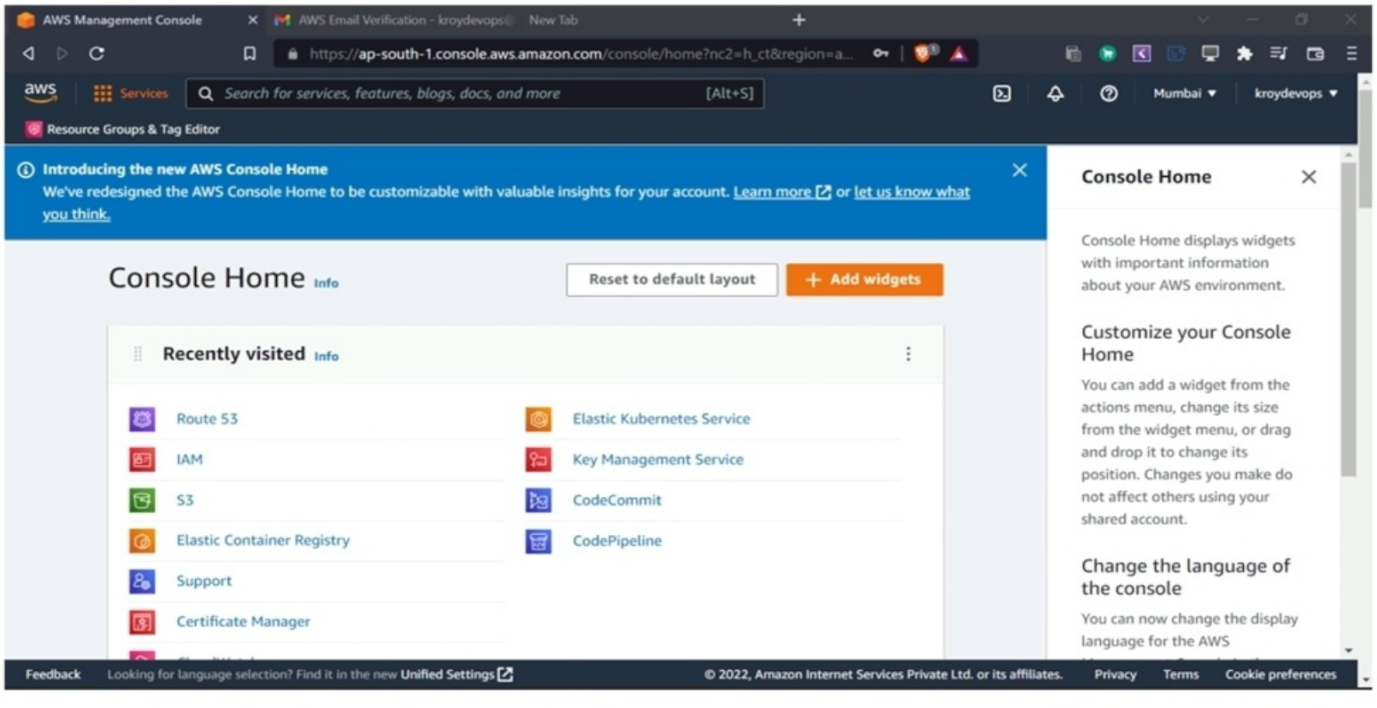
AWS → Public cloud

To create a new AWS account :-

Steps :-

(i) Open your browser.
(ii) Open google.com
(iii) Search "aws free Tier"
(iv) go on link "https://aws.amazon.com/free"
(v) Create a free account
(vi) Proceed with the steps shown in document.



→ Once Account is Ready and if you are able to Aws console then we are all set to go.

Learn how to launch an EC2 instance on cloud.

→ Proceed with the steps shown in document to how to create and connect with EC2 instance.

EIP – Elastic Ip address


→ An Elastic Ip address is a reserved public Ip address that you can assign to any EC2 instance in a particular

region, until you choose to release it.

→ we get Dynamic Ip address when we launch ec2-instance.
Elastic Ip provides a static public IP.

→ To launch and connect windows server we will see in lab.

## Aws Storage Service

Amazon S3 — Amazon simple Storage Service

→ Object level Storage
↳ can store any type of file
→ acess it from anywhere. (URL)

Elastic File System (EFS) — Network based Storage System.

→ Similar to NFS - Network File System
→ We share the files across the network.
→ Available of linux Machine.

## Elastic Block Storage (EBS) – Block level Storage.

→ It can be accessed through EC2-instance only.
→ Similar to Hard-disk of your computer.
→ To access the data of your hard-disk you have to use your computer only.

