**AWS**

**DAY-1**

In earlier days if any organization needs servers means they used to go to IBM, HP any other company they used to get servres and they were forming datacenter with an organization.

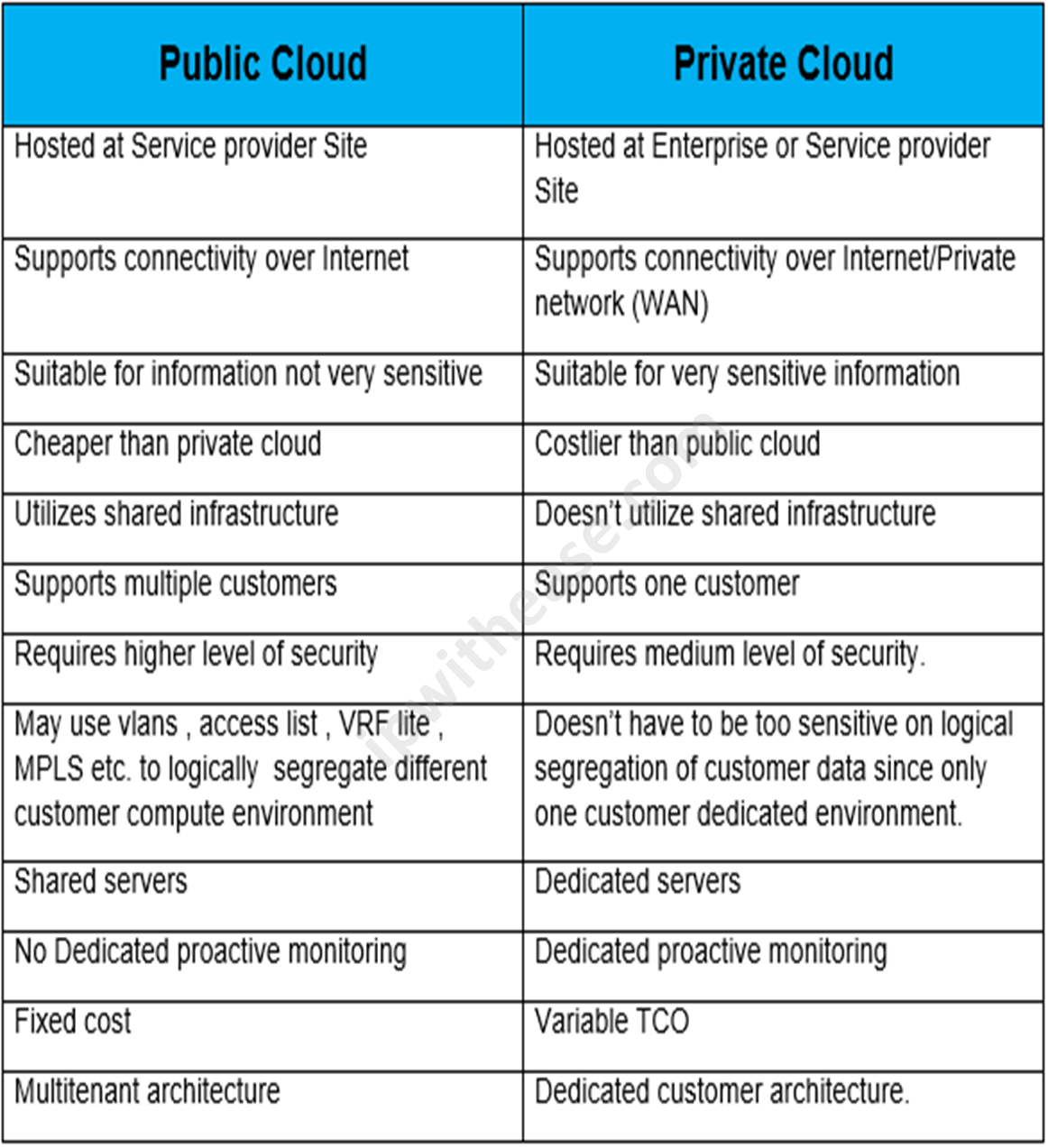
a data center is a physical facility that organizations use to house their critical applications and data. A data center's design is based on a network of computing and storage resources that enable the delivery of shared applications and data. The key components of a data center design include routers, switches, firewalls, storage systems, servers, and application-delivery controllers.

Virtualization creates a simulated, or virtual, computing environment as opposed to a physical environment. Virtualization often includes computer-generated versions of hardware, operating systems, storage devices, and more.

Why virtualization came?(create virtual servers for each application)

On each server if we run one application means it will get wasted other resources so with the help of virtualization we can run multiple applications on same server.

All these setups are doing privately within our organisation so called as private cloud.



If its public cloud means security breach can be there that we can overcome by creating virtual private cloud inside public cloud.

Why public cloud is so popular?

Cost optimization-maintaining data center is too costly keeping power on always, Ac , hacker and all.

It is also pay as you go.

How AWS is better that others?

1.First-Mover Advantage

2. Extensive service portfolio(AWS offers 200 fully featured services ranging from computing, storage and database to Machine Learning.

3. Global infrastructure

AWS has largest and most well distributed global infrstructure. It operates in 31 geographic regions.

4. Scalability and flexibility(AWS is desired to scale with businesses of all sizes from startups to enterprise.

5. Reliability and security(AWS is known for its robust security measures, compliances, certifications and reliability.

6. Cost optimization(pay as you go)

7.Innovation(Introducing new services)

8. Developer friendly(with a rich set of API, SDKs)

Are people going back from cloud?

Yes but very few due to security issues, cost optimization.

**DAY-2**

IAM:(Identity access Management)

It’s a service that allows you to securely control access to AWS resources. With IAM you can manage users, groups, roles and their permissions to ensure your resources are protected.

Core concepts:

1. Users:

Individual accounts with credentials to access AWS services.

1. Groups: (like dev, QA,prod)

Collections of users with common permissions.

1. Roles:  
   Used to grant temporary access to AWS services

(If you want to access any application that is in private cloud we should give temporary access i.e. roles)

1. Policies:

JSON document that define permissions for users, group and roles.

1. IAM identity center(SSO):

Allows signle sign on access to AWS accounts.

Authentication and authorization are both security processes that work together to ensure the security of applications, systems, and data:

* **Authentication**

Verifies a user's identity. For example, logging in to Facebook with a username and password or using TouchID on a phone.

