ITIM

Name: Khan Mohammad Waseem **Roll No.** 233414

Amazon Web Services (AWS):

AWS stands for Amazon Web Services. It is a comprehensive cloud computing platform provided by Amazon. AWS offers a wide range of cloud services, including computing power, storage, database management, networking, machine learning, artificial intelligence, analytics, and more.

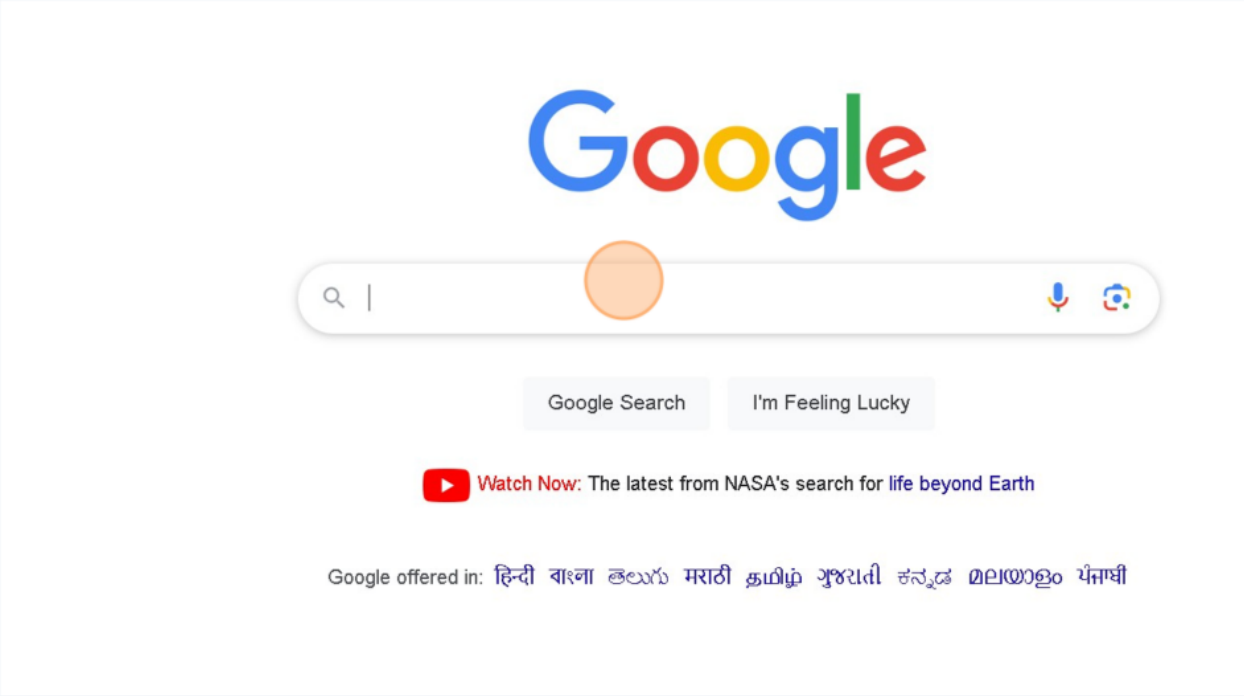
Here are some key components and services offered by AWS:

1. **Elastic Compute Cloud (EC2):** It provides virtual servers in the cloud, allowing users to run applications and services.
2. **Simple Storage Service (S3):** It offers scalable object storage for data backup, archiving, and application hosting.
3. **Relational Database Service (RDS):** It provides managed database services for popular relational database engines such as MySQL, PostgreSQL, Oracle, and SQL Server.
4. **Lambda:** It is a serverless computing service that allows you to run code without provisioning or managing servers.
5. **Amazon DynamoDB:** It is a fully managed NoSQL database service that provides fast and predictable performance at any scale.
6. **Amazon SNS:** It is a fully managed messaging service that enables the distribution of messages to a large number of recipients.
7. **Amazon Redshift:** It is a fully managed data warehousing service that allows you to analyze large volumes of data using SQL queries.
8. **Amazon Glacier:** It is a low-cost storage service for data archiving and long-term backup.
9. **Amazon Elastic Beanstalk:** It is a platform as a service (PaaS) that allows developers to deploy and manage applications easily.
10. **Amazon Machine Learning:** It is a service that provides tools and wizards for creating machine learning models and generating predictions.

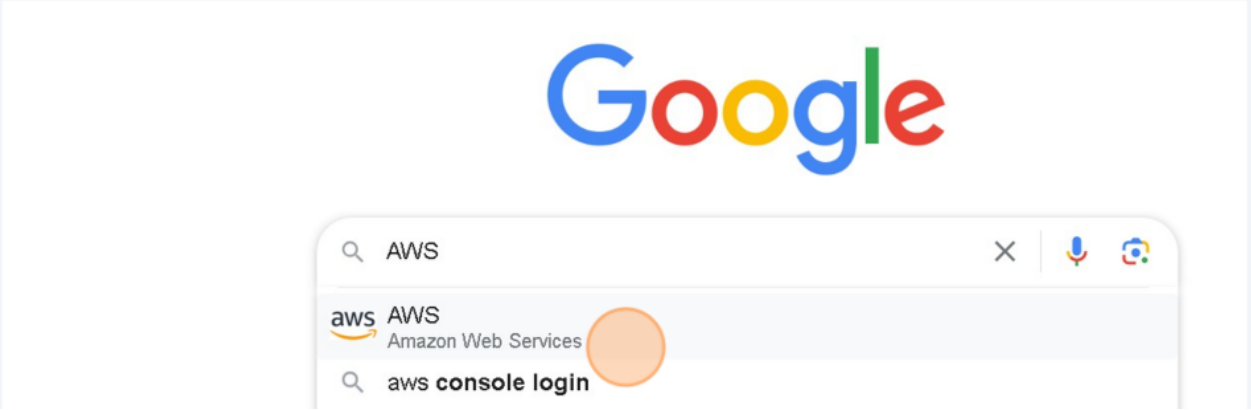
Create an AWS account.

1. Navigate to www.google.com

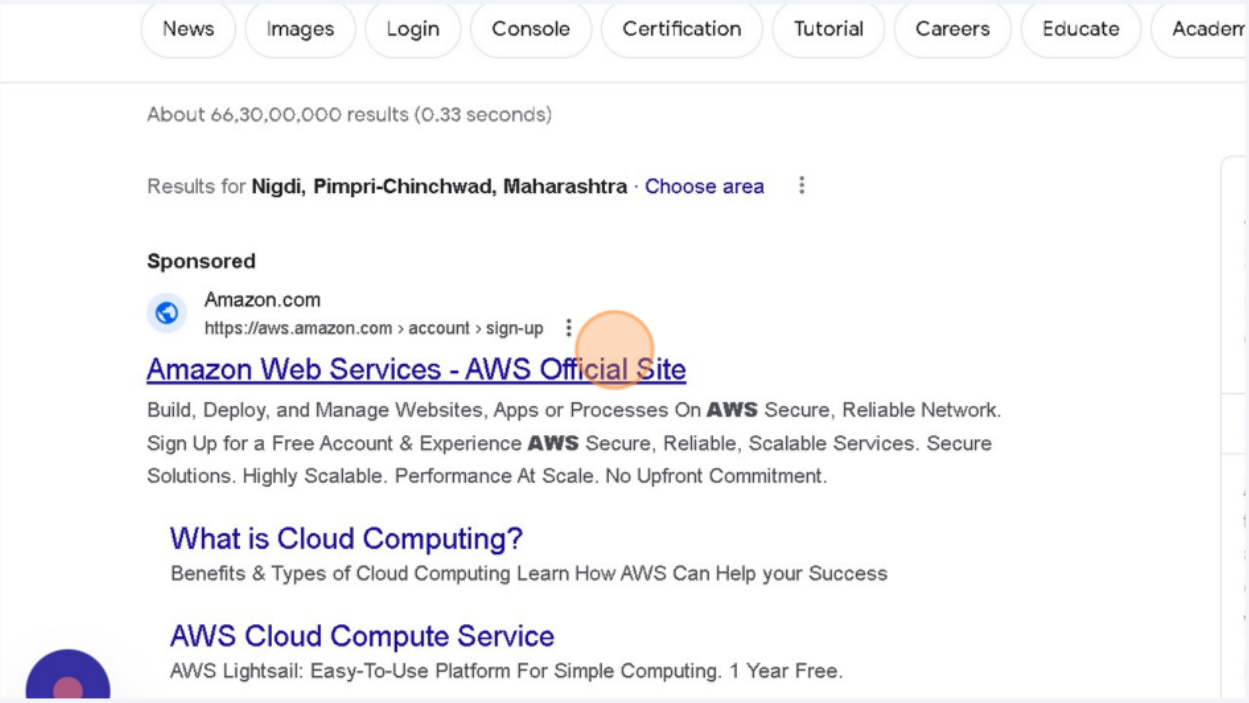
2. Click on the “Search” field.



