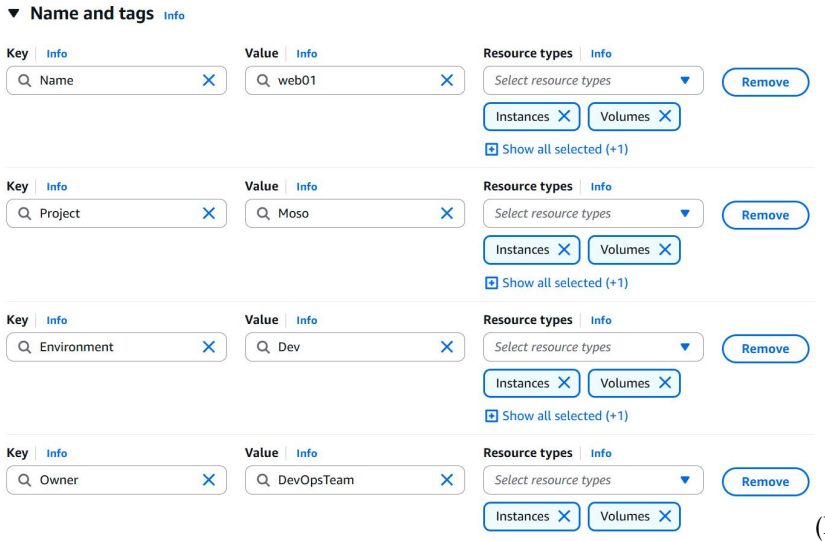
- **Availability Zone (AZ):**
  - Physically located Data center (or group of them) within a AWS region.
  - **us-east-1 is region**. **us-east-1a, us-east-1b ..etc are AZ**.
  - Each region contains 2 or more AZ.
  - **EBS & EC2 are tied to a specific AZ, not just a region.**
  - **NOTE:**
    - Let 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.

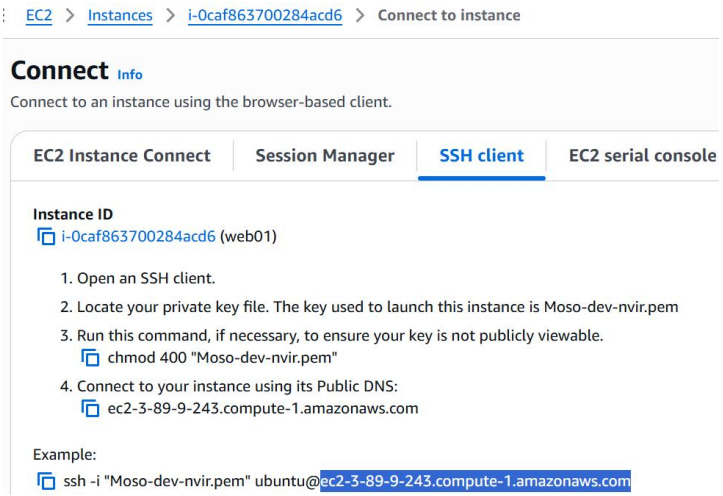


– (like this)

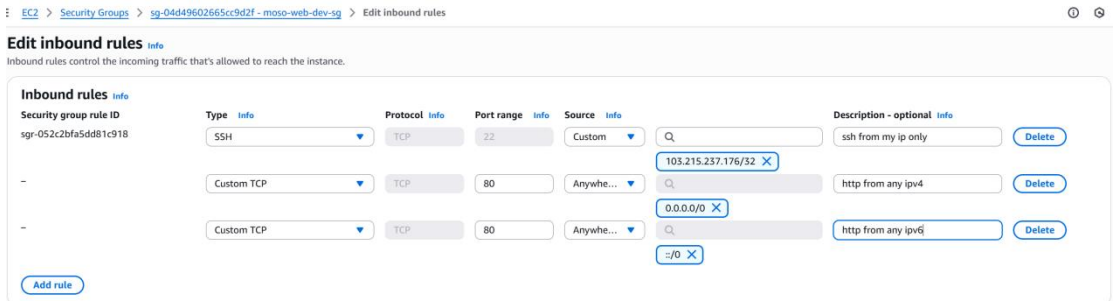
- Now select the OS image (For now I am selecting Ubuntu Server 24)
- Instance type: **t2.micro**, it is basically the need of storages and all for the instance.
- Now add the key pair that we have created earlier.
- In Network Setting (below the Key Pair section while launching instance), click **edit** 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).

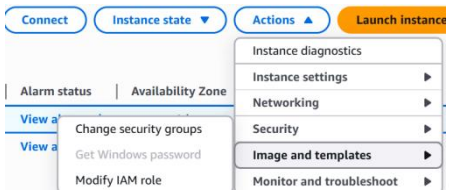
➢ Now, the Instance is created. You need to login to it's terminal using **ssh** now.

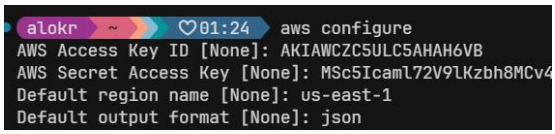
➢ Go to the instance, and click **connect** button, you'll get some **ssh** command.

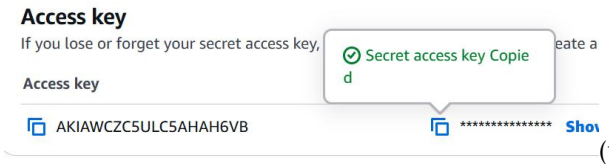


- Instead of that highlighted dns link, you can give the public IP of the instance.
- That "Moso-dev-nvir.pem" is the path of the **Private key** file that was downloaded after creating the **Key Pair** in the beginning of these setups.

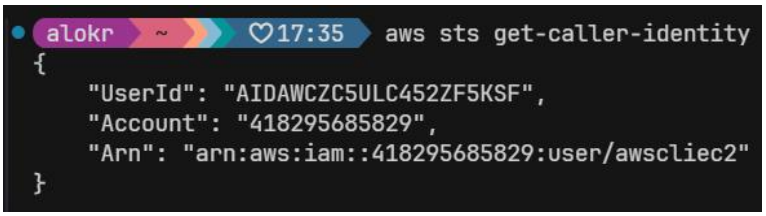
- NOTE: If you are not able to ssh the terminal, check if the private key file (i.e. **.pem** file) is having **read** permission is there for user. If not, **chmod 400 <filename>.pem**
- Now, host any static site (like downloading the files from tooplate.com and pasting those inside **/var/www/html**)
- 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:**
  - ## 🚀 Instance Lifecycle Commands

Purpose	Command
Launch a new instance	`aws ec2 run-instances`
List all instances	`aws ec2 describe-instances`
Start an instance	`aws ec2 start-instances --instance-ids i-xxxxx`
Stop an instance	`aws ec2 stop-instances --instance-ids i-xxxxx`
Terminate an instance	`aws ec2 terminate-instances --instance-ids i-xxxxx`
Reboot an instance	`aws ec2 reboot-instances --instance-ids i-xxxxx`

  - ## 🔐 Key Pairs

Purpose	Command
Create key pair	`aws ec2 create-key-pair --key-name my-key`
Delete key pair	`aws ec2 delete-key-pair --key-name my-key`
List key pairs	`aws ec2 describe-key-pairs`

🔒 Security Groups

Purpose	Command
Create security group	`aws ec2 create-security-group`
Authorize inbound rule	`aws ec2 authorize-security-group-ingress`
Revoke rule	`aws ec2 revoke-security-group-ingress`
Delete security group	`aws ec2 delete-security-group`
List security groups	`aws ec2 describe-security-groups`

📸 AMI & Snapshots

Purpose	Command
List public AMIs	`aws ec2 describe-images --owners amazon`
Create AMI from instance	`aws ec2 create-image --instance-id i-xxxxx --name "my-ami"`
Describe AMIs	`aws ec2 describe-images`
Deregister AMI	`aws ec2 deregister-image --image-id ami-xxxxx`

🗂️ Volumes (EBS)

Purpose	Command
Create a volume	`aws ec2 create-volume`
Attach to instance	`aws ec2 attach-volume`
Detach volume	`aws ec2 detach-volume`
Delete volume	`aws ec2 delete-volume`
Describe volumes	`aws ec2 describe-volumes`

🌐 Elastic IP (Optional)

Purpose	Command
Allocate Elastic IP	`aws ec2 allocate-address`
Associate with instance	`aws ec2 associate-address`
Release Elastic IP	`aws ec2 release-address`

🔍 Describe & Query (Monitoring)

Purpose	Command
List instances (with filtering)	`aws ec2 describe-instances --filters`
Get instance public IP	`aws ec2 describe-instances --query "Reservations[*].Instances[*].PublicIpAddress"`
List availability zones	`aws ec2 describe-availability-zones`
Describe instance types	`aws ec2 describe-instance-types`