Few more like — slower
- S3 Glacier (cheaper, data we don't require frequently)
- Snowball
  - ↳ To transfer large amount of data out of AWS (10TB+)

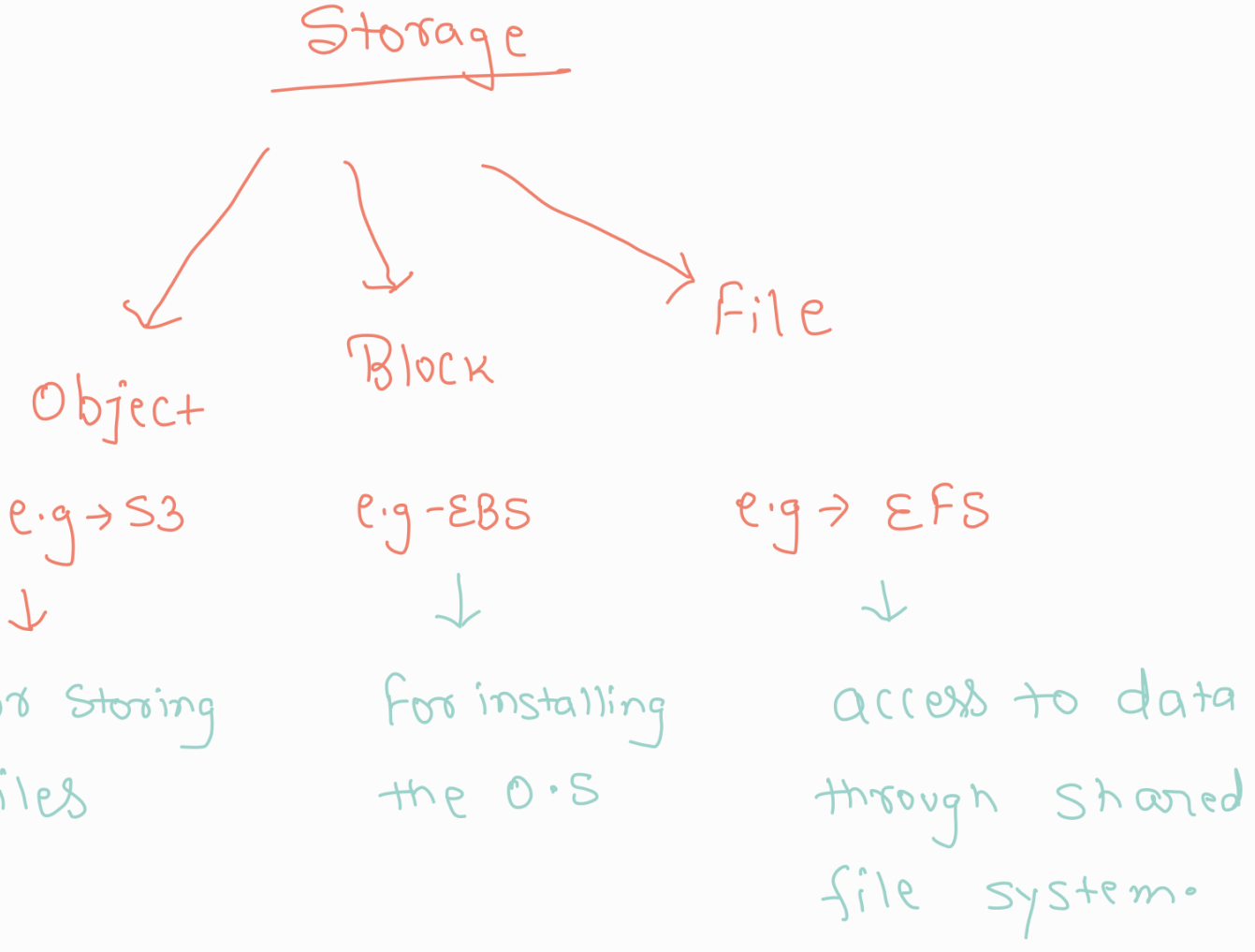
Storage → core requirement for O.S

↳ To Keep data permanent / persistent

- flash drive
- SSD
- Pen drive
- CD/DVD

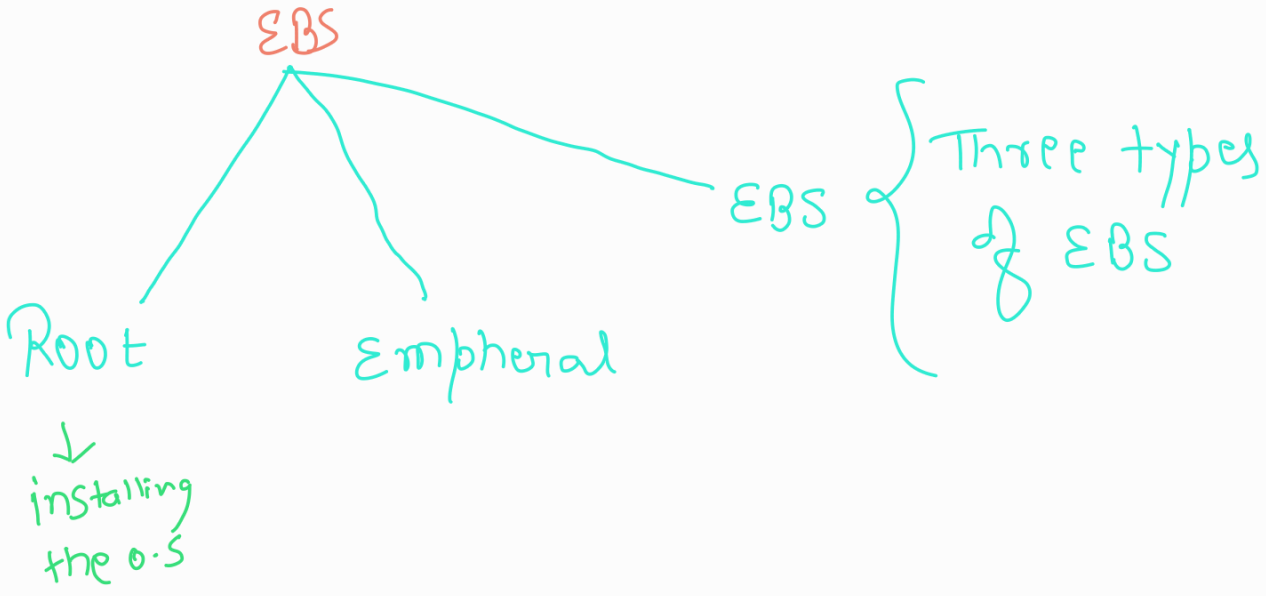
Storage ≠ Memory

Memory = RAM = Volatile memory



# EBS

→ To install O.S EBS will be required, hence we find EBS under compute Section.




To install O.S :-

Minimum Requirement

(i) Should have a H.D

(ii) atleast one partition should be there.

Root → Main partition where O.S installs is known as root block Storage.

Add Storage:

Volume Type : root

Windows — C:/ — we install OS here
        — D:/ — file here

EBS → Similar to connecting a new external HD/P.D to instance.

To store data in EBS type ⇒ Buy the P.D
- partition
- format

Mount ↳ = USB

H.D → EC2-instance
↳ Partition
↳ Format ↳ Mount → data

Empheral (instance store) — ideal for temporary storage like cache, tmp etc.

S3 — Object storage
- Serverless
- Everything will be managed

EBS → Everything we have to manage

→ Buy volume, Attach volume, Partition volume, Format volume, Mount volume, Dettach volume.

S3 – (storage type) – we will use it to store data.

data —stored in→ file —stored in→ folder

—S3→ Object —S3→ Bucket

↓

name should be specific in region.

# Similar to gdrive, you upload the file in folder, then you make that folder public and you share the link of folder to others.

S3 URL is unique. → unique Name Bucket

https://testdevops123.s3.ap-south-1.amazonaws.com/a.jpg


↳ That's why same bucket name in a particular region can't be created.

* EBS volume can't attach to multiple instances at a time.

→ Therefore to access our data from multiple instances at a time we use S3 or NFS.

when to use S3 and when NFS ?

→ Put a python code as a file in S3. To run this code you have to first download that code to local volume from S3 then only it can be run.

→ To run any code file, that file must be part of O.S file system.

→ Nfs attach to instance in such a way that instance treat Nfs as a local storage only.
(Nfs Share I/O table)

→ Nfs gives us a folder that can be attached to multiple instances at a time and all the instances will treat that folder as its their own local volume.