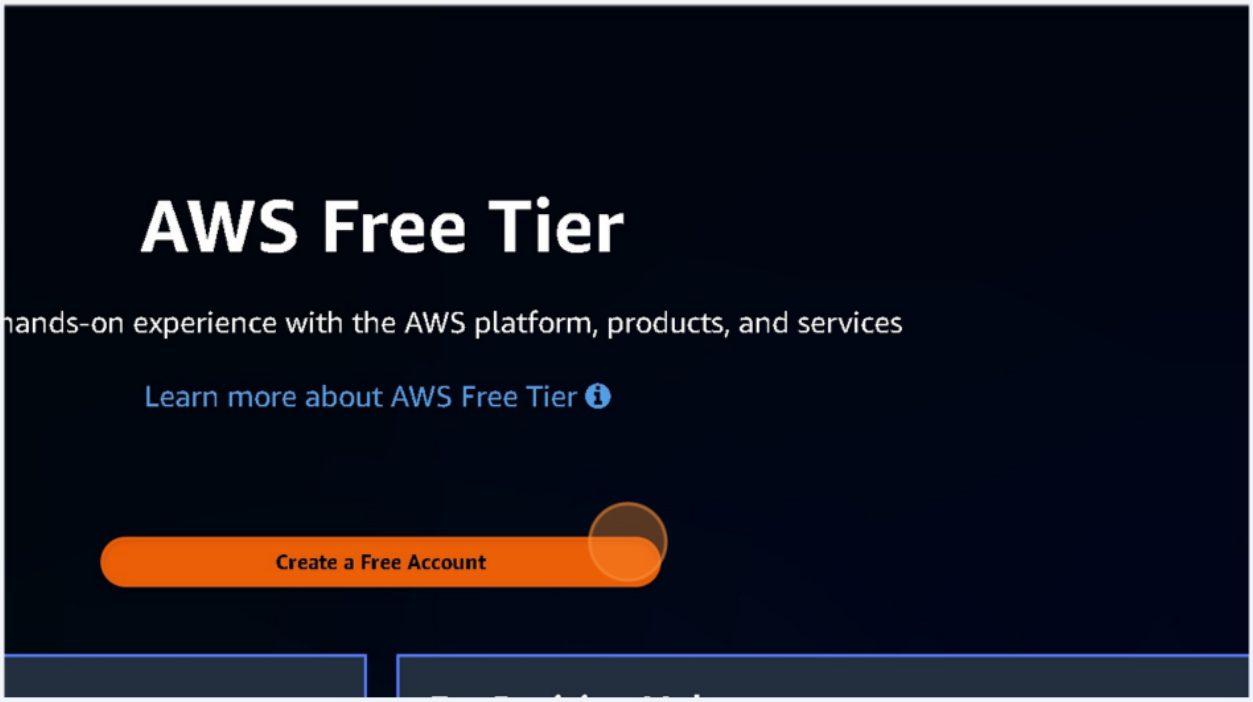
4. Type AWS and Click: AWS



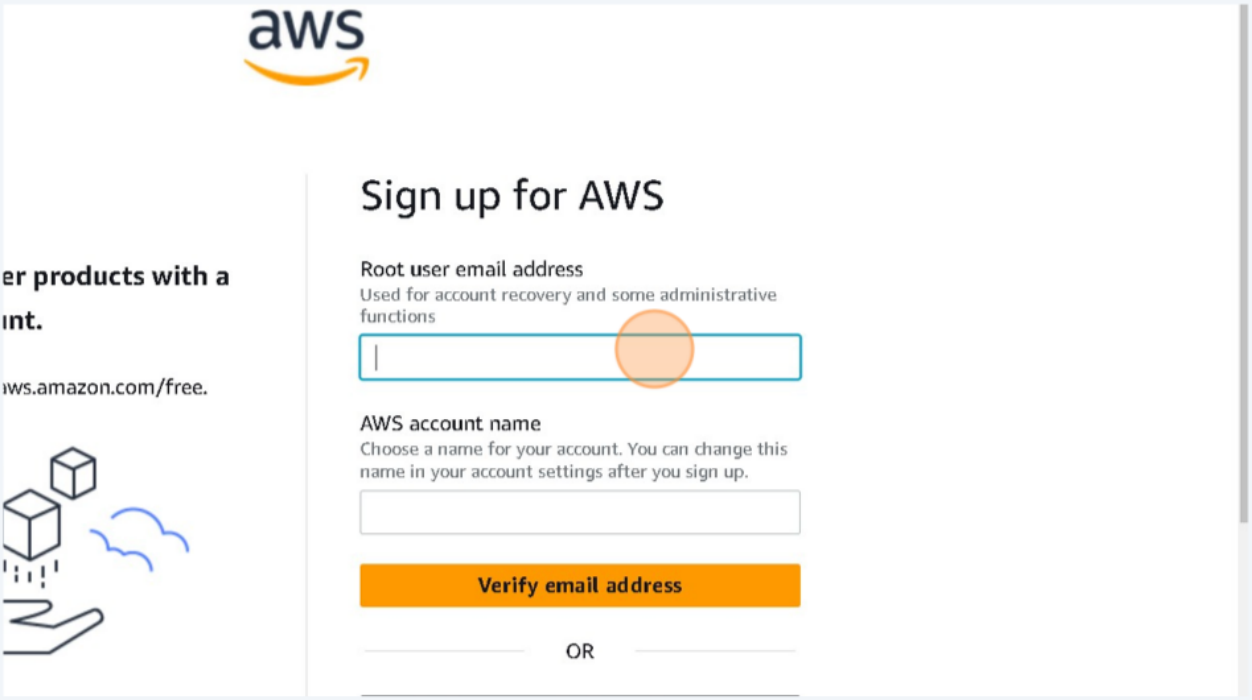
5. Click: "Amazon Web Services - AWS Official Site"



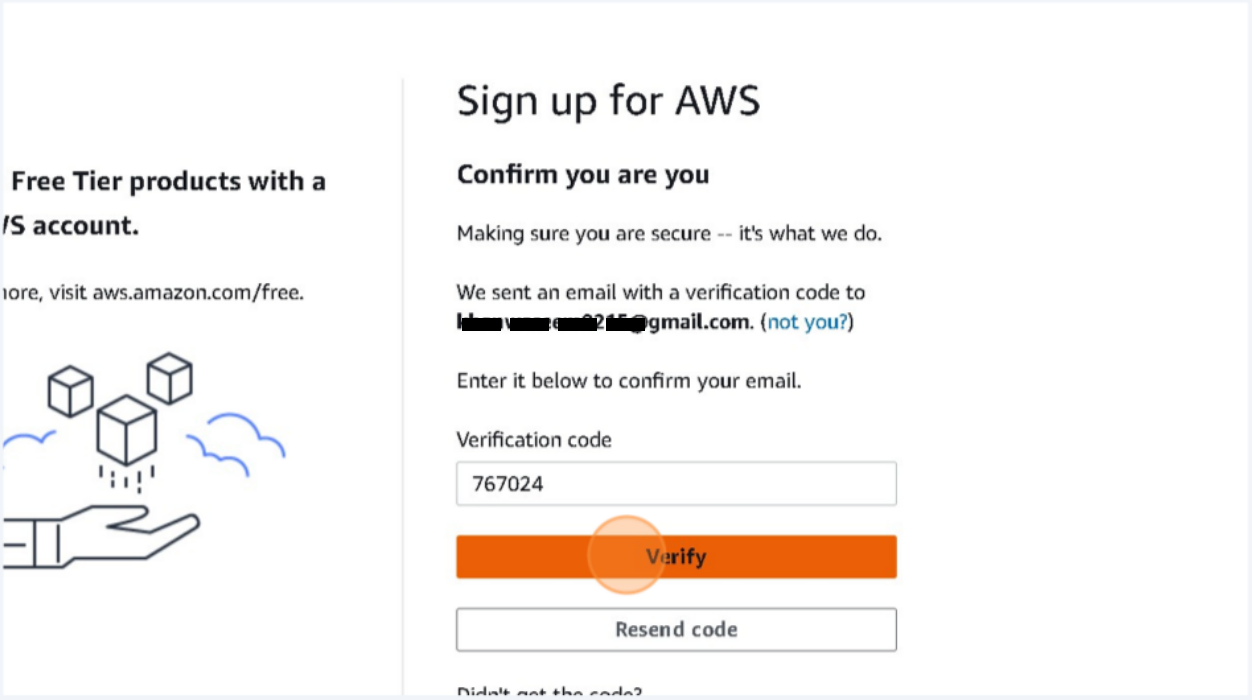
6. Click: "Create a free account"



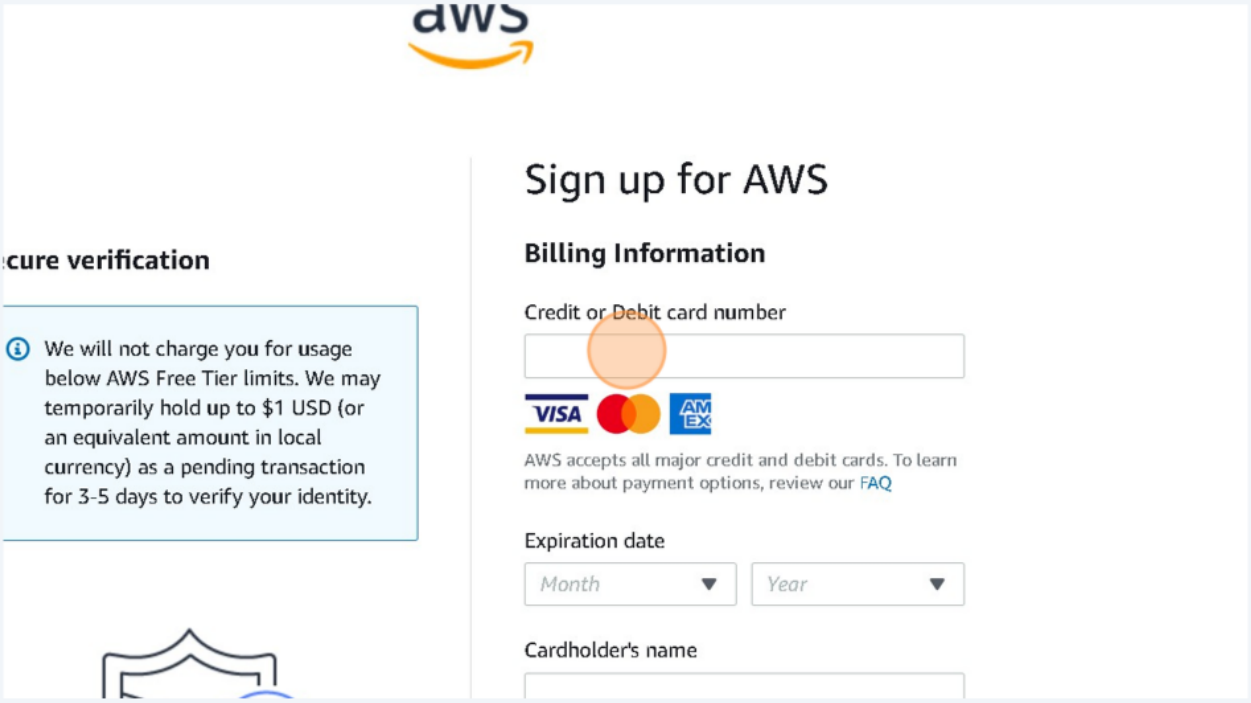
7. 1. Click the "Root user email address" field and enter your: email address.
   2. Click the "AWS account name" field and enter AWS account name.
   3. Click "Verify email address"