- **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 :**

```
json

{
  "KeyFingerprint": "1a:2b:3c:4d",
  "KeyMaterial": "-----BEGIN RSA PRIVATE KEY-----\n...",
  "KeyName": "my-key",
  "KeyPairId": "key-0abc123456789"
}
```

    - (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)
  -

```
root@awsvm:/awscli# aws ec2 create-security-group --group-name "test-sg" --description "test-sg-description"
{
    "GroupId": "sg-0627e05c5374b8f2b"
}
```

- 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/ this just give your current public IP
  - **ingress** means inbound.
  - Port 22 is for **SSH**.
  -

```
root@awsvm:/awscli# aws ec2 authorize-security-group-ingress --group-name "test-sg" --protocol tcp --port 22 --cidr "$(curl https://checkip.amazonaws.com/)/32"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    16  100    16    0     0      1      0  0:00:16  0:00:15  0:00:01     4
```

-

```
root@awsvm:/awscli# aws ec2 describe-security-groups
{
    "SecurityGroups": [
        {
            "Description": "default VPC security group",
            "GroupName": "default",
            "IpPermissions": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "UserIdGroupPairs": [
                        {
                            "GroupId": "sg-028f587f1e10c8952",
                            "UserId": "418295685829"
                        }
                    ]
                }
            ],
            "OwnerId": "418295685829",
            "GroupId": "sg-028f587f1e10c8952",
```

- Here, we need only the GroupName and GroupId.
- So, we can give --query for that.
  - **aws ec2 describe-security-groups --query "SecurityGroups[*].[GroupName,GroupId]"**

```
root@awsvm:/awscli# aws ec2 describe-security-groups --query "SecurityGroups[*].[GroupName,GroupId]"
[
    [
        "default",
        "sg-028f587f1e10c8952"
    ],
    [
        "test-sg",
        "sg-0627e05c5374b8f2b"
    ],
    [
        "moso-web-dev-sg",
        "sg-04d49602665cc9d2f"
    ]
]
```

  -

```
root@awsvm:/awscli# aws ec2 describe-security-groups --query "SecurityGroups[*].[GroupName,GroupId]" --output "table"
-------------------------------------------
|          DescribeSecurityGroups          |
+------------------+-----------------------+
|  default         |  sg-028f587f1e10c8952 |
|  test-sg         |  sg-0627e05c5374b8f2b |
|  moso-web-dev-sg |  sg-04d49602665cc9d2f |
+------------------+-----------------------+
```

- **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)

🧠 Real-World Analogy

Storage	Real-Life Example
EBS	A hard drive plugged into your laptop
S3	Google Drive or Dropbox — you just upload files and share links

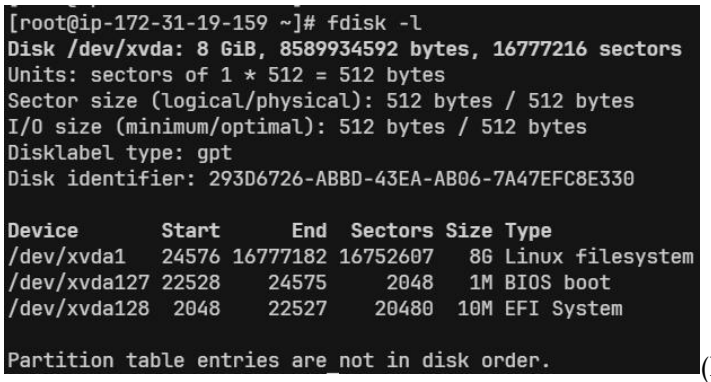
- There are 2 common types of storage used for different jobs:
  - **Block Storage** (like a computer's hard disk)
  - **Object Storage** (like Google Drive or Dropbox)
- **Block Storage:**
  - Stores data in small chunks called blocks.
  - You can create folders, read, and write inside it directly.
  - 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

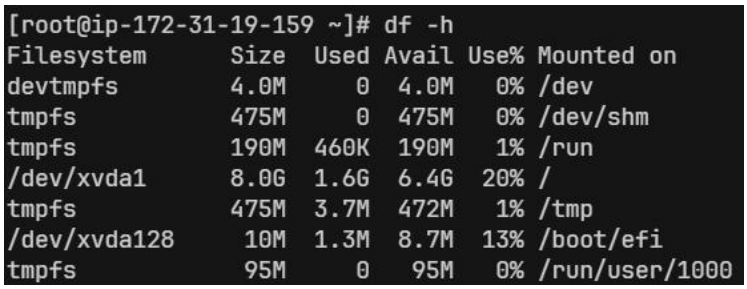
- Stores OS data & other data also of EC2.
- The AZ of EBS should be same as that of EC2 instance. (AZ: Availability Zone)
- **EBS Snapshot is the state of an EBS volume at a particular point in time. AWS uses S3 internally to store snapshots in a durable and replicated way. You can manage snapshots 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).

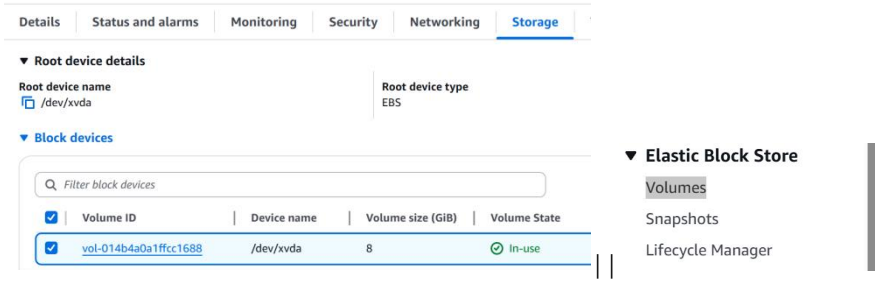
- Types of EBS:

Type	Use Case	Key Feature
gp3 (General Purpose SSD)	Default	Balanced performance & cost
io2 (Provisioned IOPS SSD)	High-performance DBs	Very fast, reliable
st1 (Throughput HDD)	Big data, logs	Good for sequential reads/writes
sc1 (Cold HDD)	Rarely accessed data	Cheapest, slowest

- 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**
  - 
```
[root@ip-172-31-19-159 ~]# fdisk -l
Disk /dev/xvda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 293D6726-ABBD-43EA-AB06-7A47EFC8E330

Device       Start      End  Sectors Size Type
/dev/xvda1   24576 16777182 16752607   8G Linux filesystem
/dev/xvda127 22528    24575     2048   1M BIOS boot
/dev/xvda128  2048    22527    20480  10M EFI System

Partition table entries are not in disk order.
```
  (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..
  - 
```
[root@ip-172-31-19-159 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  460K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  20% /
tmpfs           475M  3.7M  472M   1% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
```
- 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)
  -
```
[root@ip-172-31-19-159 ~]# fdisk -l
Disk /dev/xvda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 293D6726-ABBD-43EA-AB06-7A47EFC8E330

Device       Start      End  Sectors Size Type
/dev/xvda1   24576 16777182 16752607   8G Linux filesystem
/dev/xvda127 22528    24575     2048   1M BIOS boot
/dev/xvda128  2048    22527    20480  10M EFI System

Partition table entries are not in disk order.


Disk /dev/xvdf: 5 GiB, 5368709120 bytes, 10485760 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```
(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**.
    -
```
[root@ip-172-31-19-159 ~]# fdisk -l
Disk /dev/xvda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 293D6726-ABBD-43EA-AB06-7A47EFC8E330

Device       Start      End  Sectors Size Type
/dev/xvda1   24576 16777182 16752607   8G Linux filesystem
/dev/xvda127 22528    24575     2048   1M BIOS boot
/dev/xvda128  2048    22527    20480  10M EFI System

Partition table entries are not in disk order.


