***AWS***

***Day1***

***Cloud:***

Cloud generally refers to servers that are accessed over the internet, and the software and databases that run on those servers.

***Cloud Computing:***

Cloud Computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the internet (the cloud) to offer faster innovation, flexible resources, and economies of scale.

So, instead of having everything on your local computer, the cloud allows you to access it from anywhere with an internet connection. This makes it a very attractive option for businesses and individuals alike

**Regions: (geographic locations)**

What it is: Large geographical areas spread around the globe, each containing its own isolated set of data canters.

Purpose: Provides a complete set of AWS services within a specific geographic location. You choose the region closest to your users for optimal performance and data residency needs.

Number: 33 launched regions [AWS Global Infrastructure].

**Availability Zones (AZs): (collection of datacentres with in a region)**

What it is: Multiple, physically separate data canters located within a single region. (Connected with each other)

Purpose: Ensures high availability of your resources. Even if one AZ encounters an outage, your resources in other AZs within the region remain operational.

Number: Over 105 Availability Zones across all regions [AWS Global Infrastructure].

[Note: Region is a geographic location which hosts two or more Availability Zones]

**Edge Locations: (where end users access services located at AWS)**

It's like booster which boosts range and speed of services, like wifi booster which boosts range and speed of networks.

Purpose: Primarily focus on services like content delivery networks (CDNs) to deliver content with low latency (delay in network communication) to end users, particularly for geographically dispersed audiences.

Number: Over 600 CloudFront Points of Presence (PoPs) and 13 Regional Edge Caches [AWS Global Infrastructure].

Used for low latency for end users and also saves cache (frequently used data will be stored as cache, helps to access faster.

***Day2***

***Cloud Computing:***

* Store data/Apps on remote servers
* Process data/Apps on remote servers
* Access data/Apps on remote servers

***Cloud services and Types:***

Saas --> Software as a service (consume)

ready-to-use applications accessible through a web browser or app.

eg) Gmail, Dropbox, and Salesforce.

Paas --> Platform as a service (build)

development workbench in the cloud. It offers a complete environment for developing, deploying, and managing applications.

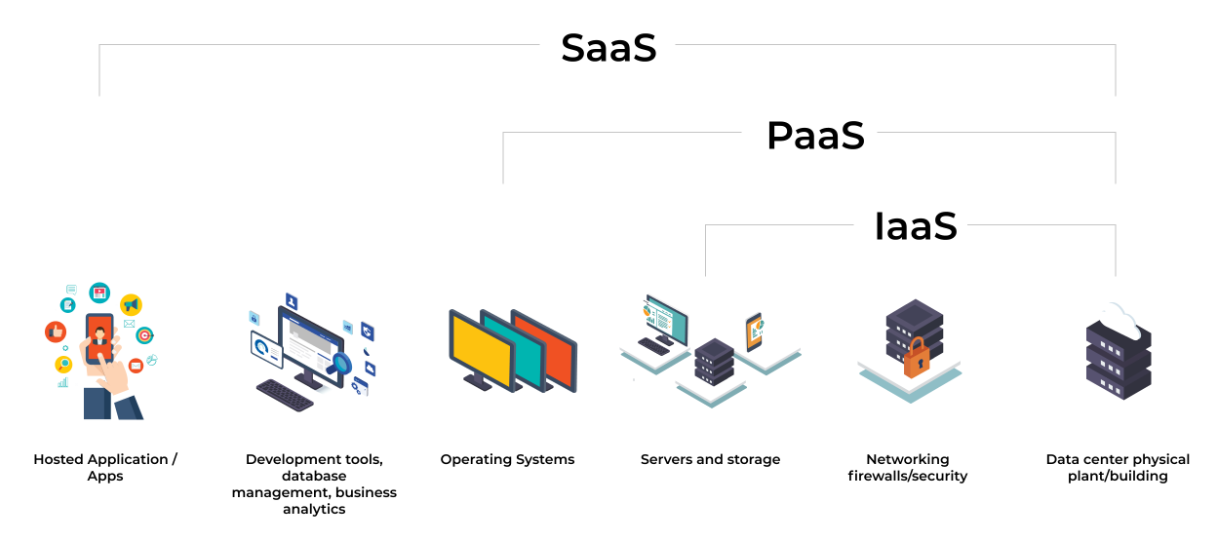
eg) GitHub, Kubernetes, docker

Iaas --> Infrastructure as a service (host)

Imagine renting a virtual data canter. IaaS provides the fundamental building blocks for computing, like virtual machines (VMs), storage, and networking.

eg) AWS, Azure

|  |  |  |
| --- | --- | --- |
| IaaS | PaaS | SaaS |
| Ownership:  Cloud provider manages infrastructure.  We manage Platform and Software | Ownership:  Cloud provider manages infrastructure and platform.  We manage software. | Ownership:  Cloud provider manages all: infrastructure, platform & software.  We manage nothing |
| Infrastructure: networking, hardware & virtualization.  Platform: operating system, middleware, runtime  Software: data and applications |  |  |
| Use cases:  Migration of workloads.  Test and deployment.  Storage, backups and  recovery. | Use casees:  Development framework.  Analytics & Business intelligence. | Use cases:  Buying applications like outlook, ondrive, skype |



* Figure is with respect to cloud provider.
* If cloud provider provides all then it's SaaS, if provide data center, security, storage, operating system then PaaS, and if provide Data center, security, servers and storages then it's IaaS.

***AWS: (Learn)***

1. Application Integration --> Simple Notification Services (SNS), we use this for sending automated messages like bank send automated messages if any transactions happen.
2. Compute --> EC2, Elastic Beanstalk, Lambda
3. Database --> RDS
4. Developer Tools --> codebuild, codecommit, codedeploy, codepipeline
5. Management and Governance: AWS auto scaling, cloudtrail, cloudwatch
6. Network and Content delivery --> Route 53, VPC
7. Security, Identity and Compliance --> IAM, Key management service (KMS), WAF and Shield
8. Storage --> EFS, S3, S3 types, EBS

***EC2 Dashboard:***

***Amazon Machine Images (AMI):***

* An Amazon Machine Image (AMI) is a supported and maintained image provided by AWS.
* It's like template to launch ec2 instances (required packages)
* You must specify an AMI when you launch an instance.
* You can launch multiple instances from a single AMI when you require multiple instances with the same configuration.
* You can use different AMIs to launch instances when you require instances with different configurations.

***Keypairs:*** key for instance

***Diff between instance families: t2, t3, a1, c1, c3, c4, c5***

T2/T3: Flexible, burstable performance for general use.

A1: Cost-effective, energy-efficient with Arm processors.

C1/C3/C4/C5: Increasingly powerful compute-optimized instances for heavy CPU tasks. (High Performance)

For high Performance we can go with c or m series.

|  |  |  |
| --- | --- | --- |
| Instance Family | Key Features | Use Cases |
| T2 | Burstable performance | Web servers, small databases |
| T3 | Improved burstable performance, cost-effective | Developer environments, microservices |
| A1 | Arm-based, energy-efficient | Web servers, containerized applications |
| C1 | Older compute-optimized | Batch processing, media transcoding |
| C3 | Enhanced networking, additional memory | HPC, scientific modeling |
| C4 | High CPU performance, cost-efficient | Ad serving, web servers |
| C5 | Latest generation, high performance | Machine learning, video encoding |

[Note:

Burstable Performance: Allows instances to handle occasional spikes in CPU demand by using performance credits.

Arm Processors: Energy-efficient CPUs that provide cost savings and are ideal for specific workloads, particularly in cloud environments]

***EBS: (Elastic Block Store)***

***Storage (root volume): diff between general purpose SSD, provisional purpose SSD and magnetic (standard)***

Magnetic storage offers cost-effective storage for light workloads, General Purpose (SSD) storage strikes a balance between performance and price for moderate workloads, and Provisioned IOPS (PIOPS) storage provides the highest level of performance for demanding applications.

**General Purpose SSD (gp2/gp3):**

* Think of it as: A well-balanced athlete. It offers a good mix of performance and affordability for most workloads.
* Pros: Faster than magnetic storage for booting up your instance, loading applications, and frequently accessed data. Good value for the price.
* Cons: Not the absolute fastest option. Might not be the most cost-effective for infrequently accessed data.
* By default, SSD supports 3 IOPS (Input Output Operations per Second)/GB means 1 GB volume will give 3 IOPS, and 10 GB volume will give 30 IOPS.

**Provisioned IOPS SSD (io1/io2 Block Express):**

* Think of it as: A specialized sprinter. Delivers the highest performance and consistent throughput for demanding applications. (throughput: measures the volume of data that passes through a network in a given period).
* Pros: Ideal for workloads requiring extremely fast response times and consistent performance, like high-frequency trading or real-time databases.
* Cons: Most expensive option. Not necessary for most applications.\By default, IOPS SSD supports 30 IOPS/GB means 10GB volume will give 300 IOPS.

**Magnetic (Standard HDD):**

* Think of it as: A workhorse for bulk storage. Offers the lowest cost per GB for data that doesn't need lightning speed. Best suited for workloads where data is accessed infrequently.
* Pros: Most cost-effective option for storing large amounts of data that are accessed infrequently, like backups or archives.
* Cons: Slowest option. Not ideal for applications requiring fast boot times or frequent data access.
* Its storage capacity of one volume ranges from 10GB to 1TB.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | General Purpose SSD (gp2) | Provisioned IOPS SSD (io1/io2) | Magnetic (Standard HDD) |
| Performance | Balanced | Highest & Consistent | Lowest |
| Cost | Moderate | Highest | Lowest |
| Ideal for | Most workloads | Demanding applications | Bulk storage, archives |

[Note:

For most applications, a general-purpose SSD (gp2) is a good starting point due to its balanced performance and affordability.

If you need the absolute fastest and most consistent performance, consider provisioned IOPS SSD. However, be prepared for a higher cost.

Use magnetic storage for data that doesn't require frequent access and prioritize cost-effectiveness]

Day3

EBS (AWS Elastic Block Storage):

* + It is an AWS service that it Provides Storage Volumes.
  + Provides block level storage volumes for use with Amazon EC2 instances.
    - Multiple volumes can be mounted to the same instance.
    - EBS volumes are network-attached, and independently from the life of an instance.
    - Storage volumes behave like raw, unformatted block devices, allowing users to create a file system on top of Amazon EBS volumes or use them in any other way you would use a block device (like a hard drive).
    - EBS volumes at one time can be attached to only one instance, it cannot be used with two or more instance at the same time.
    - Each EBS volume will have a volume id, which will be used by cloud watch and other services,
    - EBS volumes are placed in a specific Availability Zone and can then be attached to instances also in that same Availability Zone.
    - Each storage volume is automatically replicated within the same Availability Zone.
    - EBS provides the ability to create point-in-time snapshots of volumes. All snapshots are stored in s3 service of AWS.

What is difference between Instance Store Volume (Root Volume) and an EBS Volume?

***Instance Store Volume (Root Volume):***

* + An Instance Store Volume is temporary storage that is used to store the temporary data required by an instance to function.
  + By default, Root Volume will be Created when we Launch the instance.
  + It is the Volume where the OS and applications Normally live.
  + The data is available as long as the instance is running. As soon as the instance is turned off, the Instance Store Volume gets removed and the data gets deleted.
  + It Depends Upon Instance Lifecycle as Instance Deletes root volume get deleted.

***EBS (Elastic Block Storage) Volume / User Volume:***

* + On the other hand, an EBS Volume represents a persistent storage disk.
  + This is also known as User Volume it is meant to store the user’s data.
  + The data stored in an EBS Volume will be available even after the instance is turned off.
  + It Won’t depends upon instance lifecycle.

Main Components of EBS:

1.Volumes

2.Snapshots

3.Images

***EBS: Volume***

Create instance for AWS Linux

Login to AWS Linux: ssh -i pemfile\_name.pem ec2-user@PublicIP

Check disk drive: df -h

Filesystem Size Used Avail Use% Mounted on

/dev/xvda1 8.0G 1.6G 6.5G 20% /

(root drive)

New volume attach to ec2 instances:

* + **Create volume** --> AWS Console --> volumes --> create --> gp3 --> vol size in GB --> availability zone (same as instance AZ) --> tags (for reference) --> create

**Attach Volume in frontend** --> select volume that's created and is available --> actions --> Attach Volume --> Instance (Select instance to attach vol) --> device name (select any in it like /dev/sdb) --> attach volume

**Check volume is attached or not** --> go to instance --> storage --> can see all volumes

[Note: Make sure to delete volume as soon as practise is done or else will be charged]

**Check attached vol in Linux (CLI)**

lsblk (list block) --> lists all the volumes of instance

We have created volume and attached it to instance but need to mount it so we need to create file system first, then create directory/ choose existing directory to mount vol

* + **Create the file system**

sudo mkfs -t xfs /dev/xvdb

(mkfs --> make fuse, xfs --> extension of file system (or) ext4, /dev/xvdb --> path)

**Create a new directory for mounting**

mkdir new\_vol

**mounting the volume to directory**

sudo mount **/dev/xvdb** */home/ec2-user/New\_vol*

* + Vol will be mounted to new\_vol directory so directory will be having that amount of storage we given while creating volume in frontend
  + df -h (Check disk drive: to check all mounted vol)

Similarly, can create multiple volumes at frontend and mount it for the instance in same directory.

**[Steps to create and attach volume,**

* + **create vol at frontend in AWS**
  + **Attach to the instance in AWS**
  + **Check attached vol in CLI using "lsblk"**
  + **Create filesystem for mounting using "sudo mkfs -t xfs /dev/xvdb"**
  + **Create directory/choose existing directory to mount vol and mount using "sudo mount /dev/xvdb /home/ec2-user/New\_vol"**
  + **Check mounted vol using "df -h"]**

* + **Modify/extending the existing volume ec2:**

If volumes is full for the directoey then we may need to extend volume then we need to modify.

EC2 instance --> volume --> select vol that’s need to be modified/extended --> actions (modify volume)

[Note: Size of vol can only be increased, can't decrease]

steps:

**lsblk**

Cheks added/modified vol

**sudo xfs\_growfs /dev/xvdb**

Resize it in CLI to update changes

df -h --> check vol of disk

**[Steps to modify volume,**

* + **Modify vol in frontend (AWS)**
  + **Check vol is updated or not in CLI using "lsblk"**
  + **Resize vol in CLI using "sudo xfs\_growfs /dev/xvdb"**
  + **Check modified vol using "df -h"]**

**[Note: These vol are temporary, will be detached if system gets rebooted and we need to attach again]**

**So, as make vol attached in instance permanently then we need to follow few steps,**

* + ***To mount volume permanently*** 
    - **sudo blkid** --> to check UUID of volume

eg) (xvdb xfs--> /dev/xvdb: UUID="7bf92b51-1efb-4f93-bae9-83a3a1e4af3d")

* + **Backup file**

To backup any files, copy

sudo cp /etc/fstab /etc/fstab\_bkup

To bring back backup, rename

sudo mv /etc/fstab\_bkup /etc/fstab

Verify: cd /etc --> ls

* + **cd etc --> sudo vim fstab**

**(or)**

**sudo vim /etc/fstab**

Edit: (UUID=8220a4cd-a61d-43ab-9248-657439f537db /home/ec2-user/new\_vol xfs defaults,nofail 0 2)

* + **Mount and unmount**
    - sudo mount -a
    - sudo umount /home/ec2-user/dir\_name

* + **sudo reboot**

Reboots system, we can still see volume mounted to directory

**[Note: we need to reboot system only if we modify root volume, not required for other volumes (ie., EBS)]**

[Note: sudo !! --> executes previous command]

* + ***Snapshots:***

Capture/backup of volume

AWS --> ec2 --> snapshot --> create snapshot --> select volume to make backup --> create snapshot

* + We can use snapshot to copy all data when we create new volume by giving snapshot id.
  + We can attach new vol created using snapshot to new instance/existing instance where we can get all data. (it works as recovery)
  + Once we add new vol to new instance in frontend then need to mount it in CLI.