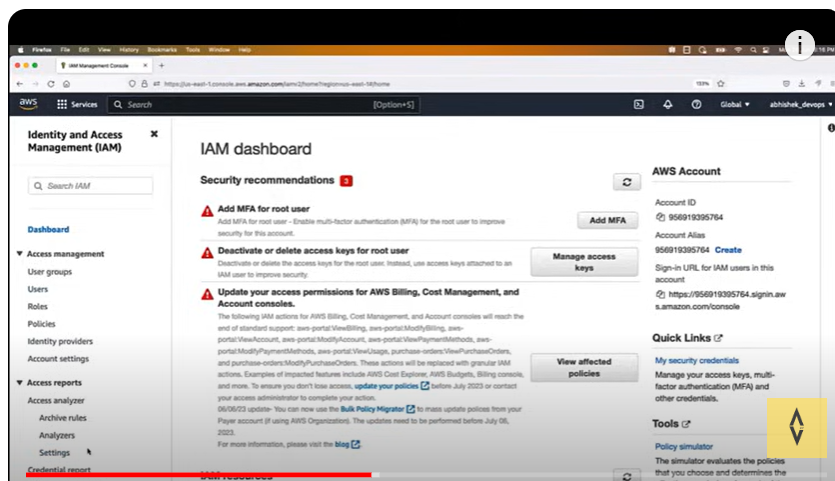
* **Authorization**

Determines a user's level of access and grants access based on that level. For example, using a card to enter a room and access specific services

IAM practice:

First login to AWS account with root access

Go to IAM



Create user

A screenshot of a computer

Description automatically generated

A screenshot of a computer

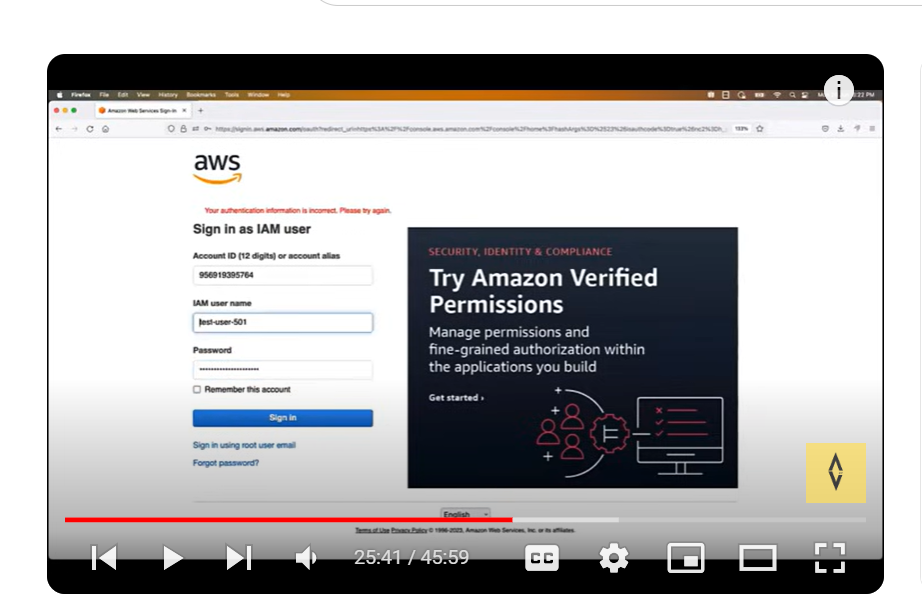
Description automatically generated A screenshot of a computer

Description automatically generated

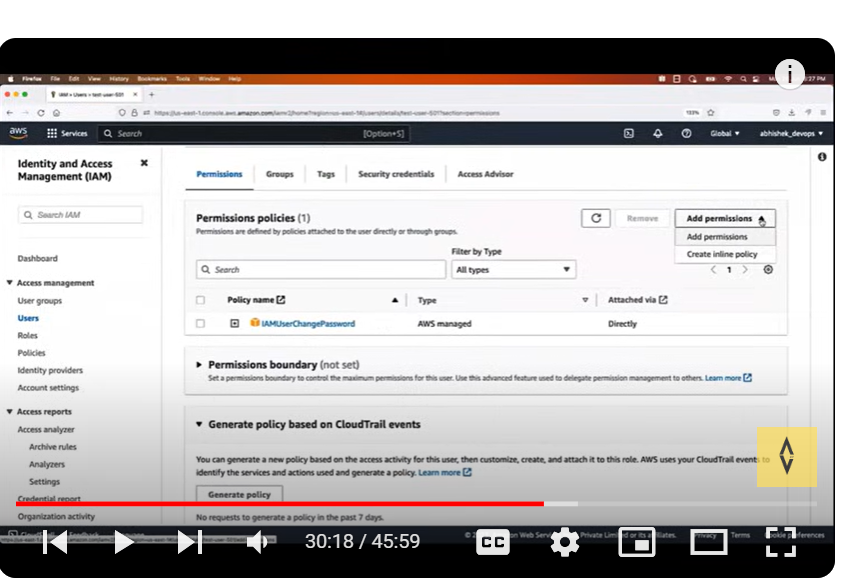
A screenshot of a computer

Description automatically generated

Dowload CSV file so that you can send for your reference

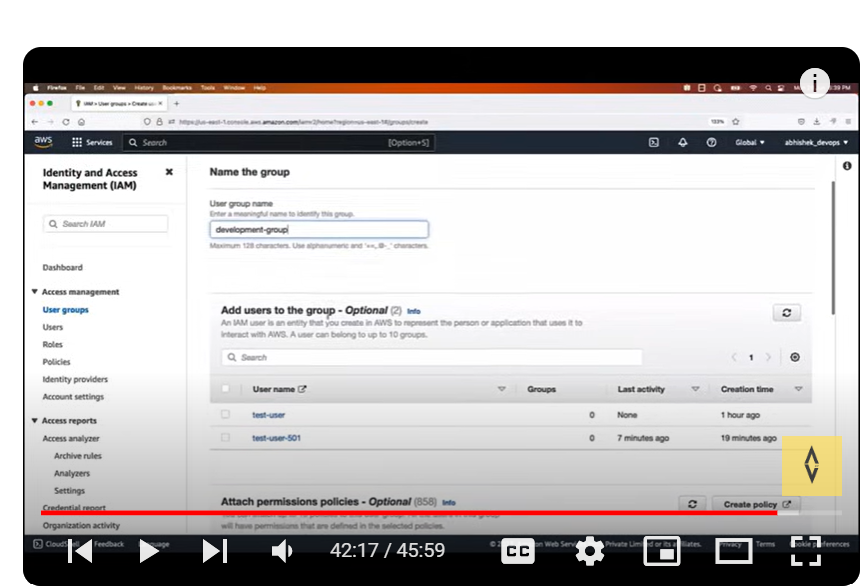


As a devops engineer I gave authentication but not authorization for that go back to Rakesh account via root login

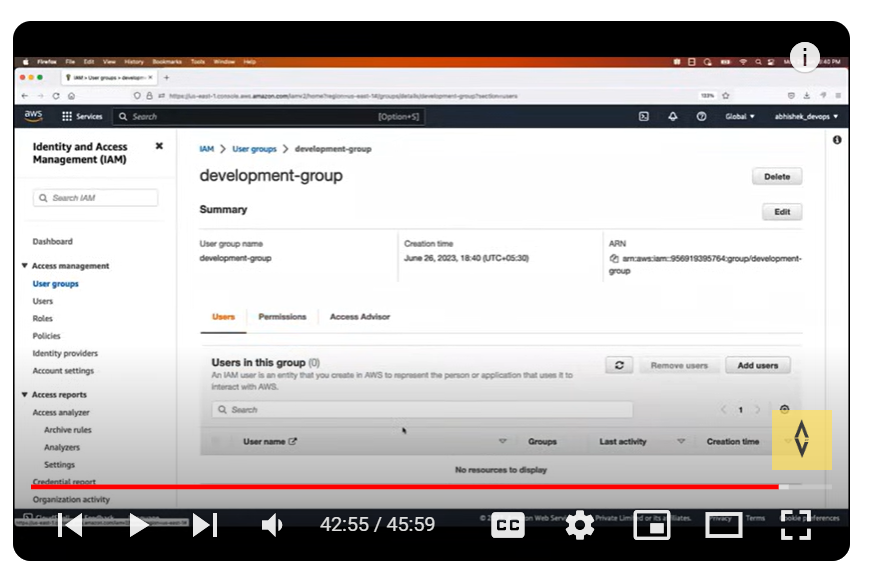


It is always good practice to use IAM user instead of root user .in interview you should tell that they provided some access to devops group with limited roles so based on that we are creating it.