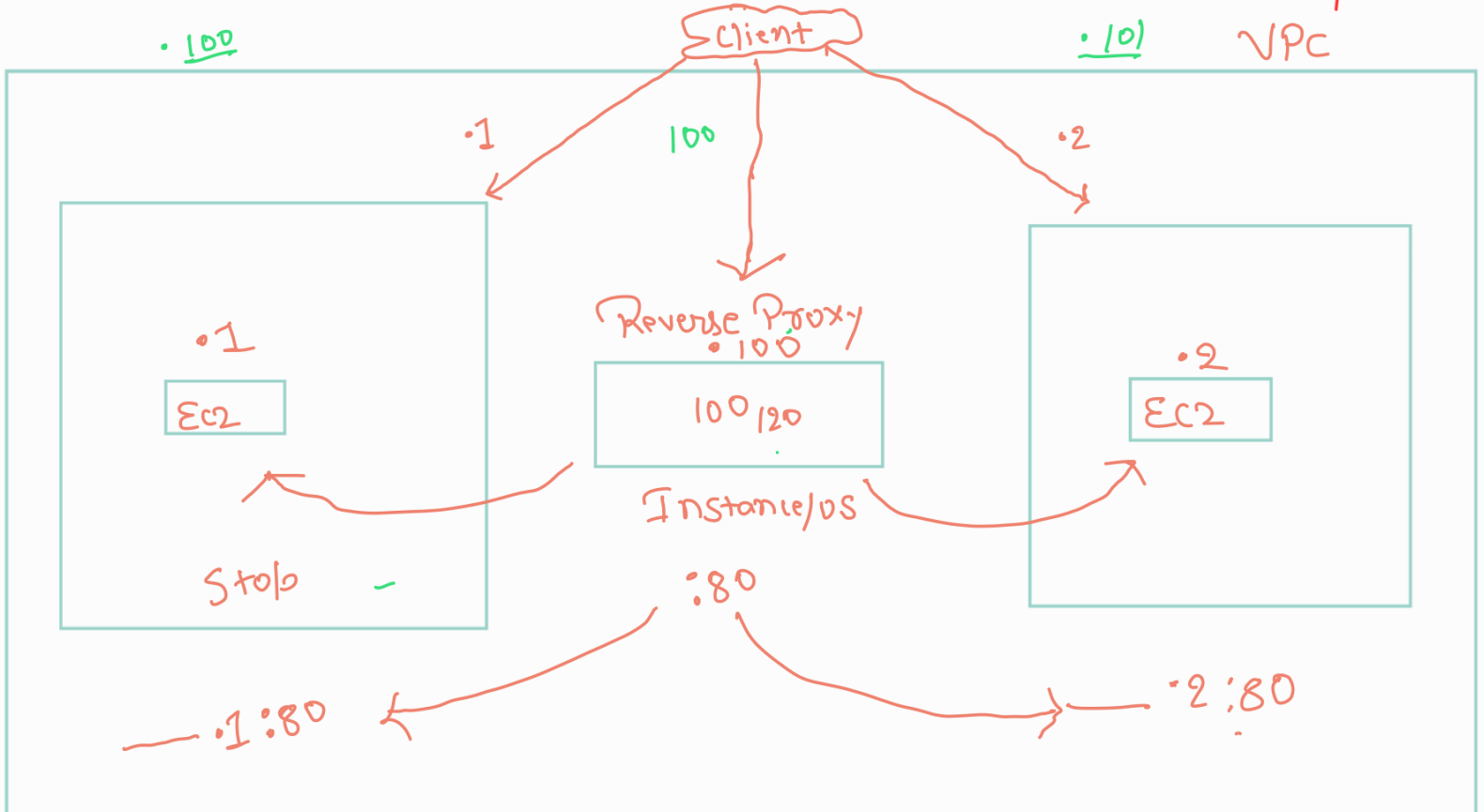
→ If file system have Io table then we have to always mount that storage before using it.

ELB – Elastic load Balancer

→ If you launch a web-service for your client then definitely you will think that it should be always up.

→ If your webservice running on a single EC2-instance and somehow if that ec2-instance fails or crashes your webserver will be down.

→ Now takes an example that we are running two ec2-instances and on both the instances our webserver are running then ever if one instance will fail our webserver will be still up.



Reverse proxy - Provides Single end point

Health check → To keep monitoring instance health.

→ If one instance fails then it will stop sending traffic to that instance till it comes.

HAproxy is a software which helps us to setup Reverse proxy on our on-permise env.

Since we are working on cloud so we don't have to setup Haproxy and then configuring it to use Reverse proxy.