**[Steps to create snapshot,**

* + **Create snapshot in AWS using volume which we need snapshot**
  + **Create new volume using snapshot ID so all data in snapshot will be copied to new volume.**
  + **We can attach that volume to either new instance or existing instance.**
  + **Go to CLI, mount that volume to any directory]**

CounterWebApp.war

<https://github.com/hareeshab/stable_war/blob/4eef8c99c5a79f0efda3261788515f568265c566/CounterWebApp.war>

**[Note: Snapshot will take snap of only data od volume, not configuration or operating system]**

* + ***Image (AMI)***

AMI are copy/backup of whole Instance. (Volumes + Operating system + files)

* + Create AMI

Instance --> Image and Templates --> create image --> image name --> add tag --> create image

Once AMI is created of any instance then can create new instance using AMI where everything of that instance will be copied.

* + Create Instance

Launch Instance --> name --> AMI (OS) --> select created AMI --> add key pair --> Launch Instance

* + Validate it in CLI

Login to that instance in CLI

df -h --> disk storage (to check volumes)

ls --> lists all files/directories

***AMI and Snapshot difference***

Both are used for backup and recovery but Snapshot is used for volumes and AMI is used for Instances.

When we create either of these, all data that were present during creation will only be saved…Later working on instances/volumes will not get reflected. If we need updated snapshot/AMI then we need to create new one.

Snapshot: A snapshot captures the state of a single EBS volume at a specific point in time.

AMI: An AMI is a template of an entire EC2 instance, including the operating system, configuration, applications, and data on all attached EBS volumes.

Day4

***1. diff AMI and snapshot***

Scope:

Snapshot: A snapshot captures the state of a single EBS volume at a specific point in time.

AMI: An AMI is a template of an entire EC2 instance, including the operating system, configuration, applications, and data on all attached EBS volumes.

Use Cases:

Snapshot:

Data backup and recovery: Restore a volume to a previous state if data gets corrupted or accidentally deleted.

Data migration: Easily copy data volumes between EC2 instances.

AMI:

Provisioning new instances: Quickly launch new instances with the same configuration as the original AMI, saving time on setup.

Disaster recovery: Restore a complete server environment in case of a major outage.

Cost and Efficiency:

Snapshot: Generally cheaper since you're only storing the data on a single volume. .

AMI: Can be more expensive due to the larger size encompassing the entire instance configuration. AMIs store all data on attached volumes, so the cost reflects the total storage used.

Bootable:

Snapshot is non-bootable

AMI is bootable

***2. TCP and UDP protocol***

TCP and UDP are both fundamental protocols that handle data transmission over networks, but they take different approaches:

***Connection-Oriented vs. Connectionless:***

TCP (Transmission Control Protocol): TCP is connection-oriented. It establishes a connection between sender and receiver before sending data. This connection allows features like:

Reliability: TCP guarantees data arrives in order and without errors. It checks for missing or corrupt packets, resending them if necessary.

Flow control: TCP paces data transmission to avoid overwhelming the receiver.

Examples: web browsing, file transfer, email.

UDP (User Datagram Protocol): UDP is connectionless. It sends data packets directly without creating a connection. This makes it:

Faster: Less overhead compared to TCP's connection setup and management.

Simpler: Less complex protocol, easier to implement.

Unreliable: UDP doesn't guarantee delivery or order of packets. Applications using UDP need to handle these aspects themselves.

Examples: online gaming, video streaming, DNS lookups.

|  |  |  |
| --- | --- | --- |
| **Feature** | **TCP** | **UDP** |
| Connection Oriented | Yes | No |
| Reliability | High | Low |
| Speed | Slower | Faster |
| Error Checking | Yes | No |
| Order of Delivery | Guaranteed | Not Guaranteed |
| Best for | Important messages | Quick updates |

***3. Error code 1XX, 2XX, 3XX, 4XX, 5XX***

|  |  |  |  |
| --- | --- | --- | --- |
| Code Range | Meaning | Description | Example |
| 1XX (Informational) | Request Received | The server has acknowledged the request and is processing it. | 100 Continue |
| 2XX (Success) | Request Successful | The request was received, understood, and processed successfully. | 200 OK |
| 3XX (Redirection) | Further Action Needed | The server needs further information to complete the request, such as redirecting to a different URL. | 301 Moved Permanently |
| 4XX (Client Error) | Request Error | There's an issue with the request itself, like a typo in the URL or missing data. | 404 Not Found |
| 5XX (Server Error) | Server Error | The server encountered an error and couldn't complete the request. | 500 Internal Server Error |

|  |  |  |  |
| --- | --- | --- | --- |
| **Status Code Class** | **Code** | **Meaning** | **Description** |
| **1XX: Informational** | 100 | Continue | Request received, continue sending the request body |
|  | 101 | Switching Protocols | Switching to the protocol requested by the client |
|  | 102 | Processing (WebDAV) | Server has received and is processing the request, but no response is available yet |
| **2XX: Success** | 200 | OK | Request succeeded |
|  | 201 | Created | Request succeeded and a new resource was created |
|  | 202 | Accepted | Request accepted, but processing not complete |
|  | 203 | Non-Authoritative Information | Request succeeded, but the information may come from a third party |
|  | 204 | No Content | Request succeeded, but no content is being returned |
|  | 205 | Reset Content | Request succeeded, reset the view that sent the request |
|  | 206 | Partial Content | Request succeeded, but only part of the resource is returned |
|  | 207 | Multi-Status (WebDAV) | Multiple independent responses in a single response body |
|  | 208 | Already Reported (WebDAV) | Member of a collection has already been reported in a previous response |
|  | 226 | IM Used | Request succeeded, content has been transformed as indicated in the response headers |
| **3XX: Redirection** | 300 | Multiple Choices | Multiple options for the resource, user can choose one |
|  | 301 | Moved Permanently | Resource has been permanently moved to a new URL |
|  | 302 | Found | Resource temporarily moved to a different URL |
|  | 303 | See Other | Resource can be found at another URL, use GET method to retrieve it |
|  | 304 | Not Modified | Resource has not been modified since last requested |
|  | 305 | Use Proxy | Resource must be accessed through a proxy specified in the response |
|  | 306 | (Unused) | Previously used, no longer used |
|  | 307 | Temporary Redirect | Resource temporarily moved to another URL, use the original URL for future requests |
|  | 308 | Permanent Redirect | Resource permanently moved to another URL, use this new URL for future requests |
| **4XX: Client Error** | 400 | Bad Request | Server cannot process request due to client error |
|  | 401 | Unauthorized | Authentication required to access the resource |
|  | 402 | Payment Required | Reserved for future use |
|  | 403 | Forbidden | Server understood the request but refuses to authorize it |
|  | 404 | Not Found | Server cannot find the requested resource |
|  | 405 | Method Not Allowed | Method used in the request is not allowed for the resource |
|  | 406 | Not Acceptable | Requested resource not capable of generating acceptable content |
|  | 407 | Proxy Authentication Required | Client must authenticate with the proxy |
|  | 408 | Request Timeout | Server timed out waiting for the request |
|  | 409 | Conflict | Request could not be processed due to conflict with the current state of the resource |
|  | 410 | Gone | Resource is no longer available and will not be available again |
|  | 411 | Length Required | Length of content is required and was not specified |
|  | 412 | Precondition Failed | Server does not meet a precondition specified in the request |
|  | 413 | Payload Too Large | Request payload is too large for the server to process |
|  | 414 | URI Too Long | URI provided in the request is too long for the server to process |
|  | 415 | Unsupported Media Type | Media type of the request is not supported by the server |
|  | 416 | Range Not Satisfiable | Client requested a range not satisfiable by the server |
|  | 417 | Expectation Failed | Server cannot meet the requirements of the Expect request-header field |
|  | 418 | I'm a teapot | Joke status code indicating the server is a teapot, not a coffee machine |
|  | 421 | Misdirected Request | Request was directed at a server unable to produce a response |
|  | 422 | Unprocessable Entity (WebDAV) | Request was well-formed but could not be followed due to semantic errors |
|  | 423 | Locked (WebDAV) | Resource that is being accessed is locked |
|  | 424 | Failed Dependency (WebDAV) | Request failed due to failure of a previous request |
|  | 425 | Too Early | Server is unwilling to risk processing a request that might be replayed |
|  | 426 | Upgrade Required | Client should switch to a different protocol |
|  | 428 | Precondition Required | Server requires request to be conditional |
|  | 429 | Too Many Requests | User has sent too many requests in a given time period |
|  | 431 | Request Header Fields Too Large | Server unwilling to process the request because its header fields are too large |
|  | 451 | Unavailable For Legal Reasons | Access to the resource is denied due to legal reasons |
| **5XX: Server Error** | 500 | Internal Server Error | Generic server error message |
|  | 501 | Not Implemented | Server does not recognize or cannot fulfill the request method |
|  | 502 | Bad Gateway | Server received an invalid response from the upstream server |
|  | 503 | Service Unavailable | Server is currently unable to handle the request due to temporary overload or maintenance |
|  | 504 | Gateway Timeout | Server did not receive a timely response from the upstream server |
|  | 505 | HTTP Version Not Supported | Server does not support the HTTP protocol version used in the request |
|  | 506 | Variant Also Negotiates | Internal configuration error on the server |
|  | 507 | Insufficient Storage (WebDAV) | Server unable to store the representation needed to complete the request |
|  | 508 | Loop Detected (WebDAV) | Server detected an infinite loop while processing the request |
|  | 510 | Not Extended | Further extensions to the request are required for the server to fulfill it |
|  | 511 | Network Authentication Required | Client needs to authenticate to gain network access |

***4. port numbers***

|  |  |  |  |
| --- | --- | --- | --- |
| **Protocol** | **Port Number** | **Full Form** | **Description** |
| HTTP | 80 | Hypertext Transfer Protocol | Standard web traffic for communication between browsers and servers. |
| HTTPS | 443/  8443 | Secure Hypertext Transfer Protocol | Secure encrypted web traffic, often used for sensitive information like online banking. |
| SSH | 22 | Secure Shell | Secure remote login protocol for managing servers. |
| SMTP | 25 | Simple Mail Transfer Protocol | Sending emails. |
| DNS | 53 | Domain Name System | Translates website names (like [invalid URL removed]) into IP address for browsers. |
| DHCP | 67 | Dynamic Host Configuration Protocol | Assigns IP addresses to devices on a network  automatically. |
| IMAP | 143 | Internet Message Access Protocol | Accessing emails on a server. |
| LDAP | 389 | Lightweight Directory Access Protocol | Managing user accounts and permissions in a network. |
| MSSQL | 1433 | Microsoft SQL Server | Relational database management system by Microsoft (default port). |
| MySQL | 3306 | MySQL | Open-source relational database management system (default port). |
| RDP | 3389 | Remote Desktop Protocol | Remote access to a graphical user interface of another computer. |
| NFS | 2049 | Network File System | Sharing file systems across a network. |
| RDS | Varies/  1150-65535 | Remote Desktop Services | Microsoft technology for remote desktop access (default port depends on configuration). |
| PostgreSQL | 5432 | PostgreSQL | Open-source relational database management system (default port). |

Day5

Major Components of Aws VPC:

1.Subnet (Public & Private Subnet)

2.NACL

3.Internet Gateway

4.NAT gateways

5.Route tables (Public & Private Route Tables)

6.Security Group

Internet --> VPC --> Internet Gateway --> route table --> subnet --> security groups --> NaCL --> instance

NAT Gatewat --> Private subnet

Virtual Private Cloud (VPC)

Our own private cloud/network within the cloud.

Isolates our resources from everyone else's.

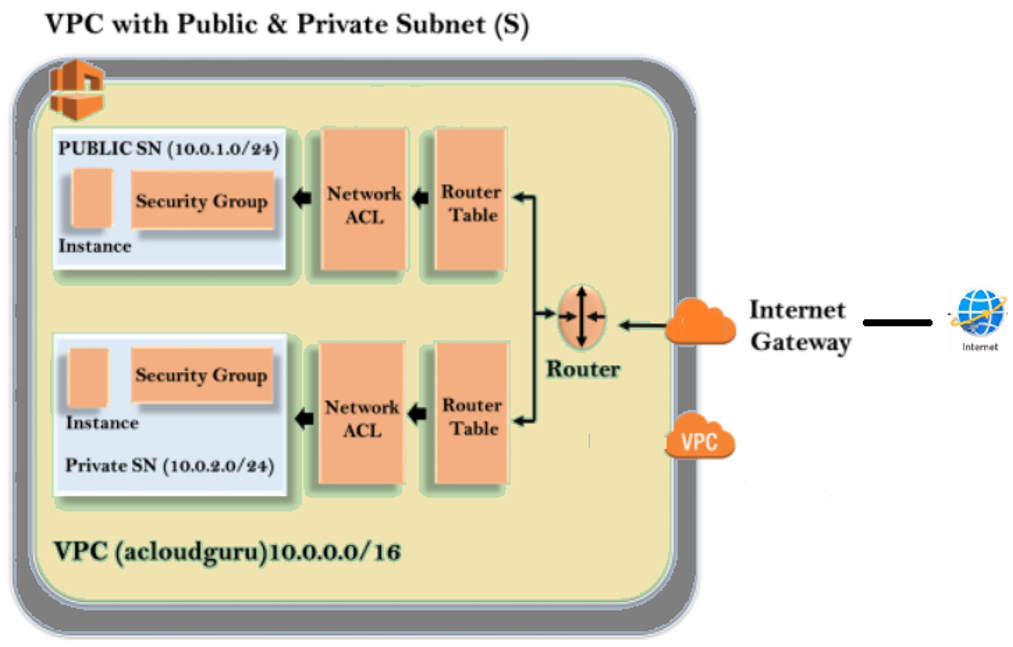
èA VPC is a Virtual network that you create in the cloud. It allows you to have your own private section of the internet, just like having your own network within a larger network.

Within this VPC, you can create and manage various resources, such as servers, databases, and storage.

Size of VPC can be defined by IP address range.

Max of 65536 IP address can be allocated in VPC

***Architecture of VPC***



VPC -->

1. subnets: A range of IP address in your VPC.(splitting ranges of IP addresses to different subnets --> different projects)

A subnet must reside in a single Availability Zone. After you add subnets, you can deploy AWS resources in your VPC.

(public (application server will be in public)

and private(Databases will be in private, no access to anyone

as all information will be stored here))

Public Subnets: If a subnet has route table that is associated with Internet Gateway it is called a public subnet.

