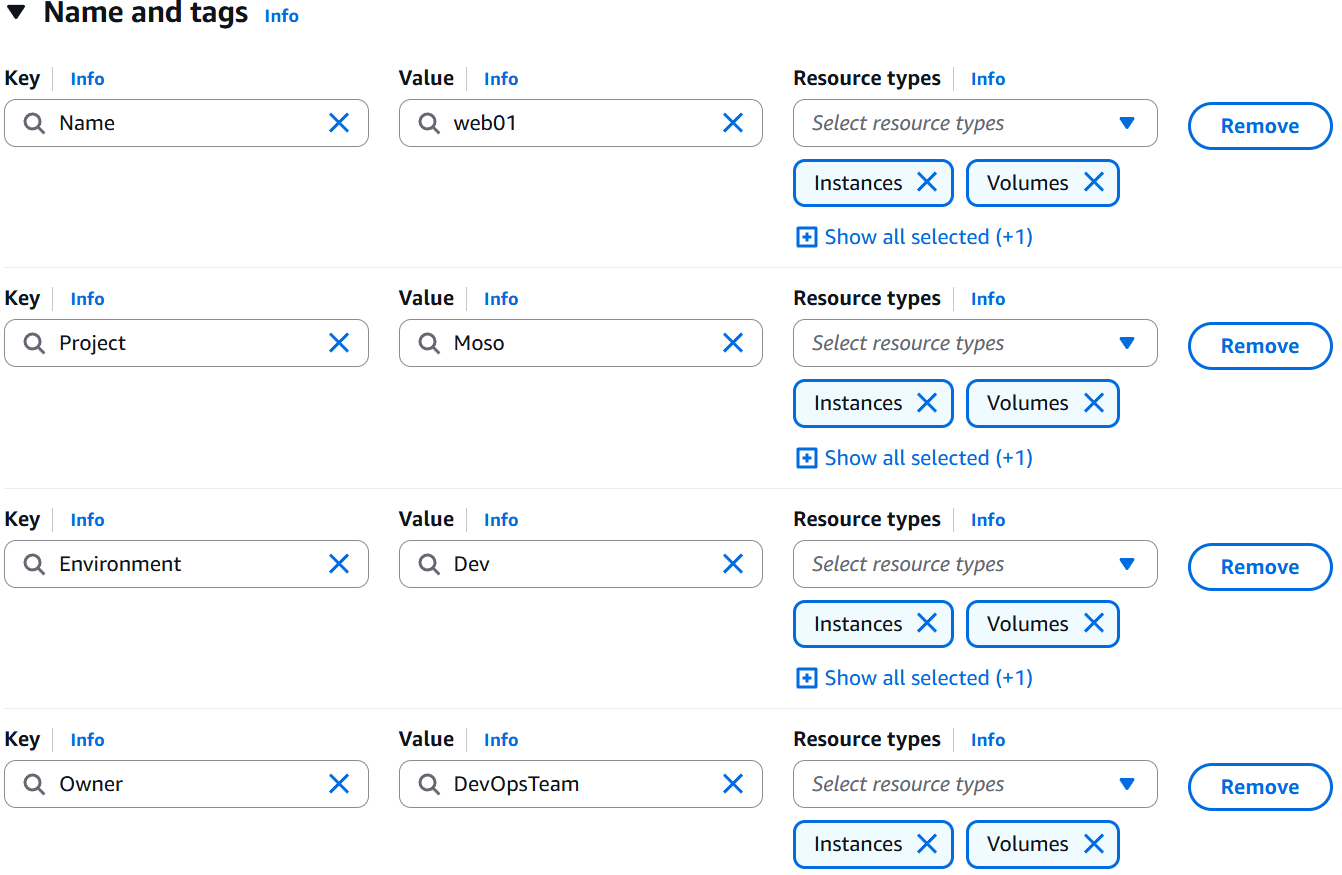
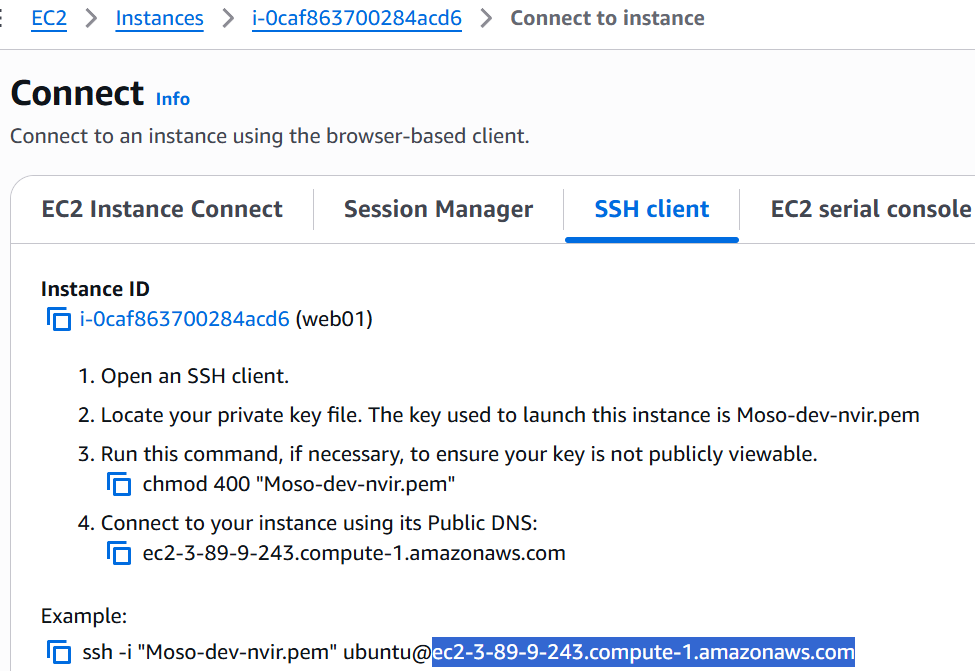
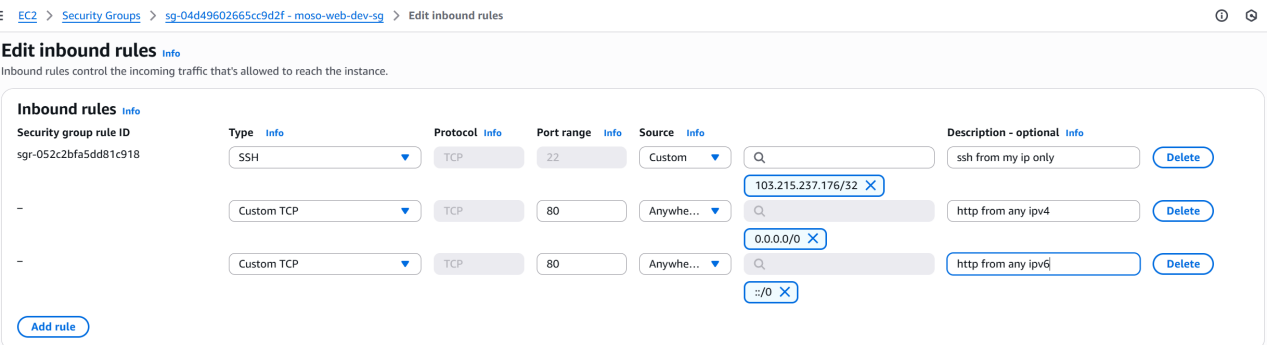
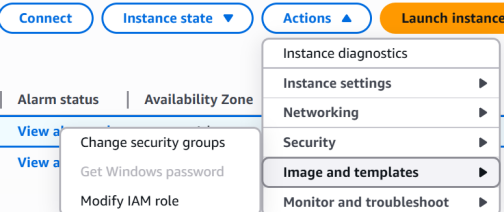
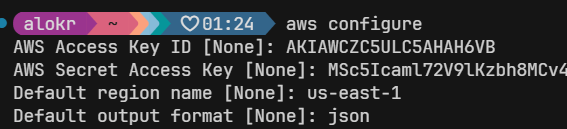
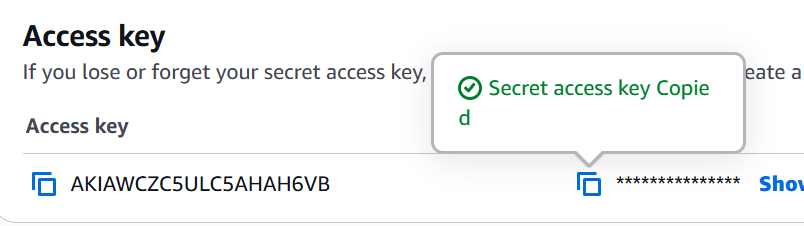
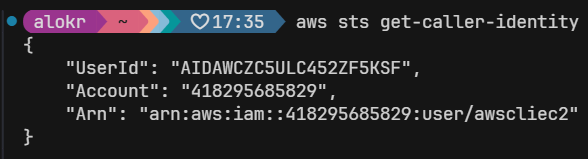
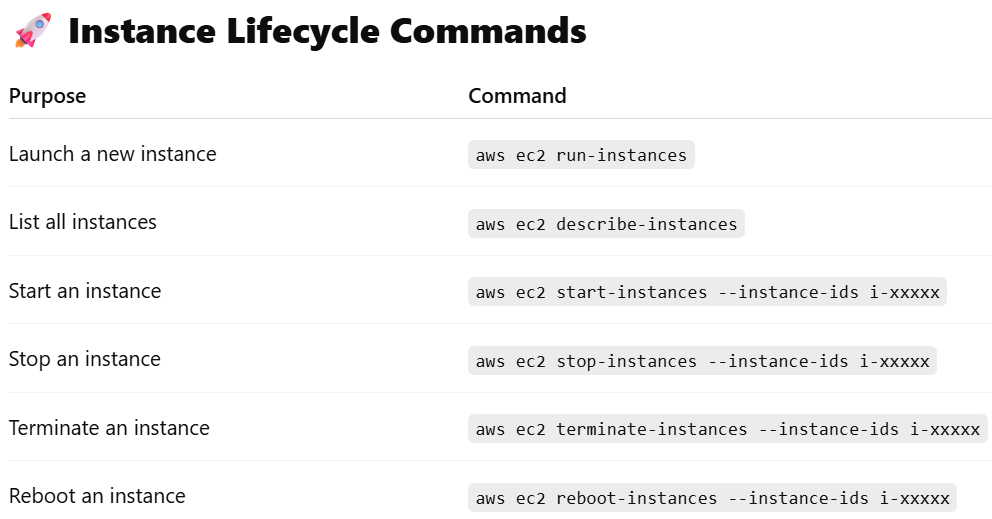
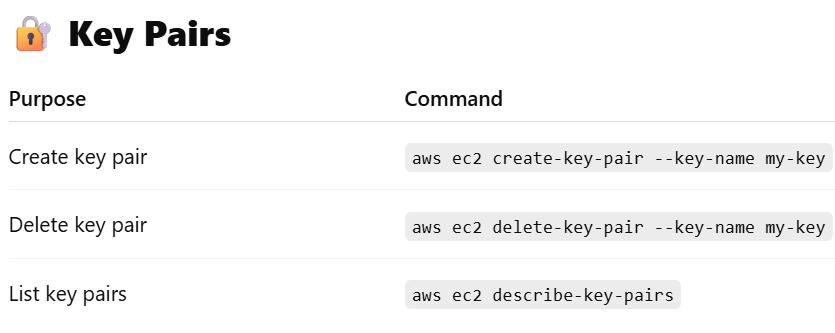