To create groups sign in root user



We can add users later as well and give permission down there.



We can add same policy to whole development group instead of each user.

Day-3

**EC2-Elastic cloud compute**

Compute means you are asking AWS to provide cpu, ram, disk virtual server.

If that service can be scaled up or scaled down based on application needs that service named as elastic.

(EC2-you are asking AWS cloud to create a virtual server with elastic feature )

To check Timely upgrade, security issue, server up or not all these cannot be done by one devops engineer.

EC2 providing scalability, cost, maintenance.

**EC2 types:**

General purpose(Usually we use general purpose ec2 instance)

General Purpose instances are designed to deliver a balance of compute, memory, and network resources. They are suitable for a wide range of applications, including web servers,

small databases, development and test environments, and more.

Compute Optimized

Compute Optimized instances provide a higher ratio of compute power to memory. They excel in workloads that require high-performance processing such as batch processing,

scientific modeling, gaming servers, and high-performance web servers.

Memory Optimized

Memory Optimized instances are designed to handle memory-intensive workloads. They are suitable for applications that require large amounts of memory, such as in-memory databases,

real-time big data analytics, and high-performance computing.

Storage Optimized

Storage Optimized instances are optimized for applications that require high, sequential read and write access to large datasets.

They are ideal for tasks like data warehousing, log processing, and distributed file systems.

Accelerated computing

Accelerated Computing Instances typically come with one or more types of accelerators, such as Graphics Processing Units (GPUs),

Field Programmable Gate Arrays (FPGAs), or custom Application Specific Integrated Circuits (ASICs).

These accelerators offload computationally intensive tasks from the main CPU, enabling faster and more efficient processing for specific workloads.

Latency:

Time taken to reach an application and giving response back to application.

You should create an EC2 instance based on application demand which region they need .

Regions: like Mumbai, us-east-1

Availability zones:

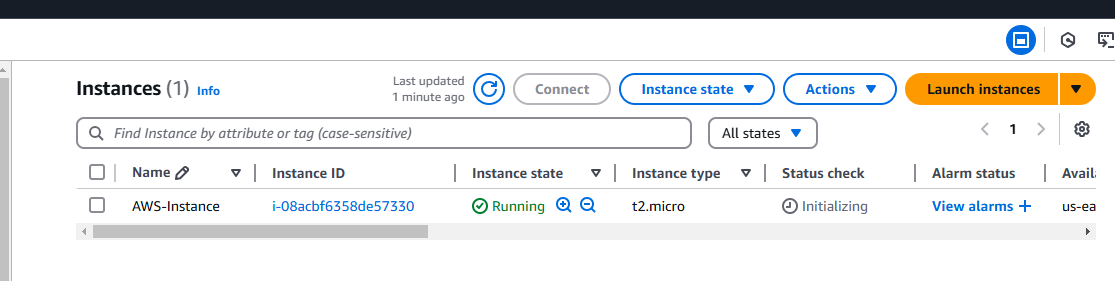
We have availability zones within a region

For example, us-east-1a, us-east-1b

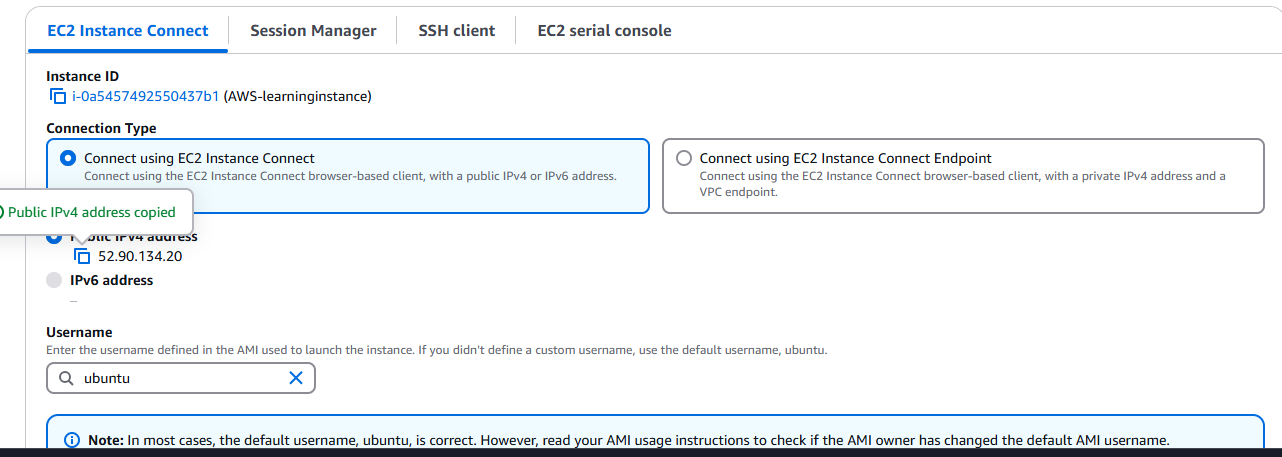
So any region goes down other availability zone works fine

Key-pair-combination of public and private key

EC2 instance creation

Create an EC2 instance with Ubuntu os

Created Ec2 instance login user as Ubuntu

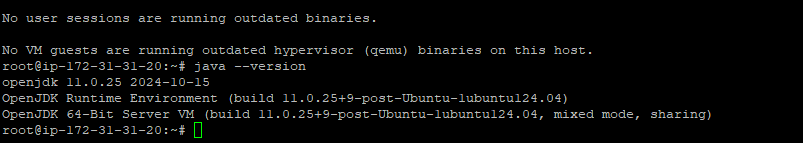


Go to root user sudo su –

Apt update

Apt install openjdk-11-jdk

Once its done we can check java version



Copy weekly release from Jenkins.io website

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

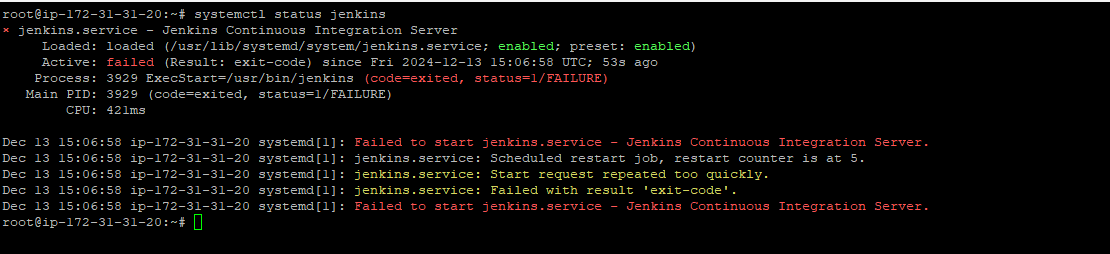
https://pkg.jenkins.io/debian/jenkins.io-2023.key

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \

https://pkg.jenkins.io/debian binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install jenkins

DAY-4

VPC(Virtual private cloud)

As per the availability of your application you will ask to create vpc cloud with specific range of IP address that IP addrees will be split to multiple servers as per application request that is called as subnet.