Private Subnets: If a subnet has route table which is associated with Nat Gateway it is called a private subnet or doesn't have route to Internet Gateway. -->

[Note: Servers --> Application & Data(sensitive)]

1. IGW (Internet Gateway): main component that allows to connect public IP's to internet. Communicates with internet(outside) and Route table (inside --> VPC).

1. RT(Route Table): routing from request to end point, from internet or instance.

Need for public and private subnet.

Instance communicate through route table(RT) to internet gateway(IGW).

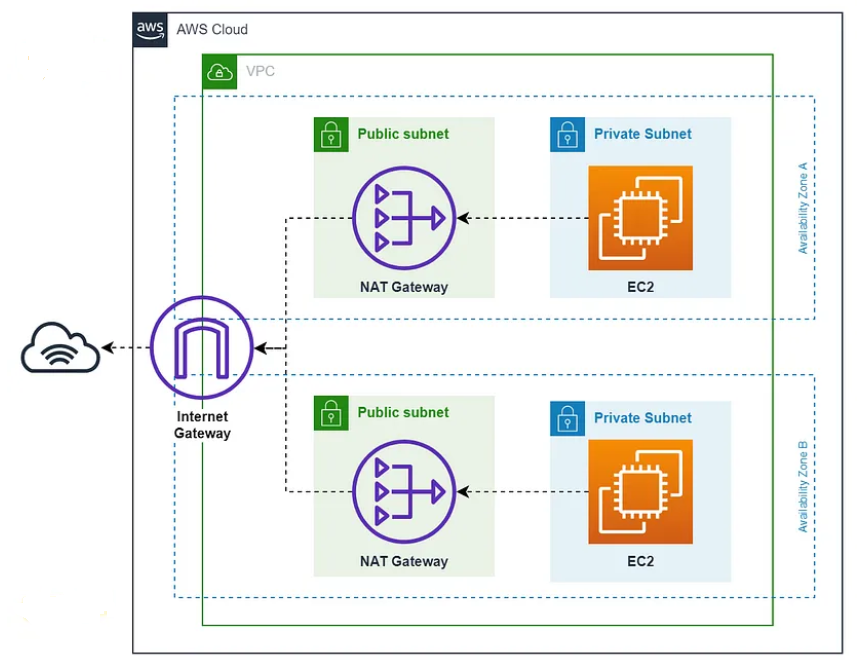
1. security groups: instance firewall
   1. Security Group - security layer/Virtual Firewall to EC2 instances that control both inbound and outbound traffic at the instance level.
   2. Security groups allow you to define rules that permit or restrict traffic based on protocols, ports, and IP addresses.
   3. Security groups act at the instance level, not the subnet level.

1. NaCl: subnet firewall
   1. A network access control list (NACL) is an optional/Additional layer of security for your VPC.
   2. Nacl is Associated security under subnet level.
   3. It is a stateless firewall that controls inbound and outbound traffic at the subnet level.

1. NAT Gateway (NAT): unidirectional (instance to internet only)
   1. Its created in Public Subnet only.
   2. Used to download packets/resources from internet and mask the IP address so no one can get actual address of Application/server.
   3. It changes Public IP instance to either load balancer or Route table.
   4. If it does with load balancer then it's SNAT and if it does with Route Table then NAT Gateway.

**Public Subnet --> NAT --> Internet Gateway --> Private Subnet**

[Note: only request from Instance will be taken by NAT and through Internet Gateway it will download/upgrade required resources to the Private instance but any request coming from Internet will be denied]



Practical:

CIDR Block

10.0.0.0/16

2^32-16 == 2^16 == 65536 --> IP address (can be created)

Range is from, 10.0.0.0 - 10.0.255.255

10.0.0.0/24

2^32-24 == 2^8 == 256 --> IP address

10.0.1.0/24 --> range is from, 10.0.1.0 - 10.0.1.255

10.0.1.0/24

2^32-28 == 2^4 == 16 --> IP address (Range, 10.0.1.0 - 10.0.1.15)

1. Create VPC:

Your VPC

VPC only --> Name Tag (demo\_VPC) --> IPv4 CIDR block (10.0.0.0/16) --> Tags (key:name, value:demo\_VPC) --> create

1. Subnets:

Public subnet

VPC ID (select demo\_VPC) --> subnet settings --> subnet name (pub1) --> availability zone (choose any) --> IPv4 VPC CIDR block (10.0.0.0/16) --> IPv4 subnet CIDR block (10.0.1.0/28 = 16IPs) --> Tags (key:name, value:pub1)

(add new subnet)

Private subnet

VPC ID (select demo\_VPC) --> subnet settings --> subnet name (pri1) --> availability zone (choose any) --> IPv4 VPC CIDR block (10.0.0.0/16) --> IPv4 subnet CIDR block (10.0.2.0/28 = 16IPs) --> Tags (key:name, value:pri1)

--> create subnet

1. Internet gateway:

Name tag (demo-igw) --> create internet gateway

Once created then attach it to created VPC

Select IGW --> Actions (Attach to VPC) --> select a VPC (demo\_VPC)

1. Route Table: need to create for each subnet

Name (RT-pub) --> VPC (select VPC) --> tag --> create Route table

**Create another route table for private**

Name (RT-pri) --> VPC (select VPC) --> tag --> create Route table

**Created Route table for pub and pri but need to associate it with respective subnets**

Select RT-pub --> subnet associations --> edit subnet associations --> select pub1 --> save

Similarly, do for private RT

**Need to give routes for Route table for public subnet**

Select RT-pub --> routes --> add route --> Destination (0.0.0.0/0 = permissions for all IP addresses) --> Target (Internet gateway --> select created internet gateway) --> save

1. NAT Gateways:

Create NAT gateway --> subnet (select public subnet) --> connection type (public) --> elastic IP (static IP = Allocate elastic IP) --> tags --> create NAT

Route Table --> select RT-pri --> roots --> destination (0.0.0.0/0), Target (NAT Gateway) --> save

**Create instance for public subnet**

Launch instance --> name (pub\_server) --> OS (linux/ubuntu) --> keypair (available keypair) -->

network settings --> VPC (select created VPC) --> subnet (created subnet) --> Auto-assign public IP (Enable) --> launch instance

**Similarly, create instance for private subnet**

Validation:

Connect public instance in CLI

sudo yum update -y

ping google.com

Connect private instance in CLI

Need to copy paste pemfile key from any public instance that's part of VPC.

1. From local system copy pemfile key --> cat pemfilename.pem --> copy full key
2. Login to any public instance from VPC

vim aws.pem --> paste pemfile key --> save

1. Login to public instance

ssh -i aws.pem ec2-user@elasticIP

chmod 400 aws.pem

ping google.com

sudo yum update -y

[Note: It will work only if NAT Gateway is linked to private instance or else it won’t accept any request from instance and keep on loading]

Day6

***Firewalls:***

NACL --> subnet level

Security Groups --> Instance leve**l**

* 1. ***NACL:* subnet level firewall**
     1. A network access control list (NACL) is an optional/Additional layer of security for your VPC.
     2. NACL is Associated security under subnet level.
     3. NACL controls inbound and outbound traffic at the subnet level.
     4. Network ACLs are stateless. This means any changes applied to an incoming rule will not be applied to the outgoing rule.

**VPC --> Security --> Network ACLs**

By default, inbound type as All Traffic (opening all ports like http, ssh, etc)

If we give inbound type as SSH and source as system IP or any IP range then only those IP will be able to access the instance.

eg) System IP --> 198.18.12.11

IP range --> 10.0.1.0/28 were it ranges from 1 to 16

(2^32-28 = 2^4 = 16)

If we allow then it allows only these IP's and if we deny then it will deny all these IP's.

Similarly, we can add many rules by adding different ports, IP's, allow/deny.

NACL can be used for both Public and Private subnets.

**[Note: Inbound is incoming from Internet to Instance**

**Outbound is outgoing from Instance to Internet]**

**Create network ACL**

Name (demo-NACL) --> select VPC --> tags --> create

Select demo-NACl --> Subnet associations --> edit --> pub-sn

**Similarly, we can create another NACL for private subnet**

* 1. ***Security Groups:* instance level firewall**
     1. Security Group - security layer/Virtual Firewall to EC2 instances that control both inbound and outbound traffic at the instance level.
     2. Security groups allow you to define rules that permit or restrict traffic based on protocols, ports, and IP addresses.
     3. Security groups act at the instance level, not the subnet level.
     4. Security groups are state full. This means any changes applied to an incoming rule will be automatically applied to the outgoing rule.

**VPC --> Security --> Security groups**

**[Note: In organization level, we won't be using All Traffic instead will be using Custom TCP]**

**When we create any instance using subset then 5 Ips will be reserved**

**Some IP-Addresses are Reserved they as follows:**

**10.0.0.0/28: 5 IP will be reserved (first 4 and last 1)**

* 1. 10.0.0.0 – Network Address
  2. 10.0.0.1 –Reserved by AWS for VPC Router
  3. 10.0.0.2 – Reserved by AWS for mapping to AWS-provided DNS Server (Domain Name system)
  4. 10.0.0.3 – Reserved by AWS for Future usage.
  5. 10.0.0.15 –N/w broadcast (AWS does not support broadcast in a VPC)

* 1. ***Elastic IP addresses (EIPs):***

Static, public IPv4 addresses provided by Amazon that you can associate with an EC2 instance, elastic network interface, or AWS resource.

**Create Elastic IP**

EC2 --> Network & Security --> Elastic IPs --> create elastic IP --> tag --> create

Select elastic IP --> actions --> Associate Elastic IP --> select instance --> associate

**[Note: Once elastic IP is associated with Instance then IP of that instance will be static/permanent)]**

**Day7**

1. ***Web server and application server***

|  |  |  |
| --- | --- | --- |
| Feature | Web Server | Application Server |
| Main Function | Serve static content | Generate dynamic content |
| Data Processing | Minimal | Complex processing and logic |
| Protocols | Primarily HTTP, may support others | Supports various protocols including HTTP |
| Resource Usage | Lower | Higher |
| Examples | Apache, Nginx | JBoss, WebLogic |

**3 types of application:**

1. Static: static content in websites, used for viewing purpose.
   * 1-Tier Architecture
   * These are web servers
   * eg) Apache, nginx
   * Used for small applications

1. Dynamic: dynamic content like insert/update/delete
   * 2-Tier Architecture
   * Servlets and Java server pages (JSPs): Java programs that run on a Java application server and extend the capabilities of the Web server.
   * used to connect between application and Database
   * eg) tomcat, Apache TomEE
   * Used for medium applications

1. Business Logic:
   * the custom rules or algorithms that handle the exchange of information between a database and user interface.
   * define how a business system or application performs calculations and executes transactions.
   * For example, when you purchase a website, business logic determines how much you should pay for shipping or taxes before providing you with a final total.
   * eg) Jboss, WebLogic
   * Used for large applications where lot of logics, calculations are required like e-commerce, stock market

[Note: Depending upon application size and type, we need to choose application to deploy]

***Practise:***

***1) apache2 server***

Default port --> 80 (HTTP)

Checking Apache web server for deploying pages,

Launch ubuntu instance in AWS so we can install Apache

Login to that instance in CLI

Step1: before installing any servers, we need to update

sudo apt-update -y (or) sudo apt updatec

--> installs/updates all security patches

Step2: Install Apache2 server

sudo apt install apache2

Step3: check status of service

sudo service apache2 status

Step4: add inbound rule for apache2 in the instance

Select instance --> security --> click security groups --> edit inbound rule --> add rule --> Type(HTTP), source(anywhere) --> save

Step5: open apache server in browser

Copy paste instanceIP:80 in URL --> default port no for http/apache2

2) Launch application in apache2 server

Step1: **cd /var/www/html/** --> folder where we deploy application

inside this folder codes are saved in files (eg., index.html) which will be displayed as application

Step2: **ls**

**index.html** --> any content(codes) stored inside this file will be reflected in apache server

Step3: backup original file

**sudo mv index.html index\_bkup.html**

Step3: **sudo vim index.html** --> add any content/codes

eg) testing web server

Step4: refresh page in browser

3) Change port

By default, port is 80

Step1: **cd** --> to get into home path

**cd /etc/apache2/** --> all configurations will be present here

**ls** --> lists all configuration related files

Step2: sudo vim ports.conf --> port config is present

Listen 8081 --> changed to 8081 from 80

Step3: If done any config. Modifications done then need to restart service

sudo service apache2 restart

Step4: Since, we have changed port to 8081 so need to add inbound rule for this port to work in the instance

EC2 --> Select instance --> click security groups --> edit inbound rule --> add --> Type (Custom TCP), port range (8081), source (Anywhere-IPv4) --> save

Step5: open apache2 server in browser

instanceIP:8081

netstat -na --> list out ports that's listening (occupied)

[Note: One application can have only 1 port,

if multiple applications have same port, then change the port]

***4) nginx server***

Default port --> 80 (HTTP PORT)

Step1: Install nginx server

sudo apt install nginx -y

Step2: check status of server

sudo service nginx status

Step3: open in browser

instanceIP:80 (or) instanceIP -->since, 80 is default port so we may or may not give

Launch application in nginx server

Step1: cd /usr/share/nginx/html/ (or) cd /var/www/html/

--> folder where we deploy application

Step2: backup original file

sudo mv index.html index\_bkup.html

Step3: sudo vim index.html --> add any content/codes

eg) testing web server

Step4: refresh page in browser

3) Change port

By default, port is 80

Step1: cd --> to get into home path

cd /etc/nginx/ --> all configurations will be present here

ls --> lists all configuration related files

Step2: port/root configuration

cd /etc/nginx/sites-available

Step3: sudo vim default --> can change port no, root location, see files like index.html

[Note: apache2 and nginx are almost similar but difference is in configuration and folder structure and in nginx we can host both static as well as dynamic websites]

Day8

Application Server:

Tomcat

***1) Installing Tomcat server in CLI***

1. sudo apt update -y --> installs/updates all security patches
2. sudo apt-get install default-jdk --> Install java for Servlets and Java server pages (JSPs)

java --version --> to check it's installed or not

1. sudo groupadd tomcat --> add tomcat to user group
2. sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat

--> restricts bash shell access (/bin/false) and ensure the user is associated with a particular group and directory, adds user to tomcat group

1. cd /tmp --> moving to tmp folder
2. sudo wget <https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.91/bin/apache-tomcat-9.0.91.tar.gz> --> download tomcat
3. sudo mkdir /opt/tomcat --> create folder to make root folder of tomcat
4. sudo tar -xzvf apache-tomcat-9.0.91.tar.gz -C /opt/tomcat --strip-components=1 --> extract tar.gz file
5. cd /opt/tomcat --> move to tomcat folder
6. sudo chown -RH tomcat: /opt/tomcat --> changing owner of these folders to tomcat from root
7. sudo sh -c 'chmod +x /opt/tomcat/bin/\*.sh'
8. sudo update-java-alternatives -l
9. sudo vim /etc/systemd/system/tomcat.service --> creates file tomcat.service to provide configuration used in service and add in it given below:

[Unit]

Description=Apache Tomcat Web Application Container

After=network.target

[Service]

Type=forking

User=tomcat

Group=tomcat

Environment="JAVA\_HOME=/usr/lib/jvm/java-1.21.0-openjdk-amd64"

Environment="JAVA\_OPTS=-Djava.security.egd=file:///dev/urandom -Djava.awt.headless=true"

Environment="CATALINA\_BASE=/opt/tomcat"

Environment="CATALINA\_HOME=/opt/tomcat"

Environment="CATALINA\_PID=/opt/tomcat/temp/tomcat.pid"

Environment="CATALINA\_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC"

ExecStart=/opt/tomcat/bin/startup.sh

ExecStop=/opt/tomcat/bin/shutdown.sh

[Install]

WantedBy=multi-user.target

1. sudo systemctl daemon-reload --> any configuration changes in system will be reloaded and applied to system
2. ssudo systemctl start tomcat --> restart service
3. sudo systemctl status tomcat --> check status of service

Once status is active,

Add inbound rule in instance under security so as to allow traffic for tomcat port.

instanceIP:8080 --> check tomcat webpage in browser

==================================================================================

Day9

Assignment:

What is load balancer? why we need load balancer?