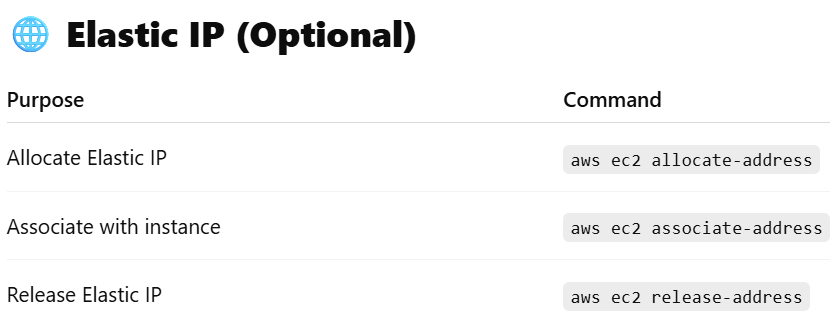
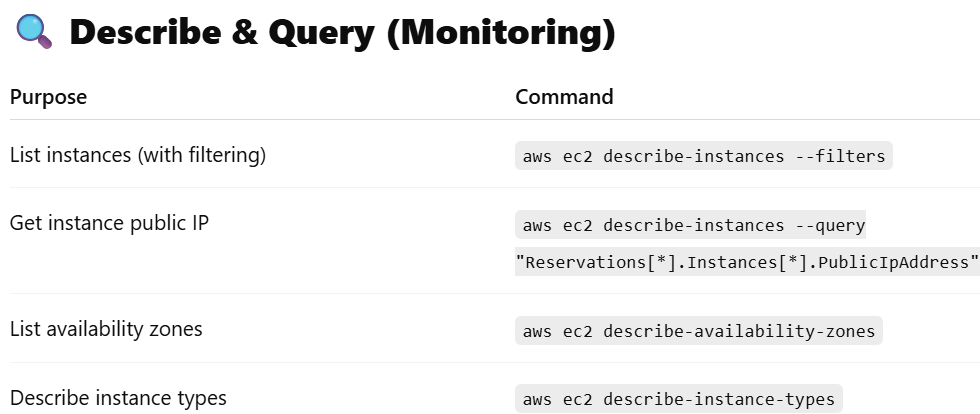
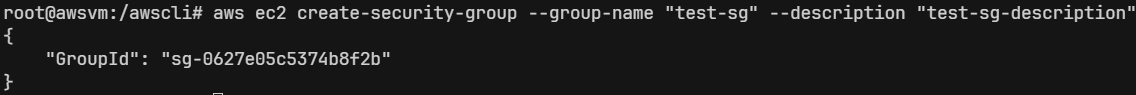
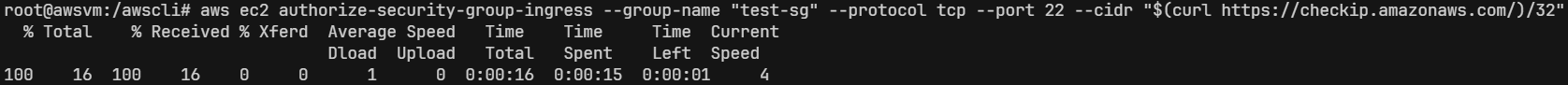
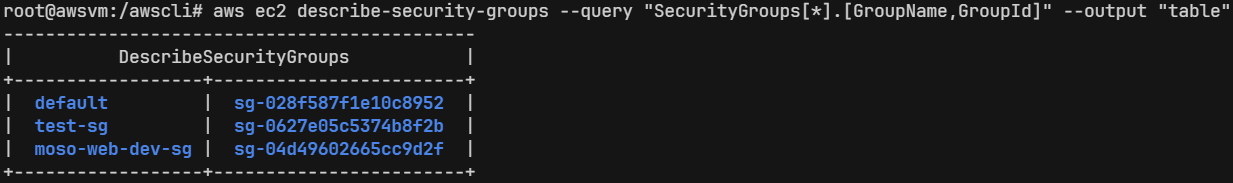
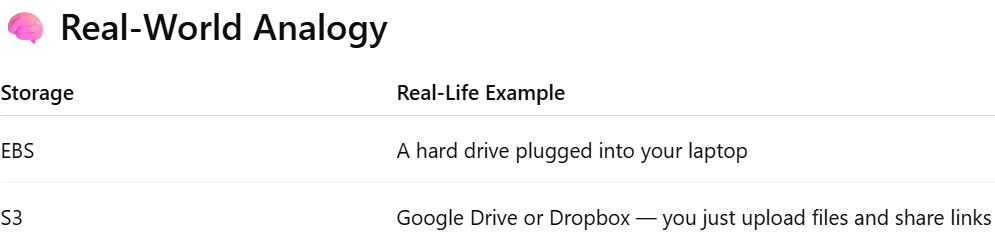
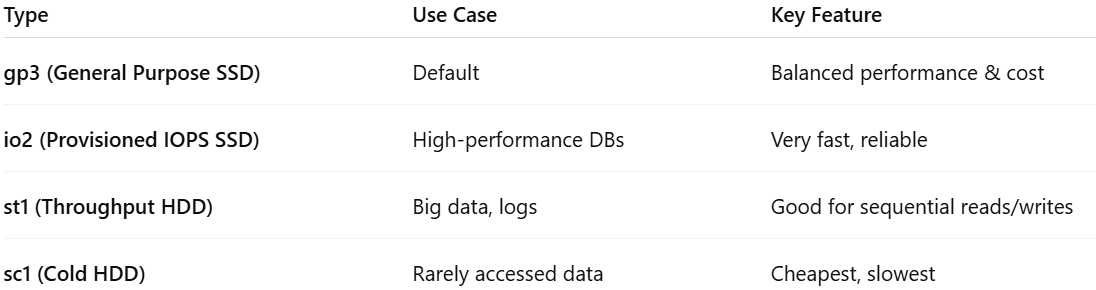
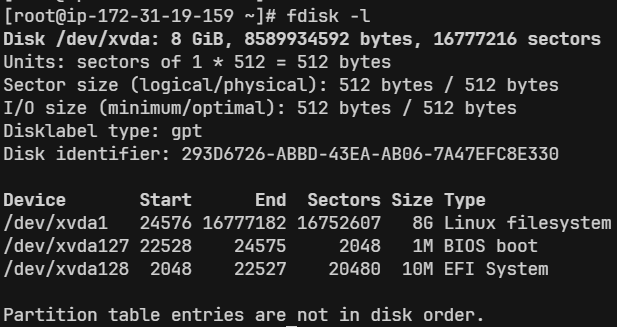
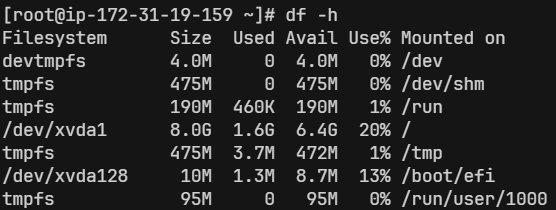
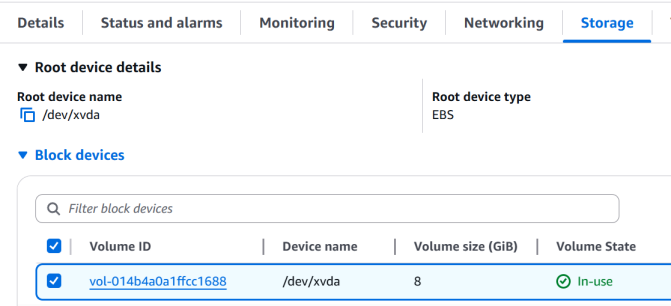