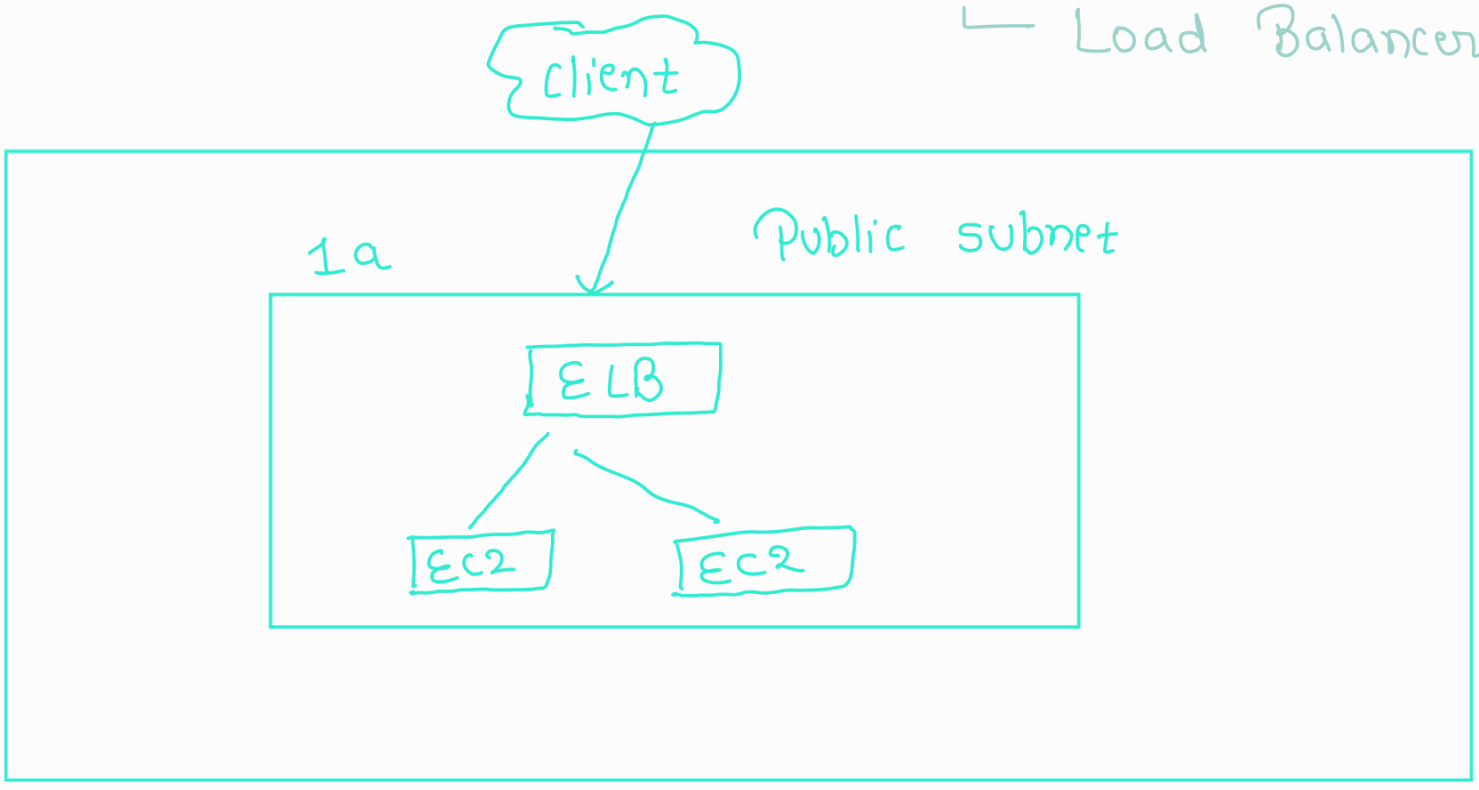
→ Load Balancer always comes with Reverse proxy.

→ Aws provides a Managed service with name ELB (Elastic load Balancer)

EC2 → Sub-service

↳ Load Balancing

└ Load Balancers




* ELB should be put in public subnet because it will require internet access for communicating with client.