Diff b/w application and network load balancer?

The process of distributing traffic among multiple servers to improve a service or application's performance and reliability

Load balancers improve application performance by increasing response time and reducing network latency.

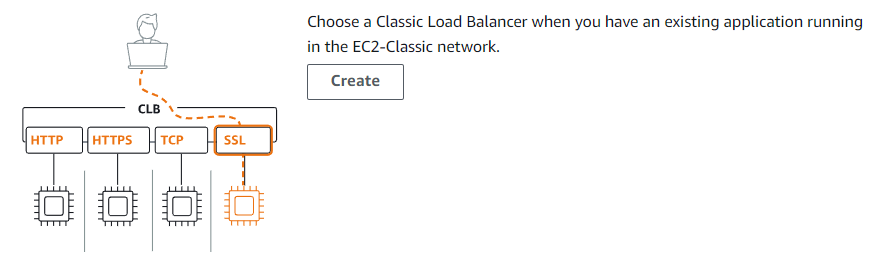
They perform several critical tasks such as the following:

Distribute the load evenly between servers to improve application performance.

Redirect client requests to a geographically closer server to reduce latency.

Classic Load Balancer:

Can give group servers, it works in round-robin fashion where takes request within page, can't route to other targets. Path based routing isn't applicable in classic.



Application Load Balancer (ALB):

Path based routing, create multiple groups, create multiple targets where we can give customised path to route that.

Layer: Operates at the application layer (Layer 7) of the OSI model.

Protocol Awareness: Understands application-level protocols (e.g., HTTP/HTTPS).

Routing: Can make intelligent routing decisions based on content, such as URL paths, headers, and cookies.

Features: Supports features like SSL termination, WebSockets, and HTTP/2.

Use Cases: Ideal for web applications, microservices, and APIs where content-based routing is needed.

Network Load Balancer (NLB):

Same as ALB but support different protocols like TCP, UDP and TLS but used for best-performance as can handle millions of requests/second.

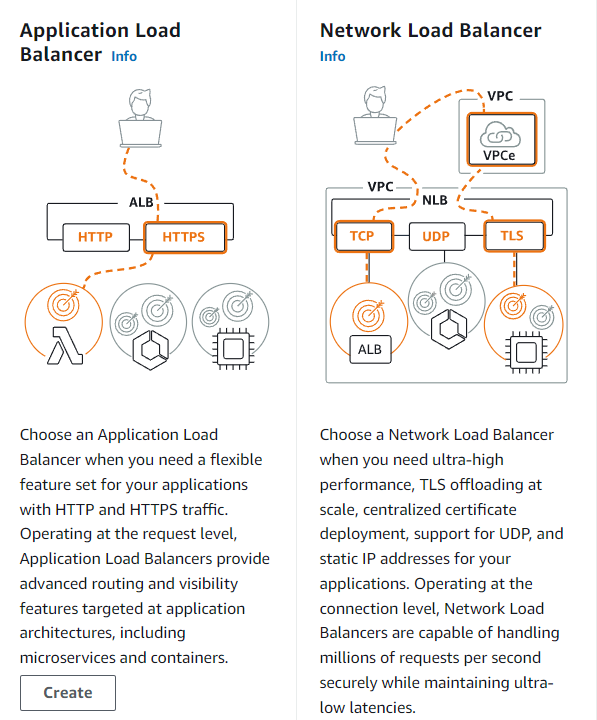
Layer: Operates at the transport layer (Layer 4) of the OSI model.

Protocol Awareness: Limited to basic transport protocols (e.g., TCP, UDP).

Routing: Makes routing decisions based on IP addresses and TCP/UDP ports.

Performance: Designed for handling high-throughput and low-latency applications, can handle millions of requests per second.

Use Cases: Suitable for applications that require extreme performance and need to handle TCP/UDP traffic, such as real-time streaming and gaming servers.



***Practical:***

1. Create 3 instance --> name, ubuntu, keypair, allow HTTPs and HTTP traffic from the internet, Advanced details --> add script

#!/bin/bash

apt update -y

apt install apache2 -y

service apache2 start

echo "Testing webserver-01" > /var/www/html/index.html

Check in browser,

instanceIP --> as HTTP default port is 80 so can give/ignore 80 after instance

1. sudo hostnamectl set-hostname server-01 --> changing host name from IP to server-01 after ubuntu (i.e., ubuntu@server-01)

Similarly, can do for other 2 servers after connecting it in CLI respectively so as to identify easily.

1. Create load balancer (classic)

Ec2 --> load balancer --> create classic --> name (demo-lb), select atleast 2-3 availability zones, instances ( add instances --> select all 3 instances --> confirm), enable cross-zone load balancing, enable connection draining --> create load balancer

Once load balancer is created, add http in inbound in security

[Note: once load balancer is created then it creates one DNS name]

[Health --> if server responds then healthy

Unhealthy --> if server doesn't respond then it's unhealthy

Unhealth--> 2, healthy --> 10

If 10 requests are responded 10 times then it's healthy so it will be sent to load balancer and if it don’t respond for 2requests then it's unhealthy

enable cross-zone load balancing --> used to distribute load to all zones.

enable connection draining --> used to detach instance if instance is failed for some reasons]

Select load balancer --> security --> security groups --> add inbound rules --> add http --> add --> copy dns name (details) --> paste in browser ---> keep refreshing(requesting) so servers will change every time based on availability (round-robin fashion)

1. ***Load Balancer (Application Load Balancer (ALB)):***

Similar to classic but need to add target here and has only 2 ports, http and https

Listening and routing --> create target group --> select instances --> target group name (tg-01), HTTP, Health Check (/) --> next --> instances (add 1st 2 instances to this target group) --> include as pending below --> create target group

Another target group

Listening and routing --> create target group --> select instances --> target group name (tg-02), HTTP, Health Check (/) --> next --> instances (add 3rd instance to this target group) --> include as pending below --> create target group

Listening and routing --> default action --> select tg-01 --> create load balancer

Add rule

Once created --> go to rules(1rule) --> add rule --> next --> add condition --> path (/order/\*) --> confirm --> actions (target group --> tg-01) --> next --> priority (1) --> next --> create

Add another rule

Once created --> go to rules(1rule) --> add rule --> next --> add condition --> path (/payment/\*) --> confirm --> actions (target group --> tg-02) --> next --> priority (2) --> next --> create

Copy dns name and paste it in browser URL to check:

If we don’t specify path then it goes to tg-01

Go to CLI and create order folder where index.html file should be present

Server1 and server2 --> order

cd /var/www/html

sudo mkdir order

cd order/

sudo vim index.html

<h1>this is order page server1</h1> --> server1

<h1>this is order page server2</h1> --> server2

sudo su --> switching to another user

Server3--> payment

cd /var/www/html

sudo mkdir order

cd order/

sudo vim index.html

<h1>this is payment page</h1>

* If no path is mentioned after dns name then it takes tg-01 as tg-01 is default server.
* If order path is given followed by dns name then it takes tg-01 as we gave order for tg-01.
* If payment path is given followed by dns name then it rakes tg-02 as we gave payment for tg-02.

[classic: it can get redirected within same application but can't get redirected to other applications.

ALB: if we want to link micro-applications in one application (mega-application) then we need to go with ALB.

Based on requirements, we need to choose load balancer]

***Day10***

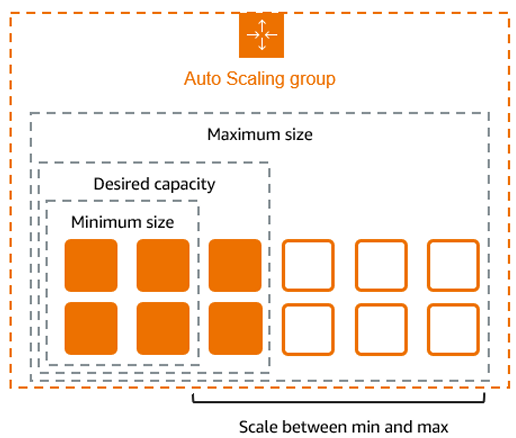
***Auto Scaling:***

Auto Scaling is a service offered by Amazon Web Services (AWS) that helps you automatically manage the number of EC2 instances (virtual servers) in your application based on demand. This means it can automatically scale your resources up or down to meet the changing needs of your application.

Here's how EC2 Auto Scaling works:

* Auto Scaling Groups: You create a collection of EC2 instances called an Auto Scaling group. You can specify the minimum, desired, and maximum number of instances for the group.
* Scaling Policies: You define scaling policies that tell Auto Scaling when to add or remove instances. These policies can be based on various metrics, such as CPU utilization, network traffic, or custom application health checks.
* Health Checks: Auto Scaling monitors the health of your instances and automatically replaces unhealthy ones to maintain your desired capacity.

For example, an e-commerce website experiencing high traffic during holiday sales. Auto Scaling would automatically add instances to handle the surge, ensuring website availability and performance.



Practical:

***Before creating Auto-scaling, need to create launch template,***

EC2 --> Instances --> launch template (similar to instance creation)

Create launch template --> name(demo-LT), description (demo-LT), Application and OS Images (browse more AMIs --> select ubuntu), Instance Type (t2.micro), key pair (create or give available keypair), Network settings (security group --> create or add created security group), Storage (8GB), Resource tags (Key: name, Value: ASG), Advance details (copy paste script:

#!/bin/bash

yes | sudo apt update -y

yes | sudo apt install apache2 -y

echo "<h1>Server Details</h1><p><strong>Hostname</strong>${hostname}</p><p><strong>IP Address</strong>$(hostname -I | cut -d " " -f1)</p>" > /var/www/html/index.html

sudo systemctl restart apache2

) --> launch template

***Creating Auto-Scaling Groups:***

EC2 --> Auto-scaling groups --> create ASG

Step1 - Chose launch template/configuration:

Name (demo-ASG), Launch template (select created template) --> next

Step2- Choose instance launch options (picked from launch template): Network (VPC --> default/choose created VPC, Availability Zones --> selecting 4 AVZ and subnets) --> next

Step3 - Configure advanced options:

Load balancing (choose created load balancer using 'Attach to an existing LB'/ create LB 'Attach to a new load balancer' --> LB type(application LB), LB name (demo-ASG), LB scheme (internet facing), Listeners (protocols: HTTP, Port:80, Default routing (create a target group)), Health checks (turn on Elastic LB health checks)) --> next

Step4 - Configure group size and scaling:

Desire capacity (min instance to maintain: 2) --> scaling (ranging instances to maintain and scale in/out within the range and limiting it, min desired capacity: 2, max desired capacity: 4), Instance maintenance policy (no policy) --> next

[Note:

Scale-in --> deleting instances

Scale-up --> creating instance]

Step5 - Add notifications

To get any notifications in mail when instances are created/deleted

Step6 - Add tags

Tags (Key: Name, Value: ASG-Demo) --> next

Step7: Review

Create auto scaling group

Select ASG --> activity (shows all activities performed by auto scaling, like logs)

Auto scaling will create --> Instances, Load balancer, target groups

Validate servers running and on which instance:

Copy paste load balancer DNS in browser URL, refresh to check instances on which application is running

Validate instances added to target groups:

EC2 --> Load Balancing (Target groups) --> select target group --> details

[Note: In auto-scaling when we delete any instance manually, we can see it in activity under auto-scaling. Even though we delete instance, auto-scaling will create instances by itself again, will create min. desired capacity no of instances…In our case, it's 2 which will help out to keep servers up]

Day11

* 1. ***Creating auto-scaling using Image(AMI):***

* 1. Create image of instance (AMI)

Create image of instance having any server in it like apache2, nginx or tomcat (select instance --> actions --> Image and template --> create image)

* 1. Step2: Create launch template

Name and desc (ASG-LT), OS (My AMIs --> select created AMI), Instance type (t2.micro), keypair(select available/create), network settings (security groups --> Select existing SG (MySG)/create SG) --> create LT

* 1. Now create Auto-scaling

* 1. ***Auto-Scaling policies:***

* 1. Dynamic scaling policies
     1. Target tracking policy:

Increase and decrease the current capacity of the group based on an Amazon CloudWatch metric and a target value.

Auto-scaling will decide when to create instances.

Target is tracked by cloudwatch

Metrics:

* 1. ASGAverageCpuUtilization: Average CPU utilization of the Auto Scaling group
  2. ASGAverageNetworkIn: Average number of bytes received by a single instance on all network interfaces.
  3. ASGAverageNetworkOut: Average number of bytes sent out from a single instance on all network interfaces
  4. ALBRequestCountPerTarget: Average Application Load Balancer request count per target.

[Note: Target tracking Automatically creates cloudwatch to monitor the metrics]

* 1. Step scaling policy:

Increase and decrease the current capacity of the group based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.

Requires to create cloudwatch explicitly.

Eg) We can configure as per step Vise, when to create instance

* 1. Step1: when ASGAverageCpuUtilizaion b/w 30-40%, add 1 ec2 instance
  2. Step: when ASGAverageCpuUtilizaion b/w 50-60%, add 1 ec2 instance
  3. Step3: when ASGAverageCpuUtilizaion b/w 70-80%, add 1 ec2 instance

* 1. Simple scaling policy:

Increase and decrease the current capacity of the group based on a single scaling adjustment, with a cooldown period between each scaling activity.

Requires to create cloudwatch explicitly.

We can decide when to create instances.

eg) we can configure how many instances should be created when it reaches threshold point.

when ASGAverageCpuUtilization = 50%, add 2 instances

* 1. Predictive scaling policies:

Predictive scaling uses machine learning to predict future traffic and adjust the number of instances in advance to handle anticipated demand.

* 1. Scheduled actions policies:

automatically adjusts the capacity of your Auto Scaling group based on a predetermined schedule

Example: Increasing the number of instances every Monday at 9 AM and reducing them every Friday at 5 PM to handle weekly business cycles.

