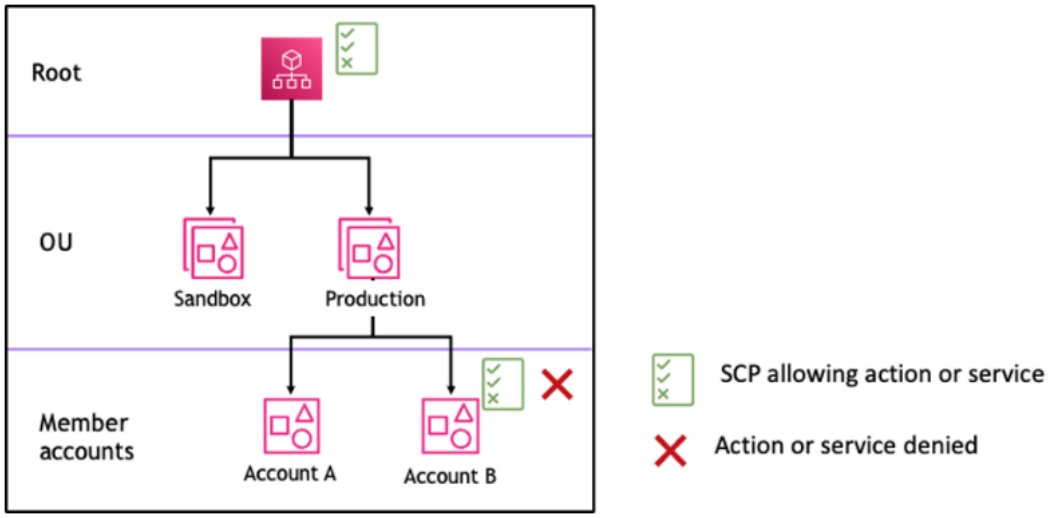
AWS CloudHSM extends the same physical security component to the cloud, allowing organizations to generate and use encryption keys within a dedicated, highly secure hardware device. It gives users complete control over their keys, from creation to management and cryptographic operations, a level of control not provided by KMS, Secrets Manager or other AWS services designed to handle sensitive data.

Now, while AWS CloudHSM can be integrated with other AWS services, it is best understood as an isolated offering, independent from most other AWS services. This is because the primary function of CloudHSM is to secure cryptographic keys and perform cryptographic operations, a function which is jeopardized by unnecessary connections and external dependencies that could introduce vulnerabilities. For example, integrating CloudHSM with a service like AWS CloudTrail (which logs and monitors AWS API calls) might expose unnecessary details about sensitive cryptographic operations to unwanted sources.

Service Control Policies (SCPs)

We have already discussed AWS Organizations and how we can use it to manage a multi-account environment, but there may be situations where enforcing governance and compliance standards across all accounts is critical. This is typically achieved using **Service Control Policies (SCPs)**, which allow administrators to define fine-grained permission boundaries at the organizational level.

SCPs in AWS can be attached to Organizational Units (OUs), individual accounts, or even the root of the organization. They ensure that the member accounts under the defined structure cannot exceed the permissions set by the SCPs, providing centralized control over what actions and services can and cannot be used within each account. It may therefore be thought of as being used to set a kind of upper limit for the entities attached to it, overriding any and all permission policies that go against it. A diagram that illustrates an organization hierarchy where an entity on each level has an SCP associated with it is shown below:



Note again that the purpose of SCPs is to define permission boundaries and not to grant permissions to the constituent identities of an organization. To grant permissions to the accounts within an organization, identity and/or role based policies like the ones discussed in the IAM section should be used instead. SCPs are built not to provide but rather to limit an organization's permission boundary, it is meant for setting guardrails and creating hard red lines for an organization making sure that unauthorized actions do not take place.

AWS Shield

AWS Shield is a managed Distributed Denial of Service (DDoS) protection service that safeguards applications running on AWS, and it can be applied to critical AWS resources such as load balancers, CloudFront distributions, and Route 53. Now AWS Shield comes in two flavors, both of which are pretty self-explanatory: **AWS Shield Standard** and **AWS Shield Advanced**.

AWS Shield Standard, which is automatically included at no extra cost for all AWS customers, provides protection against common, smaller-scale network and transport layer DDoS attacks, ensuring continuous availability of your applications.