OSI Model - 7 layers

L7 - Application layer

L6

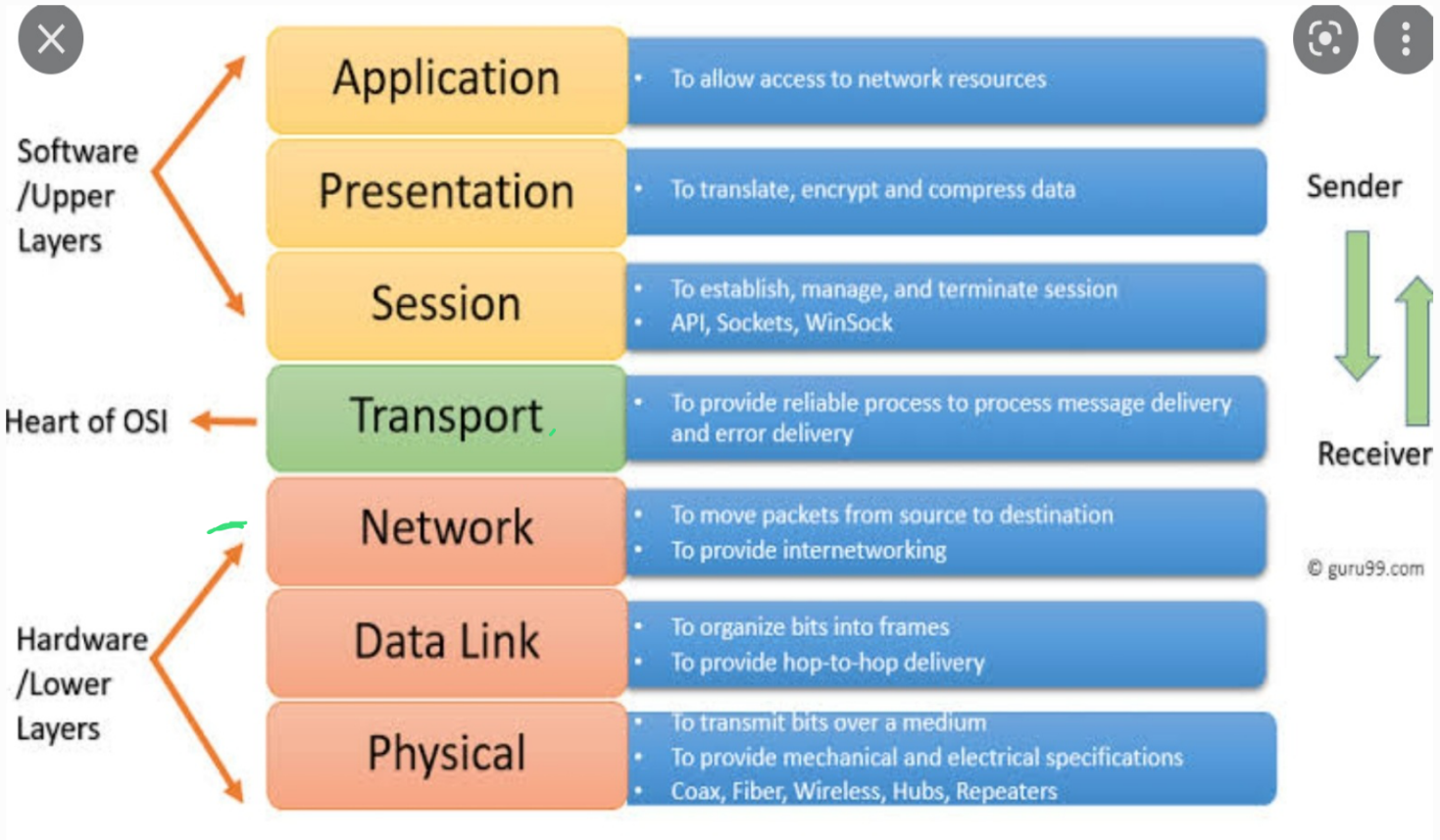
L5

L4 – Transport layer - TCP/UDP

L3 – N/w layer

L2

L1




CLB – classic/traditional – works on layer 4 and 7.

↳ works on checking of only port number.

↳ little bit slower

↳ cross-zone by default enabled (we can disable if require)

ALB → works on layer 7 (Path based routing)

↳ Slower than CLB
↳ cross zone enabled. (Always)

NLB → work on layer 3 and 4.

— Fastest

— cross zone by default disabled. (can be enabled)

ALB

↳ one single endpoint for user for all diffirent application access.

e.g- • 100:8080/chat

• 100:8080/Search


CLB - port configuration

↳ Edit Stickness

(user will stick to an instance till logout.)

NLB

└ Used for high ultra performance.

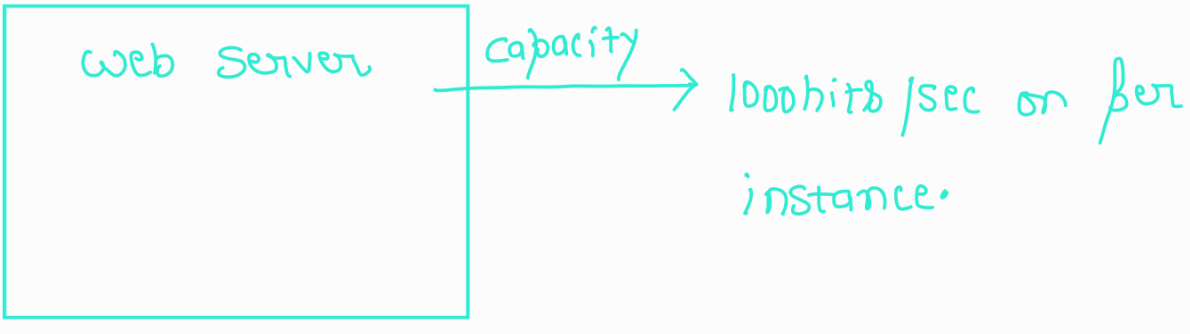
→ just forward the request whereas application load Balancer examines the contents of HTTP request to determine where to route the request.