As a security breach internet gateway is present . Once you enter within vpc you will have public ip to access outside then via elastic load balancer you will communicate to your instance path that we communicate to reach instance is route table.

If that instance wants to communicate outside world to update any packages that Ip address we do not share so masking Ip address called as NAT.

Easy Explaination

### What is a VPC? (Simplified Explanation)

A **VPC (Virtual Private Cloud)** is like your **own private network** in the cloud. Imagine you're building a house (your application) in a big city (the cloud). A VPC is like the plot of land where you build the house, complete with fences and gates for security. You control everything inside this plot—who can enter, what’s inside, and how things are connected.

### Key Points to Understand

1. **Isolation**:
   * A VPC is your private space in the cloud. No one else can access it unless you allow them.
2. **Subnets**:
   * Subnets are like dividing your land into smaller sections. For example:
     + **Front yard (Public Subnet):** Accessible by anyone (e.g., a web server).
     + **Backyard (Private Subnet):** Only you can access it (e.g., databases).
3. **Gateways**:
   * **Internet Gateway:** A door that allows some parts of your VPC to connect to the internet.
   * **NAT Gateway:** A special door for private areas that lets you send messages (requests) out but keeps you hidden.
4. **Security**:
   * **Security Groups:** Like locks on your doors, controlling what comes in and goes out.
   * **Network ACLs:** A bigger security wall around your land to keep out unwanted traffic.
5. **Routing**:
   * Think of routing tables as maps for traffic inside and outside your VPC. They define where data should go.

### Real-Life Example: Hosting a Website in a VPC

1. **You create a VPC** with an IP address range, say 10.0.0.0/16.
2. **Divide it into subnets:**
   * A **public subnet** for your website.
   * A **private subnet** for your database.
3. **Set up an internet gateway** to let people access your website.
4. Use **security groups** to:
   * Allow HTTP/HTTPS traffic to your web server.
   * Block all traffic directly to your database (for security).
5. Use a **NAT gateway** so your web server can talk to the database without exposing the database to the internet.

### Why Use a VPC?

* **Privacy**: Your resources (like servers and databases) are isolated.
* **Control**: You decide who can enter or leave your network.
* **Customization**: You can design the network to fit your exact needs.

[Internet]

|

[Internet Gateway]

|

[VPC]

|-------------|-------------|

[Public Subnet] [Private Subnet]

| |

[Web Server] [Database]

\\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

[NAT Gateway]

### What is a VPC?

A **VPC (Virtual Private Cloud)** is a private, isolated network within a cloud environment, such as AWS, Azure, or Google Cloud. It allows you to host resources like servers, databases, and applications in a secure and customizable virtual network.

### Why Use a VPC?

* **Privacy:** Your resources are isolated from others.
* **Customization:** You control the structure of your network.
* **Security:** You can set up firewalls and control access.
* **Scalability:** Easily add or remove resources within your VPC.
* **Hybrid Networking:** Connect your on-premises data center to the cloud.

### Components of a VPC

1. **Subnets**(range of IP address)
   * Divide your VPC into smaller sections:
     + **Public Subnet**: Resources (like web servers) can be accessed via the internet.
     + **Private Subnet**: Resources (like databases) are hidden from the internet.
2. **Internet Gateway (IGW)**:
   * A gateway that allows public subnet resources to access the internet.
3. **NAT Gateway**:
   * Allows private subnet resources to access the internet without being exposed to it.
4. **Route Tables**:
   * Maps that direct traffic within your VPC and to the internet.
5. **Security Groups**:
   * Firewalls at the instance level. They control what traffic is allowed to or from your instances.
6. **Network ACLs (Access Control Lists)**:
   * Optional firewalls at the subnet level for broader traffic control.
7. **VPC Peering**:
   * Allows you to connect two VPCs to share resources.
8. **VPN Gateway**:
   * Securely connects your on-premises network to your VPC over the internet.

### **Security Groups vs. NACL (Network ACL)** - Simplified Explanation

Both **Security Groups** and **NACLs (Network Access Control Lists)** are used to control network traffic in a VPC, but they work differently and serve different purposes. Here's an easy-to-understand comparison:

### **1. Security Groups (SG)**

**Think of Security Groups as "guards at the door of your house."**

* **What it does:** Controls traffic to and from specific resources like EC2 instances.
* **Scope:** Works at the **instance level**.
* **Stateful:** It remembers the connection. If you allow incoming traffic, the response is automatically allowed (no need to configure separately).
* **Direction:**
  + Controls **inbound** (traffic coming in) and **outbound** (traffic going out) separately.
* **Rules:** Only **allow rules**—you cannot explicitly deny traffic.
* **Example Use:**
  + Allow SSH (port 22) access only from your IP.
  + Allow HTTP (port 80) traffic from the internet.

**Example Analogy:** If your EC2 instance is a house, a Security Group acts like a personal bouncer for the house, deciding who can knock on the door or leave.

### **2. Network ACLs (NACL)**

**Think of NACLs as "fences around your neighborhood."**

* **What it does:** Controls traffic to and from subnets within the VPC.
* **Scope:** Works at the **subnet level**.
* **Stateless:** It doesn’t remember connections. You must configure both inbound and outbound rules separately.
* **Direction:**
  + Controls **inbound** and **outbound** traffic separately.
* **Rules:** Can have both **allow rules** and **deny rules**.
* **Example Use:**
  + Block all traffic from a suspicious IP range to an entire subnet.
  + Allow only specific protocols (e.g., HTTP, HTTPS) for subnet resources.

**Example Analogy:** If your VPC is a neighborhood, a NACL acts like a neighborhood gatekeeper, controlling access to all houses in the area.

### **Key Differences**

| **Feature** | **Security Groups (SG)** | **Network ACLs (NACL)** |
| --- | --- | --- |
| **Level** | Instance level (e.g., EC2 instances) | Subnet level (applies to all resources) |
| **State** | Stateful: remembers connections | Stateless: doesn’t remember connections |
| **Rules** | Only allow rules | Both allow and deny rules |
| **Applies To** | Specific resources like instances | All resources in a subnet |
| **Use Case** | Fine-grained control over resource access | Broad control at the subnet level |