AWS Shield Advanced on the other hand, is designed for DDoS attacks of a greater sophistication and larger scale and offers more enhanced features compared to AWS Shield Standard, including more advanced attack detection, near-real-time attack visibility, and tailored mitigations to reduce application downtime. Additionally, opting into the service also provides us with **24/7 access to the AWS DDoS Response Team (DRT)**, a team of DDoS and AWS experts who can engage in real time to assist with threat mitigation and offer detailed post-attack diagnostics.

TLDR;

AWS Shield provides us with protection against Distributed Denial of Service (DDoS) attacks.

AWS WAF

Short for **Web Application Firewall**, AWS WAF helps protect web applications against malicious actors by providing us with a firewall that filters and monitors HTTP requests, allowing us to block **SQL injection**, **cross-site scripting (XSS)**, and other common web exploits. It can also be used to restrict access to our website by IP addresses, geographic locations, or other criteria, making it an essential layer of defense for our web resources.

Note that AWS WAF operates on the application layer of the OSI model, and is a different service from AWS Network Firewall (discussed in a later section) which operates primarily on the network layer (consult the Background on Networking if a refresher on the OSI layer model is needed).

Additionally, AWS WAF also has an easy-to-configure feature called **rate-based limiting**, which detects source IP addresses that make large numbers of HTTP requests within a 5-minute time span and automatically blocks requests from the offending source IP until the rate of requests falls below a set threshold. This can help us mitigate against DDoS style attacks on the application layer, an area not covered by services like AWS Shield, which focus primarily on network and transportation layer protection.

Now, AWS WAF is an account-specific service and is limited to being managed by a single account though some of the service's shortcomings can be circumvented using **AWS Firewall Manager**, a service which allows us to manage firewalls across multiple AWS accounts, allowing us to create and enforce WAF policies across a multi-account AWS environment.

GuardDuty

Amazon GuardDuty is a managed, continuous **threat detection service** designed to help you identify malicious or unauthorized activity in your AWS accounts, workloads, and services. GuardDuty exists to address the growing need for proactive and automated security monitoring across cloud environments, where manual threat detection can be both time-consuming and ineffective against increasingly sophisticated attacks.

The service leverages machine learning, anomaly detection, and integrated threat intelligence from sources like AWS Security Hub, AWS Web Application Firewall (WAF), and global threat databases to detect a wide range of security issues. These include unusual API calls, unauthorized access attempts, suspicious network traffic patterns, and potential data exfiltration.

Basically, GuardDuty is designed to reduce the complexity of threat detection by providing organizations with real-time security findings without the need to manage or deploy additional infrastructure.

Additionally, GuardDuty can also be used to enable fast and/or automated responses to potential threats, with integration support for event-driven workflows and initiating lambda functions for the purposes of automated remediation or prevention of the detected threads.

AWS Network Firewall

AWS Network Firewall is a managed service that allows us to deploy network firewalls to the VPCs within our AWS environment, providing them with scalable network protection. It enables **traffic inspection and filtering** at the network and transport level of the OSI model using customizable rulesets, with each rule relating to one of the following five factors:

- **IP addresses** (for example, block all traffic from a specified IP range)
- **Protocols** (for example, allow only HTTPS traffic through the firewall)
- **Port** (for example, block all incoming traffic from non-standard ports)
- **Domain name** (for example, block access to all requests from a specific domain name)
- **Pattern-matching** (for example, block requests with malicious patterns in its payload)

Note also that the Network Firewall service integrates seamless with AWS Firewall Manager, enabling us to centrally apply the firewall policies and rulesets across both multiple VPCs and multiple accounts. A similar arrangement was also discussed in a previous section on AWS WAF.

MACHINE LEARNING
IN
AWS

Machine Learning: Cheat Sheet

The shortest chapter in the book, this section seeks to familiarize the reader with some services within the still expanding list of Artificial Intelligence and Machine Learning related services that are being put out by AWS.

Now, Amazon also has a rather expansive set of services designed for various Artificial Intelligence and Machine Learning tasks. A rather recent addition to the AWS Services arsenal, these services are, frankly speaking, **not** my area of expertise. Nonetheless, an attempt has been made at explaining the ML services particularly those that I believe to be of the highest import. As I see it, most machine learning services boil down to the same task: **Prediction**. The major difference between most services being in the output and the input i.e. what the service is going to predict, and what information is that prediction based off of.