CLB → can't forward traffic on more than one port per instance and it doesn't support forwarding to Ip address.

Cloud — Scalability
└ Reliability

AutoScaling — Automation in scalability.

e.g



(i) In starting 10-15 hits per sec was coming.

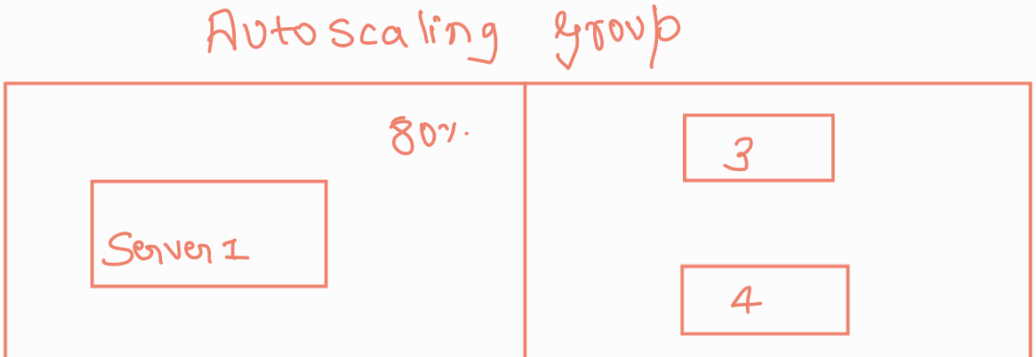
(ii) Now it increased to 600 hits per second which means cpu utilization 60%.

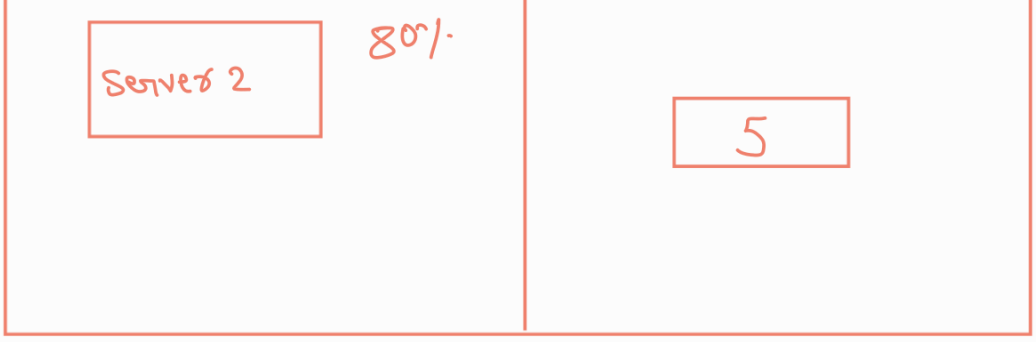
(iii) Now suddenly cpu ultilization increased to 80% that may cause failure of instance, so now to reduce that we have to scale our webserver.

Now we scaled to 3 instance of webserver.

→ 1000+1000+1000 = 3000 hits/second can be handled

* Cpu ultilization will be decreased automatically.






80% > +1

$\frac{80+80}{2} = 80\%$ ⟶ 1 more instance

$\frac{80+80}{3} = \frac{160}{3} = 53.33\%$

Again load decreased like 30, 30, 30 then

$\frac{30+30+30}{3} = 30\%$ ⟶ 1 less instance

$\frac{30+30+30}{2} = 45\% < 80\%$

→ Creating group of EC2 instances that can scale up or down depending on conditions you set.

→ Enable elasticity by scaling Horizontally through adding or terminating EC2 instances.

→ Autoscaling helps you save cost by cutting

down the number of EC2 instances, when not needed, and scaling out to add more instances only when it is required.

## Autoscaling components

① Launch configuration (can not edit) – like Instance Type, AMI, Keypair, Security group.