Disk /dev/xvdf: 5 GiB, 5368709120 bytes, 10485760 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x17584e4d

Device     Boot Start      End  Sectors Size Id Type
/dev/xvdf1       2048 10485759 10483712   5G 83 Linux
```
  - 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)
    -
```
[root@ip-172-31-19-159 ~]# mkfs
mkfs          mkfs.cramfs  mkfs.ext2    mkfs.ext3    mkfs.ext4
[root@ip-172-31-19-159 ~]# mkfs.ext4 /dev/xvdf1
```
  - But, even now you have not mounted the disc to any folder. So, it won't be displayed after hitting the command **df -h**.
    -
```
[root@ip-172-31-19-159 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  464K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  20% /
tmpfs           475M     0  475M   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
[root@ip-172-31-19-159 ~]#
```

- 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/**
  -
```
[root@ip-172-31-19-159 ~]# mkdir /tmp/img-backup
[root@ip-172-31-19-159 ~]# mv /var/www/html/images/* /tmp/img-backup/
[root@ip-172-31-19-159 ~]# ls /var/www/html/images/
[root@ip-172-31-19-159 ~]# mount /dev/xvdf1 /var/www/html/images/
[root@ip-172-31-19-159 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  464K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  20% /
tmpfs           475M  652K  475M   1% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
/dev/xvdf1      4.9G   24K  4.6G   1% /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/**
    -
```
[root@ip-172-31-19-159 ~]# umount /var/www/html/images/
[root@ip-172-31-19-159 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  472K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  20% /
tmpfs           475M  652K  475M   1% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
```
  - 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.
    -
```
# 
UUID=8ccb215f-5a99-42c1-8ecd-1a3ec537135b     /          xfs    defaults,noatime  1   1
UUID=5A01-AD97        /boot/efi       vfat    defaults,noatime,uid=0,gid=0,umask=0077,shortname=winnt,x-systemd.automount 0 2
```
    -
```
[root@ip-172-31-19-159 ~]# cat /etc/fstab
#
UUID=8ccb215f-5a99-42c1-8ecd-1a3ec537135b     /           xfs    default
UUID=5A01-AD97        /boot/efi       vfat    defaults,noatime,uid=0,gid
/dev/xvdf1      /var/www/html/images    ext4    defaults        0 0
```
      - 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**)

```
[root@ip-172-31-19-159 ~]# mount -a
[root@ip-172-31-19-159 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  472K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  20% /
tmpfs           475M  652K  475M   1% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
/dev/xvdf1      4.9G  684K  4.6G   1% /var/www/html/images
```

- 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

```
[root@ip-172-31-19-159 ~]# lsof /var/www/html/images/
[root@ip-172-31-19-159 ~]# cd /var/www/html/images/
[root@ip-172-31-19-159 images]# lsof /var/www/html/images/
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash    22160 root  cwd    DIR 202,81     4096    2 /var/www/html/images
lsof    22550 root  cwd    DIR 202,81     4096    2 /var/www/html/images
lsof    22551 root  cwd    DIR 202,81     4096    2 /var/www/html/images
```

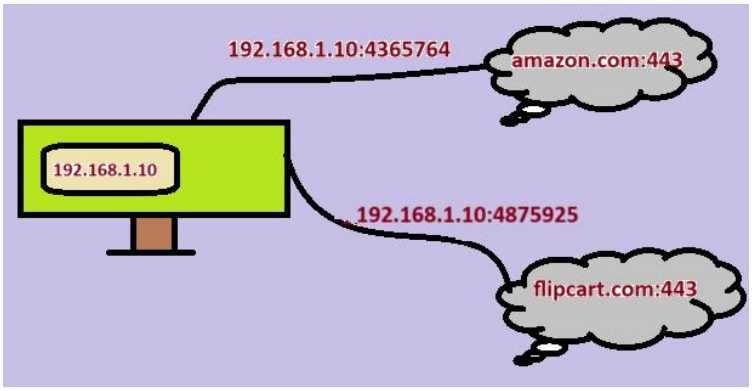
  - I was in some other directory and did **lsof**. It was not being used by any process at that time.
  - Then I did cd into that directory and checked with **lsof**. Now its showing someone has done **cd** into that directory.

```
[root@ip-172-31-19-159 ~]# lsof /var/www/html/images/
[root@ip-172-31-19-159 ~]# cd /var/www/html/images/
[root@ip-172-31-19-159 images]# lsof /var/www/html/images/
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash    22160 root  cwd    DIR 202,81     4096    2 /var/www/html/images
lsof    22550 root  cwd    DIR 202,81     4096    2 /var/www/html/images
lsof    22551 root  cwd    DIR 202,81     4096    2 /var/www/html/images
[root@ip-172-31-19-159 images]# umount /var/www/html/images
umount: /var/www/html/images: target is busy.
```

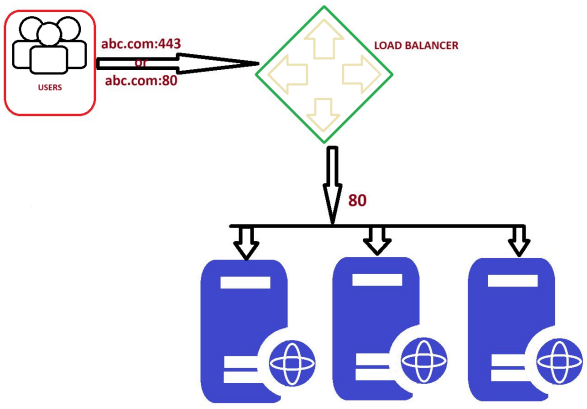
- Now as I have done **cd** into that directory and trying to unmount it, its showing **target is busy**.

- 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.

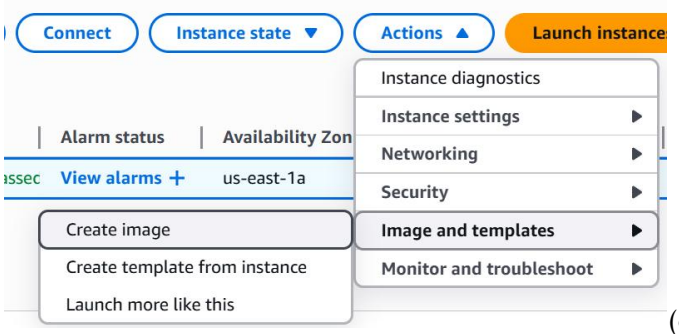


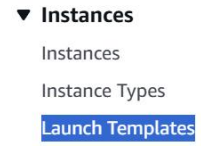
  - Those 443 or 80 are front-end port used by the users.
  - After that the load balancer will forward the traffic to the particular web server via the back-end port (here **80**).
  - NOTE: Web servers will be having different IP but having same port (here **80**)
- Load balancer:
  - It is a device 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.

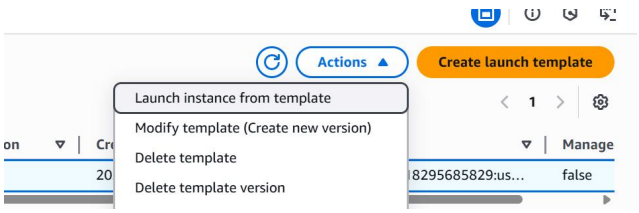
- Load balancer is not the real server. It acts like a proxy between the users and server(s).
- **There is a frontend port by which the users access the load balancer (e.g., port 80 for HTTP or 443 for HTTPS). And there is a backend port by which the load balancer forwards the traffic to the different 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

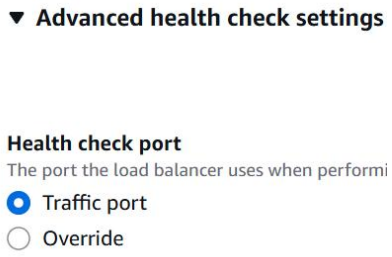
- **HANDS-ON**
  - Launch an instance hosting a static website using httpd. (security group: sg-web (let))

- 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/ .
      - 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.

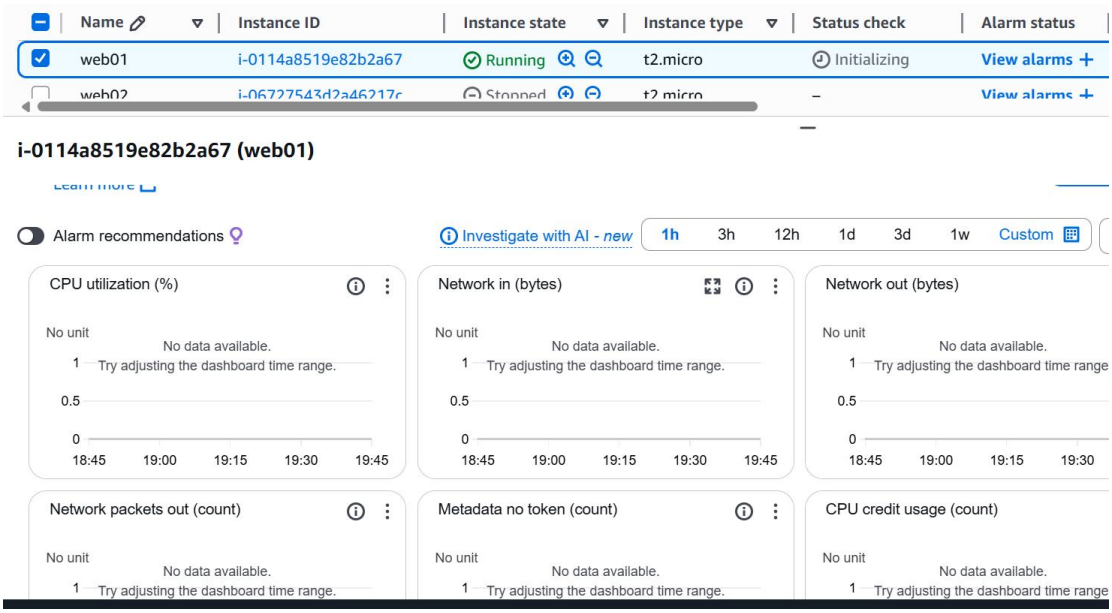
- Experiment 3 **(Important)**:
  - I added the outbound rule of **sg-elb** as "**My IP**".
  - ❌❌❌ I was thinking, if I set some IP in the outbound rule means:
    - When those **IP**, which are valid for the **outbound** rule of the SG, sends traffic to the particular SG… then only the **SG** will 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….