While a more detailed look at some of the ML services will be done in the future sections of this chapter, this section should be seen simply as a collection of extremely brief overviews on some of the more popular Machine Learning services, a sort-of “Cheat sheet” for AWS Machine Learning Services, if the reader will:-

Amazon Rekognition

An image recognition tool developed and made readily available by Amazon themselves. Can find objects, people, text, scenes in images and videos using Machine Learning. Use cases: Labelling, Content moderation, text detection, face detection and analysis, face search and verification, celebrity recognition and pathing.

Amazon Transcribe

Used to transcibe/turn speech into text using machine learning. Has a feature called redaction which can automatically remove PII (Personally Identifiable Information from the audio recordings).

Amazon Polly

Used to turn text into human-like speech using machine learning, a TTS (Text-To-Speech) service developed and made readily available by Amazon.

Amazon Translate

Used to translate text from one language to another, allows you to localize content - such as websites and applications - for international users, and easily translate large volumes of text efficiently.

Amazon Lex

The same technology that powers Alexa, and is used to create conversational AI chatbots. Uses ASR (Automatic Speech Recognition) to convert speech to text.

Amazon Connect

Used to create virtual call centers, i.e. services that can recieve calls, create contact flows, etc. Can integreate with other CRM Systems or AWS.
Amazon ComprehendA serverless service for Natural Language Processing (NLP) tasks such as sentiment analysis, text classification/grouping, etc.

Amazon Comprehend Medical

Amazon Comprehend Medical detects and returns useful information in unstructured clinical text. Uses NLP to detect Protected Health Information (PHI). From an architecture POV, you could store your medical documents in S3, stream

real-time medical data through Kinesis Data Firehose or use Amazon Transcribe to transcibe patient narratives into text and then use Amazon Comprehend Medical for analyzing it.

Amazon SageMaker

Fully managed service for developers/data scienctists to build ML models. May be understood as an alternative to Jupyter Notebooks that utilize Amazon's infrastructure for the training and operation of their models instead of the system it is being run on.

Amazon Forecast

Fully managed service to deliver highly accurate forecasts using time-series forecasting.

Amazon Kendra

Fully managed document search service powered using Machine Learning, can be used to say...Extract answers from within a document (text, pdf, HTML, ppt, etc). It gives users the ability to integrate documents with natural language search capabilities. Additionally, it learns from user interactions/feedbacks to promote preferred results and also has the ability to manually fine-tune search results.

Amazon Personalize

Fully-Managed ML-Service to build apps with real-time personalized reccomendations using the same reccomendation algorithms that is used for the suggestion of products to customers on Amazon.com

Amazon Textract

Automatically extracts text, handwriting and data from any scanned documents using AI and ML.

Macie

Amazon Macie is a security service that uses machine learning and pattern matching to discover, classify, and protect sensitive data stored in AWS. It automatically identifies and alerts you to sensitive data such as personally identifiable information (PII), helping you meet compliance requirements and reduce the risk of data breaches.

It continuously monitors our Amazon S3 buckets for anomalies and unauthorized access, providing detailed dashboards and alerts to help you quickly respond to potential security threats. By enhancing visibility into the security of our data, Macie plays a crucial role in maintaining the privacy and integrity of sensitive information.

TLDR;

Amazon Macie is used to detect **PII** (Personally identifiable information).

Rekognition

Amazon Rekognition is a powerful and fully managed image and video analysis service provided by AWS. It utilizes deep learning models to analyze and extract meaningful information from images and videos, enabling developers to incorporate visual recognition capabilities into their applications easily.

Key features of Amazon Rekognition include:

1. **Object and Scene Detection**: Automatically identify objects, scenes, and activities within images and videos.
2. **Facial Analysis**: Recognize, analyze, and compare faces for a variety of use cases, including verification and identification.
3. **Facial Recognition**: Create and manage a database of faces to perform real-time face searches against a repository of known faces.
4. **Moderation**: Automatically moderate images and videos by identifying inappropriate content such as violence, adult content, and suggestive imagery.

Basically, Amazon Rekognition is a AWS proprietary service for image recognition purposes.