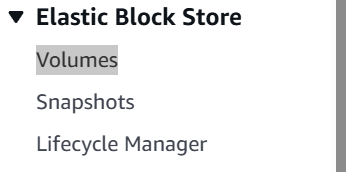
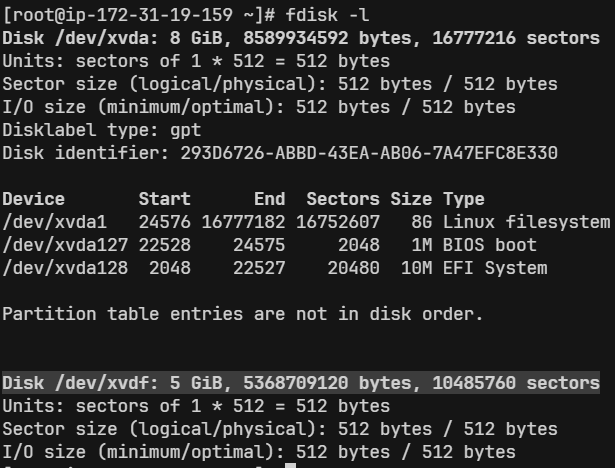
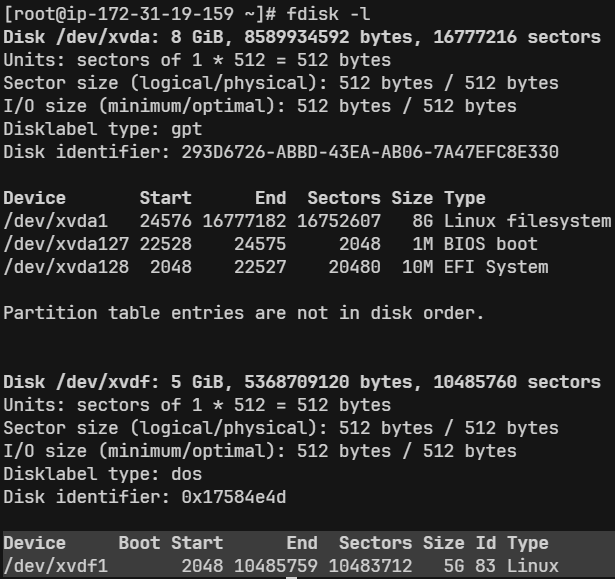
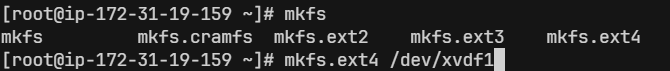
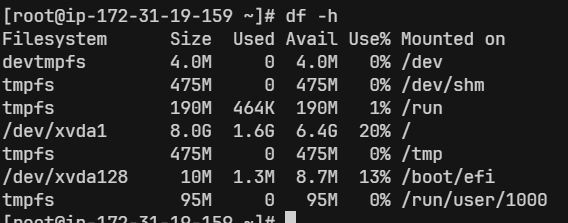
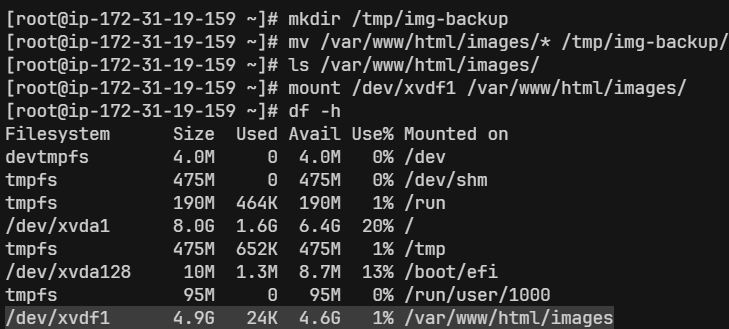
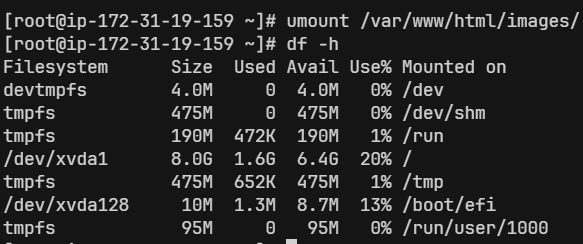
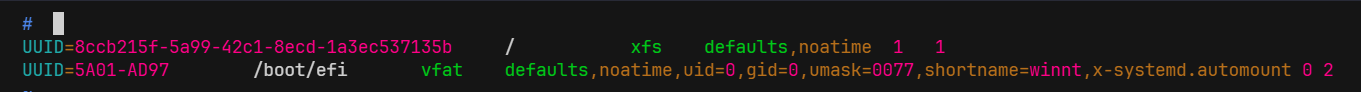
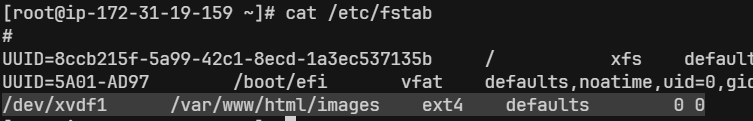