➢ CLOUD WATCH

- Monitor performance of AWS environment - standard infrastructure metrics.
- Metrics: AWS cloud watch allows you to record metrics for services such as EBS, EC2, ELB, Route53 Health checks, RDS, Amazon S3, cloudfront etc etc ..

🔍 What Does CloudWatch Do?

1. **Monitoring Metrics**
   - Collects and tracks metrics like **CPU usage**, **memory**, **disk**, **network**, etc., from AWS services such as EC2, RDS, Lambda, ECS, etc.
2. **Log Collection**
   - Collects and stores **logs** from applications, services, and operating systems (like `/var/log/messages` or app logs).
3. **Alarms & Notifications**
   - You can set **CloudWatch Alarms** to trigger actions (like send an email via SNS or auto-scale instances) when metrics cross a threshold.
4. **Dashboards**
   - Create visual dashboards to **view real-time graphs** of metrics.
5. **Events (Now called EventBridge)**
   - Respond to changes in your AWS environment, like an EC2 instance state change or an EBS snapshot being created.
6. **CloudWatch Agent**
   - A custom agent installed on EC2 or on-prem servers to collect **more detailed system-level metrics and logs**.



- By default the monitoring happens in a interval of **5 mins**. If you want to reduce it to **1 min**, then **Manage Detailed Monitoring > Detailed Monitoring (ENABLE)**.

- We'll try to make one cloud watch so that if CPU utilization is more than expected then it'll send one mail.
  - **EC2 Instance** ---- (create alarm) ---- **Amazon Cloudwatch** -------- **Alarm** ---- (alarm triggered) ---- **Email Notification(SNS)**
  - There is a package **"stress"** which can be used to give stress to the CPU. (it's not preinstalled. You need to install it)

```
[root@ip-172-31-81-10 ~]# stress
`stress' imposes certain types of compute stress on your system

Usage: stress [OPTION [ARG]] ...
 -?, --help         show this help statement
     --version      show version statement
 -v, --verbose      be verbose
 -q, --quiet        be quiet
 -n, --dry-run      show what would have been done
 -t, --timeout N    timeout after N seconds
     --backoff N    wait factor of N microseconds before work starts
 -c, --cpu N        spawn N workers spinning on sqrt()
 -i, --io N         spawn N workers spinning on sync()
 -m, --vm N         spawn N workers spinning on malloc()/free()
     --vm-bytes B   malloc B bytes per vm worker (default is 256MB)
     --vm-stride B  touch a byte every B bytes (default is 4096)
     --vm-hang N    sleep N secs before free (default none, 0 is inf)
     --vm-keep      redirty memory instead of freeing and reallocating
 -d, --hdd N        spawn N workers spinning on write()/unlink()
     --hdd-bytes B  write B bytes per hdd worker (default is 1GB)

Example: stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 10s
```

- **nohup <command> <arguments> &**
  - **nohup** => prevent the process from being killed when terminal session ends
  - **&** => runs the process in the background

```
[root@ip-172-31-81-10 ~]# nohup stress -c 4 -t 300 &
[1] 868
[root@ip-172-31-81-10 ~]# nohup: ignoring input and appending output to 'nohup.out'
```

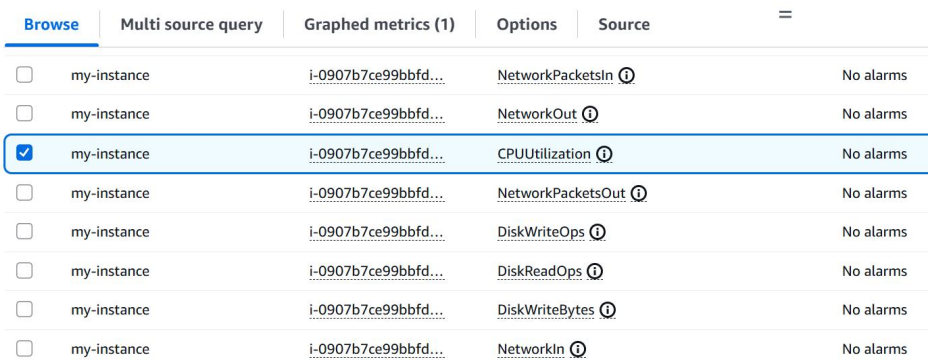
- **top**: The top command in Linux is a real-time **system monitoring tool** that shows **running processes** and their **resource usage**, such as **CPU**, **memory**, and **load average**.

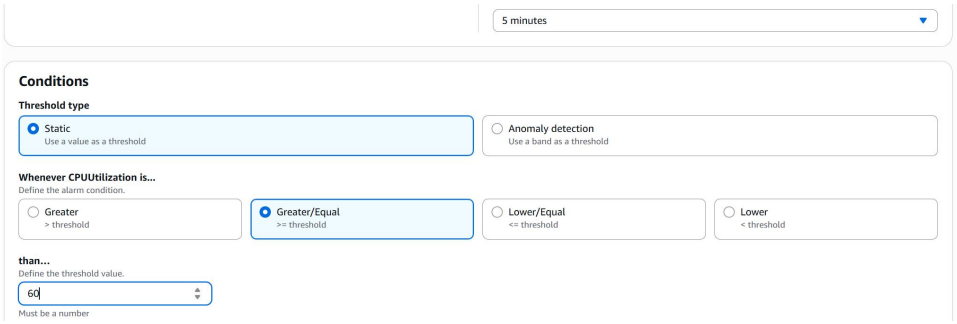
```
[root@ip-172-31-81-10 ~]# top
top - 19:58:00 up  5:50,  1 user,  load average: 3.36, 2.46, 1.10
Tasks:  94 total,   1 running,  51 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   975520 total,   404112 free,    85932 used,   485476 buff/cache
KiB Swap:        0 total,        0 free,        0 used.   745636 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0  123480   5376   3884 S  0.0  0.6   0:02.36 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 kworker/0:0H-ev
```

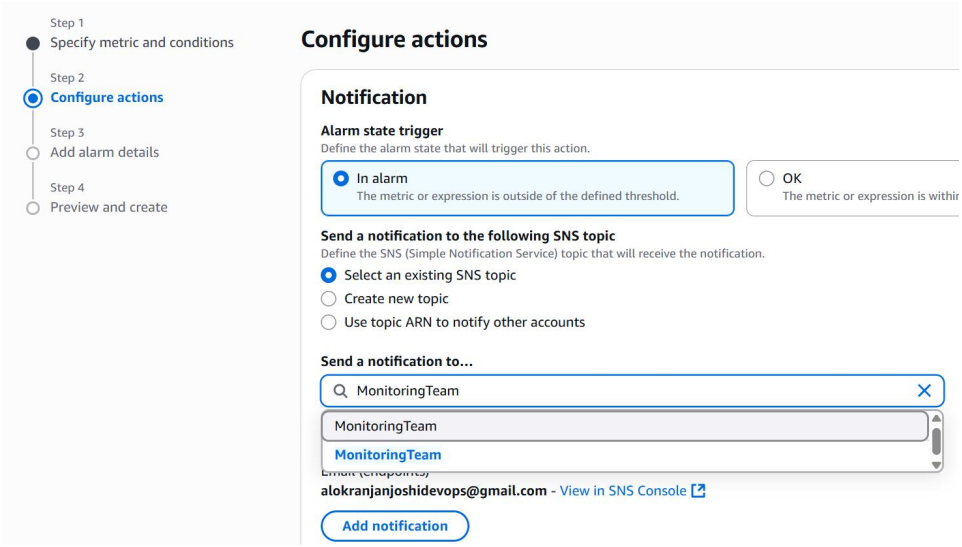
- Search for **Cloud Watch** in AWS console
- Alarms > All alarms
- Create alarm (button)

- Select metric: EC2 => Pre-instance metrics => select the alarm you want to the specific instance (CPU utilization for me)

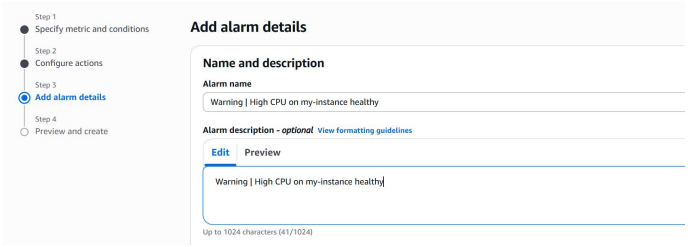


- 

(you can select the conditions as well)

- 

(as I had already created the SNS topic, so I am selecting this).. so many things can be done.. like under the section EC2, if you want to reboot or do something to your instance when the alarm appears u can do that as well.