Textract and Comprehend Medical

Amazon Textract is a machine learning service that automatically extracts text and data from scanned documents. It can analyze a variety of document types, such as forms, invoices, and receipts, extracting key information like text, tables, and form fields. Textract makes it easier to process and analyze large volumes of document data quickly and accurately, enabling businesses to automate document workflows and extract valuable insights.

Amazon Comprehend Medical is a natural language processing service specifically designed for healthcare and life sciences applications. It can extract medical information from unstructured text, such as doctor's notes, clinical trial reports, and patient records.

Basically,

Textract is used to extract text from pdfs and such, while Comprehend Medical is an Amazon service designed especially to understand prescriptions and diagnosis slips. Also, Comprehend Medical can also be used to identify PHI (Protected Health Information).

CONCLUSION AND MULTI-CLOUD

A Case for Technological Diversity

> **Note:** The following is an opinion piece on the perils of relying solely on a single cloud provider, be it AWS or another entity. It was written out of a concern that readers may limit themselves to an AWS-centric attitude and not fully explore what the world of cloud computing has to offer.

The bulk of this book has been focused on explaining fundamental concepts related to cloud computing and the way in which they relate to AWS and AWS-adjacent services, therefore it might not come as a surprise to the reader that I am quite a big fan of AWS.

But I am a fan of AWS in the same way that an alcoholic might be especially fond of vodka. After all society, in its infinite wisdom, would still call that person an alcoholic and not an vodkaholic.

At my core, I find the convenience of spinning up physical infrastructure and using them without having to buy and setup any hardware myself, an ability which cloud computing provides me with to be worth the faustian bargain of having to pay rent and not completely owning or controlling the systems my applications are dependent on. It is therefore, the cloud that I am addicted to and not AWS.

AWS is simply the cloud provider that I (and evidently, 31% of the industry as a whole) am most fond of, it is my poison of choice. And it need not be yours. In fact, the whole concept that a business should attach itself to a single cloud provider, i.e. have a poison of choice, is dumb.

Just as any smart investor diversifies their portfolio in order to hedge against potential catastrophes and not be over-reliant on any single company/industry, good businessmen too should diversify their technology in order to hedge against potential catastrophes and not be beholden to any single entity.

Not to be a snob, but there is a quote from the 1965 novel Dune that I believe most succinctly describes my attitude on the matter:

`"He who can destroy a thing, has real absolute control over it."`

If a person's business can be completely put to halt by a single power outage in an Amazon data center in California, or be made entirely unprofitable by an update in AWS pricing policy, then that person might own the business on paper, but AWS has the real absolute control over it.

There are three major reasons why I believe adopting multiple cloud providers would be prudent. These are enumerated below:-

- Mitigate chances of complete infrastructure breakdown
- Protecting oneself against changes in policy and pricing
- Developing attachment with proprietary services rather than general technologies

Mitigating Infrastructure Breakdown

The most widely used region within AWS is `us-east-1` , located in the area spanning the eastern coast of the United States, concentrated particularly around areas in the north of Virginia. A great deal of very large organizations have a history of being foolishly over-reliant on AWS resources in the region, such that an outage in the `us-east-1` region can halt their entire business operation.

On June 13, 2023 for example, Amazon Web Services (AWS) experienced an incident that impacted a number of services in the US-EAST-1 region. The

incident, which lasted more than 2 hours, was first detected around 18:50 UTC, and affected organizations such as **The New York Metropolitan Transportation Authority** and the **Associated Press**. With the MTA even having to send a tweet apologizing to the residents of New York City because of the incident. Also, curiously, Amazon has still yet to publicly disclose the cause of the outage more than a year after the incident.

Though few and far between, these outages of AWS services are not an uncommon occurrence and especially when running a business which cares about minimizing down-time, it would be wise not to rely on a single cloud provider.

Additionally, though remembering the durability and up-time guarantees given to us by AWS with regards to a particular service is necessary, blindly putting one's trust in them is another matter entirely. For example, the S3 service guarantees us with 99.99% uptime, and for the most part, it does deliver on the promise. But AWS is not expecting it to be available 99.99% of the time, in fact, Amazon expects failures to happen and considers it as part of the process. It just hopes that its customers are smart enough to plan for contingencies, with AWS giving credits that you can use within AWS as compensation for whenever "shit hits the fan" as they say in more cultured parts of the world.