### **When to Use What?**

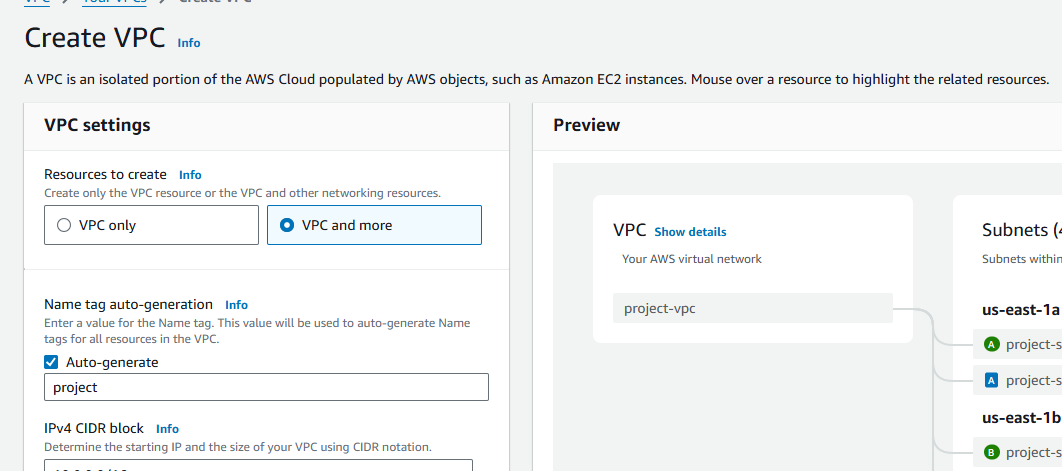
* **Use Security Groups** for:
  + Controlling access to specific instances (e.g., web servers, databases).
  + Setting fine-grained rules for each resource.
* **Use NACLs** for:
  + Broad traffic control at the subnet level.
  + Adding an extra layer of security for the entire subnet.

### **Quick Example**

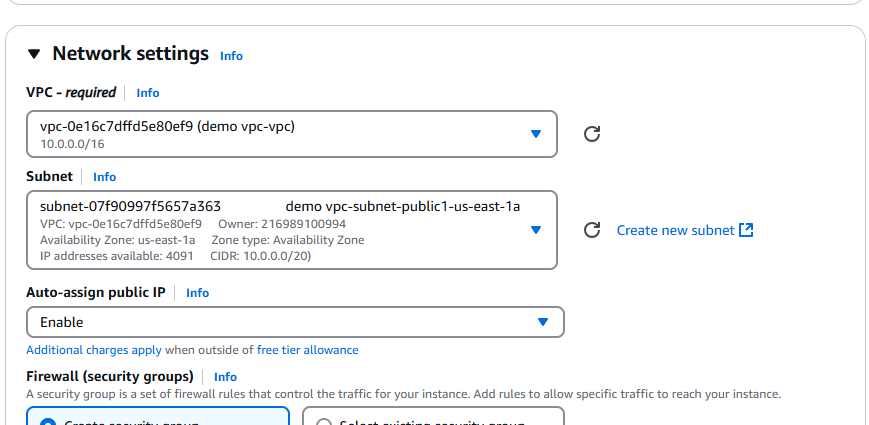
* You have a **public subnet** with a web server and a **private subnet** with a database.
  + **Security Groups:**
    - Allow HTTP (port 80) to the web server.
    - Allow MySQL traffic (port 3306) only from the web server to the database.
  + **NACL:**
    - Block all traffic from an untrusted IP range to the **public subnet**.

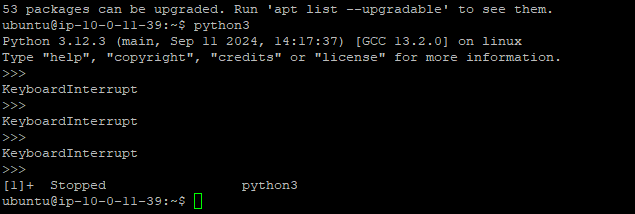
VPC Creation

Create VPC



Other things go with default settings and create vpc.

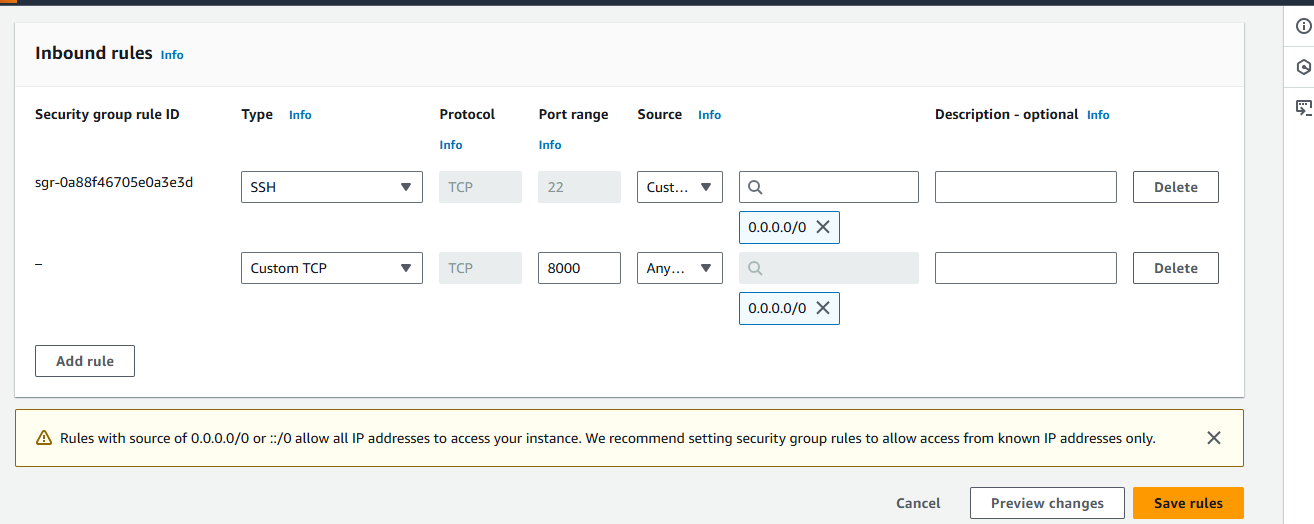
Now create EC2 instance in Ubuntu in network settings add vpc that you created

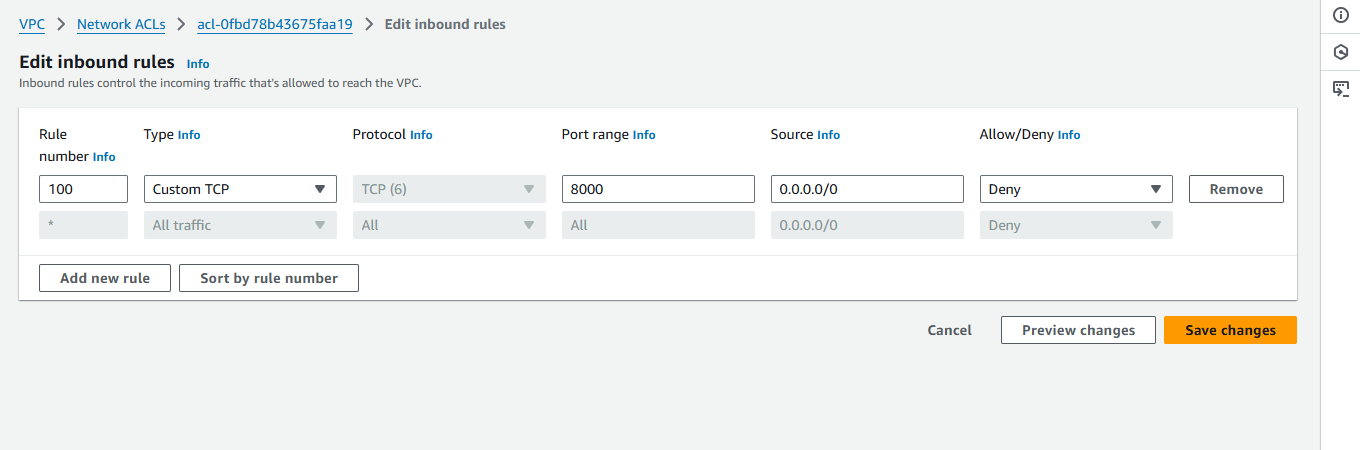
Connect to that Ubuntu instance check python installed or not