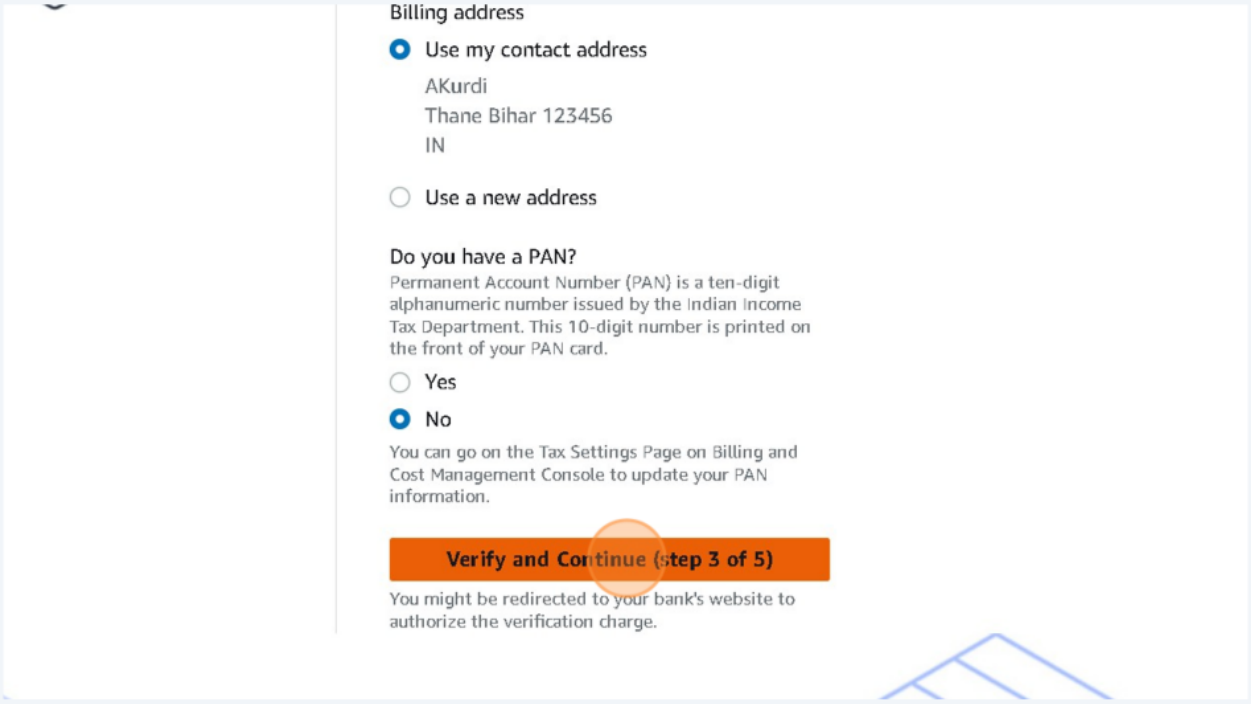
8. 1. Click the "Verification code" field.
   2. Enter the verification code sent in your email.
   3. Click "Verify"



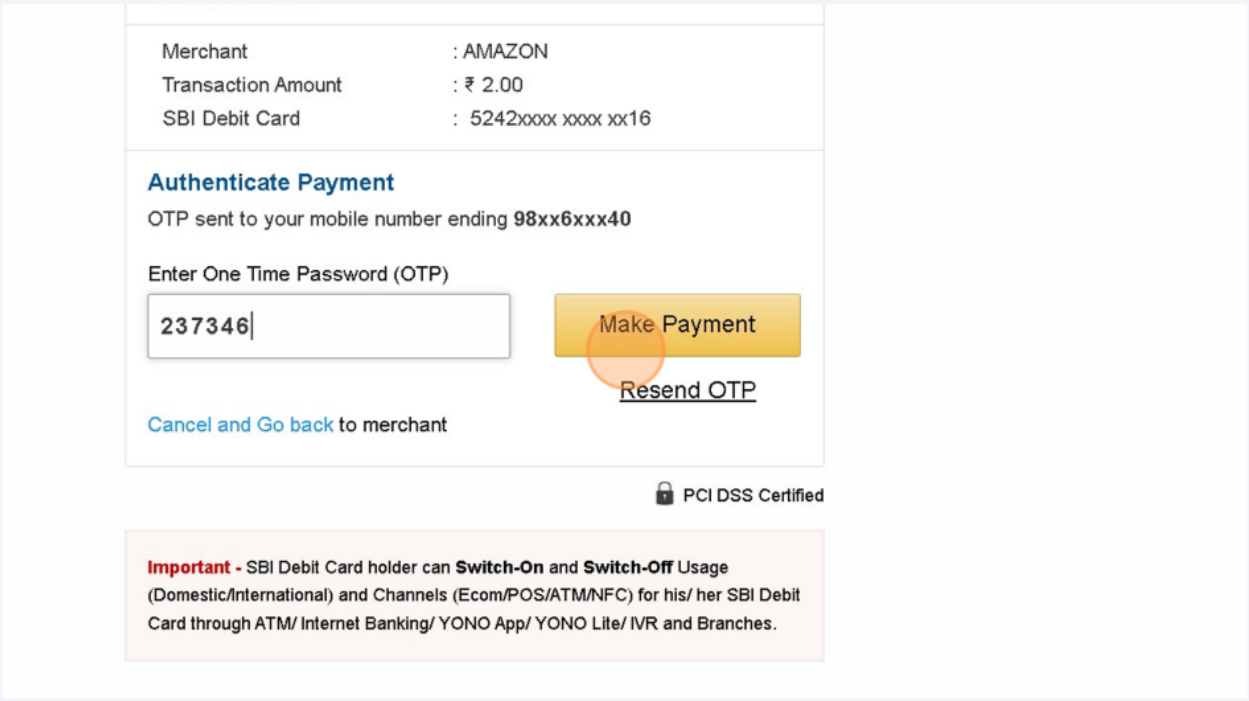
9. On this page: fill all the credentials.



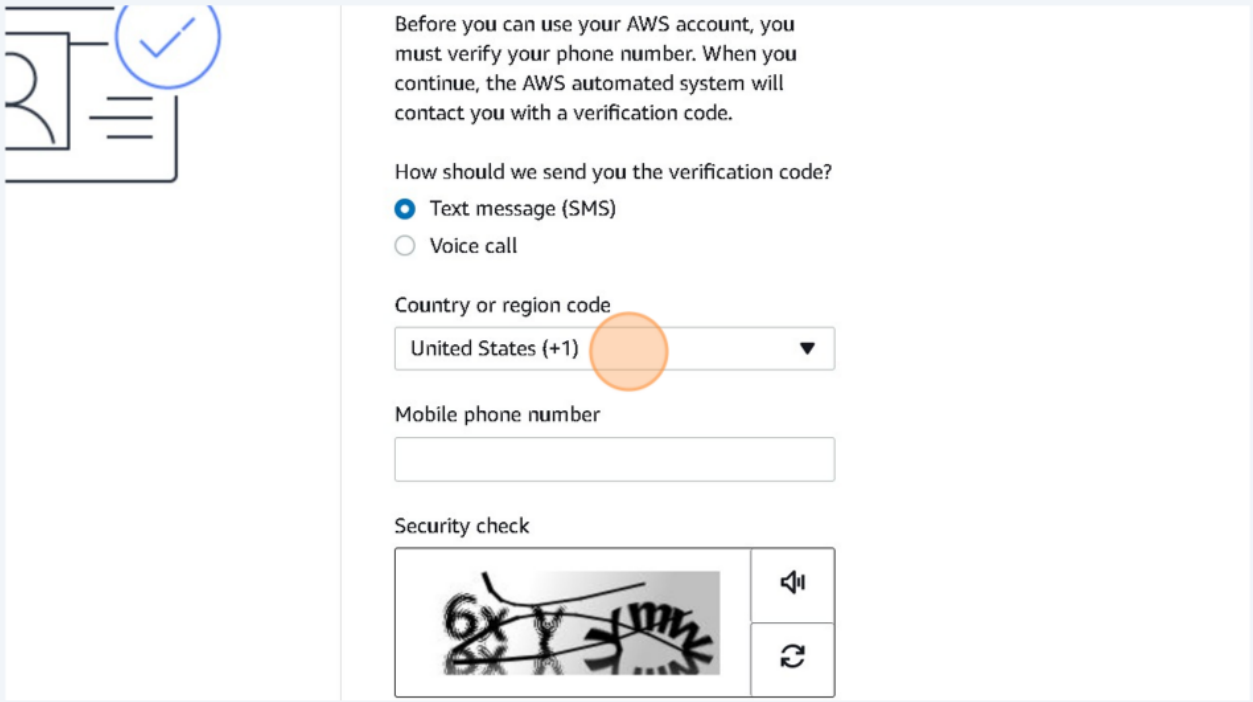
10. Click “Verify and continue (step 3/5)”



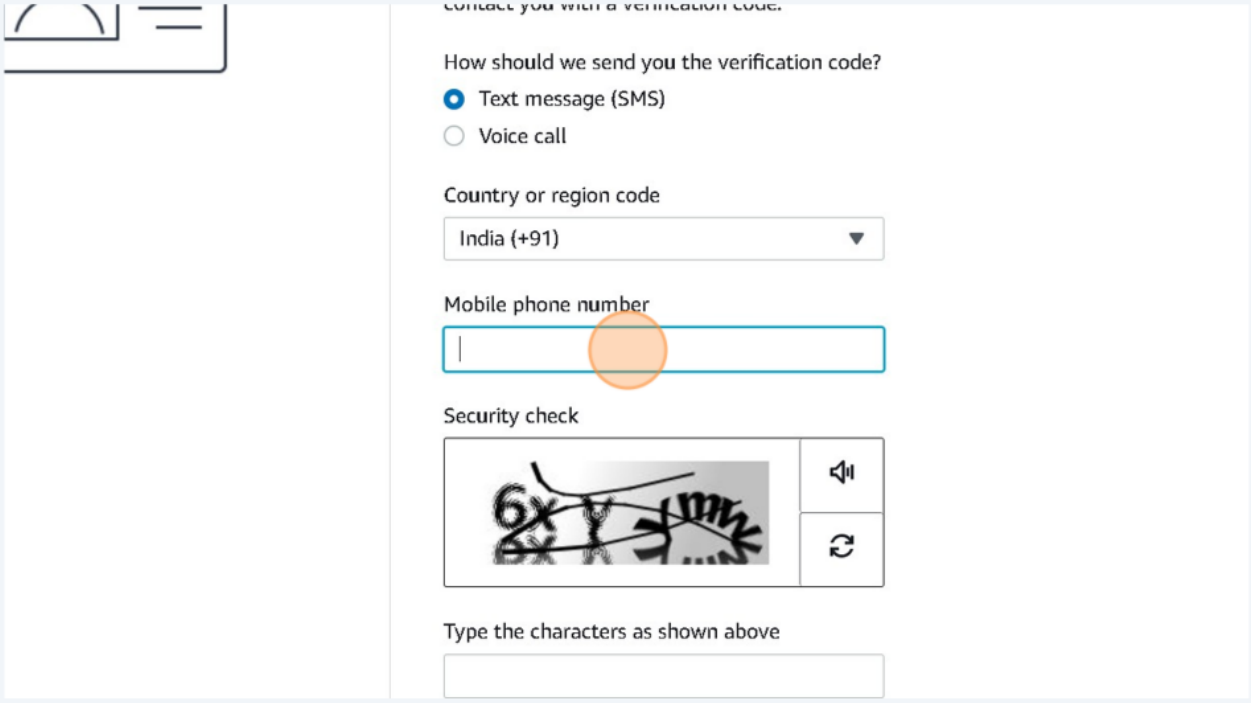
11. Fill OTP and Click “Make Payment”