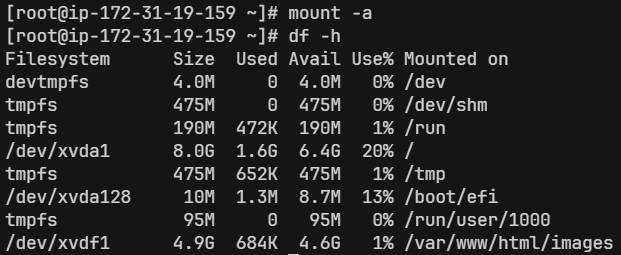
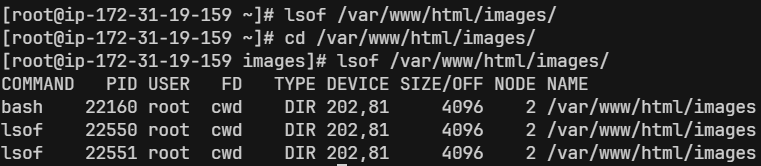
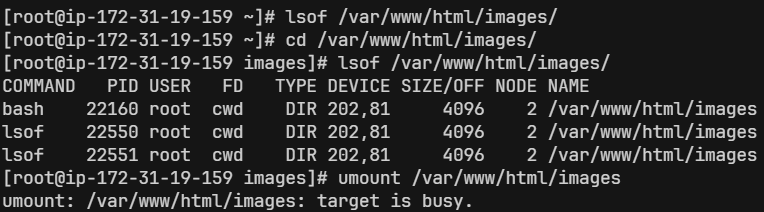
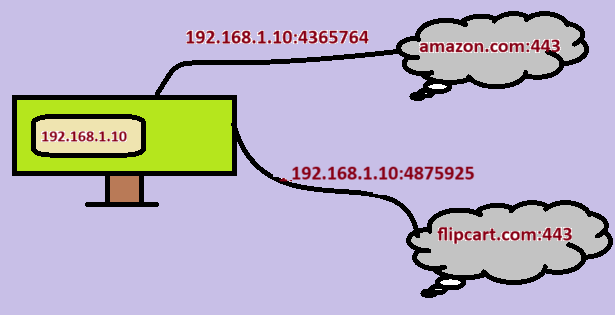
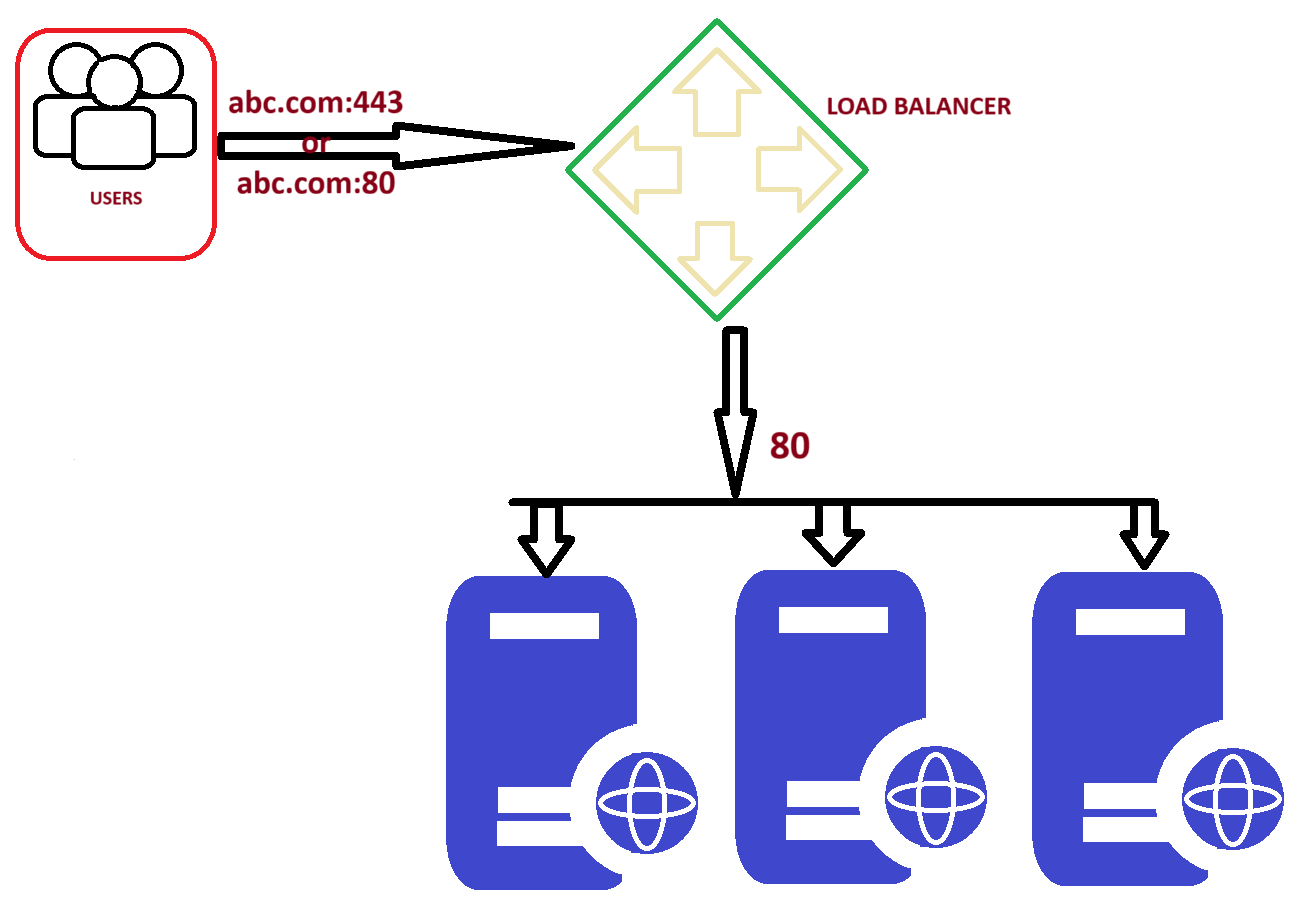
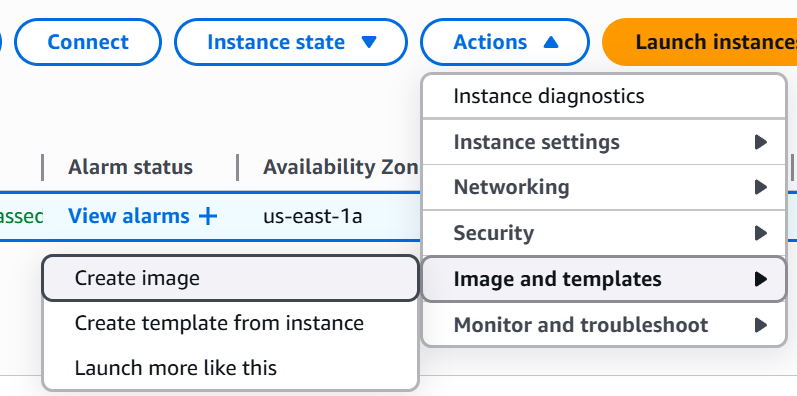
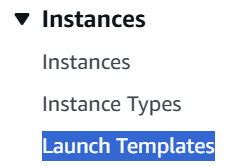
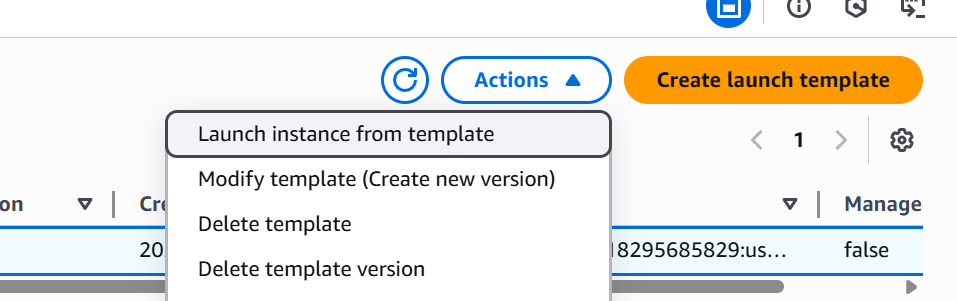
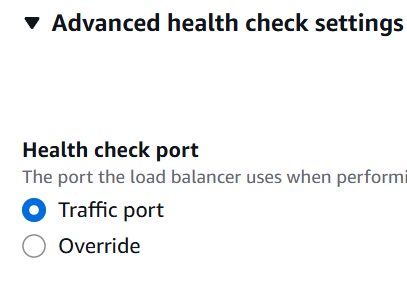
* **Availability Zone (AZ):** 
  + Physically located Data center (or group of them) within a AWS region.
  + **us-east-1 is region**. **us-east-1a, us-east-1b ..etc are AZ**.
  + Each region contains 2 or more AZ.
  + **EBS & EC2 are tied to a specific AZ, not just a region.**
  + **NOTE:** 
    - Let some data centers (AZ) are there inside the region **us-east-1** i.e. **us-east-1a, us-east-1b, us-east-1c**.