python3 -m http.server 8000

To enable security group

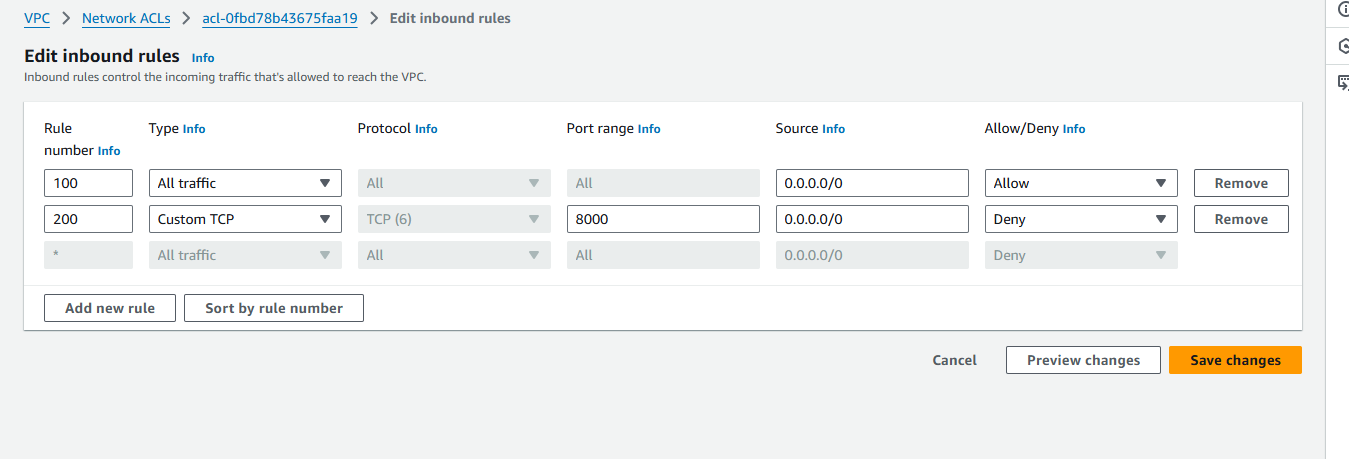
Go to security group –select security group-edit inbound rules



Now go to Nacl edit inbound rules remove it and add new rule

Now you cant’ access it so devops engineer blocked it

Go to network ACL again edit inbound rules



Even though we deny it gives output because it follows first come first serve.

Route 53

IP address can be changed based on your network so we have domain names.

### **Route 53 in Simple Words**

**Amazon Route 53** is a service that helps you connect **domain names** (like www.example.com) to the servers or resources where your website or application is hosted. It works like a **phonebook for the internet**, helping computers find where to go when users type a domain name.

### **How the Internet Works (Simplified)**

When you type a website like www.example.com in your browser:

1. Your browser doesn’t understand the domain name (www.example.com).
2. It needs the **IP address** (like 192.0.2.1) of the server where the website is hosted.
3. Route 53 provides this IP address and connects you to the correct server.

### **Why is it Called Route 53?**

* **Route** refers to routing traffic (directing traffic).
* **53** is the port number used for DNS (Domain Name System) queries.

### **Basic Terms to Understand**

1. **Domain Name**
   * A human-friendly name like www.example.com.
   * Think of it as a **contact name** in your phonebook.
2. **IP Address**
   * A computer-readable address like 192.0.2.1.
   * Think of it as the **phone number** behind the contact name.
3. **DNS (Domain Name System)**
   * It translates the domain name into the IP address.
   * Like looking up a name in the phonebook to find the phone number.

### **Example Use Case: Hosting a Website**

Let’s say you have a website hosted on an **AWS EC2 instance**. You want users to visit the website by typing www.example.com.

Here’s how Route 53 works step-by-step:

1. **Domain Registration**
   * You register the domain name example.com using Route 53.
2. **DNS Records**
   * You create a DNS record (A Record) in Route 53 to connect the domain name example.com to your server’s IP address, like 192.0.2.1.
3. **Routing Traffic**
   * When a user types www.example.com in their browser:
     + Route 53 looks up the IP address for example.com.
     + It sends the user to the EC2 instance where your website is hosted.
4. **Health Checks** (Optional)
   * Route 53 can check if your EC2 instance is healthy. If it fails, traffic can be routed to a backup server.

### **Types of Routing with Examples**

1. **Simple Routing**
   * Traffic goes to a single server.  
     **Example:**
   * example.com → 192.0.2.1 (your web server).
2. **Weighted Routing**
   * Split traffic between multiple servers.  
     **Example:**
   * Send 70% of users to Server A and 30% to Server B.
3. **Latency-Based Routing**
   * Direct users to the server closest to them (fastest response time).  
     **Example:**
   * US users → US Server.
   * Europe users → Europe Server.
4. **Failover Routing**
   * Route traffic to a backup server if the main server goes down.  
     **Example:**
   * Primary server → 192.0.2.1.
   * If unhealthy → Backup server → 192.0.3.1.
5. **Geolocation Routing**
   * Route users based on their location.  
     **Example:**
   * Users in India → India Server.
   * Users in the US → US Server.

### **Visual Example**

1. **You have a domain:** example.com.
2. **Your resources:**
   * **Web server (IP: 192.0.2.1)** in the US.
   * **Backup server (IP: 192.0.3.1)** in Europe.
3. **Route 53 Routing:**
   * If the US server is **healthy**, users are directed there.
   * If the US server is **unhealthy**, users are redirected to the European server.