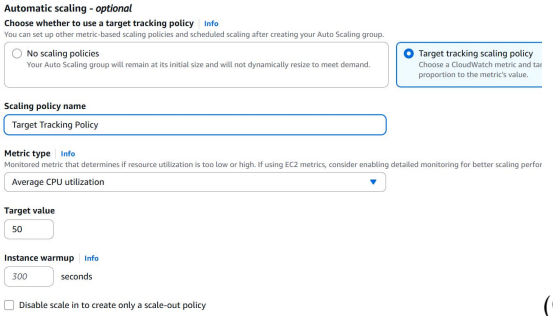
- 

- Then create alarm.
- NOTE: make sure the instance id that is mentioned in the alarm is same as that of the instance you want to monitor.

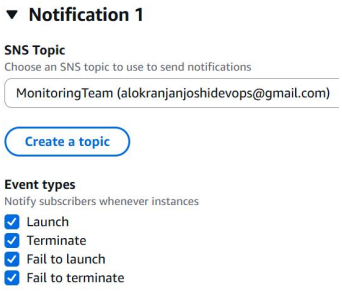
- **EFS (Elastic File System) :**
  - It's kind of same as EBS, but EFS can be **shared among multiple instances.**
  - Creating **filesystem**:
    - Create security group, protocol: **NFS**, in the **inbound** rule add the security group of the web-server **instance** so that it can access the EFS (as its shared).
    - Create EFS, attaching that Security Group, and selecting any applicable options that you want.
  - Accessing the **filesystem**:
    - I am using **Access Point** to access the filesystem. (IAM user can also be created I guess to access this..)
    - Create access point selecting the **filesystem** that you created and by giving all the details that you want.
    - Then click on **Create Access Point.**
  - Mounting **EFS** file system:
    - **EFS Mount Helper** helps in mounting the EFS file system with the instance.
    - Website for the docs: https://docs.aws.amazon.com/efs/latest/ug/installing-amazon-efs-utils.html
    - I am using Amazon Linux 2, so I can directly install it using the command **sudo yum install -y amazon-efs-utils** .
    - Website for the docs: https://docs.aws.amazon.com/efs/latest/ug/mount-fs-auto-mount-update-fstab.html
    - **sudo yum install -y amazon-efs-utils**
    - **file-system-id:/ efs-mount-point efs _netdev,noresvport,tls,accesspoint=access-point-id 0 0** (inside **/etc/fstab**)
      - fs-02d9a586c27435b88:/ /var/www/html/images/ efs _netdev,noresvport,tls,accesspoint=fsap-0088b84d01cd75d8c 0 0 (in my case)
    - **mount -fav**
    - ```
      [root@ip-172-31-82-181 ~]# df -h
      Filesystem      Size  Used Avail Use% Mounted on
      devtmpfs        468M     0  468M   0% /dev
      tmpfs           477M     0  477M   0% /dev/shm
      tmpfs           477M  472K  476M   1% /run
      tmpfs           477M     0  477M   0% /sys/fs/cgroup
      /dev/xvda1      8.0G  2.0G  6.0G  25% /
      tmpfs            96M     0   96M   0% /run/user/1000
      127.0.0.1:/     8.0E     0  8.0E   0% /var/www/html/images
      ```
      - We are seeing 127.0.0.1 instead of the filesystem dns name bcs **under the hood, the helper creates a tunnel through 127.0.0.1 to the real EFS endpoint via a process like stunnel, which is part of the TLS-based mount.**)
      ```

- Try doing **ps aux | grep stunnel**

- **Autoscaling**:
  - It'll create or delete instance depending upon the monitored value.
  - For example, if we set about the CPU utilization, it the CPU utilization exceeds from the threshold, it'll create new instances.
  - It needs a **Launch Template** so that it can launch new instances automatically by itself.
  - So, using the AMI that you created, create one launch template for this.
  - Now **Auto Scaling > Auto Scaling Group**
    - Click on **Create Auto Scaling Group**
    - Step 1:
      - Give a name & select the launch template. Then click on "next"
    - Step 2:
      - Choose the availability zones. (I selected all 6 from us-east-1a to us-east-1f)
    - Step 3:
      - Attach the load balancer (radio inputs).
      - Health checks: select **ELB**. EC2 health check is a very basic health check (hardware health check & vm health check). It doesn't guarentee if the website is up or down.
    - Step 4:
      - Select desired, minimum, maximum capacity. (I chose 2, 1, 3 respectively)
      - Automatic Scaling (policies)
        - If you choose **"No Scaling Policies"** here, then it won't scale. It means if you give all the capacity i.e. desired, min, max as same value. It will **not scale** anything. Just if the instance goes unhealthy, it'll **replace** that.
        - So, I'll choose **"Target Checking Scaling Policy"** as I want to scale up/down depending upon a metrics.

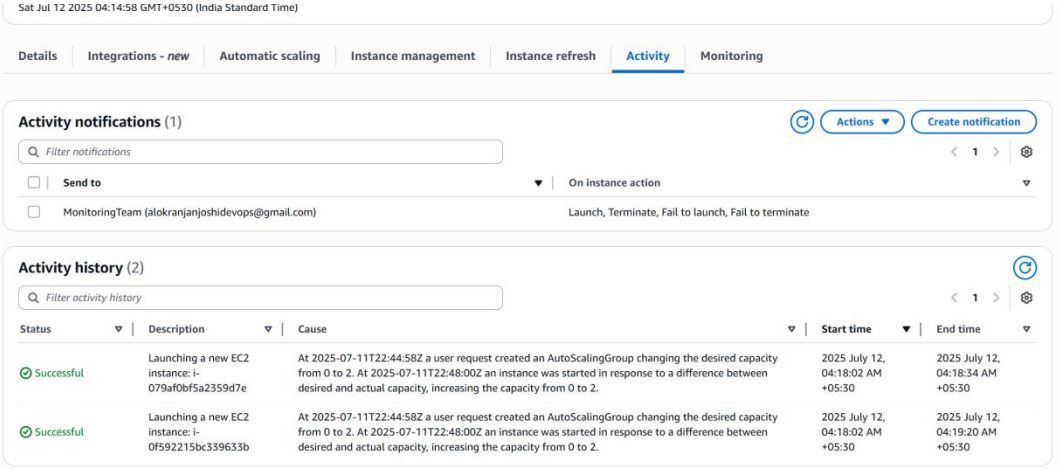


        - (CPU utilization)

- Step 5:
  - 

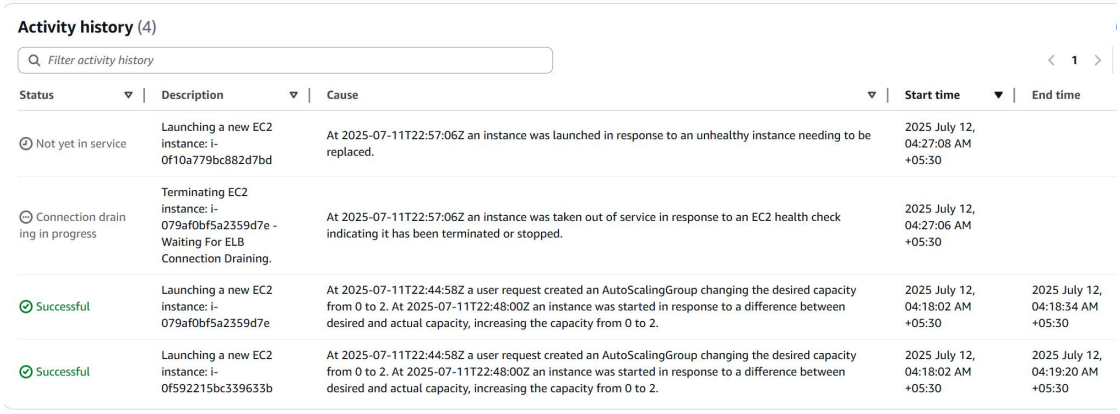
(add notification)
- Step 6:
  - You can give any tag if you want.
- Step 7:
  - Review all the details, and then **Create Autoscaling Group**.

- You can go inside the recently created “Auto Scaling Group”, under the “Activity” tab, you can see the instances will be getting created.



- Target Group will also be get updated according to the instances created.

- I stopped the instances that were created by **Auto Scaling Group**. It checked and found those **unhealthy**. So, it **terminated** those and **created new instances**.