    - Let the real name of those **AZ**s are **AZx, AZy, AZz**.
    - Let in my account **us-east-1a** maps to **AZx**. But its not sure that in someone else’s account also **us-east-1a** will be mapping to **AZx**. It might be mapping to **AZy** also.
    - Why these randomized **AZ** names are used?
      * Each **AZ** can be used by multiple users. So, it’s not the reason behind the randomized mappings of **AZs**.
      * It’s because of security concerns, load balancing, fault isolation.

* + You need not choose any particular AZ to run your instance. But you need to choose the region. If you want to be specific that your instance should run in that AZ only then you can choose the particular AZ. However, as EBS and EC2 instance should be in same AZ. So, in that case you need to choose the particular AZ.
  + *So now, Let there are 4 AZs in a region R. You are running your instance in AZ1 (let). For some reasons like power failure or something like that, that AZ (i.e. AZ1) goes down, then your instance will also goes down. AWS doesn’t migrate your instance to any other AZ in that region bcs so many dependencies might be there like EBS, subnet, IPs etc etc. You need to be smart enough to make use of those regions so that your design system will not go down. You can run your instances in many AZs. So that if one goes down then others can take it up. Use load balancer or tools like that to make sure of it.*
* If you are unable to **ssh** to **ec-2** instance in aws, check that **private key** file which is of **.pem** extension. Give **read** permission to user i.e. **chmod 400 <file name>**.
* **(Left Menu)Network and Security > Key pairs** (used to login to the instance through ssh)
  + First create one ssh key
  + You should neither create one key per instance not only one key for all the instance.
  + Better to create key per environment like for dev, q&a, etc. Each env should have separate keys.
  + Also, along with environment, by region also the key should be different.
  + For example: Moso-dev-nvir (Moso is project name, dev is devlopment environment, nvir means the region N.Virginia)
  + You can even give **tags** as well to filter it afterwards.
* (Region is not data center. Each region have at least 2 zone. These zones are data center)
* **(Left Menu)Network and Security > Security Groups** (used for managing the access ips for different protocols like http, ssh etc. You can selete any custom ip that can only access the instance or you can give all ipv4 or all ipv6.. like this)
  + Just like key pairs, you should neither create one SG (security group) per instance not one SG only for all the instances.
  + It should be per environments.
  + For example: moso-web-dev-sg (moso: project name, web: web server, dev: development enviroment).
  + 2 types of rules are there in SG:
    - **Inbound rules**: **FROM** where this security group is allowed to **RECEIVE** traffic)
    - **Outbound rules**: **TO** where this security group is allowed to **SEND** traffic
  + Better to add the inbound rule for the ssh to “My IP”. as you will have to configure the web server inside that instance. Other protocols should be added later.
  + If you change the outbound rules, the internet connectivity might be hampered on the instance as internet traffic goes out from many ports.
* Now we’ll launch our instance as Key Pair and Security Group has been created.
  + Click “Launch Instance” button in **Instances > instances**.
  + Add the tags, try to give proper tags according to project name, environment, owner and all.
    - (like this)
  + Now select the OS image (For now I am selecting Ubuntu Server 24)
  + Instance type: **t2.micro**, it is basically the need of storages and all for the instance.
  + Now add the key pair that we have created earlier.
  + In Network Setting (below the Key Pair section while launching instance), click **edit** and add the **Security Group** that we have created earlier.
  + Now, Launch the instance (you can click on that **Advanced Setting** button and give the provision commands just like vagrant provision but here I am not giving).
* Now, the Instance is created. You need to login to it’s terminal using **ssh** now.
* Go to the instance, and click **connect** button, you’ll get some **ssh** command.
  + 
  + Instead of that highlighted dns link, you can give the public IP of the instance.
  + That “Moso-dev-nvir.pem” is the path of the **Private key** file that was downloaded after creating the **Key Pair** in the beginning of these setups.
  + NOTE: If you are not able to ssh the terminal, check if the private key file (i.e. **.pem** file) is having **read** permission is there for user. If not, **chmod 400 <filename>.pem**
  + Now, host any static site (like downloading the files from tooplate.com and pasting those inside **/var/www/html**)
  + As, earlier you had only added the **ssh** in the security group, so in browser you can’t access the hosted site. So, you need to add the **http** protocol inside the **Security Group**.
  + 
* When you stop your instance, the public IP will be gone. And when you again start your instance, a new public IP will appear.
  + To freeze one public IP, you can go to **Elastic IPs** and allocate one IP. And associate this IP to your instance. (You need to release the IP otherwise it’ll charge you for this)
* You can associate multiple security groups also to an instance.
  + (instance should be running)
* NOTE: When you create one instance and attach the SG and Key Pairs; Network Interface gets created and all these things get attached to that N/W interface only not to the instance.
* Another thing that gets created is **Volume**.
* **AWS in CLI:**
  + First create one user.
    - Search for “IAM” in the search bar.
    - Click on IAM.
    - Go to **users** page.
    - **Create User** giving the necessary policies.
    - After creating user, go to that user and **create access key**(inside **Security Credentials** tab) to use this in CLI.
    - 
    - (you’d have got something like this, copy paste these things in cli)
    - After clicking the **done** button in this page, the access key will be gone. You can’t see the keys if you have not downloaded the csv file. You’ll have to delete this and create new access key if you’ve forgotten the keys.
* **aws help (not --help)**
  + To get all the commands
  + **aws ec2 help** (to get all the commands of ec2 service)
* **aws sts get-caller-identity** (sts: Security Token Service)
  + 
* **Some important commands of EC2 service in awscli:**
  + 
  + 
  + 
  + 
  + 
  + 
  + 
* **aws configure**
  + Give security access key id and key to login with that particulate user.
* **aws ec2 create-security-key --key-name “<key name>” --output text --query “KeyMaterial” > <key-pair-file-name>.pem**
  + **--query :**
    - (without query)
    - We need only the value of **KeyMaterial** key. So pass it inside the **--query** to get that value only.
    - **>** is nothing but the output redirection.
* **aws ec2 create-security-group --group-name "test-sg" --description "test-sg-description"** 
  + Create security group. (after creating you can set the rules like inbound or outbound etc etc)
  + 
* aws ec2 authorize-security-group-ingress --group-name "test-sg" --protocol tcp --port 22 --cidr "$(curl [https://checkip.amazonaws.com/)/32"](https://checkip.amazonaws.com/)/32\")
  + <https://checkip.amazonaws.com/> this just give your current public IP
  + **ingress** means inbound.
  + Port 22 is for **SSH**.
  + 
* 
  + Here, we need only the GroupName and GroupId.
  + So, we can give --query for that.