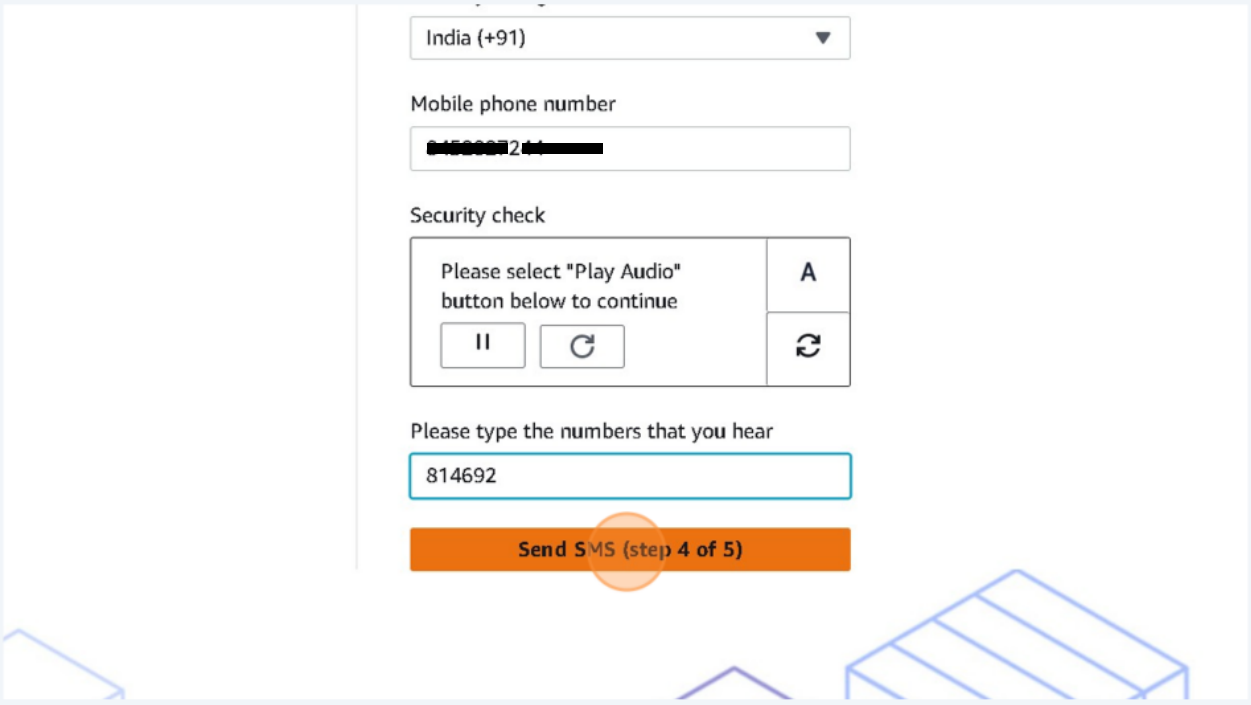
12. Change Country or region code to: India (+91)



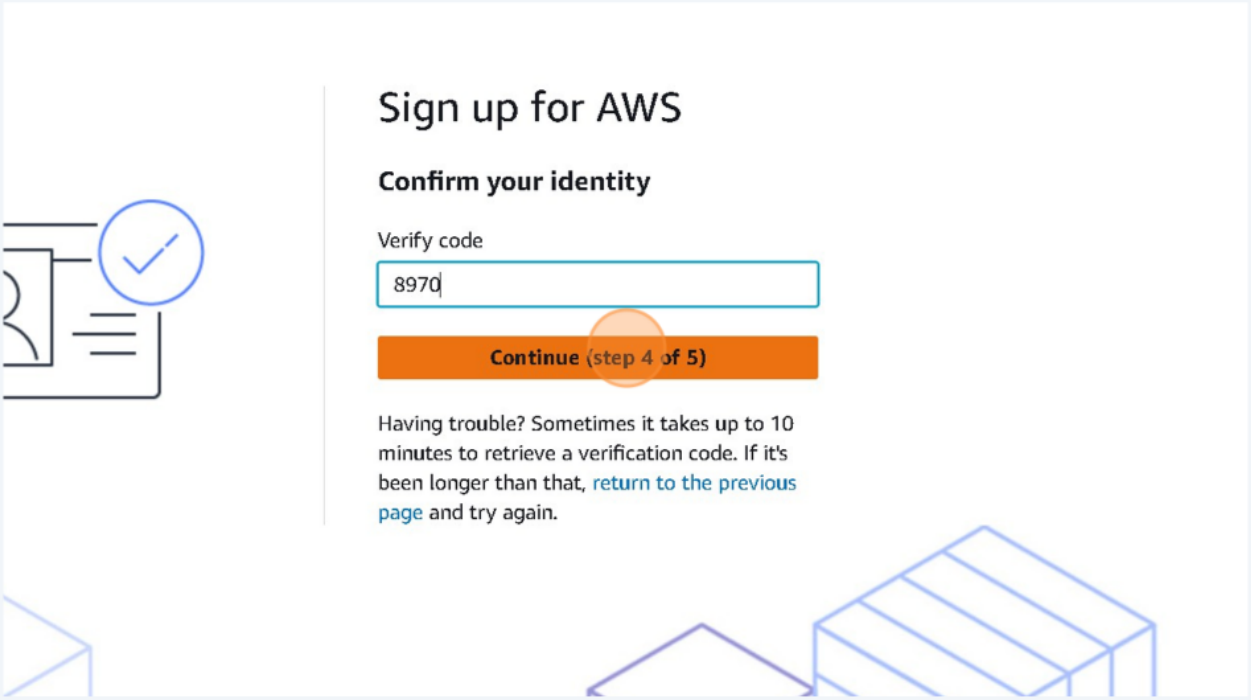
13. Provide the “”Mobile phone number”



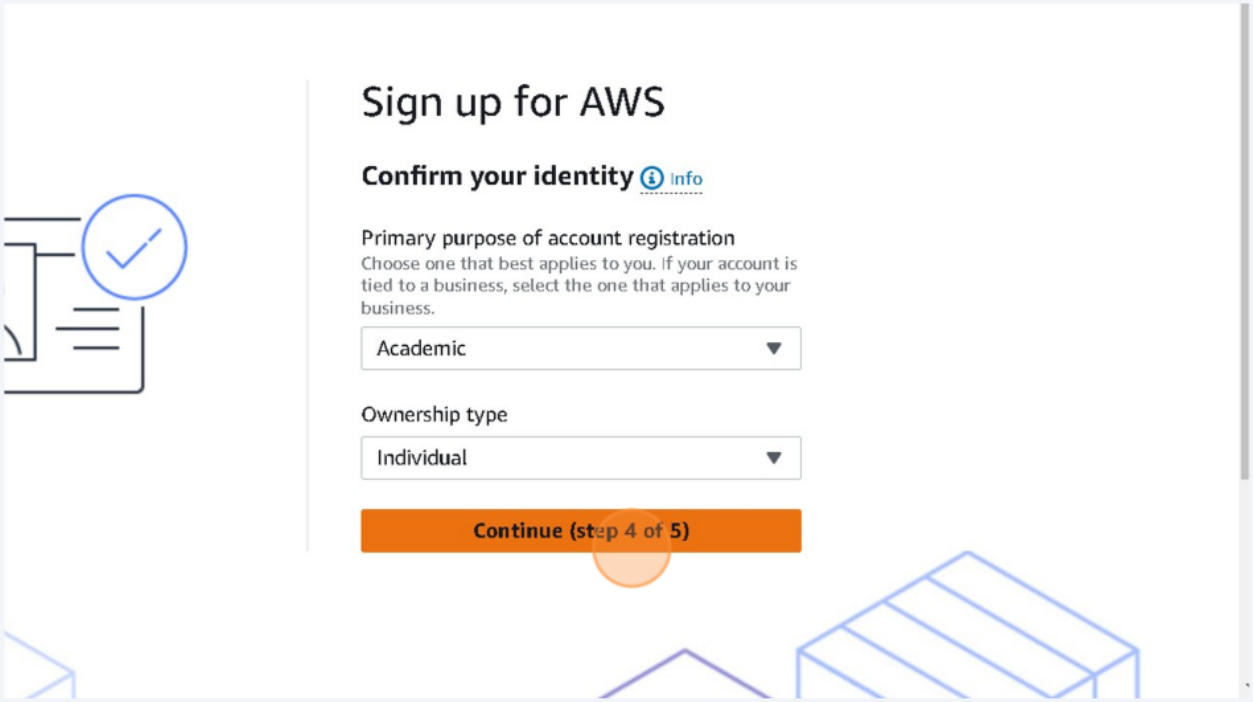
14. 1. Fill the CAPTCHA
2. Click “Send SMS (step 4/5)”