***Practical:***

***Target tracking scaling:***

EC2 --> auto-scaling group --> select created Auto-scaling --> Automatic scaling

Create dynamic scaling policy --> policy type (Select Target tracking scaling), scaling policy name (Target tracking policy), Metric type (Select Average CPU Utilization), Target Value (50), Instance warmup (60) --> create

[Note: creating Target Tracking scaling will create CloudWatch automatically

Cloudwatch --> used to monitor resources]

[Note: can create Target tracking policy while creating Auto-Scaling (or) can create it after creating Auto-scaling groups]

Validate the policy:

Login to instance in CLI,

Install stress tool to increase CPU utilization/load,

sudo apt update

sudo apt install stress -y

top --> check CPU utilization

sudo stress --cpu 12 --timeout 240s

top --> check again for cpu utilization

Check CloudWatch in frontend (AWS)

AWS --> CloudWatch --> All alarms --> can see target group would have created instance when state comes to in-alarm

(since, while giving stress we have given time as 240s Target tracking policy will wait for 240s, if still CPU utilization is more than target value(we have given as 50) then automatically instances will be created to distribute load)

Check in activity in Auto-scaling for instances logs

***Target tracking scaling --> Step scaling policy:***

Create dynamic scaling policy,

Policy type (simple scaling), Scaling policy name (Target-SSP)

***Create cloudwatch:***

Step1: Specify metric and conditions

Metric --> Metric name (CPUUtilization), AutoscalingGroupName (demo-ASG), Statistic (Average), Period (5min) --> Conditions (Threshold type (Static), whenever CPUUtilization is (Greater), than (30)) --> next

[Note: will check CPU Utilization for 5min if it crosses target value, if it's still more than target value then alarm will get triggered inturn creates instances]

Step2: Configure actions

Add auto-scaling policies --> Auto scaling actions (Alarm state trigger (in alarm), Resource type (EC2 ASG), select ASG, take the foll action (can add multiple actions) --> add ec2 instances (select multiple instances)

Step3: Add name and description

Step4: Preview and create

***Target tracking scaling --> Simple scaling policy:***

Create dynamic scaling policy

Policy type (simple scaling), Scaling policy name (Target-SSP), Cloudwatch alarm (create and add), Take the action (Add, instance(2), capacity units), and then wait (100)

[Note: If alarm triggers then 2 instances will be added and can create multiple alarms, one alarm for adding(state - alarm-in) and one for removing (state - ok)]

[Note: need to create cloudwatch for Simple and Step]

Day12

Assignment:

Instance refresh

Without Down-time (without affecting applications) and with down-time

***1) Instance Refresh:***

* Instance Refresh is a powerful feature in AWS Auto Scaling that allows you to update instances in an Auto Scaling group with minimal disruption to your application.
* It's a controlled process where new instances are launched, validated for health, and then old instances are terminated.
* This is particularly useful for updating the Amazon Machine Image (AMI), changing instance types, or updating instance configurations.

**Use Cases**

* Deploying new AMIs or user data scripts: Update application code or configuration without manual intervention.
* Migrating instance types: Optimize performance or reduce costs by switching to new instance types.
* Switching to launch templates: Modernize Auto Scaling group configuration.
* Rolling back failed deployments: Revert to the previous configuration in case of issues.

[Note: When we want to update/make any changes in application/configurations of Instance we use Instance refresh. First, we will make all the required changes in instance and take image(AMI) of it. Once taken, then we will create launch template from AMI and replace it with production Instance reducing down-time. If it takes more time for the process then client can take 15-30 min maintenance break so updated instance will be attached in Auto-Scaling. Once, updated instance is attached then old instance will be terminated]

***Update AMI:***

EC2 --> launch template --> actions --> Modify template (create new version) --> OS (change created AMI) and if any changes in instance can be changed like volume (size), instance type, etc --> create template

Auto-scaling group --> select created ASG --> Instance refresh --> warmup (50) --> Desired configuration - enable Update launch template (Launch Template (Version (2)) --> start instance refresh

Check in EC2 instances --> can see new instances will be created

Instances will be updated automatically in Target groups and AMI will be updated in Load Balancer

***2) Identity and Access Management (IAM):***

fundamental service for securely controlling access to AWS resources for user Internal and External. It allows you to manage users, groups, and roles, as well as define permissions that control what actions these identities can perform on AWS resources.

***Key Components of IAM***

* Users: Individuals who directly access AWS resources using access keys and secret access keys.
* Groups: Collections of users that can be managed as a single unit.
* Roles: Permissions that can be assumed by users or other AWS entities, providing temporary access to resources.
* Policies: Define the permissions granted to users, groups, or roles.

***(i) User:***

AWS --> IAM --> user --> create users --> Enable IAM Identity Centre --> Enable with AWS Organizations --> continue

**User**

Step1: Specify user details

User details -->

username(yatish) --> enable "Provide user access to the AWS Management console" --> choose "Specify a user in Identity Center - Recommended" --> next

Manage in identity center

Primary information

username (devops) --> password (enable "Generate a OTP that can share with user") --> fill Email address and other details --> next

Step2: Add user to groups

If any group created/ want to create then can create

Step3: Review and add user

Done

Pop-up screen will come (OTP)

AWS access URL, username and OTP --> make note of it

AWS --> IAM --> users

Step1: Specify user details

username(test123) --> enable "Provide user access to the AWS Management console" --> select "I want to create an IAM user --> con sole password (custom password (test123)) --> disable "user must create a new password at next sign-in -->next

Step2: set permissions

Attach the policies (policies are permissions for resources) --> select "AmazonEC2ReadOnlyAccess (only view/ read), AmazonS3ReadOnlyAccess" --> next

[Note: There are 1000+ policies that can be selected and given to users so can perform only those operations,

eg) AmazonEC2FullAccess (permission to create EC2 instance, load balancer, auto0scaling, cloudwatch)]

Step3: Review and create

Create user

[Note: once user is created then URL, username and console password can be seen so download it as we can't get that back once it's gone]

Validation: Login using above URL and credentials (or) can login in IAM in AWS login

We can just view/read EC2 and S3 services, can't do others operations like adding, deleting, etc.

(ii) Similarly, we can do it for group

(iii) ***Role:***

AWS --> IAM --> role --> create role

Step1: Select trusted entity

Trusted entity type(AWS service) --> Use case (EC2) --> next

Step2: Add permissions

AmazonS3FullAccess, AWSCodeCommitFullAccess, AWSLambda\_FullAccess --> next

Step3: Name, reviews and create

Role name (my\_custom\_role), Tag (Key:Name, Value:Full access) --> create role

Use case: login to instance in CLI and install aws cli

sudo apt update

sudo apt install unzip

***Install aws cli***

curl "<https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip>" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

unzip awscliv2.zip

cd aws

sudo ./install

aws s3 ls

configuring configuration in aws in cli

We need access key and secret key to configure aws in cli

**AWS Console**

*IAM -->* users --> test123 --> create access key

Step1: Access key best praises and alternatives

use case (CLI), tick "I understand the above recommendation and want to proceed to create an access key" --> next

Step2: Set Description tag

Tag (test) --> create access key

Once created, note down access key and secret key as it will show once

**CLI**

aws configure

AWS Access Key ID: copy paste created access key here

AWS Secret Ket: copy paste secret key here

Default region name: us-east-1(give region of instance)

Default output format: enter

Once we configure, it(conf and credentials) will be stored in .aws folder

cd

cd .aws

Ls

cat config

cat credentials (to check credentials like access key and secret key)

aws s3 bucket

**Create bucket in S3**

AWS --> s3 --> create bucket --> Bucket name (demobucket.test) --> create bucket --> upload some file

Can add files/folders inside bucket

[Note: Bucket is like pendrive]

aws s3 ls

(any files uploaded/ buckets created in S3 will be stored here)

Now, remove access of AmazonS3FullAccess in test123 user in AWS Console and if we come and check again in CLI, AccessDenied will show

[Note: This is configuration of AWS w.r.t user in instance level]

Remove .aws (rm -r .aws) and attach role to instance and check in CLI

**Attach role that's created to instance and check**

AWS Console (Attaching role to instance)

Select Instance --> actions --> security --> Modify IAM

IAM Role (select created role --> my\_custom\_role) --> update IAM role

CLI

aws ec2 describe-instances

(can see details of one instance, all details like instance ID, private IP, public IP, etc)

[Note: Role can't be used for external, service to service and used for internal

User is for Internal and External]

***Can use policy given by AWS (or)***

***Can Create policy (permissions) so can be attached to Role:***

IAM --> roles --> add permissions --> create inline policy

Step1: Specify Permissions

service --> can choose services like EC2, S3, etc

Manual actions --> can select what access to given for that service like List, Read, Write, Permission management and Tagging

Step2: Review and create

Policy name (my\_policy) --> create policy

Once policy is created then it will be added to the role

Practise IAM:

One role,

one user

Management console

CLI console

Day13

1. Cloudwatch:

It is monitoring service for AWS resources and applications. It enables you to collect and track metrics, monitor log files, set alarms, and automatically react to changes in your AWS resources.

AWS --> cloudwatch --> in alarm --> create alarm

select metric --> AWZ (Mumbai), Application ELB --> Per AppELB, per TG Metrics (321 metrics there) --> can give like "UnhealthyStateRouting" "UnhealathyStateDNS", etc and watch these

Cloudwatch --> in alarm --> create alarm --> select metric --> EC2 --> Per Instance metric --> paste Instance ID --> select CPUUtilization --> select metric -->

Step1: Specify metrics and conditions

threshold (40)

Step2: Configure actions

In alarm --> Select an existing SNS topic (snsdemo) --> next

Step3: Add name and description

Alarm name (Instance\_CPU\_crossed) --> Alarm description (#Alarm triggered for instance

This is SNS instance) --> next

Step4: Preview and create

Create alarm

Check cloudwatch alarm: login to instance in CLI

Give stress to the instance,

sudo apt update -y

sudo apt install stress

sudo stress --cpu 12 --timeout 240s

Top

Check in AWS

Go to instance --> monitor --> check CPUUtilization graph

Cloudwatch --> in alarms

Once the state is in "in alarm" then it will send email notification to provided maiID

Similarly, can watch any metrics based on requirements for any services

1. SNS:(Simple Notifications Service)

web service which makes it easy to set up, operate, and send a notification from the cloud to users/groups.

AWS --> SNS

Topic

Topic name (snsdemo) --> Next step

Type (Standard) --> Display name (snsTest) --> create topic

Subscription

Create subscription --> Topic ARN (snsdemo), Protocol (Email), Endpoint (mailID) --> create subscription

[Note: go to mailID and confirm after which subscription status will be confirmed]

Steps to send mail notification for metrics:

(i) create topic and subscription in SNS

(ii) create metrics under in-alarm for instance in cloudwatch

(ii) give stress in CLI for the instance so once threshold point is exceeded then mail will be sent automatically

1. ***Cloudtrail:***

Tracking user activity with respect to account level. Logs will be stored for 90days. If we want logs to be stored for more than 90 days then we need to create CloudTrail.

CloudTrail is a web service that records API (Application Programming Interface) activity in your AWS account.

It continuously logs and monitors the activities and actions across your AWS account. It also provides the event history of your AWS account including information about who is accessing your AWS services.

Logs can be checked in events history in CloudTrail

AWS --> CloudTrail --> create cloud trail

Trail name (CT\_demo) --> trail log bucket and folder (bucket inside bucket where all log files will be stored) --> create trail

Can create multiple trails for multiple actions,

Step1: choose trial attributes

Create trail --> trial name (test1) --> create s3 bucket --> SNS topic (choose created topic) --> AWS KMS alias (test) --> next

[Note: In organization level, "enable for all accounts in organization"]

Step2: Choose log events

Event type (Management events) --> Management event (API activity --> read, write, exclude AWS KMS events) --> next

Step3: Review and create

create trial

Difference between cloudwatch and cloudtrial?

|  |  |  |
| --- | --- | --- |
| Feature | CloudWatch | CloudTrail |
| Focus | Monitoring performance and health | Recording API calls and user activity |
| Data | Metrics, Logs, Events | API calls, Data events |
| Actions | Alarms, Dashboards, Trend analysis | Security auditing, Compliance, Troubleshooting, Cost optimization |
| Analogy | Car dashboard | Car black box |

1. EFS (Elastic File System):

fully managed, scalable file storage service provided by AWS. It offers shared file storage that can be accessed by multiple EC2 instances simultaneously.

EFS is designed for high throughput and low latency, making it suitable for various workloads.

EFS will charge based on how much we use only unlike EBS which used to charge for volume created.

EFS is slower than EBS comparatively, in terms of transferring.

AWS --> EFS --> select/click created EFS --> Attach --> mount via DNS -->

AWS Console

AWS --> EFS --> click created EFS --> network --> copy security groups

EC2 --> select instance --> actions --> security --> change security groups --> paste security groups --> add security groups --> save

CLI: Login to the instance in which we want to mount EFS

sudo apt update

mkdir efs

sudo apt install nfs-common

sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-04c61a697a8570565.efs.ap-south-1.amazonaws.com:/ efs

df -h

sudo cp -r \* efs/

cd efs/

ls

du -sh #check size of drive

Attach EFS for another instance: Login to 2nd instance

sudo apt update

sudo apt install nfs-common

sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-04c61a697a8570565.efs.ap-south-1.amazonaws.com:/ efs

df -h #size of filesystems

cd efs

ls

Once, mounted in multiple instances then whatever changes are made in 1 instance will be reflected in another as multiple instances are mounted in same EFS storage.

[Note: EFS capacity: 8.0E (Exabyte)

1extabyte = 1,073741824 GBs]

Day14

1. ***Amazon S3 (Simple Storage Service):***
   * ***S3 is a scalable, durable and cost-effective object storage service offered by Amazon Web Services (AWS). It is designed to store and retrieve any amount of data from anywhere on the web, making it ideal for use cases like data backup, web application hosting, etc.***
   * ***S3 is cost-effective as it's less expensive than EBS and EFS.***
   * ***It is global level as we can get same data in all regions.***

Diff b/w EBS, EFS and S3 storage

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | EBS | EFS | S3 |
| Storage Type | Block | File | Object |
| Access | Single EC2 instance | Multiple EC2 instances | Publicly accessible |
| Performance | High | Medium | Varies |
| Scalability | Up to 16 TB per volume | 8 exabytes per file | Virtually unlimited |
| Cost | Higher | Medium | Lower |
| Use Cases | Boot volumes, databases | Shared file systems, content management | Backup, static websites, content delivery |

AWS --> S3

General purpose buckets --> create bucket

Bucket type (General purpose) --> Availability zone (demo-s3bucket) --> Object ownership ("ACLs disabled" if we have one account) --> Tags (Key:Name, Value:myBucket) --> create

[Note: ACL = Access control lists (manage access to S3 buckets and objects like read, write and delete)

Block all public access --> if enabled then it will block all public access and if disabled then public can access the bucket]

Create folder inside bucket

Create folder --> folder name --> test --> create

Can Upload any files/data

1. ***Public access to bucket config***

S3 Bucket general config to make public access

***Method1: configuring bucket using ACL for individual objects***

If we want to make bucket public then we need to disable "Block all public access" and need to enable "ACLs enabled" under ownership.