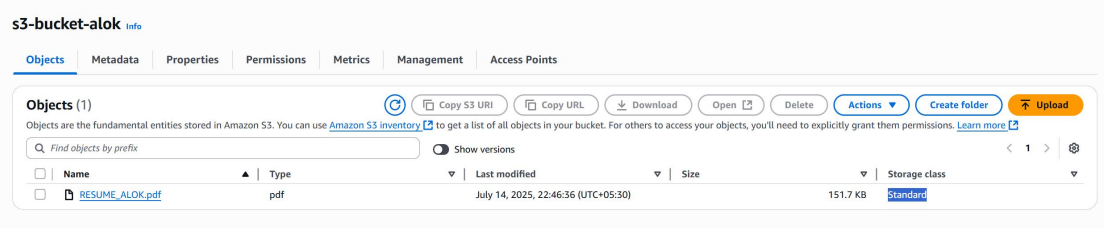
- **Only way to delete the instances is to delete the “auto scaling group” .**

- S3 (Simple Storage Service)
  - It stores data as objects.
  - Building blocks:
    - Buckets:
      - Its like a folder at the root level.
      - You must create a bucket before uploading the objects.
      - Bucket names should be globally unique.
    - Objcts:
      - These are files/data like images, videos, html files, bakcup etc… that you upload.
      - Each object consists of
        - Data (your actual data)
        - Meta-data (key-value pair)
        - A unique key (filename or path inside the bucket)
    - Keys:
      - Keys are the unique identifier of the objects
      - Think of it as the full-path of the file
    - Regions:
      - Buckets are created in specific AWS region.
      - Choose region closer to users for performance
  - 📊 **S3 Storage Classes Comparison Table**

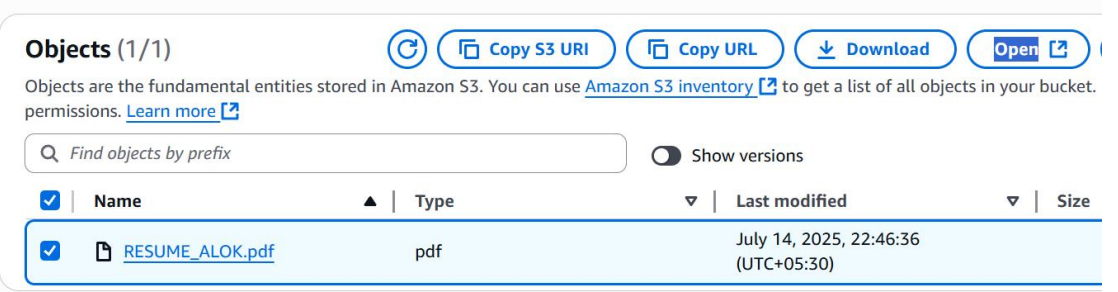
Storage Class	Best For	Cost (per GB)	Retrieval Time	Min Storage Duration	Use Case
S3 Standard	Frequently accessed data	🔥🔥 (Highest)	Immediate	None	Active app data, websites, frequently used files
S3 Intelligent-Tiering	Unknown/variable access patterns	🔥🔥	Immediate (auto-tiers)	30 days (for infrequent tiers)	Cost optimization with automatic tiering
S3 Standard-IA	Infrequently accessed but needed quickly	🔥	Immediate	30 days	Backups, DR, not-often-used files
S3 One Zone-IA	Infrequent access, less critical data	💵 (Cheaper than IA)	Immediate	30 days	Re-creatable data, logs, secondary backups
S3 Glacier	Archival with occasional access	💎 (Very Low)	Minutes to hours	90 days	Archive data, compliance storage
S3 Glacier Deep Archive	Long-term cold storage (rarely accessed)	🟠 (Cheapest)	Up to 12 hours	180 days	Deep archival, regulatory storage

  - IA: Infrequent access
  - Glacier tiers have low storage cost but higher retrieval cost & time
  - Intelligent-Tiering automatically moves data between tiers based on access patterns
  - One Zone-IA stores data in only one AZ (less durable, lower cost)

- Create Bucket
  - Bucket name should be unique worldwide
  - By default ACL (access control list) is disabled. Make it enable if it is required.
  - By default all the public access is disabled. It doesn't mean that you should disable public access. It just confirms public access is not enabled accidentally.
  - Bucket versioning: Making it enable makes it easy to recover the data from previous versions.
  - Encryption is necessary. You have just some options to select the encryption types from the options.
- After the bucket got created
  - Open that bucket and upload any file/folder. (**add file/folder** -> select the **permissions**, **properties** >>> click on **upload** button)
  - Below, in the **properties** section, you can select **storage class**, **encryption** options etc.. overriding the default settings of the buckets. Means, these overridden properties will be applicable to that particular file/folder only, not the entire bucket.



  - By default the objects that are uploaded in the buckets are private.



    - When you click on that **"Open"** button, the file will be loaded in the browser as it's opening the file as the perticular IAM user.
    - Open that file on clicking over it, copy the **URI**, it is a public accessible **URI**. If you open it in a new tab, it'll show **Access Denied**.

- **S3 URI**
  s3://s3-bucket-alok/RESUME_ALOK.pdf

  Amazon Resource Name (ARN)
  arn:aws:s3:::s3-bucket-alok/RESUME_ALOK.pdf

  Entity tag (Etag)
  6594cf955c9f8a85ec29c018c1e66ede

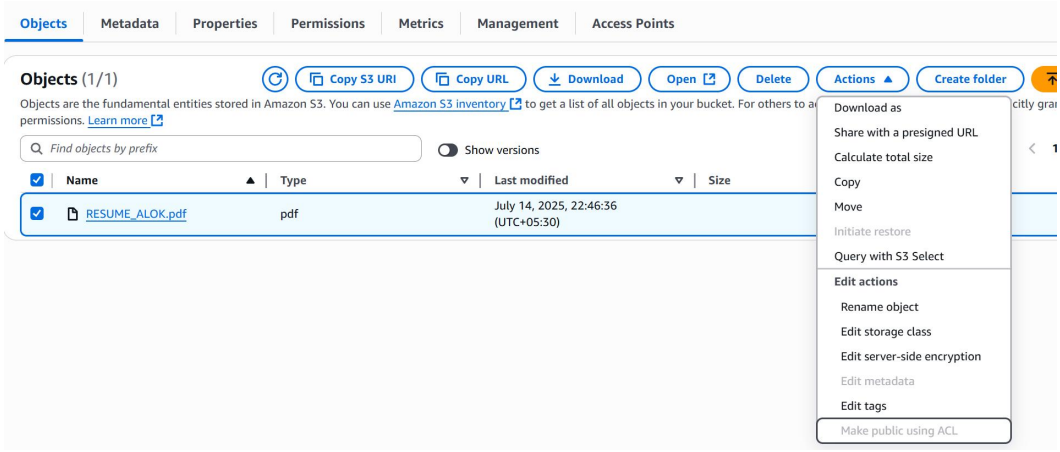
  Object URL
  https://s3-bucket-alok.s3.us-east-1.amazonaws.com/RESUME_ALOK.pdf

  (object URL)

  This XML file does not appear to have any style information associated with it. The document tree is shown below.

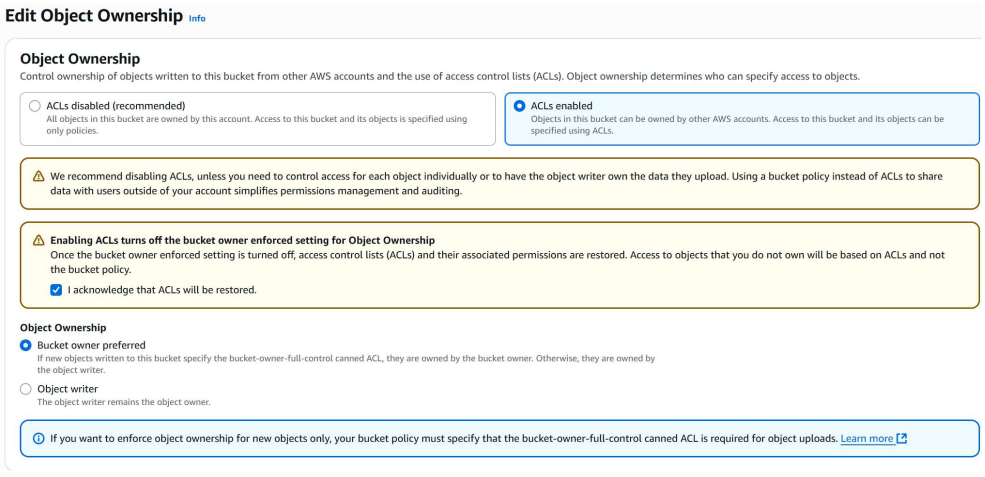
```
<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>GQE42477SJJHT5PQ</RequestId>
  <HostId>gwfzRmpegt9ETcnIiQVbA/EZtmyd2N5pRF64kPmIJKlPS+HUDK9RY5t51t3UiLl3gC0EQGDNH9g=</HostId>
</Error>
```

- To make it public, **select** the objects you want to edit permission of >> **Actions >> Make public with ACL.**