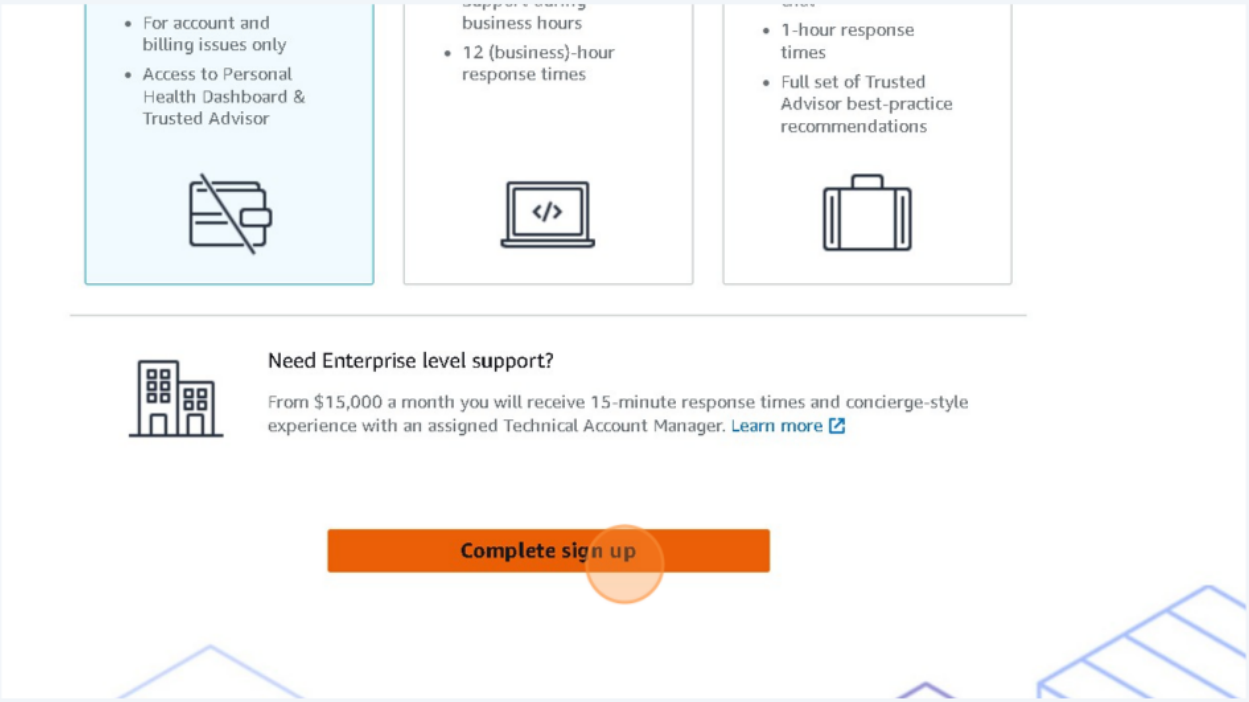
15. 1. Type Code send on your phone number.
2. Click “continue (step 4/5)”



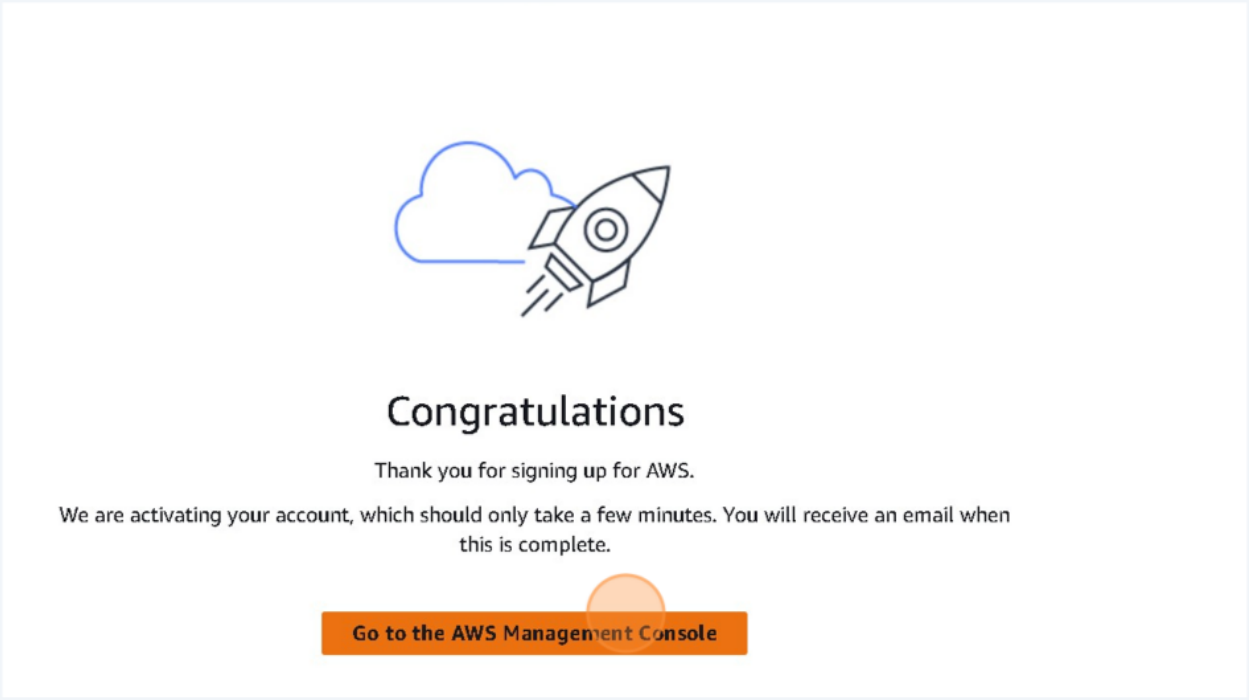
16. Select Academic in field 1. | Select Individual in field 2.
Click “Continue (step 4/5)”



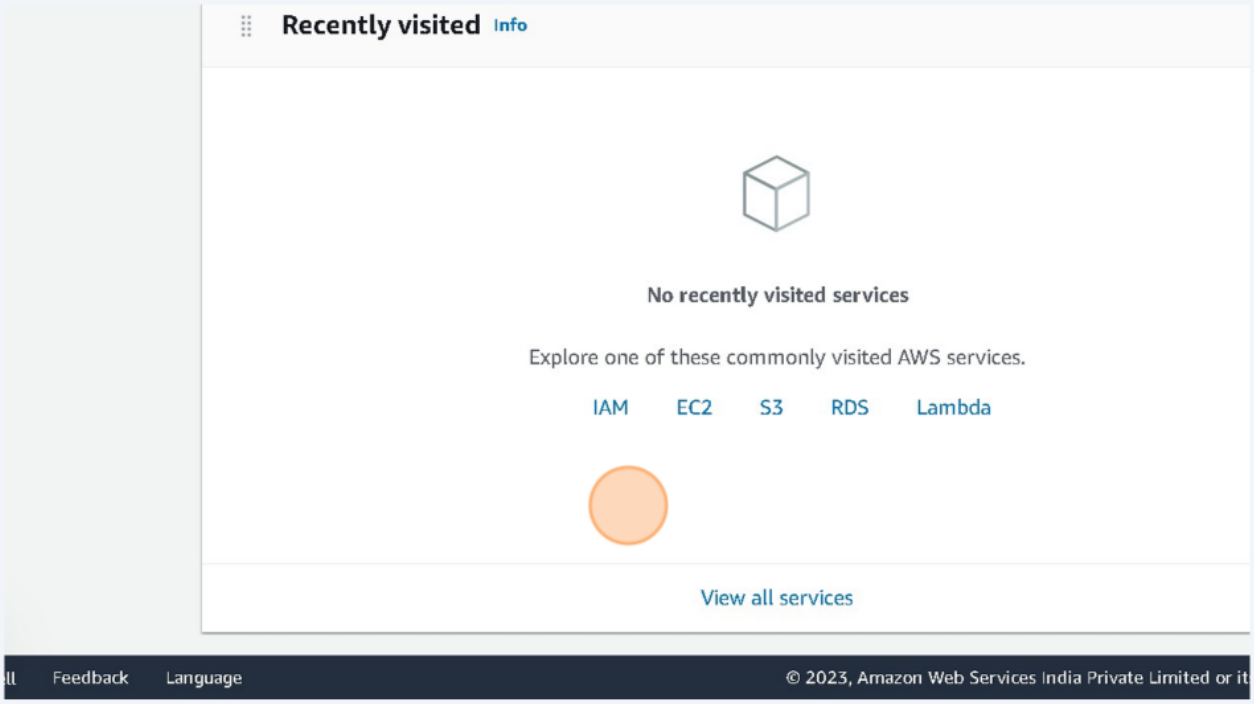
17. Click "Complete sign up"



18. Click "Go to the AWS Management Console"

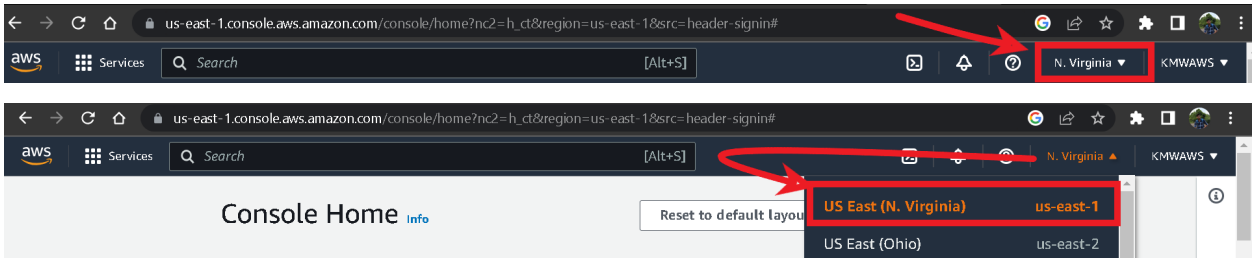


19. AWS account "Sign Up" done successfully.



—---------- AWS Account Setup End
—---------—----------

Change Location to : US East (N. Virginia).