S3 --> Bucket --> permissions

ACL --> edit --> everyone (public access): objects(list), Bucket ACL (read) --> save changes

To make individual object in bucket public access

Bucket --> Select object --> Actions --> manage public using ACL --> make public

Object Public Access

Click on object --> click URL to download file or copy paste URL in browser to open in browser without downloading

***Method2: configuring bucket using policy for all objects***

Bucket --> permissions --> Bucket policy --> edit --> policy generator

Step1: Select policy type

S3 bucket policy

Step2: Add statements

Effect (allow), principal (\* --> all (or) copy paste ARN of user/group/role)

[find ARN(Amazon Resource name) of user/group/role,

AWS --> IAM --> click user/group/role --> details --> copy ARN]

Actions (GetBucketPolicy), ARN(bucket ARN)

[to find ARN of bucket,

Select bucket --> properties --> copy ARN]

Step3: Generate policy

Generate policy (policy will be generated)

Once generated then add it in bucket --> permissions --> bucket policy --> edit --> paste generated policy --> save changes

1. ***Bucket Versioning:***

* Bucket Versioning is a feature in Amazon S3 that automatically saves previous versions of an object in the same S3 bucket when it is overwritten. This helps protect against accidental data deletion and provides a way to restore previous versions if needed.
* If we don't enable versioning then we can't track files and can't recover/roll back any files if deleted. If enabled then can retrieve back (roll back) deleted file.

Step1: enable version while creating bucket

Step2: click on bucket --> go to "show versions"

Recover back deleted file

bucket --> show versions --> select deleted file (Delete marker) --> delete option --> permanently delete objects (delete marker) --> delete objects

Once, we delete marker then we can recover the file which was deleted. This can be done only of versioning will be enabled.

1. ***Encryption***

Data encryption is a process for securing data by encoding information. Data is encoded using a password or an encryption (cypher) key and special encryption algorithms.

If an unauthorized person gets access to the encrypted data, the data is unreadable without the key or password.

3types:

1. Server-side encryption with Amazon S3 managed keys (SSE-S3)
2. Server-side encryption with AWS key management service keys (SSE-KMS)
3. Dual-layer server-side encryption with AWS key management service keys (DSSE-KMS)

SSE-S3: Uses AWS managed keys to encrypt data at rest. It's a simpler option with less control over key management.

SSE-KMS: Uses AWS KMS to manage encryption keys. It provides more control over key rotation and management.

DSSE-KMS: Employs double encryption using both an AWS managed key and a KMS managed key for enhanced security.

1. ***Lifecycle rules:***

S3 Lifecycle Rules are a set of automated actions that you can configure to manage the lifecycle of objects in your Amazon S3 bucket. These rules help optimize storage costs and ensure data compliance.

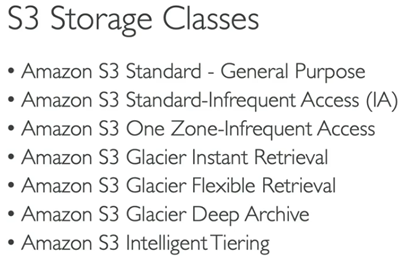
This is a cost-saving practice that can move your files to AWS Glacier (The AWS Data Archive Service) or to some other S3 storage class for cheaper storage of old data or completely delete the data after the specified time.

Create lifecycle

S3 --> bucket --> management --> lifecycle rules --> create

Lifecycle rule name (test) --> Prefix (\*) --> object size (specify minimum object size (1, mb), specify maximum object size (10, mb)) --> Lifecycle rule actions (Move current versions of objects b/w storage classes) --> choose storage class transitions (Standard-IA), Days after object creation (30) --> add transition (choose storage class transitions (one zone-IA), Days after object creation (60)) --> add transition ( choose storage class transitions (Glacier Deep Archive), Days after object creation (90)) -->

[Note: can add multiple transitions so object will be moved to that storage once 30 days, 60 days, 90days,etc., are over so as to decrease price/billing]



Day15

* 1. ***Global service (s3), zone specific services, region specific service***

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Global Services** | **Regional Services** | **Zone-Specific Services** |
| Scope | Accessible from anywhere in the world. | Available within a specific geographic area. | Limited to a specific Availability Zone (AZ) within a region. |
| Latency | Generally higher due to geographic distance. | Lower latency for users within the region. | Lowest latency for highly localized access. |
| Fault Tolerance | High availability through redundancy across regions. | Fault tolerance through redundancy within a region. | High availability through redundancy within an AZ. |
| Data Residency | Data is stored regionally, but bucket names are globally unique. | Data is stored regionally. | Data is stored regionally and within a specific AZ. |
| Scalability | Highly scalable to meet global demands. | Scalable within a region. | Limited scalability within an AZ. |
| Use Cases | Content delivery, static data storage, backups, logs. | Compute resources, databases, virtual networks. | Critical components of highly available systems. |
| Examples | Amazon S3, IAM, Route 53, CloudFront | Amazon EC2, RDS, VPC, Lambda, EC2 | Amazon EBS volumes, Network interfaces |

* 1. ***Encryption:***

Data encryption is a process for securing data by encoding information. Data is encoded using a password or an encryption (cypher) key and special encryption algorithms.

If an unauthorized person gets access to the encrypted data, the data is unreadable without the key or password.

3types:

* 1. Server-side encryption with Amazon S3 managed keys (SSE-S3)
  2. Server-side encryption with AWS key management service keys (SSE-KMS)
  3. Dual-layer server-side encryption with AWS key management service keys (DSSE-KMS)

SSE-S3: Uses AWS managed keys to encrypt data at rest. It's a simpler option with less control over key management.

SSE-KMS: Uses AWS KMS to manage encryption keys. It provides more control over key rotation and management.

DSSE-KMS: Employs double encryption using both an AWS managed key and a KMS managed key for enhanced security.

* 1. ***object lock:***

Permanently allows objects in this bucket to be locked. Additional Object Lock configuration is required in bucket details after bucket creation to protect objects in this bucket from being deleted or overwritten.

* 1. ***web hosting:***

Web hosting is a service that allows individuals and organizations to make their website accessible on the internet.

When you decide to create a website, you need a place to store all the files that make up your site. This is where web hosting comes in.

Step1: create s3 bucket

create s3 bucket disabling block public (giving access to public)

Step2: enable Static website hosting

click on bucket --> properties --> Static website hosting --> edit --> enable "Static website hosting", Index document (index.html), Error document (error.html) --> save changes

Once changes are made, then we can get Bucket website endpoint (URL)

Step3: add bucket policy

Permissions --> Bucket policy --> edit --> copy paste policy code

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::test-webhosting-ol/\*"

}

]

}

Step4: upload files into bucket

upload index.html file into bucket from local system

Similarly, upload error.html file into bucket from local system

Step5: validation

Paste Bucket website endpoint (URL) in browser

If .html file matches then index.html file will be executed or else if doesn't match then error.html file will be executed

[Note: index.html and error.html file must match document in properties (static website hosting) of bucket]

* 1. ***Lambda:***

Lambda is a serverless compute service that lets you run code without provisioning or managing servers. With AWS Lambda, you can run code for virtually any type of application or backend service with zero administration.

Use case: we can start or stop instances by providing crone job (Triggers) which can reduce cost of company.

Can take AMI's of instances at end of the day

**Practical:**

Before Lambda, we need to create inline-policy and roles in IAM,

Policy

IAM --> policies --> create policy

Step1: specify permissions

Service (EC2) --> allowed functions (write --> StartInstances, StopInstances) --> Resources (Any in this account) --> next

Step2: Review and create

Policy name (ec2-start-stop-inline-policy) --> create policy

Roles

IAM --> roles --> create roles

Step1: Select trusted entity

AWS service --> Service or use case (Lambda) --> next

Step2: Add permissions

Policy name (AWSLambdaFullAccess, ec2-start-stop-inline-policy) --> next

Step3: Name, review and create

Role name (lambda-role-for-instance) --> Description (Allow Lambda functions to call AWS services on your behalf) --> create role

AWS --> Lambda --> create function

Author from the scratch --> function name (my-demo-instance-function) --> runtime (Python 3.12) --> change default execution role (use an existing role --> lambda-role-for-instance) --> create function

Copy paste below code in code:

[

import boto3

region = 'ap-south-1'

instances = ['i-02df39e715a5af15a', 'i-0cb06d9ff6a978660']

ec2 = boto3.client('ec2', region\_name=region)

def lambda\_handler(event, context):

ec2.stop\_instances(InstanceIds=instances)

print('stopped your instances: ' + str(instances))

]

Test --> create new event --> event name (demo-stop) --> save --> deploy

[Note: change timing for initiation as by default it will be 3sec

Configuration --> general configuration --> edit --> timeout (10) --> save]

Test

Similarly, test for starting instances

[

import boto3

region = 'ap-south-1'

instances = ['i-02df39e715a5af15a', 'i-0cb06d9ff6a978660']

ec2 = boto3.client('ec2', region\_name=region)

def lambda\_handler(event, context):

ec2.start\_instances(InstanceIds=instances)

print('started your instances: ' + str(instances))

]

Adding triggers (Crone jobs)

After creating function --> add trigger --> Trigger configuration (eventbridge --> Cloudwatch events) --> create new rule --> rule name (my-start-rule) --> rule type (schedule expression) --> schedule expression (cron(22 04 ? \* MON-FRI \*)) --> add

cron(22 04 ? \* MON-FRI \*) = 22min, 4am(hours), ?=no specific day-of-month, \*=every month, run from Mon-Fri, \*=every year (Run the task at 4:22 AM UTC every Monday, Tuesday, Wednesday, Thursday, and Friday.)

[Note: time is as per UTC not IST

\* = minutes

\* = hours

\* = day-of-month

\* = month

\* = day-of-week

\* = year]

Day16

Diff b/w jenkins and AWS pipeline?

Developer Tools --> codebuild, codecommit, codedeploy, codepipeline

***Codepipeline:***

It is continuous delivery service used to model, visualize, and automate the steps required to release your software. CodePipeline automates the steps required to release your software changes continuously.

Use case:

Continuous Integration and Continuous Delivery (CI/CD): Automate the build, test, and deployment process for frequent code change.

***Practical:***

***(1) Manual: deploying using CLI***

Create instance --> linux, network settings (select existing security group) --> launch instance

CLI:

login to instance

vim Script.sh #prerequisites code

(#!/bin/bash

sudo yum -y update

sudo amazon-linux-extras install epel -y

curl -sL <https://rpm.nodesource.com/setup_6.x> | sudo -E bash -

sudo yum install nodejs -y

sudo npm install express

sudo yum -y install ruby

sudo yum 0y install wget

cd /home/ec2-user

wget <https://aws-codedeploy-ap-south-1.s3.amazonaws.com/latest/install>

sudo chmod +x ./install

sudo ./install auto)

chmod +x script.sh

sudo su

./script.sh

ls

sudo yum install git -y

git clone "paste repo link"

cd AWS-Code-Commit-Deploy

ls

cd scripts

ls

chmod -R 777 ../scripts/

ls

./before\_install.sh

cd ..

ls

cd ..

ls express-app/

ls

cd express-app/

cp -r ../AWS-Code-Commit-Deploy/\* .

ls

cd scripts

./application\_start.sh

npm --version

cd ..

ls

node app.js

netstat -na|grep 3000

Browser:

ipaddress:3000

ipaddress:3000/product

***(2) Automation: Codepipeline and codedeploy:***

Codepipeline

Developer tools:

(i) Source

(ii) Artifacts

(iii) Build

(iv) Deploy

(v) Pipeline

Before deploying we need create 2 roles in IAM, one for EC2 and another for CodeDeploy.

and EC2 instance connected with EC2 Role (IAM) to deploy code and run in server,

(i) EC2 role

IAM --> roles --> create role

Step1: select trusted entity

AWS service --> Service or use cases (EC2) --> next

Step2: Add permissions

AmazonEC2RoleForSSM, AmazonSSMFullAccess --> next

Step3: name, review and create

Name(role-for-Codedeploy) --> create role

(ii) CodeDeploy role

IAM --> roles --> create role

Step1: select trusted entity

AWS service --> Service or use cases (CodeDeploy) --> next

Step2: Add permissions

AWSCodeDeployRole, AWSCodeDeployFullAccess --> next

Step3: name, review and create

Name(role-for-CD) --> create role

(iii) create instance to deploy code

Create instance --> linux, network settings (select existing security group) --> advanced details (IAM instance profile --> role-for-codedeploy) --> copy paste below code for installing required prerequisites

(#!/bin/bash

sudo yum -y update

sudo amazon-linux-extras install epel -y

curl -sL <https://rpm.nodesource.com/setup_6.x> | sudo -E bash -

sudo yum install nodejs -y

sudo npm install express

sudo yum -y install ruby

sudo yum 0y install wget

cd /home/ec2-user

wget <https://aws-codedeploy-ap-south-1.s3.amazonaws.com/latest/install>

sudo chmod +x ./install

sudo ./install auto)

--> launch instance

(iv) create application (codedeploy)

Developer tools --> CodeDeploy --> applications --> create application

Application name (Express-App-Node-JS) --> compute platform (EC2) --> create applications

Deployment group --> create deployment group

Deployment group name (My-deployment\_express-group) --> service role (role-for-CD) --> deployment type (In place) --> environment configuration (Amazon EC2 instances) --> Tag group (Key:Name, Value:InstanceName) --> Agent configurations (Install AWS CodeDeploy Agent --> never) --> deployment settings (COdeDeployDefault\_AllAtOnce) --> disable load balancing --> create deployment group

(v) Pipeline --> get started --> create pipeline

Step1: create pipeline settings

Pipeline name (My-express-app-pipeline) --> execution mode (Superseded) --> next

Step2: add source stage

Source provider (GitHub (version2)) -->connection (connect to GitHub --> create GitHub app connection (My-GitHub-connection) --> connect to GitHub --> authorise AWS connector --> Install a new app --> login to GitHub account --> select repository (select repo having source code) --> save --> taken no will be generated --> connect) --> select created GitHub-connection (copy paste arn) --> Repository name (choose repo) --> default branch (main) --> trigger (no filter) --> next

Step3: add build stage

Skip build stage

Step4: add deploy stage

Deploy provider (AWS CodeDeploy) --> region (select working region) --> Application name (Express-App-Node-JS) --> next

Step5: review

Create pipeline

Check logs

Deploy --> view details --> click execution details link

Check in browser

instanceIP:3000

instanceIP:3000/products

Day17

WAF:

AWS WAF is used to protect application server/web application server from common web exploits.