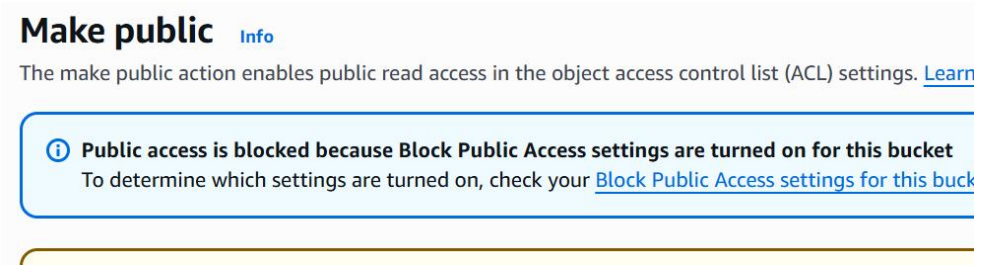
(It is grayed out because we have disabled ACL of the bucket)

- To enable ACL, open the **Bucket >> Permission** tab **>> Object Ownership >> ACLs enabled**



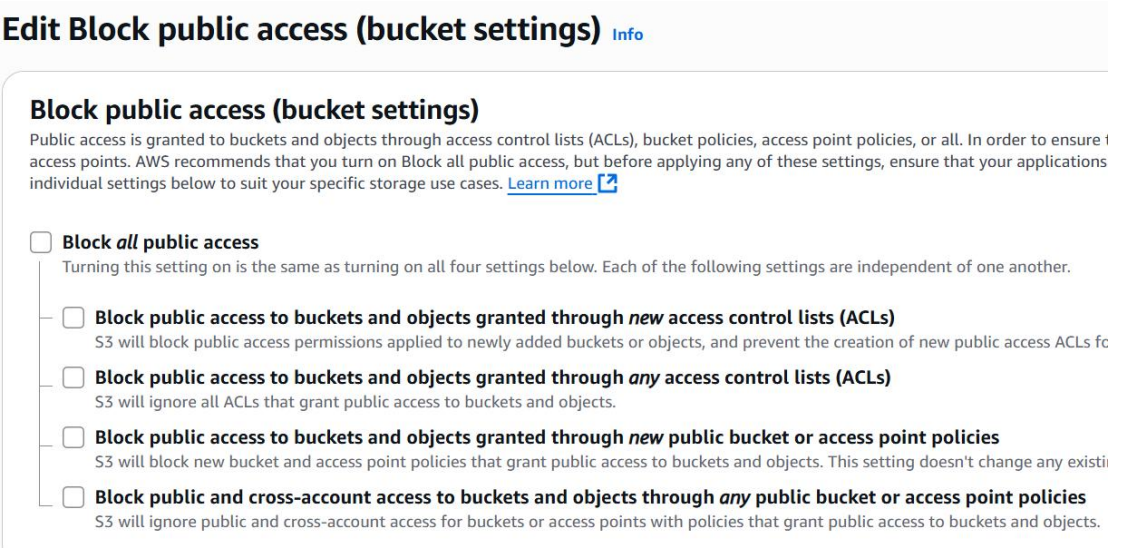
- Now, go inside the bucket, select the file >> **Make public with ACL.**

  You'll get an error

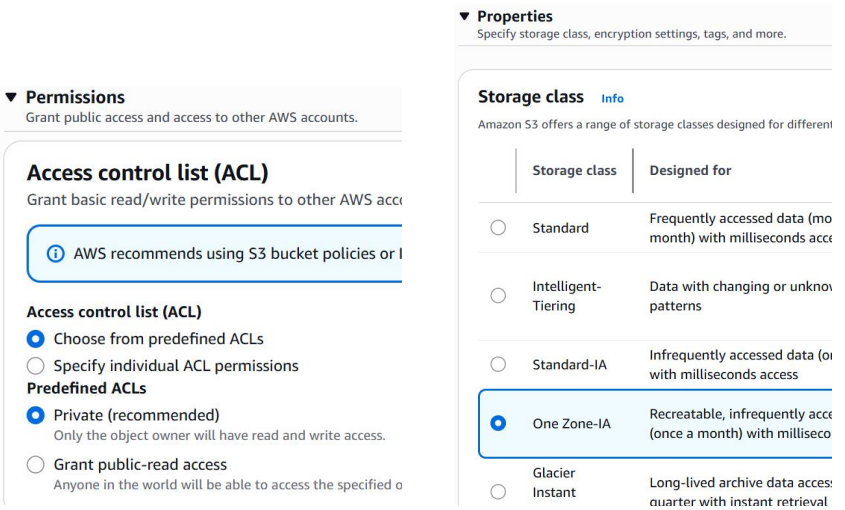


  (Because, the public access is blocked)

- Now, go to the bucket, **Permissions** >> **Block Public Access (bucket setting)** >> Un-check the **block all public access**



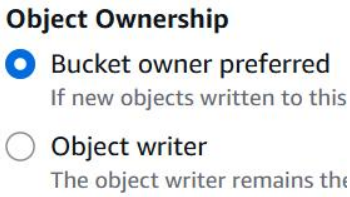
- Now, you can make the object publicly accessible.
- Now, upload one more file



- Upload the file selecting these fields.

- If you try to access the newly uploaded file using the **Object URI** now, you'll get **Access Denied**. (because this file is not edited for public access)
- Means, even the ***Bucket is public***, ***ACLs are enabled*** but ***Buckets are private***.

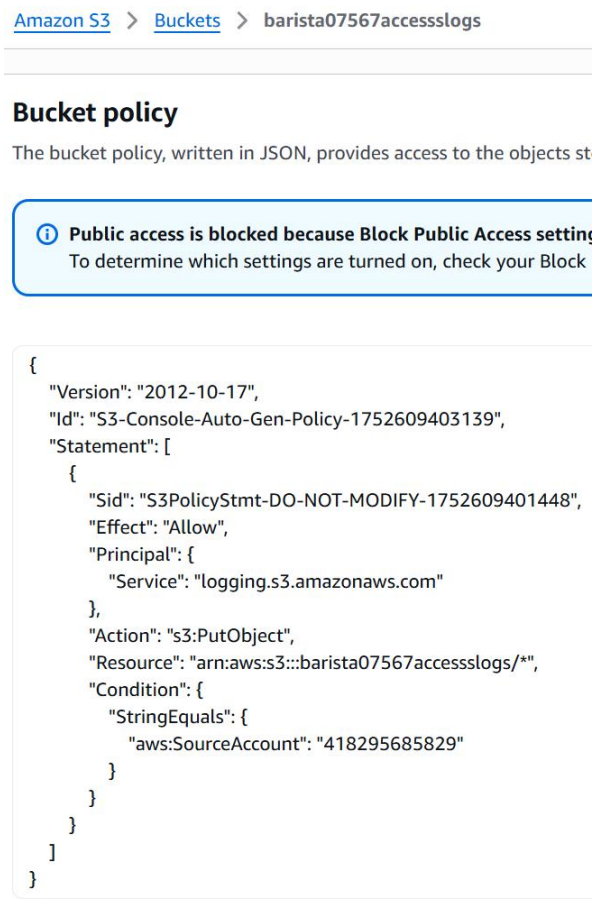
- **NOTES**
  - ACL disabled: Objects owner will be the Bucker owner (its fixed)
  - ACL enabled: You can choose whether the **Object Creater** or the **Bucket Owner** will be the **Object Owner**. (in the above example of uploading files, **Bucket owner preferred** was selected)

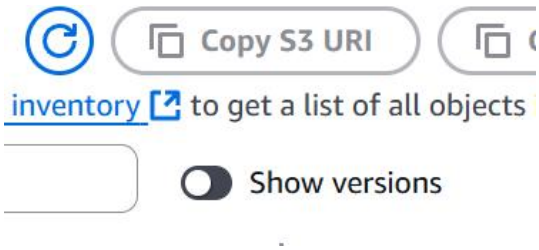
    

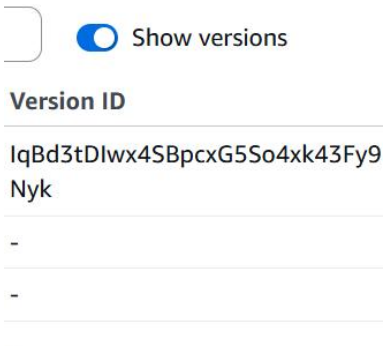
    - -------- this is why ACL is there inside the **Object Ownership**
  - **Bucket Access Control** can be managed by **IAM** & **Bucket policies**. If the **ACL** is enabled, then through **ACL** also **Access Control of Bucket can be managed**.
  - If you allow the public access unchecking the checkbox **"Block all public access"**, now the bucket is publicly accessible, Not the objects. Objects will be still private only.