    - **aws ec2 describe-security-groups --query "SecurityGroups[\*].[GroupName,GroupId]"**
    - 
    - 
  + **aws ec2 run-instances --image-id ami-0a7d80731ae1b2435 --security-groups test-sg --key-name test-key --instance-type t2.micro --count 1**
    - Count 1 means only run one instance.
    - Give proper ami-id otherwise the instance will not be created.
* **EBS (Elastic Block Storage)** vs **S3 (Simple Storage Service)**
  + 
  + There are 2 common types of storage used for different jobs:
    - **Block Storage** (like a computer’s hard disk)
    - **Object Storage** (like Google Drive or Dropbox)
  + **Block Storage:**
    - Stores data in small chunks called blocks.
    - You can create folders, read, and write inside it directly.
    - It behaves like a normal disk, which needs to be formatted and mounted.
  + **Object Storage:**
    - You upload files from anywhere via browser, API, or CLI.
    - You don’t manage folders or file systems — you just upload the object.
    - Each file (object) is stored with:
    - A unique key (like a filename)
    - Metadata (info about the file)
  + In AWS:
    - **EBS (Elastic Block Store)** → Block Storage
      * Acts as the hard drive of an EC2 instance
      * You attach it to EC2 and use it like a disk (e.g., install OS, save DB)
    - **S3 (Simple Storage Service)** → Object Storage
      * Used for storing static files, media, logs, backups, and even static websites
      * Each file gets a unique URL to access over the internet or programmatically
* **EBS**
  + Stores OS data & other data also of EC2.
  + The AZ of EBS should be same as that of EC2 instance. (AZ: Availability Zone)
  + **EBS Snapshot is the state of an EBS volume at a particular point in time. AWS uses S3 internally to store snapshots in a durable and replicated way. You can manage snapshots from the EC2 dashboard, but you can't access them directly through the S3 console.**
  + *It is persistent*. Data stays even if the EC2 is stopped (just like hard-drive).
  + Types of EBS:
    - 
  + In Linux, when you create any partition or attach any new hard-drive, the hard-drive will be linked to (which is called mounting) to a specific folder. Just imagine you are passing a variable to a function as call by reference (in C++)
    - int myfun(int &x) {}
    - Here the same variable will be used as different name. Like this, the srive will be used as some folder like **/mnt/data/**
  + **fdisk -l** 
    - (list all the disc partitions & details)
  + **df -h** 
    - List details about the discs & partitions.
    - How much storage is full or empty, to which directory they are mounted etc etc..
    - 
  + You can check the volume attached to the EC2 instance in AWS console i.e.
    - Click on the instance ID => storage tab => click on the volume ID
    - (or) Elastic Block Store(EBS) => volumes
      * ||
  + Create one volume clicking on the “Create Volume” button in the Volume page.
    - Make sure you select the same AZ as of the EC2 instance.
    - (In free tier, EBS can be at most 30gb. Otherwise you’ll be charged)
  + Select the checkbox on the left of the newly created volume => action => Attach Volume (To attach the volume to the EC2 instance)
    - (highlighted part; after attaching the volume of 5gb)
    - Now we’ll **partition** these volume
  + **NOTE:** 
    - When you attach the EBS volume, it’ll not be mounted. A disk must have a filesystem to be mountable; even if you don’t partition it.
    - **ANALOGY:** Imagine buying a blank notebook — before writing, you draw lines and sections so it’s organized.
      * The disk = blank notebook
      * The file system = lined pages (rules for storing and reading files)
    - **df -h** shows the mounted directory only after the disc is formatted with the file system.
    - So, Now you must be thinking if it the disc is not mounted till now, then why is that **/dev/xvdf** being displayed.
      * That’s not a directory, that’s a device.
      * You need to mount it to “**/mnt/mydata**”. not specifically this folder only, you are free to choose any folder to which the partition will be mounted.
  + **/dev/** directory contains so many types of **devices**.
    - Ex:
      * **/dev/sda** : Hard drives
      * **/dev/xvda** : Root EBS volumes
      * **/dev/xvdf** : Extra EBS volumes
  + **fdisk /dev/xvdf**  : to perform many things. I am doing for partitioning.
    - If you skip the **FIRST** & **LAST sector** with its default value while creating partition, it’ll create only **ONE** partition taking whole disc size.
    - Now, one partition is created. You can see this using **fdisk -l**.
      * 
    - But, now the partition is **raw**, is not having any filesystem within it. So, you need to add the filesystem.
    - To add the filesystem, **mkfs** command is used.
    - In Linux, mostly **ext4** filesystem is used.
      * + **mkfs.ext4 /dev/xvdf1** (shorthand of **mkfs -t ext4 /dev/xvdf1** )
      * Here **xvdfi** means **i**th partition of the device **xvdf**. (**i** is numeric)
      * 
    - But, even now you have not mounted the disc to any folder. So, it won’t be displayed after hitting the command **df -h**.
      * 
    - I want to mount it on **/var/www/html/images/**, so that all the images of my website will be stored in this new drive.
      * **mount** <**partition name**> <**directory path**>
      * **mount /dev/xvdf1 /var/www/html/images/**
      * 
    - This is a temporary mount. If you reboot the instance, this mount will be gone.
      * First unmount the current mount.
        + **umount /var/www/html/images/**
        + 
      * There is a file, **/etc/fstab** (filesystem table), it contains the details about the mounted folders, device names, disc partitions and all so that the file systems should be automatically mounted at boot time.
        + 
      * 
        + I added this line.
        + /dev/xvdf1 : device name
        + /var/www/html/images : mount point (where partitions will appear in filesystem)
        + ext4 : filesystem type
        + defaults : mount options (like read/write, noexec, etc.. )
        + 0 : dump (rarely used; set to 0 (no backup by dump))
        + 0 : fsck order (Set to 0; don’t check filesystem on boot)
      * **mount -a** (it’ll mount everything listed in **/etc/fstab**)
        + 
  + **lsof** : List Open Files
    - In Linux everything is a file.
    - lsof list all the opened files:
      * Which files a process has open
      * Which process is using a specific file/device
      * Which ports are being used
    - Common uses:
      * lsof /dev/xvdf
        + If u get something like “device is busy” errors, (like umount or wipefs)
      * lsof -u ec2-user
        + List what files the user ec2-user is using
      * lsof -i :80
        + Show which process is using port 80
      * lsof
        + List all open files
    - 
      * I was in some other directory and did **lsof**. It was not being used by any process at that time.
      * Then I did cd into that directory and checked with **lsof**. Now its showing someone has done **cd** into that directory.
    - 
      * Now as I have done **cd** into that directory and trying to unmount it, its showing **target is busy**.