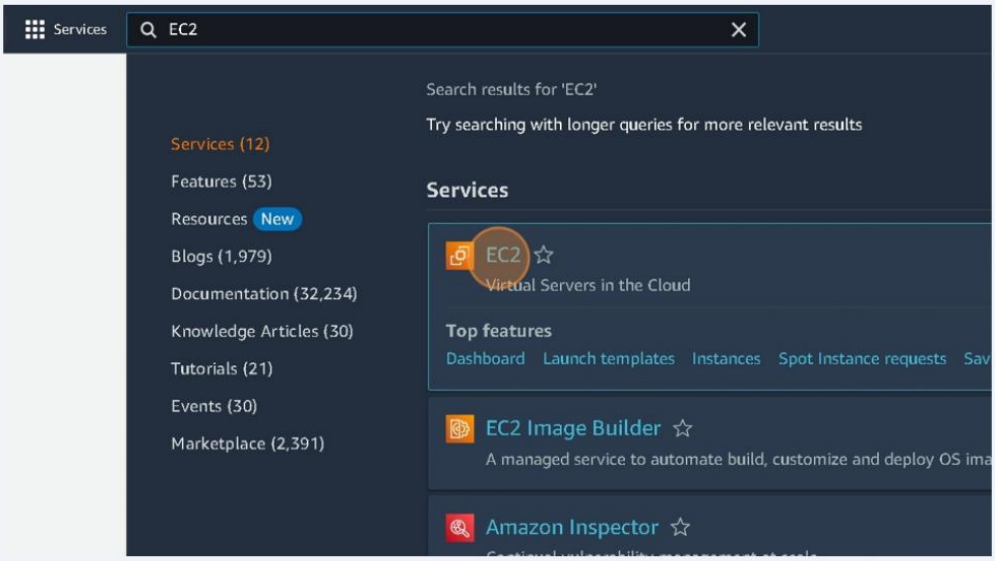


—---------- Location Change End
—---------—----------

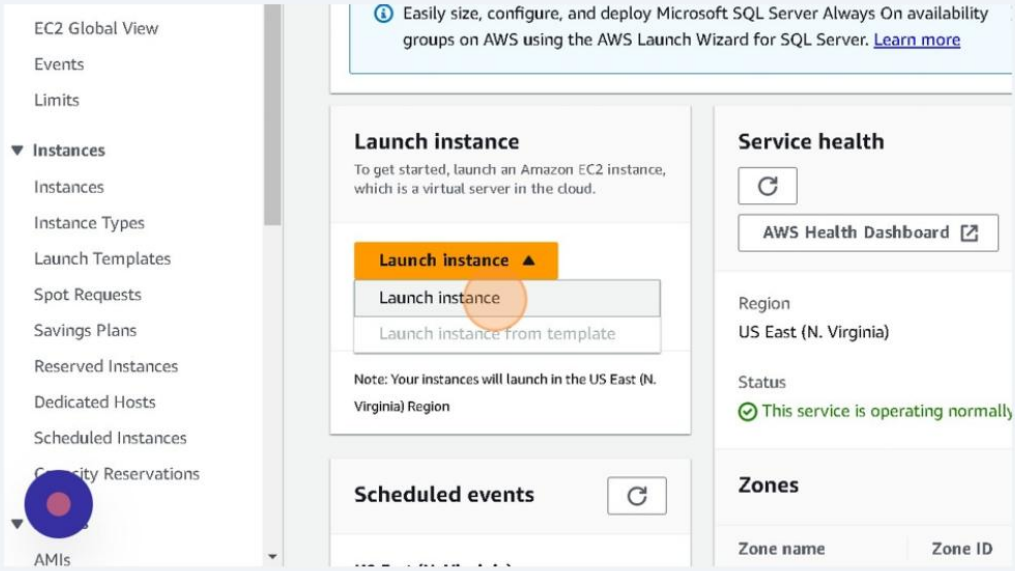
Setup EC2 Instance in AWS:

→ AWS → EC2

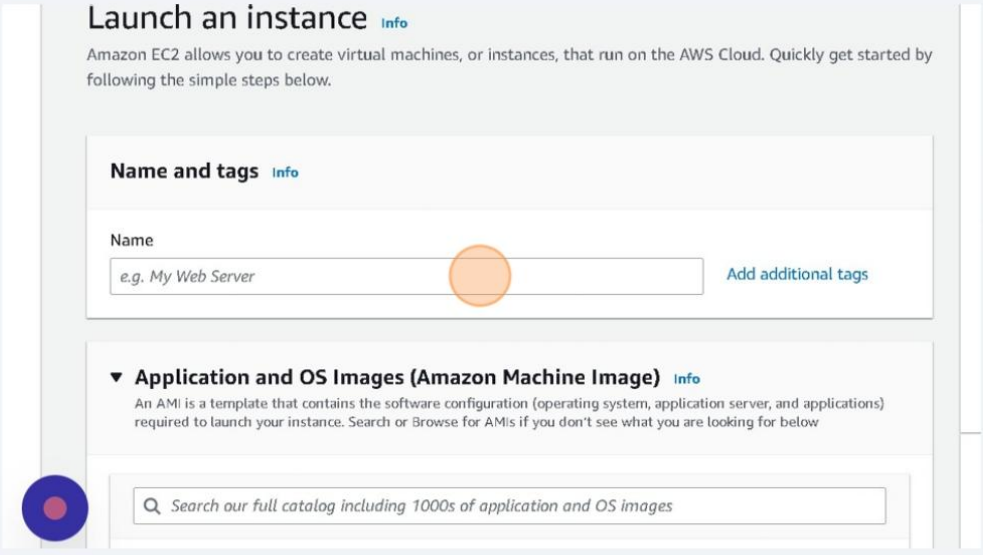
1 Click "EC2"



2 Click "Launch instance"

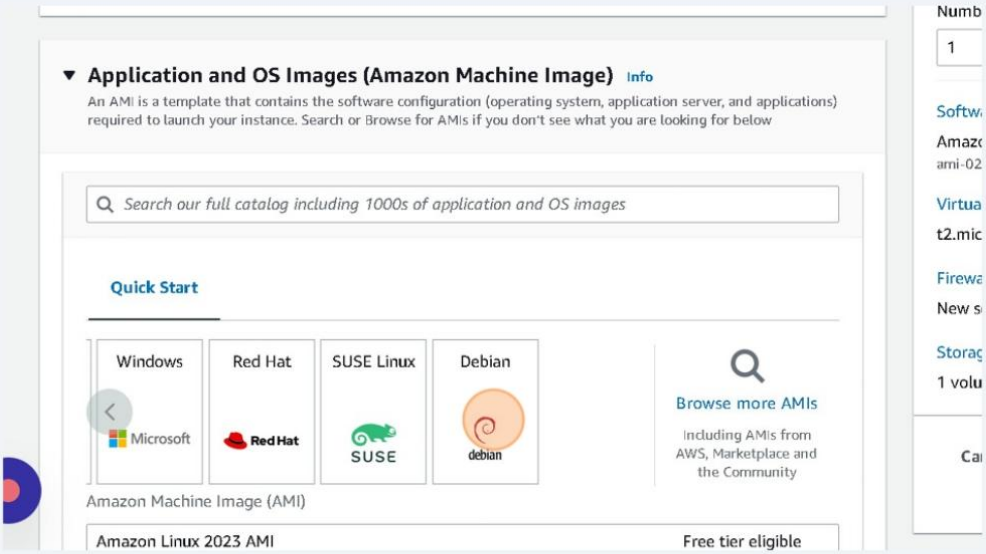


3 Click the "Name" field.

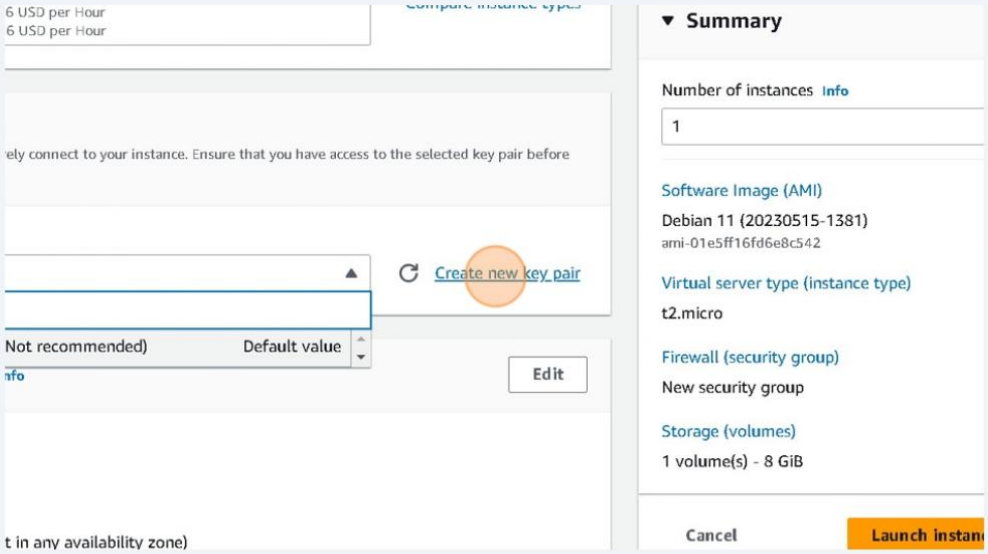


4 Type "Jenkins"

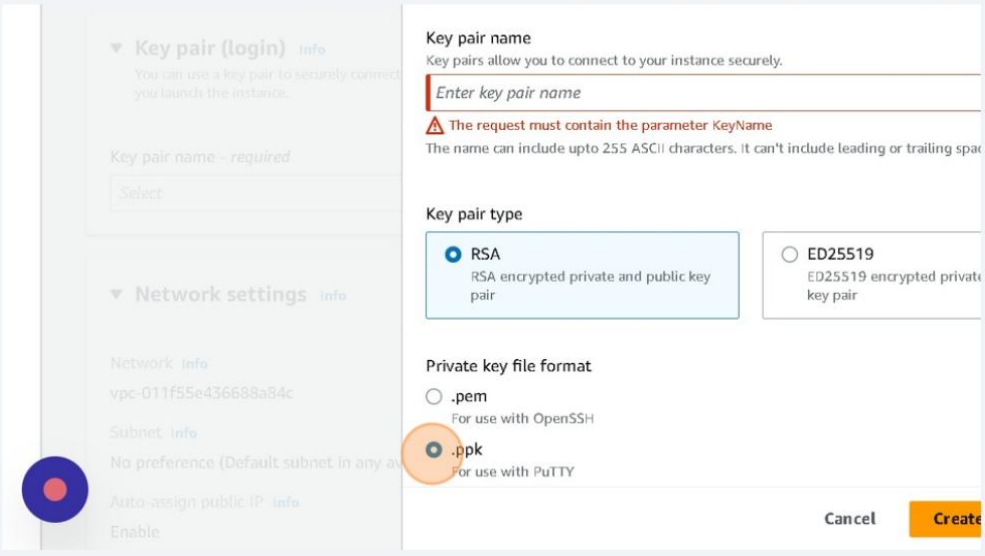
5 Click here.