-

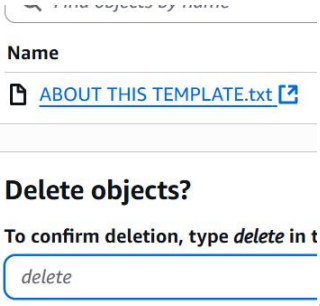
- **Static Website Hosting** using S3:
  - Create 2 buckets
    - one is for hosting the website. Enable the versioning of this
    - Another is for keeping the **Access Logs**.
  - After creating the bucket, upload the static files like html, css, js, images etc etc.. and make them all public.
  - Go to the properties tab, under **static website hosting** section, make it enable and select the index & error file.
  - Now, you'll get a link through which the static website can be accessed.
  - Under the **properties** tab, another option is there **Server Access Logging**. Enable it and select the perticular bucket that we created to store the **Access Logs**.
  - It takes a little time to update the access logs inside the bucket. You can see it when someone access the static website.

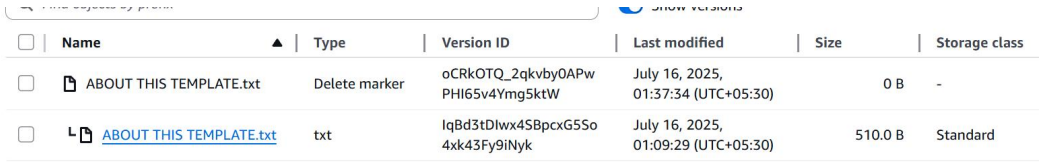


- (bucket policy also gets updated)
- **Versioning** :
  - 

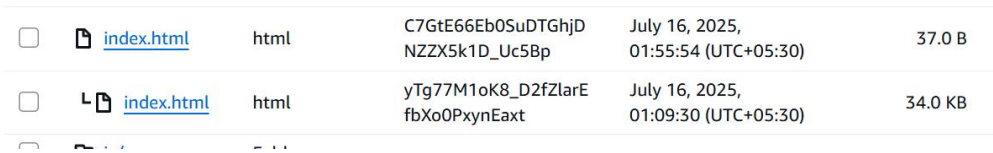
    (toggle option is there to see with/without versioning)….
  - 

    (With enabling that toggler, one more column “Version ID” will be displayed)

- I am trying to delete one file.
  - 

(here, **delete** is coming)
- Now, you can't see the file inside the bucket. But when you enable the toggler to see the versions you will be able to see the file.
  - 

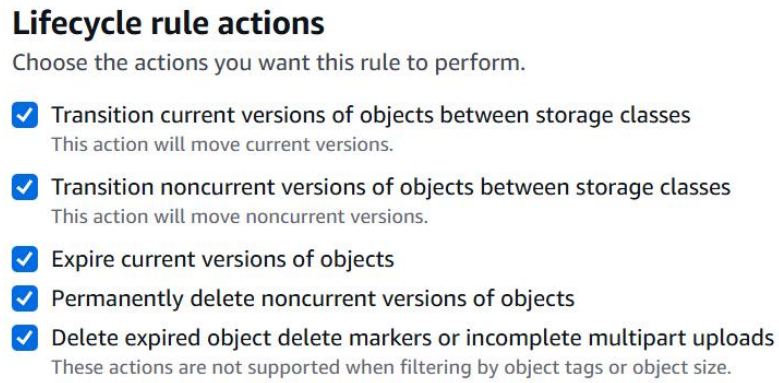
Name	Type	Version ID	Last modified	Size	Storage class
ABOUT THIS TEMPLATE.txt	Delete marker	oCRkOTQ_2qkvby0APw PHI65v4Ymg5ktW	July 16, 2025, 01:37:34 (UTC+05:30)	0 B	-
ABOUT THIS TEMPLATE.txt	txt	IqBd3tDIwx4SBpcxG5So 4xk43Fy9iNyk	July 16, 2025, 01:09:29 (UTC+05:30)	510.0 B	Standard

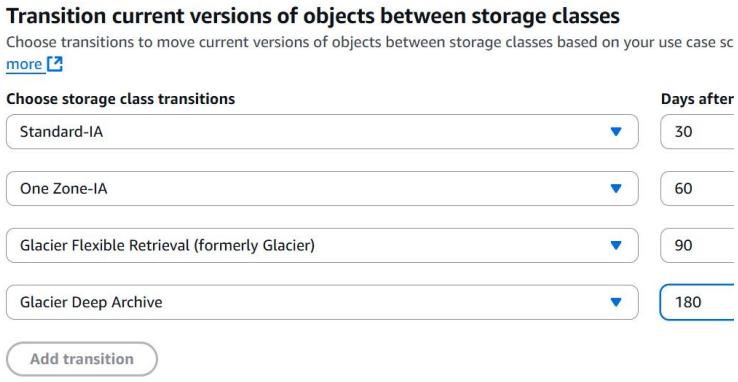
  - You can see, the file(delete marker) is having size **0B**, but the txt is having **510.0B**. (means the file has gone no where, its there only. Just one **delete marker** was introduced.)
    - Whenever you override one file, one version of that file will get created.
  - Now, if I delete that **delete marker**, the file will come back again inside the bucket.
  - When bucket versioning is enabled, deleting a file from the normal view (i.e., without turning on the "Show versions" toggle) will add a delete marker, hiding the file. But when you turn on the "Show versions" toggle, you can see all versions, including delete markers, and get the option to permanently delete any version.
  - Now overriding case,
    - There is one index.html file there. I uploaded one more index.html file to the bucket, so that the previous index.html got replaced by the new one.
    - 

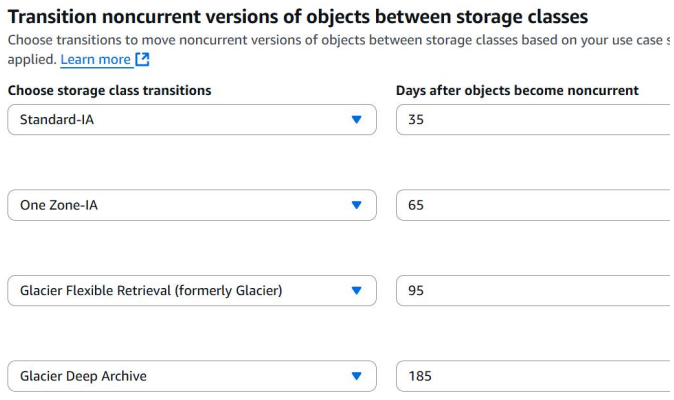
index.html	html	C7GtE66Eb0SuDTGhjD NZZX5k1D_Uc5Bp	July 16, 2025, 01:55:54 (UTC+05:30)	37.0 B
index.html	html	yTg77M1oK8_D2fZlarE fbXo0PxynEaxt	July 16, 2025, 01:09:30 (UTC+05:30)	34.0 KB

    - Now, we have 2 index.html files. First one is the new one and 2nd one is the old one.

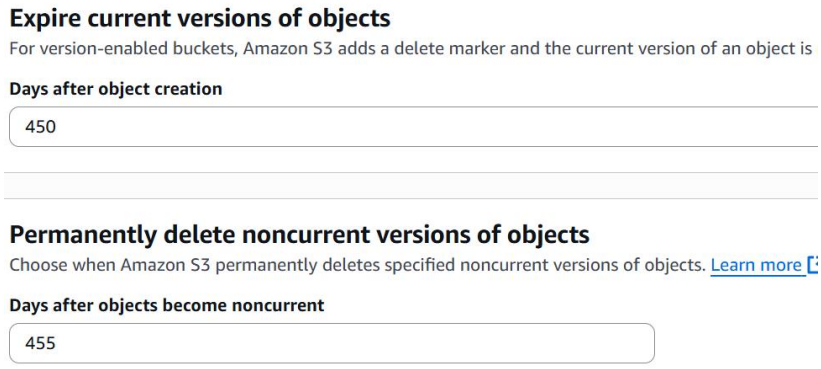
- If you want to revert, then just remove the upper **index.html** **(**with **version toggler on,** otherwise it'll create one **delete marker)**, and its done.

- **Life Cycle Rule** :
  - It is used to transfer the objects to another storage class, or destroying the objects after some days etc etc.
    - On the **management** tab of the bucket, Under **Lifecycle Configuration**, click on **Create Lifecycle Rule**.
    - 

    - You can do this for the current versions and/or previous versions as well.
    - 

    (for current version of the objects)
    - 

    (non current versions of the objects)



(you can select expiration date as well)

- **Disaster Recovery** :
  - It might happen that your data will be lost in case of any disaster.
  - You can create a S3 bucket in another region (like currently I am using N Virginia, we can create on Oregon (for example)).
  - We'll replicate the objects of current bucket to that new bucket.
  - Under **management** tab, there is a section called Replication rules.
  - Create one **replication rule**, select the bucket of **another region**. Either you can select all the objects or any particular.

- **RDS (Relational Database Services)** :

  ـ