  + Using **EBS Snapshot**:
    - Create one instance
    - Create one volume of 5gb
    - Attach the volume to the instance (**/dev/sdh** device; u can take any)
    - Create one folder, /var/lib/mysql.
    - Make partition (**fdisk /dev/xvdh**) and mount the partition to **/var/lib/mysql/** (**mount /dev/xvdh1 /var/lib/mysql**)
    - Install mariadb105-server (as it store the required files inside /var/lib/mysql)
    - **systemctl start mariadb**
    - Now, you can see some files should have come inside **/var/lib/mysql** directory.
      * Those files are inside that new EBS volume as we had mounted that directory to that device.
    - Now, create one EBS Snapshot out of that EBS Volume.
    - Now, go and delete all the files inside the directory **/var/lib/mysql**
    - Now try doing **systemctl restart mariadb** .. it’ll fail because all the required things had been deleted inside the directory **/var/lib/mysql** ..
    - Unmount the disc (EBS Volume) from the instance. Detach it and delete.
    - Now create one volume out of that EBS Snapshot.
    - Attach that volume to the instance.
      * NOTE: This volume contains all the details like before (partition is also there… so no need to make partition again)
    - Mount this device to that directory **/var/lib/mysql**.
    - Now, try doing **systemctl restart mariadb**
      * It’ll succeed now.
* **ELB (Elastic Load Balancer)**
  + 
    - Here, 2 websites are opened inside the PC.
    - PC, maps its current ip with a random port to the website ip with that port.
      * It means the port **192.16.1.10:4365764** means **amazon.com:443** and **192.16.1.10:4875925** means **flipkart.com:443**.
      * This random ports are local to the computer only. The computer use these ports to keep track of the websites.
      * NOTE: ports are in the range 0 to 65535. here the ports that I’ve mentioned are wrong.
  + 
    - Those 443 or 80 are front-end port used by the users.
    - After that the load balancer will forward the traffic to the particular web server via the back-end port (here **80**).
    - NOTE: Web servers will be having different IP but having same port (here **80**)
  + Load balancer:
    - It is a device or software that distributes network traffic across multiple servers or applications to optimize performance and capacity.
    - It acts as a **proxy** between the user and the servers, ensuring that all servers are used equally and that no single server becomes overloaded.
    - Load balancer is not the real server. It acts like a proxy between the users and server(s).
    - **There is a frontend port by which the users access the load balancer (e.g., port 80 for HTTP or 443 for HTTPS). And there is a backend port by which the load balancer forwards the traffic to the different web-servers managing the traffic.**
      * **NOTE: All the web-servers in the backend will be listening on same port.**
  + Types of **ELB**:
    - Classic LB:
      * Takes request from frontend port (443) and routes to the backend server port (80).
      * Ideal for simple solution
      * Works on **layer 4**.
      * Older generation. Only used for backward compatibility.
    - Application LB:
      * Works on **Layer 7**.
      * Intelligent routing based on content.
      * Best suited for **HTTP** & **HTTPS** web traffic.
      * Path based, host based routing.
    - Network LB:
      * Work on **Layer 4**.
      * Handles millions of requests.
      * Used in low-latency or non HTTP traffic.
      * Static IP
      * Very expensive
    - Gateway LB:
      * Works on **Layer 3**.
      * It enables you to deploy, scale & manage virtual appliances such as
        + Firewalls
        + Intrusion detection
        + Prevention system
        + Deep packet inspection systems
  + **HANDS-ON**
    - Launch an instance hosting a static website using httpd. (security group: sg-web (let))
    - Create one AMI out of it.
      * (create image)
    - NOTE: from snapshot you can create a volume, but from AMI you can create one instance.
    - Create one launch template (instances > Launch Templates), so that you don’t need to type all the things while launching instance. (select the created AMI also in that template)
      * 
    - **Launch instance from Template**
      * 
    - Now it comes the **LOAD BALANCER** part …
      * Before creating Load Balancer, create one **Target Group** (Load Balancing > **Target Groups**)
      * Fill all the details; in my case:
        + Target type: instances,
        + Health Checks: **/**

It might vary, it checks if the web-server is healthy or not depending upon the success codes given as input.

I’ve given **/** because my website remains directly in the root path i.e. [http://18.212.102.198](http://18.212.102.198/)**[/](http://18.212.102.198/)** .

You can give the path where your website remains i.e. <http://<ip>:<port>/<any>\_path> like <http://18.212.102.198:80/v1/web.> for this the the health checks path will be **/v1/web**

* + - * + 

You can override this if your web-servers are running on different port.

* + - * Now create the **Load Balancer**. (Load Balancing > Load Balancers)
        + In my case:

I created Application Load Balancer.

Selected us-east-1a to us-east-1f as AZs.

Create one security group (sg-elb (let)) allowing all IPv4 and IPv6 address as HTTP (in my case).

NOTE: this **sg-elb** should be added inside the **sg-web**. It means, **“Allow inbound traffic to instances in sg-web only if it originates from instances in sg-elb”**.

* + Some experiments I did:
    - Experiment 1:
      * Forget about the load balancer thing at all for now.
      * I added “My IP” in sg-web and tried to access the web server from my laptop. It was **accessible**.
      * Then I tried to access it in my mobile. The expectation was that it shouldn’t be accessible from my mobile. But it was **accessible**.
      * ISSUE:
        + I had connected my laptop to my mobile’s hotspot.
        + So, my mobile & laptop were having same public IP.
        + So, the web-server was accessible from my mobile as well.
        + [Laptop] > [Mobile hotspot] > [Internet] > [EC2]
    - Experiment 2:
      * **sg-web** was having the **sg-elb** for HTTP in its inbound rule. (Custom TCP, port 80)
      * **sg-elb** was having all IPv4 and IPv6 address in its inbound rule.
      * But when I was trying to access from my mobile, it was not accessible where I can see it was **healthy** inside the target groups and it was **accessible** from my laptop.
      * ISSUE:
        + In my mobile, the DNS was not able to get resolved.
        + Fetched the IP of the load balancer from its domain (**nslookup <domain>**) … domain means everything except the **http://** or **https://** .
        + Then I tried to access it from my mobile, and it was accessible now.
    - Experiment 3 **(Important)**:
      * I added the outbound rule of **sg-elb** as “**My IP**”.
      * **❌❌❌** I was thinking, if I set some IP in the outbound rule means:
        + When those **IP**, which are valid for the **outbound** rule of the SG, sends traffic to the particular SG… then only the **SG** will forward the traffic further. It is totally wrong **❌❌❌**
      * I set **"My IP"** as the only allowed outbound destination in **sg-elb**. So when I accessed the Load Balancer domain, it couldn't forward the request to the web server because the **web server's IP wasn't permitted** in the **outbound rule** — resulting in an inaccessible server….