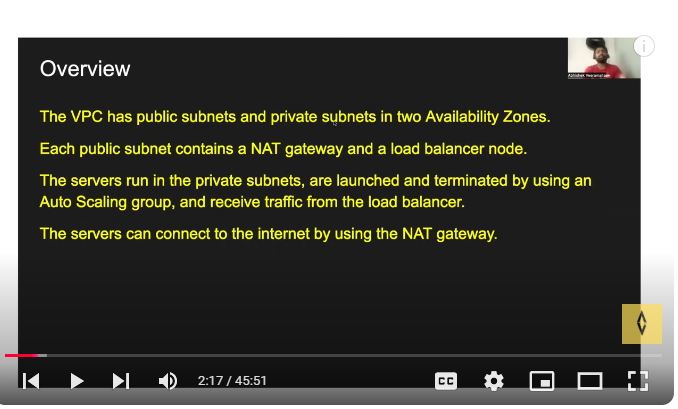
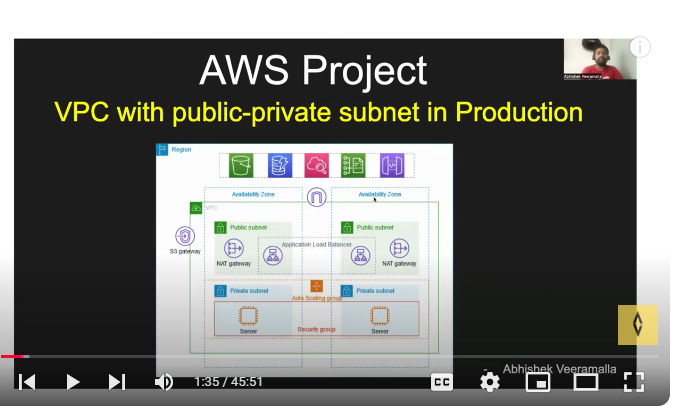
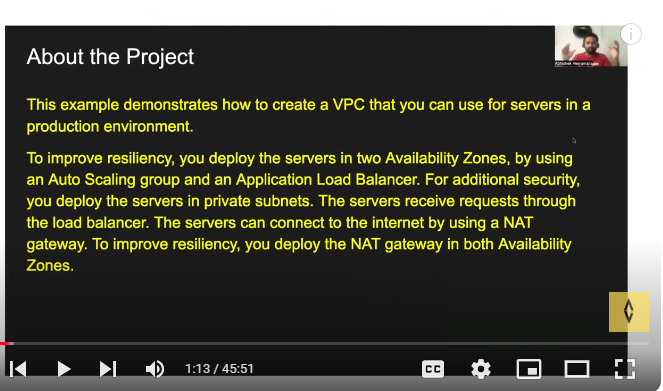
### **Why Use Route 53?**

* **Fast**: Low-latency, global DNS service.
* **Reliable**: Highly available, built on AWS infrastructure.
* **Flexible**: Supports advanced routing (weighted, failover, geolocation, etc.).
* **Integrated**: Works easily with other AWS services.

### **Summary**

Route 53 connects your **domain name** to the right server (IP address), so users can access your website or application. It also helps control traffic routing based on health, location, or other rules.

**AWS project :VPC with public-private subnet in production**

****

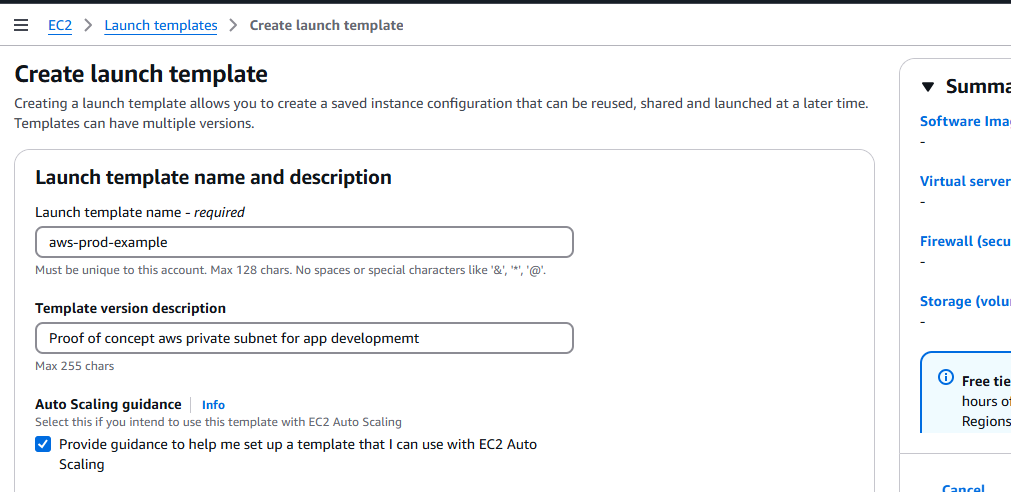
### **What is a Bastion Host?**

A **bastion host** is a **secure server** that acts as a **gateway** to access other servers (like EC2 instances) in a private network. It is used to **safely manage and connect** to resources that are **not directly accessible from the internet**.

Think of it as a **security checkpoint** or a **jump server** that allows you to enter a private area (private servers) in a controlled way.

First we need to create VPC-NAT gateway 1 per AZ

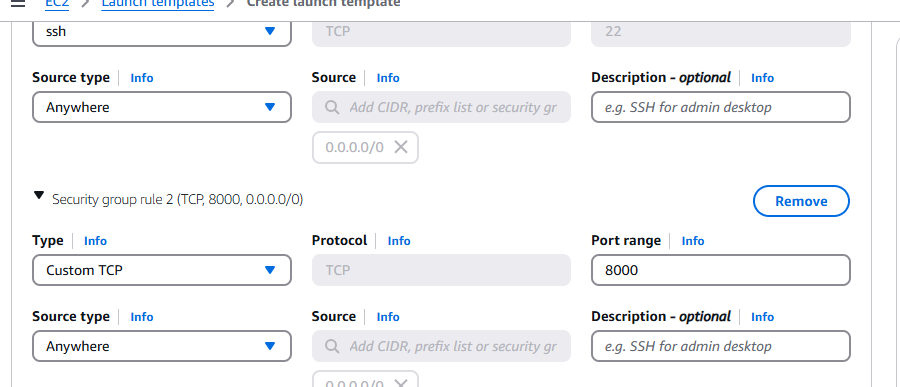
Then create auto scaling group that you will get in Ec2.

Before creating auto scaling check on create launch template. 

Clcik on recently launched

Network setting-create subnet group and select vpc created today for project

Give inbound security groups as



Create Ec2-auto scaling groups-choose launch template. Select vpc that you created availability zones us-east1-a and us-eastb-b private subnet

Create bastion instance in ec2 network settings

Click on network settings and create in same vpc.

Connect to bastion instance via putty.

Project setup

Before we start:

First we need to create VPC as vpc more.

-Autoscaling group:

Ec2- Auto scaling groups

-load balancer

-target group

-Bastion host or jump server

**S3(Simple storage space)**

As a devops engineer S3 bucket name should be unique across all aws accounts so  
app1-feature-prod or dev or qa environment-domain name  
[app1-payments-prod-example.com](http://app1-payments-prod-example.com/)  
anything that you pt to s3 called as object that can be file, video or image.  
The file that we uploaded can be universaly accessible even though we lost that file from local we can get from s3 bucket.  
S3 advantages:  
availability & durability, scalability, security, cost effective, performance.  
Multipart upload in Amazon Simple Storage Service (S3) is a feature that allows users to upload large objects in parts, rather than as a single object:  
How it works  
The object is uploaded in parts that can be uploaded independently, in any order, and in parallel. Once all the parts are uploaded, S3 combines them into the original object.  
Benefits  
Multipart uploads offer several advantages over traditional single-part uploads, including:  
Improved performance: Multipart uploads enable parallelization of data transfer, which improves overall upload speed.  
Reduced upload failures: Multipart uploads are more robust and reduce the chances of upload failures.  
Quick recovery from network issues: Multipart uploads can recover from a network error more quickly by only restarting the upload for the failed parts.  
When to use it  
Amazon S3 customers are encouraged to use multipart uploads for objects greater than 100 MB. You can use multipart upload for objects from 5 MB to 5 TB in size.  
we van have multiple versions of objects like git but we need to enable versioning in s3.  
To enable versioning in an Amazon S3 bucket, you can:  
Sign in to the AWS Management Console  
Open the Amazon S3 console  
Select the S3 bucket you want to enable versioning for  
Select the Properties tab  
Under Bucket Versioning, select Edit  
Select Enable versioning  
Select Save changes.