6 Click "Create new key pair"

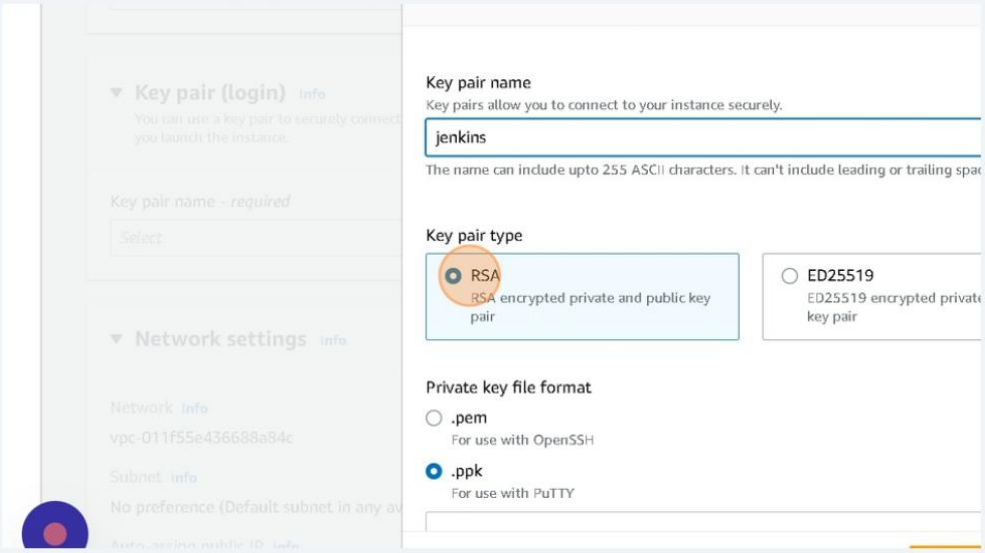


7 Give: "Key pair name"

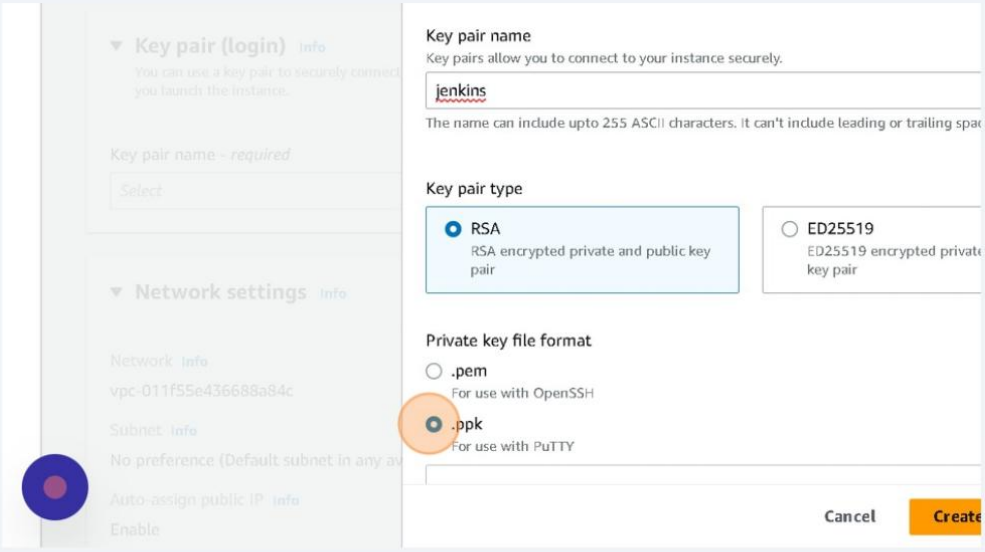


8 Type "jenkins"

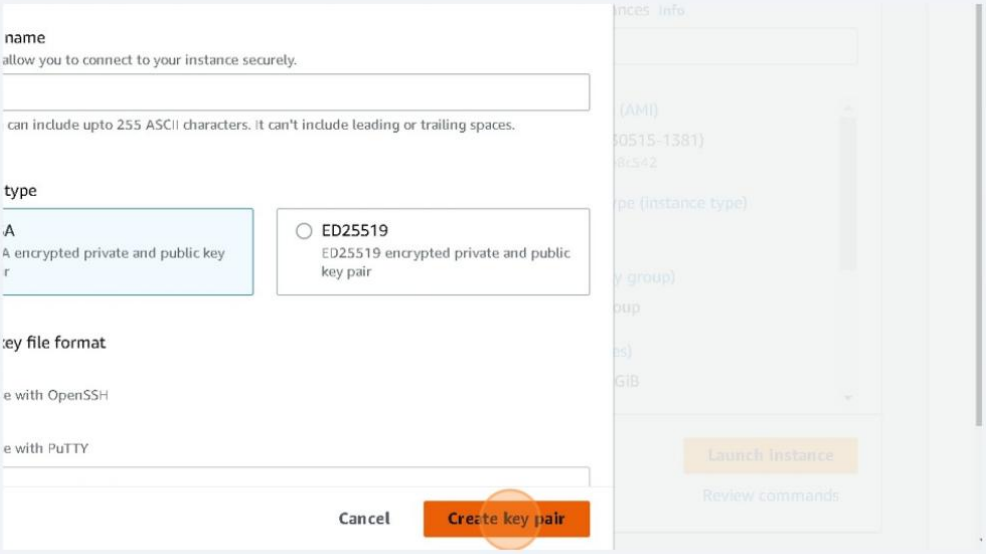
9 Click "RSA"



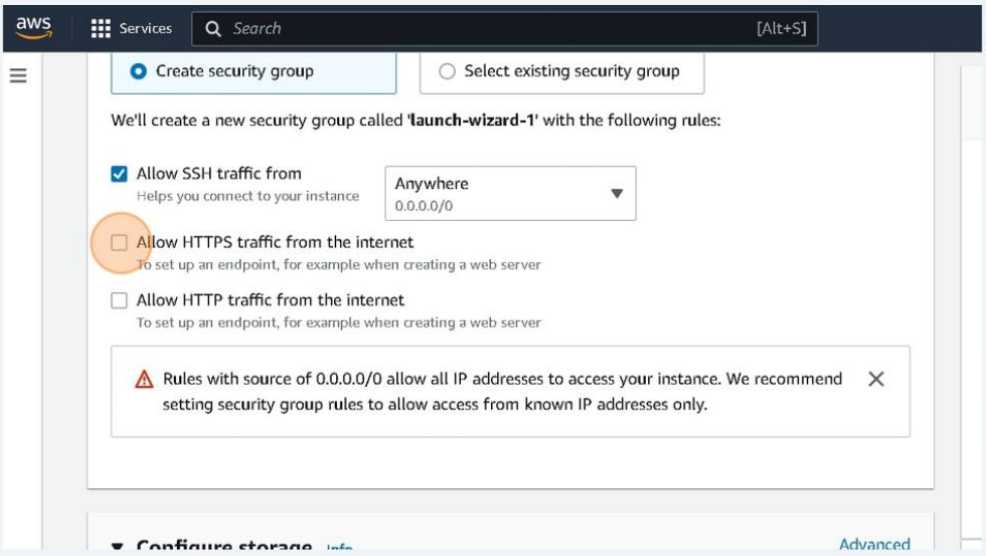
10 Click " .ppk ".



11 Click "Create key pair"

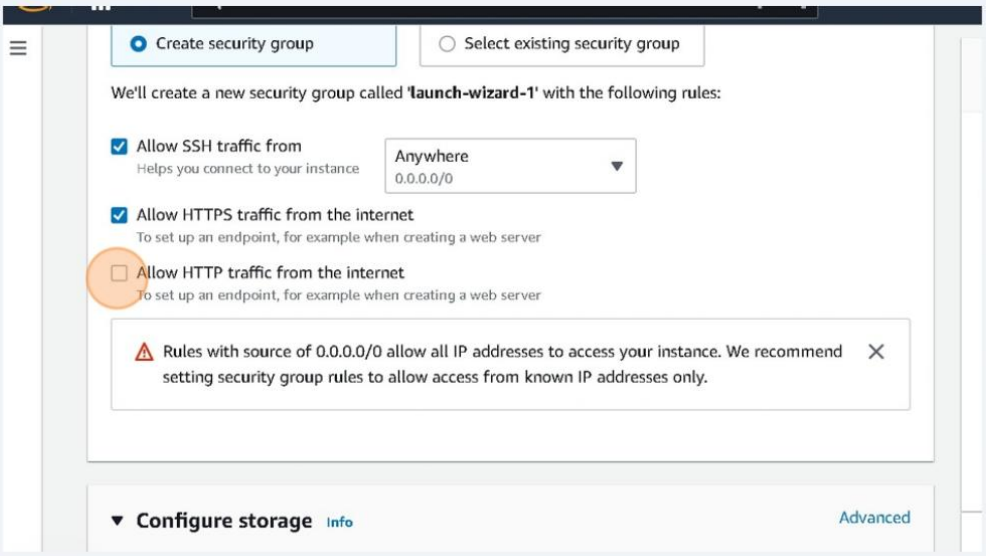


12 Under Networking.

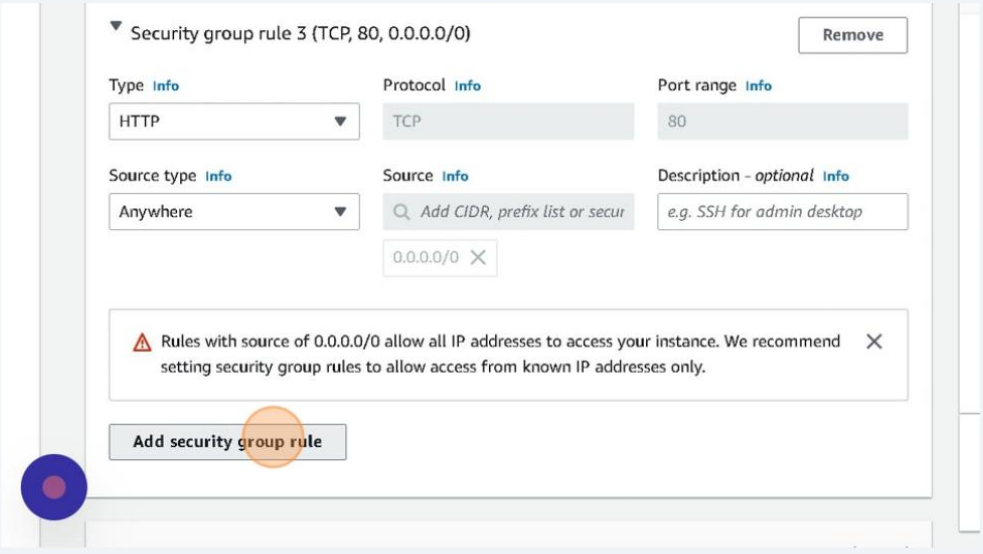


13

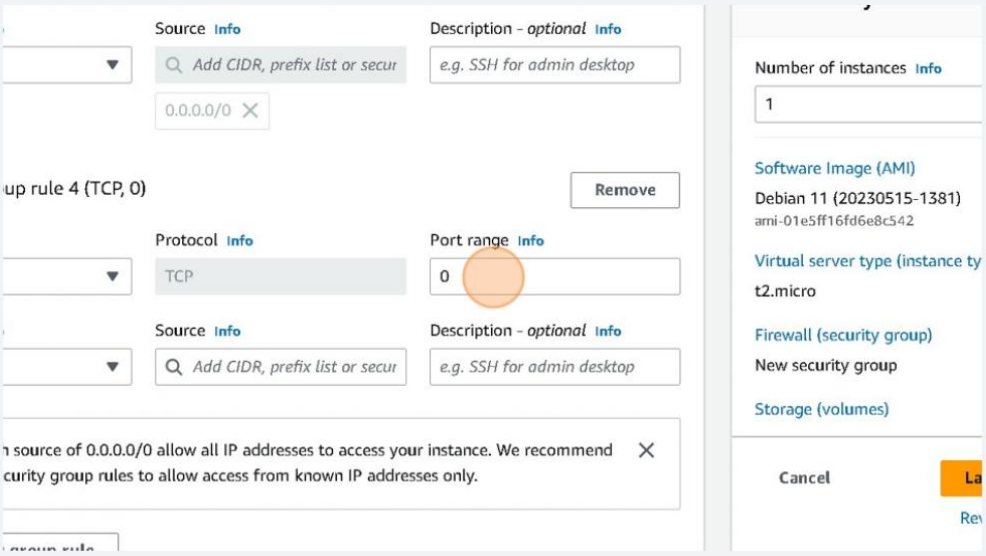
1. Click "Allow HTTP traffic from the internet"
2. Click "Allow HTTPS traffic from the internet"