Rules: creating own rules

Practical:

(i) Create 2 instances for Load Balancers to check different servers

Launch instance --> Instance name --> Ubuntu --> keypair --> allow traffic for http and https --> advanced details (add script)

#!/bin/bash

sudo apt update -y

sudo apt install apache2 -y

echo "WPL test" > /var/www/html/index.html

sudo service apache2 restart

validate:

copy paste instanceIP in browser

instanceIP

(ii) Create Load balancer

Create load balancer --> application LB

LB name (demo-ALB) --> VPC (select default) --> mappings (select atleast 3 availability zone or subnet) -->

create security groups (name (demo-sg) --> description (demo-sg) --> inbound (http, anywhere IPv4) --> create SG)

--> select created SG (demo-sg) -->

Create target group (instances --> TG name (demo-tg) --> next --> select 2 instances --> click on "include as pending below" --> create TG) -->

Select created TG (demo-tg) --> create LB

Validate:

Connect to respective instances and change in index.html to check

Instance1:

CLI --> connect instance --> vim /var/www/html/index.html

WLF test server1

Instance2:

CLI --> connect instance --> vim /var/www/html/index.html

WLF test server2

Copy load balancer DNS name and paste it in browser,

Refresh page to see both servers

(iii) WAF:

Create web ACL

Step1: Describe web ACL and associate it to AWS resources

Resource type (Regional resources) --> region (choose working region) --> name (demo-WACL) -->

Add AWS resources (resource type (Application load balancer) --> choose "demo-ALB" --> add --> next

Step2: Add rules and rule groups

Rules --> Add rules (Add managed own groups --> select rules as per requirement (enable "Admin protection" and "amazon IP reputation list")) --> add rules -->

Default web ACL action for requests that don't match any rules (can choose any based on what action we want "Allow" or "Block" (allow)) --> next

[Note: can add rules, maximum of 5000 capacity

More capacity = more rules]

Step3: set rule priority

next

Step4: configure metrics

next

Step5:Review and create web ACL

Create web ACL

(iv) IP sets

WAF & Shield --> IP sets --> create IP sets

Name (demo-IPset) --> IP addreses (copy paste systemIP --> systemIP/32 (for 1 IP, IP/31 for 2 IP (2^1))) --> create IP set

[Note: myIP in browser to get systemIP]

(v) attach IP set to WAF:

Web ACLs --> click on created WAF --> rules --> add rule --> add my own rules and rule group -->

rule type (IP set) --> name (demo-rule1) --> IP set (select created IP set) --> Action (Block) --> add rule

--> save in WAF

Validate:

copy paste load balancer DNS name in browser

(can see forbidden as application is blocking our IP which we have set in rule)

(vi) Changing rule (Block to Captcha)

Web ACL --> select created rule (demo-rule1) --> edit --> change action from block to CAPTCHA --> save rule

Validate:

LB DNS name --> can see captcha,

once chosen correct captcha then application will be opened

Assignment:

Cloudfront and WAF, load balancer, instance and application inside instance

Day18

***(1) CloudFront:***

CloudFront is a Content Delivery Network (CDN) service provided by Amazon Web Services (AWS).

It's designed to speed up the distribution of our static and dynamic web content to users around the world with low latency and high transfer speeds, such as images, videos, and HTML files.

How does it work?

CloudFront has a vast network of data centers called edge locations distributed globally.

When a user requests content, CloudFront directs the request to the nearest edge location that has the content cached. This significantly reduces latency and improves loading times for users.

Practical:

(i) create S3 bucket

Bucket name (cf-bkt) --> ACL's enabled --> disable "Block all public access" --> bucket key (disable) --> create bucket

Upload file --> any image/HTML file --> select object --> actions --> make public using ACL --> upload

Validate:

Click on uploaded object --> copy paste URL in browser (can see some latency while loading)

(ii) create cloudfront

Create distribution --> origin domain (select S3 bucket source) --> web application firewall (Do not enable security protections) --> create distribution

Validate:

Copy paste distribution domain name URL in browser followed by object name in bucket

[Note: first time there will be some latency but latency will be low if we keep on using it]

***(2) AWS CLI:***

* + AWS CLI (Command Line Interface) is a powerful tool that allows you to interact with various AWS s ervices directly from your command line. It provides a consistent interface for mana
  + ging your AWS resources, enabling you to automate tasks, script interactions, and perform complex operations efficiently.
  + Can perform all operations/actions in CLI that we used to perform in AWS and also it's faster in CLI.

***Practical:***

Create ubuntu instance in AWS and connect it in CLI (GitBash).

* + ***S3***

(i) export --> To define variables (local system)

|  |  |
| --- | --- |
| eg) export user=ubuntu@InstanceIP  echo $user | # defining variablee  #cdisplaying variable |

|  |  |
| --- | --- |
| (ii) Install AWS   * + sudo apt update -y   + sudo apt install awscli   + sudo apt install unzip   + curl "<https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip>" -o "awscliv2.zip"   + unzip awscliv2.zip   + sudo ./aws/install   + aws --version | #install aws cli                  #validate |

[Note: documentation to install aws cli

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>]

|  |  |
| --- | --- |
| (iii) aws s3 ls | #list the bucket |

(iv) Since, don't have permission for user to access aws s3 in CLI so to give permission for user, create user in IAM in AWS,

IAM --> user --> create user

Step1: username(new\_user) --> next

Step2:Attach policies directly --> AmazonS3FullAccess, AmazonEC2FullAccess, IAMFullAccess --> next

Step3: tag (key:name, value:AWS-CLI\_ --> create user

Click on created user (new\_user) --> create access key

Command Line Interface (CLI) --> Description tag value (AWS-CLI) --> create access key

Copy access key and secret key

(v) give permission to user in CLI

aws configure

(give access, secret key and region)

aws s3 ls

(vi) create bucket

|  |  |
| --- | --- |
| aws s3 mb s3://mytestbucket13  ls  aws s3 cp awscliv2.zip s3://mytestbucket13 | #mb=make bucket    #copy file to bucket |

(vii) adding policy (rules) for bucket

|  |  |
| --- | --- |
| * + - vim policy.json   {  "Statement": [  {  "Effect": "Allow",  "Principal": "\*",  "Action": "s3:GetObject",  "Resource": "arn:aws:s3:::MyBucket/\*"  },  {  "Effect": "Deny",  "Principal": "\*",  "Action": "s3:GetObject",  "Resource": "arn:aws:s3:::MyBucket/MySecretFolder/\*"  },  {  "Effect": "Allow",  "Principal": {  "AWS": "arn:aws:iam::123456789012:root"  },  "Action": [  "s3:DeleteObject",  "s3:PutObject"  ],  "Resource": "arn:aws:s3:::MyBucket/\*"  }  ]  }   * + - aws s3 mv policy.json s3://mytestbucket13 | #adding policy                                                        #moving policy to s3 bucket |

(vii) delete bucket

|  |  |
| --- | --- |
| aws s3 rm s3://test-bucket --recursive | #remove objects (empty bucket) |
| aws s3 rb s3://test-bucket | #remove bucket |

(v) sync --> checks differences in folder and copies that only

aws s3 mb mybucket.test

mkdir test

cd test

touch f1 f2

cd

aws s3 sync test s3://mybucket.test

Making sync automated using crontab, if any new files are created in test folder then it will automatically push into bucket (mybucket.test)

|  |  |
| --- | --- |
| crontab -l | #lists crontab |
| crontab -e  2 (vim.basic)    45 03 \* \* \* aws s3 sync /home/ubuntu/test/\* s3://mybucket.test/ >> /home/ubuntu/s3\_bucket\_upload.log    touch f3 f4 f5 | #edit crontab      #time as per UTC    #testing crontab |

* + ***EC2:***

|  |  |
| --- | --- |
| aws ec2 describe-instances --query "Reservations[\*].Instances[\*].[InstanceID]" | #shows all instance IDs |
| aws ec2 describe-instances | #shows complete details of all instances in region |
| aws ec2 describe-instances --query "Reservations[\*].Instances[\*].[InstanceID]" --output text    aws ec2 describe-instances --query "Reservations[\*].Instances[\*].{PublicIpAddress}" --output text | #shows output in text format as by default it comes in json format |
| aws ec2 describe-instances --query "Reservations[\*].Instances[\*].{PublicIP:PublicIpAddress, Name:Tags[?Key=='Name'] | [0].Value,Status:State.Name,Type:InstanceType}" --output table | #shows in table format |

* + ***Create Instance:***

**(i) create keypair (pem file)**

aws ec2 create-key-pair --key-name demo --query 'KeyMaterial' --output text > demo.pem

chmod 400 demo.pem

**(ii) create instance**

aws ec2 run-instances --image-id ami-04a81a99f5ec58529 --instance-type t2.micro --key-name demo --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=demo2}]

Validate: check in AWS and connect instance in cli using privateIP from ubuntu

* + ***Create Instance using AMI:***
    - Create Instance
    - Create Image (AMI) of that Instance
    - Using AMI create Instance

|  |
| --- |
| Step1: Launch an EC2 Instance and save the instance ID into an environment variable  instance\_id=$(aws ec2 run-instances --image-id ami-55ef662f --instance-type t2.micro --key-name demo --user-data [file://userdata.txt](file:///\\userdata.txt) --query 'Instances[\*].[InstanceId]' --output text ) |
| Step 2: Check the user data worked and the web server is running by typing the web server IP on a browser and verifying you see ÒHello WorldÓ |
| Step 3: Create an image from that instance ID and save the image id to a variable image\_id  image\_id=$(aws ec2 create-image --instance-id $instance\_id --name "My server" --description "An AMI for my webserver" --query ImageId --output text) |
| step 4: use that image to launch an instance  instance\_id=$(aws ec2 run-instances --image-id ami-04a81a99f5ec58529 --instance-type t2.micro --key-name demo --query 'Instances[\*].[InstanceId]' --output text) |

Day19

***Route 53:***

Amazon Route 53 is a scalable and highly available Domain Name System (DNS) web service provided by Amazon Web Services (AWS). It is designed to provide reliable and cost-effective routing to domain names.

Steps:

(i) create 2 ubuntu instance with user data

(ii) create Application Load Balancer and link 2 instances

(iii) create Route 53

Register domain (example.com --> can buy available domains)

[Note: Once we buy website then can use it by giving website name (domain name) instead of giving IPs or dns name]

Hosted zones --> create record

Used for giving policies like Simple, Weighted, Latency, Failover, Geolocation

Instance -> allow http -> userdata

#!/bin/bash

yes | sudo apt update

yes | sudo apt install apache2

echo "us-webserver-1" > /var/www/html/index.html

sudo systemctl restart apache2

-> launch instance [ create 2 instance]

CREATE APPLICATION LOAD BALANCER

loadbalancername -> internet facing -> ipv4 -> default vpu -> edge location 2 region -> SG -> TG -> creat ALB

ROUTE 53

Register name -> check [ if it will show it avaliable r else NO ] -> select domain -> check box to[ AUTOrenewal] proceed to checkout -> next -> personal details -> next [ billing charge] ->

***(2) RDS:***

Amazon Relational Database Service (Amazon RDS) is a collection of managed services that makes it simple to set up, operate, and scale databases in the cloud.

Default port for mySQL is 3306

(i) Create one linux instance

(ii) create RDS --> create database

standard create -> MySQL -> MySQL community -> engine version default -> free tier -> database name (username --> password) -> connect to an ec2 compute resource -> select existing instance nor create new [ aws linux ][sg -> inbound [mysql/aurora],SSH -> save rules] -> create databASE

***CLI:***

Connect to linux instance in CLI

(1) sudo yum update -y

(2) sudo yum install -y <https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm>

(3) sudo yum install mysql-community-server

(4) sudo dnf install mariadb105

(5) mysql --version

(6) mysql -h database-1.ch22eqeemrr9.us-east-1.rds.amazonaws.com -P 3306 -u admin -p

[Note: once we give password, mysql will get connected]

***mysql commands***

|  |  |
| --- | --- |
| show databases; | # to check databases |
| create database <database\_name> | # create database |
| use <database\_name> | # get into database |

**DynamoDB:**

Called as serverless NoSQL database, can create table in AWS in DynamoDB.

For large tables not used/recommended

Day20

**Elastic Beanstalk**

Amazon Elastic Beanstalk is platform-as-a-service (PaaS) offering from Amazon Web Services (AWS). Easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.

**Why Elastic Beanstalk?**

Elastic Beanstalk is a service for deploying and scaling web applications and services. Upload your code and Elastic Beanstalk automatically handles the deployment—from capacity provisioning, load balancing, and auto scaling to application health monitoring.

**Pricing:**

There’s no additional charge for Elastic Beanstalk. You pay for Amazon Web Services resources that we create to store and run your web application, like Amazon S3 buckets and Amazon EC2 instances.

**Key Benefits:**

**Simplified Deployment:** Handles infrastructure setup and management, allowing you to focus on your application code.

**Auto-Scaling:** Automatically adjusts the number of instances based on traffic to optimize costs and performance.

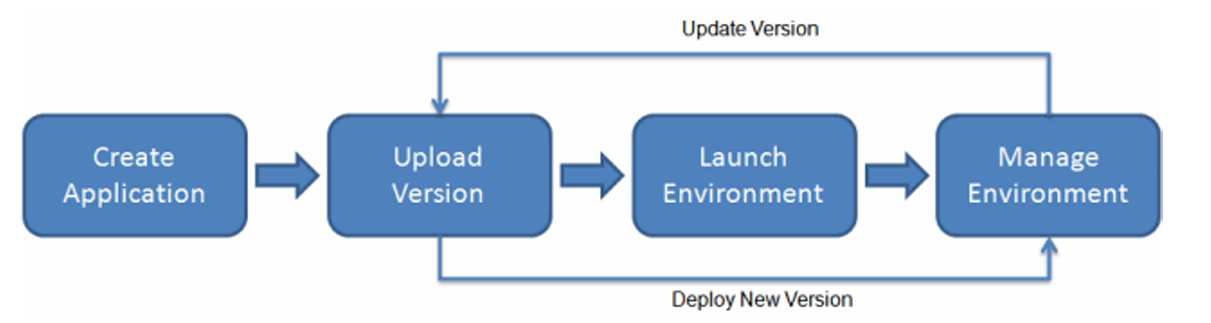
**Load Balancing:** Distributes incoming traffic across multiple instances for improved performance and reliability.

**Platform Support:** Supports a wide range of programming languages and frameworks.

**Cost-Effective:** Only pay for the resources you use.

**How Does It Work?**

To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application. Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code. After your environment is launched, you can then manage your environment and deploy new application versions.



[Note: Elastic Beanstalk will create Instances, Load balancers, Target groups, Auto Scaling, S3 bucket, Elastic IP]

**Setting up:**

Step 1: Create an application

Elastic Beanstalk --> create application --> app name (demo-app) --> platform (python) --> sample application --> presets (Single Instance) --> next

Step2: Configure service access

Create and use service role --> choose key pair --> EC2 instance profile (create IAM roles, open "View permission details" and in role add 3 policies (roles --> create role --> EC2 --> AWSElasticBeanstalkWebTier,AWSElasticBeanstalkWorkerTier, AWSElasticBeanstalkMulticontainerDocker) --> select created IAM role --> next

Step3: Setting up networking, databases and tags

Select VPC (select default) --> select instance subnets (availability zones) --> next

Step4: configure instance traffic and scaling

Root vol (choose as per requirement) --> EC2 security groups (default) --> Instance types (t3.micro)

--> next

Step5: Configure updates, monitoring and logging (optional)

Monitoring (Enhanced) --> managed updates (Activated, Friday at 22:21 UTC) --> platform software (ngnix) --> next

Step6: Review

Submit

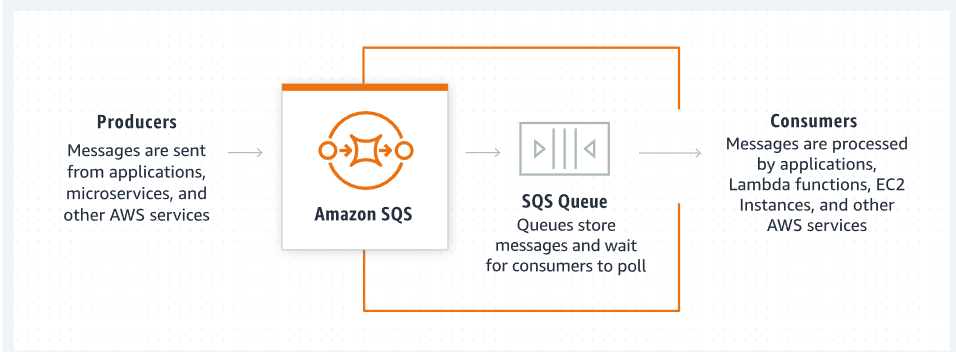
***Day21***

# ***SQS (Simple Queue Service):***

SQS is a fully managed message queuing service provided by AWS. It enables to send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available.