### **1. What is an S3 Bucket?**

* **Definition**: An S3 bucket is a container in Amazon Simple Storage Service (S3) that stores objects (files and metadata). It serves as the top-level namespace for data in S3.
* **Unique Naming**: Bucket names are globally unique across all AWS accounts and regions, ensuring no two buckets share the same name.
* **Purpose**: Buckets organize and manage data in a hierarchical structure, enabling features like access control, versioning, and storage class selection.

### **2. Key Features of S3 Buckets**

* **Data Organization**:
  + Objects in a bucket are identified by a unique key.
  + A bucket can store unlimited objects.
* **Storage Classes**: Allows selection of cost-effective storage for different access patterns (e.g., S3 Standard, S3 Glacier).
* **Security**: Access can be managed using IAM policies, bucket policies, and Access Control Lists (ACLs).
* **Versioning**: Tracks changes by storing multiple versions of an object in the same bucket.
* **Logging**: Buckets can be configured to log access requests for auditing purposes.
* **Static Website Hosting**: Buckets can serve static content directly to users.

### **3. Lifecycle of an S3 Bucket**

1. **Creation**: A bucket is created in a specific AWS region using the AWS Management Console, CLI, SDKs, or CloudFormation.
2. **Configuration**:
   * **Permissions**: Configure who can access the bucket (public/private settings).
   * **Versioning**: Enable to retain and retrieve previous versions of objects.
   * **Lifecycle Policies**: Automate transitions between storage classes or deletions.
   * **Encryption**: Enforce encryption at rest (SSE-S3, SSE-KMS, or client-side encryption).
3. **Data Storage**: Objects are uploaded, retrieved, or deleted using APIs or AWS tools.
4. **Monitoring and Optimization**:
   * **Logging**: Enable access logs for security monitoring.
   * **Metrics**: Use S3 Storage Lens or CloudWatch to monitor usage and performance.

### **4. Use Cases of S3 Buckets**

* **Backup and Disaster Recovery**: Store backups and leverage cross-region replication (CRR) for disaster recovery.
* **Data Lakes and Analytics**: Serve as a central repository for structured and unstructured data, integrated with AWS analytics services like Athena.
* **Static Website Hosting**: Host static websites by enabling public access and configuring index and error pages.
* **Media Storage and Distribution**: Store media files (images, videos) and serve them using AWS CloudFront (CDN).

### **5. Common S3 Bucket Configurations**

* **Public vs Private**: Most buckets are private by default but can be made public for hosting content.
* **Cross-Region Replication (CRR)**: Ensures data is copied to another AWS region for redundancy.
* **Object Lock**: Implements Write Once, Read Many (WORM) policies for compliance.
* **Event Notifications**: Trigger AWS Lambda, SNS, or SQS when objects are added, modified, or deleted.

### **6. Security Best Practices**

* Always keep buckets private unless public access is necessary.
* Use **IAM roles** and **bucket policies** for fine-grained access control.
* Enforce encryption (SSE-S3 or SSE-KMS) for sensitive data.
* Enable **logging** and **versioning** for tracking changes and recovery.
* Regularly review access permissions and use AWS Trusted Advisor to identify misconfigurations.

**CLOUDWATCH**

Cloudwatch is a gatekeeper for AWS cloud which helps in implementing and understanding:

Monitoring

Alerting

Reporting

Logging

Amazon CloudWatch metrics are data points that monitor the performance of AWS services. These metrics can be used to create dashboards, alarms, and events.

Amazon CloudWatch is a **monitoring and management service** provided by AWS that helps you track the performance and health of your applications, resources, and infrastructure in the cloud. Here's an easy explanation of its key features:

### What CloudWatch Does:

1. **Collects Metrics**
   * It gathers data (metrics) from AWS resources like EC2 instances, RDS databases, Lambda functions, and more.
   * Examples: CPU usage, memory utilization, network traffic.
2. **Logs Monitoring**
   * It captures and stores log files from your applications and resources.
   * You can search, analyze, and set up alerts based on specific log patterns.
3. **Set Alarms**
   * You can create alarms to notify you when certain conditions are met, such as high CPU usage or low available memory.
   * These alarms can trigger actions like sending an email or automatically scaling up/down resources.
4. **Dashboards**
   * It provides customizable dashboards to visualize metrics and logs in one place, making it easy to monitor system performance.
5. **Automated Responses**
   * CloudWatch can automatically take actions when an alarm is triggered, such as restarting an instance or invoking a Lambda function.

### Why It's Useful:

* **Proactive Monitoring:** Helps you identify and fix issues before they become big problems.
* **Cost Efficiency:** Optimizes resource usage by providing insights into under-utilized or over-utilized resources.
* **Automation:** Reduces manual intervention by automating responses to alerts.
* **Centralized Monitoring:** Offers a single view for all AWS resources.

**LAMBDA**

Lambda function follows server less architecture.

AWS Lambda is a cloud service provided by Amazon Web Services that lets you run your code without needing to manage servers. It’s often referred to as **serverless computing**, but here’s an easy breakdown:

## How Lambda Functions Fit into the Serverless World

At the heart of AWS Lambda are "Lambda functions." These are individual units of code that perform specific tasks. Think of them as small, single-purpose applications that run independently.

Here's how Lambda functions fit into the serverless world:

1. **Event-Driven Execution**: Lambda functions are triggered by events. An event could be anything, like a new file being uploaded to Amazon S3, a request hitting an API, or a specific time on the clock. When an event occurs, Lambda executes the corresponding function.
2. **No Server Management**: As a developer, you don't need to worry about managing servers. AWS handles everything behind the scenes. You just upload your code, configure the trigger, and Lambda takes care of the rest.
3. **Automatic Scaling**: Whether you have one user or one million users, Lambda scales automatically. Each function instance runs independently, ensuring that your application can handle any level of incoming traffic without manual intervention.
4. **Pay-per-Use**: One of the most attractive features of serverless computing is cost efficiency. With Lambda, you pay only for the compute time your code consumes. When your code isn't running, you're not charged.
5. **Supported Languages**: Lambda supports multiple programming languages like Node.js, Python, Java, Go, and more. You can choose the language you are comfortable with or that best fits your application's needs.

## Real-World Use Cases

Now, let's explore some real-world use cases to better understand how AWS Lambda can be applied:

1. **Automated Image Processing**: Imagine you have a photo-sharing app, and users upload images every day. You can use Lambda to automatically resize or compress these images as soon as they are uploaded to S3.
2. **Chatbots and Virtual Assistants**: Build interactive chatbots or voice-controlled virtual assistants using Lambda. These assistants can perform tasks like answering questions, fetching data, or even controlling smart home devices.
3. **Scheduled Data Backups**: Use Lambda to create scheduled tasks for backing up data from one storage location to another, ensuring data resilience and disaster recovery.
4. **Real-Time Analytics**: Lambda can process streaming data from IoT devices, social media, or other sources, allowing you to perform real-time analytics and gain insights instantly.
5. **API Backends**: Develop scalable API backends for web and mobile applications using Lambda. It automatically handles the incoming API requests and executes the corresponding functions.