Don't believe me? This was a slide prominently displayed at the AWS re:Invent conference in 2013:

Whenever we create an EC2 instance or spin up a S3 bucket we enter into a Serivce-Level Agreement (SLA) with the Amazon corporation, agreeing to accepting compensation for any downtime incurred due to faults or failures caused by AWS in the form of AWS credits, usually according to the structure laid below:

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.99% but equal to or greater than 99.0%	10% (of the estimated monthly bill)
Less than 99.0% but equal to or greater than 95.0%	30%
Less than 95.0%	100%

The truth of the matter is, AWS is rather component in handling its infrastructure and making sure its operating smoothly but in the rare cases that somehow it is not able to, AWS is also large enough to be able to pay the costs outlined in its SLA.

What AWS is not however, is an entity that will give you 24/7 uptime all day, every day. **Expect things to fail.** And prepare for contingencies. These contingencies usually offer themselves to us in one of two forms, both of which are listed and explored below:

- Multi-AZ and Multi-Region environments
- Multi-cloud environments

Multi-AZ and Multi-Region

It is definitely possible to assure against possible technological disasters whilst limiting oneself to AWS, with the most obvious manner of doing so being the spreading of AWS resources across multiple availability zones and/or multiple regions. This helps ensure high availability and is quite easy to set up. It requires learning no new cloud technologies and if planned properly, very little added cost.

Outages happening at the same time across multiple AWS regions is basically unheard of, especially for regions that are not adjacent to one another geographically. Therefore this is definitely a valid choice for enterprises that wish

to make their existing AWS infrastructure more resilient regardless of their stance on utilizing multiple cloud providers.

Multi-Cloud

If an organization is scared that its infrastructure is over-reliant on a single cloud provider, and wishes to mitigate against infrastructure breakdown, then the logical solution would be to spread the infrastructure and workload across multiple cloud providers. Indeed, according to a survey of over 1500 corporations across the world done by the Oracle corporation, **98 percent** of all business enterprises rely on more than a single cloud provider for the purposes of infrastructure provisioning and management.

This normally means keeping accounts with multiple cloud providers and handling operations across multiple cloud platforms, which does involve a whole lot more work than if an organization were relying solely on a single cloud provider. A popular method of going about this is through the utilization of **Terraform** and **Ansible**, which are both IaC (Infrastructure as Code) tools that allow us to provision and configure infrastructure in the cloud through the use of scripts that are **cloud-agnostic**. Meaning that the same programming language and code structure (sometimes even the same exact lines of code) can be used for AWS, Azure, GCP and 60+ other infrastructure providers. Both tools are liberally used by cloud engineers and DevOps teams alike.

Protection against Policy and Pricing Changes

Another major reason why not to rely on a single cloud provider would be to hedge against potential changes in (A) How a service operates and (B) How a service is priced. For example, if you are running a company that heavily utilizes Amazon's S3, then any change in how S3 is priced or operated could affect you in greatly outsized manners.

This is exactly what Dropbox, one of the largest digital storage companies in the world, realized in 2015 when it decided to move away from AWS (and it never

looked back). This allowed Dropbox to avoid paying the extremely high costs that had recently become customary when dealing with large volumes of data in S3, reportedly saving the company upwards of $75 Million over the next two years.

However, one is much less susceptible to being greatly affected by changes in policy and pricing if their business is technologically diversified and relying on a greater number of cloud providers, each with its own set of policy and pricing standards, greatly reducing the chance of a single change in policy or pricing leading to outsized impact on the business.

Proprietary Services and General Technologies

The final major reason why I believe it to be important not to rely on a single cloud provider is to avoid excessive attachment, comfort and limited expertise with a specific proprietary service rather than a general type of technology leading to a type of cloud administrator who say, realizes how to use S3 but will never bother learning what object storage is and will never explore other cloud providers and object storage services, limiting themselves to an AWS-centric world.

That is not the type of cloud user I wish for the reader to become, after all the shackles of habit are too weak to be felt until they are too strong to be broken. AWS should simply be treated as the on-boarding point for the reader's cloud journey. It provides you with the largest and most popular cloud platform in the world, but it is not however to be conflated with the only cloud platform in the world.