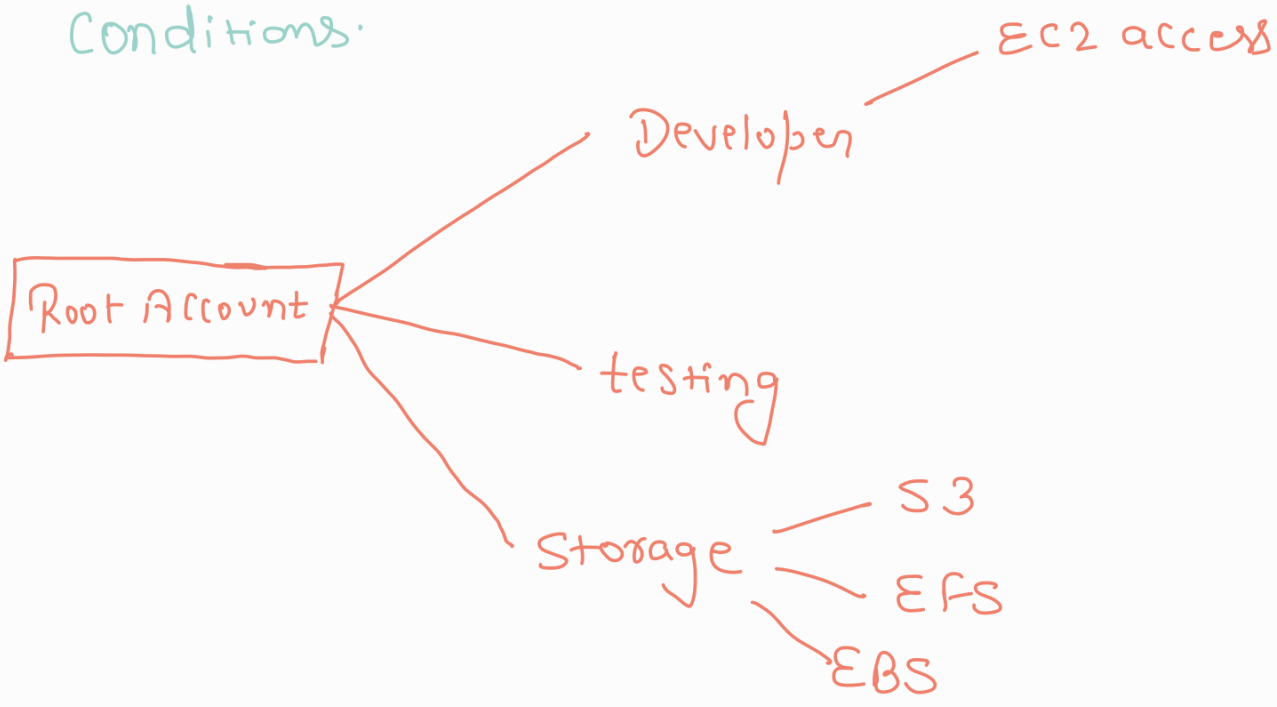
② Autoscaling group (can be edited.)
→ Group name, group size, VPC, Subnet, Health check period.
- Health check period → 300 sec
- group size → 3 (min) – 20 (max)

③ Scaling policy
– Metric type, Target value.

IAM → Identity Access Management

↳ fine-grained access control across all of aws.

↳ you can specify who can access which services and resources, and under which conditions.




IAM can handle user and roles with limited power/permissions.

→ Iam refers to a framework or policies and technologies for ensuring that the proper people in an organisation have the appropriate access to technology Resources.

→ In Starting when we create an aws

account we get root user Identity.

→ Aws strongly recommends that you do not use the root user for your everyday task, even the adminstrative ones.

## Aws account root user

* when you first create aws account you begin with a single sign-in identity that has complete access to all aws services and resources in account. (root user)
  → for sign in
    email and password required.

## IAM users

→ Represents the person or services who uses the IAM user to interact with Aws.

→ Primary use for IAM users is to give people the ability to sign in to the Aws Management console for interactive tasks and to make programmatic requests to Aws Services using API or CLI.

→ In an organization there is a main aws account and they create IAM user for each employee (having their own username and password) and Manager assign some policy having some permission.

→ Whatever permission will be written in policy, IAM user can perform only those in aws account.

## IAM user group

→ Collection of IAM users.

→ You can use user groups to specify permissions for a collection of users, which can make those permissions easier to manage for those users.

→ Let's take an example. You created a user group with "developer" name and attached some policy to it.

→ Now whenever new employee (developer) will join you just have to add IAM user of that employee to "developer" user group.

## RDS Connectivity

↳ Relational Database System

→ Aws gives fully managed database

- AWS gives fully managed database in which we don't require anything to configure.