***How it works:***

* Producers send messages to an SQS queue.
* Consumers retrieve these messages from the queue and process them.
* SQS ensures messages are delivered reliably and in order (for FIFO queues).

****

## ***Difference between SNS and SQS***

|  |  |  |
| --- | --- | --- |
| **Feature** | **SNS (Simple Notification Service)** | **SQS (Simple Queue Service)** |
| Purpose | Sends messages to multiple subscribers | Stores messages in a queue for later processing |
| Message delivery | Pushes messages to subscribers | Messages are pulled by consumers |
| Message persistence | No, messages are not stored | Yes, messages are stored for a specific duration |
| Scaling | Automatically scales to handle many subscribers | Automatically scales to handle message volume |
| Use case | Notifications, alerts, fan-out | Asynchronous processing |
| Example | Email notifications, SMS alerts | Task queues, message buffering |

# ***ACM: (AWS Certificate Manager)***

It's a service provided by Amazon Web Services (AWS) that simplifies the process of managing and deploying SSL/TLS certificates for your AWS-based websites and applications.

***How ACM Works***

1. **Request a Certificate:** You can request a certificate for a specific domain, multiple domains, or a wildcard domain.
2. **Validation:** ACM verifies your ownership of the domain through DNS validation or email validation. Once validated, ACM issues the SSL/TLS certificate.
3. **Deployment:** You can deploy the certificate to various AWS services like Elastic Load Balancing, CloudFront, or API Gateway.
4. **Renewal:** ACM automatically handles certificate renewals before they expire.

Tomcat App on Benastalk with Aurora DB v2.txt

***Elastic Beanstalk Project:***

Project: Book Seller

####################

You are an e-com book seller. Your simple starting page shows

the list of authors. Buyers can select the author. The application will

list all the books by the author with the price. Once they click order, you

have to show a Thank You message. (Note: we don't want to go to other processes

such as payments, shopping cart etc..). You need to use an Aurora DB to store all

the book lists and access it from application.

ref: https://www3.ntu.edu.sg/home/ehchua/programming/java/JSPByExample.html

Book seller

===========

Task 1: get the code ready

=========================

1) Launch Instance (Ubuntu)

2) Get the code base in tar form:

wget https://s3.amazonaws.com/klowdbay.com/YTCodeResources/book-seller.tar.gz

3) Unzip

tar -xzvf book-seller.tar.gz

(# Install tree utility.

sudo apt install tree -y

tree .

# Ensure that you have the files pom.xml, web.xml, index.jsp, order.jsp

mysql-connector-j-8.0.32.zip (mysql connector)

├── book-seller

│   ├── mysql-connector-j-8.0.32.zip

│   ├── pom.xml

│   └── src

│   └── main

│   └── webapp

│   ├── WEB-INF

│   │   └── web.xml

│   ├── index.jsp

    └── order.jsp

5 directories, 7 files)

4) Install maven package

sudo apt install maven -y

mvn --version

# It will display version no. ex: Apache Maven 3.6.0

5) cd ~/book-seller (goto your folder where the code resides)

6) View the code files

cd ~/environment/book-seller/src/main/webapp

ls

cat index.jsp

cat order.jsp

7) goto root directory

cd ~/environment/book-seller

8) Time to build the load

mvn clean package

# You should see 'Build Success' Message

9) Observe the '/target' folder to see if warfile 'book-seller-1.0.0.war' is present

ls ~/environment/book-seller/target

10) Install Tomcat

sudo apt install tomcat10

validate: add inbound rule: 8080

instanceIP:8080

11) location of webapps

sudo find / -iname "webapps"

(/var/lib/tomcat10/webapps)

12) copy and rename book-seller-1.0.0.war in webapps location

sudo cp /home/ubuntu/book-seller/target/book-seller-1.0.0.war book-seller.war

validate: instanceIP:8080/book-seller/

13) Create IAM role for S3 bucket access in Instance

IAM --> role --> create role

AmazonEC2FullAccess, AmazonS3FullAccess

14) Attach IAM role into instance

select Instance --> Actions --> security --> modify IAM role --> select created role --> save

15) Install AWS-CLI

cd (home path)

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_ 64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

16) Create an S3 bucket (or use an existing one). Copy warfile to the S3 bucket. When you create

# beanstalk platform, you can upload the code from the S3 bucket.

cd /home/ubuntu/book-seller/

aws s3 mb s3://mybeanstalkbkt #create bucket

cd target/

aws s3 cp book-seller-1.0.0.war s3://mybeanstalkbkt/myapp.war

aws s3 ls mybeanstalkbkt

# Note: trainings-2023 is my bucket which is public. You need to give your bucket name.

Task 2: Create Elastic Beanstalk Application

============================================

Go to Beanstalk console. Click on Application (left side menu)

Create beanstalk application : book-seller-101

Select tomcat platform

copy paste S3 bucket - object URL

Task 3: Setup the DB instance

=============================

Go to RDS console; Click 'Create database'

Choose 'standard create'

Choose 'MySQL'

Under templates, choose Dev/Test (free tier);

Under settings, db cluster identifier give the name as 'yourname-book-seller'

ex: unus-book-seller

Master user name can be admin

password: enter admin1234

DB instance class: choose smallest instance type (ex: burstable type - db.t3.small)

Don't create Aurora replica

Public access: choose yes

Select a security group which opens MySQL/Aurora 3306 port from 0.0.0.0/0

If not specifically mentioned here, leave other fields with default values

Click 'Create Database'

CLI

CLI: (Ubuntu)

1) sudo apt update -y

2) sudo apt install mysql-server

3) mysqld –version

4) sudo mysql\_secure\_installation

{Note: give y for all]

5) sudo systemctl status mysql

6) mysql -h database-1.ch22eqeemrr9.us-east-1.rds.amazonaws.com -u admin -p

Task 4: Create & Update table in the DB

=======================================

1. Go to Cloud9 terminal. Access the database from here.

2. Instead of Cloud9, if you are using EC2, then you have to install mySQL :

sudo apt update -y

sudo apt install mysql-client-core-8.0

If you are in cloud9, mysql is already installed

3. Connect to the Aurora RDS endpoint from EC2. You will get endpoint from RDS console. Ex:

mysql -h <write endpoint> -P <mySQL port no> -u <user id> -p

mysql -h unus-book-seller-instance-1.chkaeykcmlpn.us-east-1.rds.amazonaws.com -P 3306 -u admin -p

4. It will ask for password - please enter the password. (ex:test1234) You will get MySQL prompt

5. CREATE DATABASE ebdb;

6. Show databases;

7. Use ebdb;

8. Create a new table in 'ebdb' with following commands

create table if not exists books (

book\_id INT auto\_increment,

title VARCHAR(250) NOT NULL,

author VARCHAR(50),

Price FLOAT,

Qty INT,

PRIMARY KEY (book\_id))

ENGINE = InnoDB;

9. show tables;

10. Insert a few rows as below

INSERT INTO books VALUES (1100, 'Chamber of Secrets', 'Rowling', 11.11, 4);

INSERT INTO books VALUES (1103, 'Philosophers Stone', 'Rowling', 10.90, 8);

INSERT INTO books VALUES (1105, 'War and Peace', 'Tolstoy', 22.22, 2);

INSERT INTO books VALUES (1107, 'Romeo and Juliet', 'Shakespear', 33.33, 5);

INSERT INTO books VALUES (1109, 'Othallo', 'Shakespear', 13.99, 7);

INSERT INTO books VALUES (1109, 'KingLear', 'Shakespear', 10.79, 3);

INSERT INTO books VALUES (1111, 'Death on the Nile', 'Agatha', 44.4, 15);

INSERT INTO books VALUES (1113, 'ABC Murders', 'Agatha', 39.4, 11);

INSERT INTO books VALUES (1115, 'Anna Kareneena', 'Tolstoy', 55.55, 23);

11. List the entries in the table

SELECT \* FROM books;

+---------+--------------------+------------+-------+------+

| book\_id | title | author | Price | Qty |

+---------+--------------------+------------+-------+------+

| 1100 | Chamber of Secrets | Rowling | 11.11 | 4 |

| 1103 | Philosophers Stone | Rowling | 10.9 | 8 |

| 1105 | War and Peace | Tolstoy | 22.22 | 2 |

| 1107 | Romeo and Juliet | Shakespear | 33.33 | 5 |

| 1109 | Othallo | Shakespear | 13.99 | 7 |

| 1111 | Death on the Nile | Agatha | 44.4 | 15 |

| 1113 | ABC Murders | Agatha | 39.4 | 11 |

| 1115 | Anna Kareneena | Tolstoy | 55.55 | 23 |

+---------+--------------------+------------+-------+------+

8 rows in set (0.00 sec)

12. exit;

Task 5: EBS to RDS connection

=============================

Now, we need to establish the connection between the Beanstalk application and the DB instance.

Go to Beanstalk. Click on 'environment' on the left side menu. Click on the environment name to

open it. Open 'configuration' menu on the left.

Select category 'software'. click on edit.

Under 'Environment Properties', Add below parameters in the text boxes.

RDS\_USERNAME: admin

RDS\_PASSWORD: test1234

RDS\_PORT: 3306

RDS\_HOSTNAME: add endpoint from connectivity & Security (of writer instance)

Ex: unus-book-seller-instance-1.chkaeykcmlpn.us-east-1.rds.amazonaws.com

RDS\_DB\_NAME: ebdb

Click 'Apply'. This may take 1-2 minutes.

Task 6: Access your page

========================

# Go to beanstalk console. Choose your application. Copy URL to browser

Ex: http://books-seller.us-east-1.elasticbeanstalk.com/

Select the author and list all his/her books. Select any book and place the order.

You will get a message 'Thank you for ordering books'.

You will get a feedback link. Enter the feedback.

Click on 'Back' link and reach the main page.

As the next step, change the list of books in the DB and see if it is getting reflected in

your book seller page.

Task 7: Cleanup

===============

1. Terminate RDS

Uncheck 'Create final snapshot' and 'Retain automated backups'.

Check 'I acknowledge that....'

2. Terminate Elastic Beanstalk application

3. Terminate cloud9

#########

Code Base

#########

pom.xml

=======

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.efsavage</groupId>

<artifactId>book-seller</artifactId>

<version>1.0.0</version>

<packaging>war</packaging>

<name>Book Seller DB </name>

<description>Simple ecom built with Java</description>

<dependencies>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.32</version>

</dependency>

<!-- Thanks for using https://jar-download.com -->

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.mortbay.jetty</groupId>

<artifactId>jetty-maven-plugin</artifactId>

<version>8.1.5.v20120716</version>

<configuration>

<scanIntervalSeconds>0</scanIntervalSeconds>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.1.1</version>

</plugin>

</plugins>

</build>

</project>

index.jsp

=========

<html>

<head>

<title>Book Query</title>

</head>

<body>

<h1>Academy E-Bookstore</h1>

<h3>Choose Author(s):</h3>

<form method="get">

<input type="checkbox" name="author" value="Tolstoy">Tolstoy<br>

<input type="checkbox" name="author" value="Shakespear">Shakespear<br>

<input type="checkbox" name="author" value="Agatha">Agatha Christie<br>

<input type="checkbox" name="author" value="Rowling">J K Rowling<br><br>

<input type="submit" value="List the Books and Price"><br>

</form>

<%

String[] authors = request.getParameterValues("author");

if (authors != null) {

%>

<%@ page import = "java.sql.\*" %>

<%

// build the jdbc url

// Read RDS connection information from the environment

String dbName = System.getProperty("RDS\_DB\_NAME");

String userName = System.getProperty("RDS\_USERNAME");

String password = System.getProperty("RDS\_PASSWORD");

String hostname = System.getProperty("RDS\_HOSTNAME");

String port = System.getProperty("RDS\_PORT");

String jdbcUrl = "jdbc:mysql://" + hostname + ":" +

port + "/" + dbName + "?user=" + userName + "&password=" + password;

// Load the JDBC driver

try {

System.out.println("Loading driver...");

Class.forName("com.mysql.jdbc.Driver");

System.out.println("Driver loaded!");

} catch (ClassNotFoundException e) {

throw new RuntimeException("Cannot find the driver in the classpath!", e);

}

Connection conn = null;

Statement setupStatement = null;

Statement readStatement = null;

ResultSet resultSet = null;

ResultSet rset = null;

String results = "";

int numresults = 0;

String statement = null;

conn = DriverManager.getConnection(jdbcUrl);

readStatement = conn.createStatement();

String sqlStr = "SELECT \* FROM books WHERE author IN (";

sqlStr += "'" + authors[0] + "'"; // First author

for (int i = 1; i < authors.length; ++i) {

sqlStr += ", '" + authors[i] + "'"; // Subsequent authors need a leading commas

}

sqlStr += ") AND qty > 0 ORDER BY author ASC, title ASC";

// for debugging

System.out.println("Query statement is " + sqlStr);

rset = readStatement.executeQuery(sqlStr);

%>

<hr>

<form method="get" action="order.jsp">

<table border=1 cellpadding=5>

<tr>

<th>Order</th>

<th>Author</th>

<th>Title</th>

<th>Price</th>

<th>Qty</th>

</tr>

<%

while (rset.next()) {

int id = rset.getInt("book\_id");

%>

<tr>

<td><input type="checkbox" name="id" value="<%= id %>"></td>

<td><%= rset.getString("author") %></td>

<td><%= rset.getString("title") %></td>

<td>$<%= rset.getInt("price") %></td>

<td><%= rset.getInt("qty") %></td>

</tr>

<%

}

%>

</table>

<br>

<input type="submit" value="Order">

<input type="reset" value="Clear">

</form>

<a href="<%= request.getRequestURI() %>"><h3>Back</h3></a>

<%

rset.close();

readStatement.close();

conn.close();

}

%>

</body>

</html>

order.jsp

=========

<html>

<head>

<title>Academy Bookstore Order Management</title>

</head>

<body>

<h3>Thank you for ordering books </h3>

<h4>Your Feedback, Please </h3>

<form method="get">

<input type="checkbox" name="feedback" value="Excellent">Excellent

<input type="checkbox" name="feedback" value="Good">Good

<input type="checkbox" name="feedback" value="Average">Average

<br><br><br><input type="submit" value="Submit Feedback">

</form>

<%

String[] feedbacks = request.getParameterValues("feedback");

if (feedbacks != null) {

%>

<h3>Thank You for the Feedback</h3>

<br> <br>

<!--<a href="<%= request.getRequestURI() %>">BACK</a>-->

<a href="index.jsp">BACK</a>

<%

}

%>

</body>

</html>

WEB-INF/web.xml

===============

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

>

<!-- This is only here because Maven requires it to make a war. -->

Downloading the mySQL connector

==============================

Download mysql-connector-j-8.0.31.jar from https://dev.mysql.com/downloads/connector/j/?os=26

wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-j-8.0.32.zip

Keep the file in root directory of your maven project

Now include the same in dependency section in pom.xml (already done above)

Useful Ref Documents

====================

AWS SDK for Java to connect to RDS

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.IAMDBAuth.Connecting.Java.html