Therefore, in order to aid the reader in broadening their horizon, small tables of the most popular AWS services discussed in the book, the general technology they represent, and their alternatives is provided to the reader, so that they can take the AWS-centric knowledge that they have gained in this book and use it as a jumping off point for more broader cloud endeavors.

Storage Services

AWS Service	General Technology	Azure alternative	GCP alternative	3rd party alternative
Simple Storage Service	Object Storage	Azure Blob Storage	Google Cloud Storage	Storj
Elastic File System	File Storage	Azure Files	Google Filestore	OpenStack Manilla
Elastic Block Store	Block Storage	Azure Disk Storage	Google Persistent Disk	DigitalOcean Volumes
AWS DataSync	Data Transfer	Azure Data Box	Storage Transfer Service	Resilio Connect
Snowball and Snow family	Data Transfer + Edge	Azure Data Box	GCP Transfer Appliance	N/A

Compute Services

AWS Service	General Technology	Azure alternative	GCP alternative	3rd party alternative
EC2	Virtual Machines	Azure Virtual Machines	Google Compute Engine	DigitalOcean Droplets
ECS	Containers	Azure Containers	Google Kubernetes Engine	Docker Swarm
Lambda	FaaS	Azure Functions	Google Cloud Functions	Cloudflare Workers

Database Services

AWS Service	General Technology	Azure alternative	GCP alternative	3rd party alternative
RDS	Managed SQL DB	Azure SQL Databases	Cloud SQL	SQL on Oracle Cloud
DynamoDB	NoSQL DB	CosmosDB	Firestore	MongoDB
Redshift	Data Warehouse	Azure Synapse	BigQuery	Snowflake

Networking Services

AWS Service	General Technology	Azure alternative	GCP alternative	3rd party alternative
CloudFront	Content Delivery Network (CDN)	Azure CDN	Google Cloud CDN	Cloudflare
Elastic Load Balancer	Load Balancers	Azure Load Balancer	Google Cloud Load Balancing	Nginx
API Gateway	API Management	Azure API Management	Google Cloud Endpoints	Kong
Route53	DNS Provider	Azure DNS	Google Cloud DNS	Oracle DNS

Application Integration Services

AWS Service	General Technology	Azure alternative	GCP alternative	3rd party alternative
SQS	Message Queue	Azure Queue Service	Google Pub/Sub	RabbitMQ
SNS	Notification Service	Azure Notification Hub	Google Pub/Sub	Twilio
EventBridge	Event Bus	Azure EventGrid	Google EventArc	Zapier
SES	Email Service	Azure Communication Services	N/A	SendGrid, Mailgun
Kinesis	Real-time Data Streaming	Azure Event Hubs	Google Cloud Dataflow	Apache Kafka
Quicksight	Data Analytics	Power BI	Google Data Studio	Tableau

Conclusion

The final section of the book, this section is meant to congratulate the reader on reaching this point.

After all, a great deal of ground has been covered throughout the course of this book, and a great many concepts related to IT and Cloud Computing have been discussed, not to mention a great many AWS services. Thanking the reader for bearing with me throughout the journey, I apologize if I was ever unnecessarily over-indulgent or wasteful regarding the amount of words and sentences used.

I hope that the reader has learnt a great deal through the process and is eager to put their newfound knowledge to work. In this final section, let me suggest two possible ways in which they may go about doing so: **Certifications** and **Projects**.

Certifications: AWS itself has a very well-respected set of certifications, which usually cost between 100-300 dollars USD. The most sought after of those certifications, which indeed I would suggest even the reader to obtain, would be the **AWS Certified Solutions Architect (SAA-CO3)** and/or the **AWS Certified Developer (DVA-CO2)** certificates. Though not necessary, they are a net-positive for working with the cloud in the real world, and their respective exams will test the readers theoretical grasp over both cloud computing and AWS services quite well.

Projects: An obvious choice considering that the ultimate aim of any person working with the cloud would be building and maintaining business solutions. These projects need not be lavish or extravagant, they just need to demonstrate technical competence and some level of expertise. Here, I would like to point readers towards the **Cloud Resume Challenge**, a go-to project for many novices just starting out with the cloud.

I depart and end the book wishing the best to the reader, hope that the time they spent with the book was fruitful and thank them one last time for the support and attention that they afforded me throughout it.

Cheers.