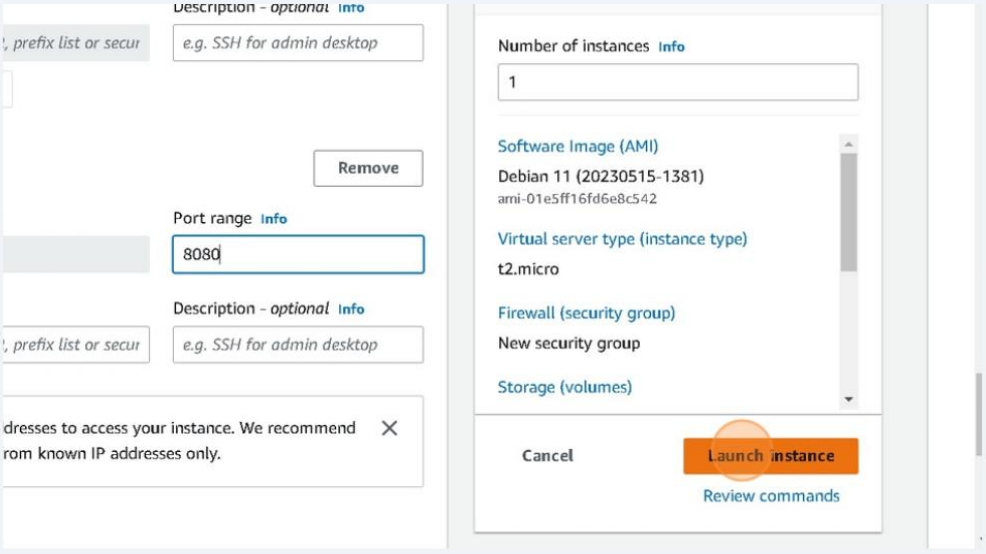
14 Click "Add security group rule"



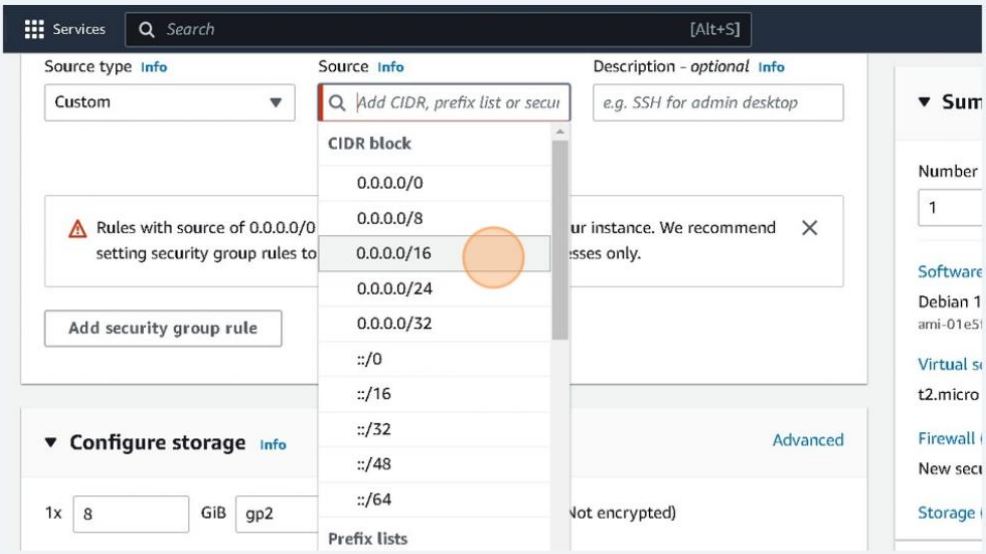
15 Click the "Port range Info" field.



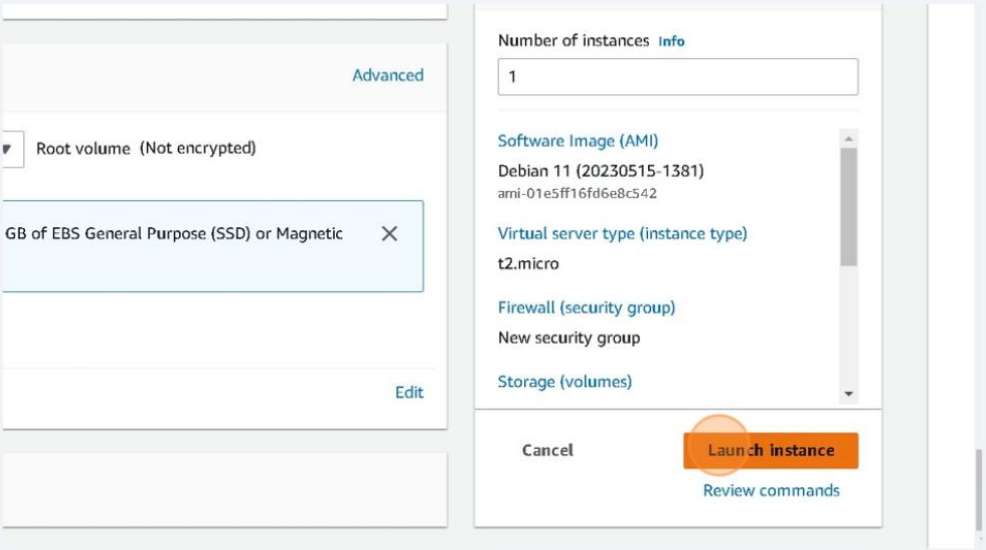
16 Type "8080"



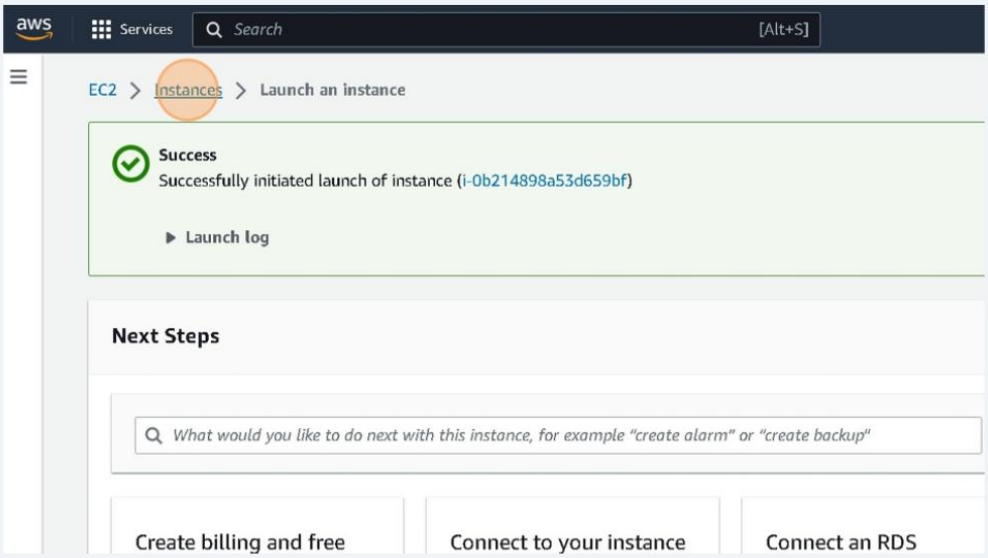
17 Click "0.0.0.0/0"



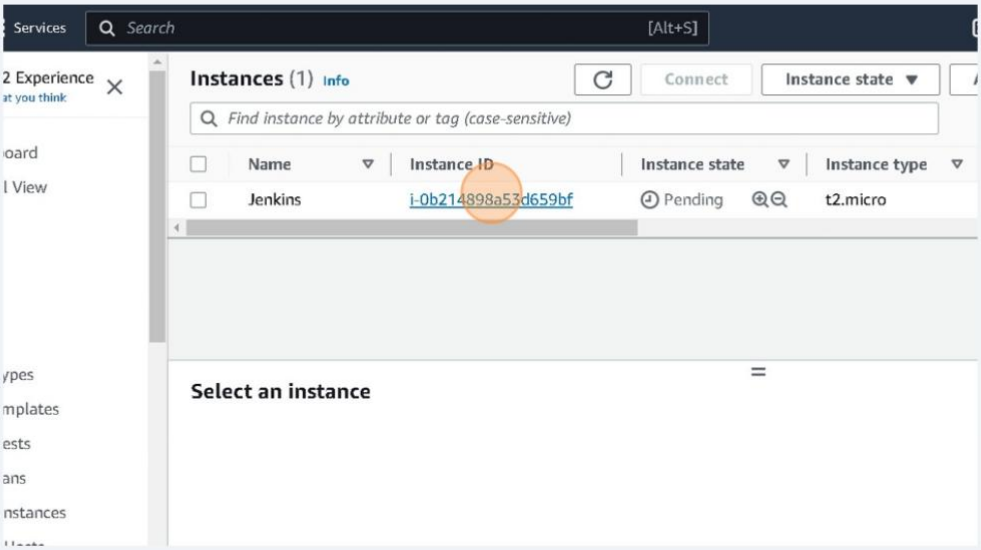
18 Click "Launch instance"



19 Click "Instances"



20 Click "i-0b214898a53d659bf"



21 EC2 Instance Setup Successful

→ Instances → Your Instance → Copy Public IP
→ Open PuTTY → Paste IP → Connections → SSH → Auth
→ Credentials → Private Key → Save Settings
→ Login → Username: `admin` [Login Succesful]

```
→ apt get upgrade
→ apt get update
→ apt get-install apache2
→ sudo systemctl restart apache2
→ sudo systemctl status apache2
```

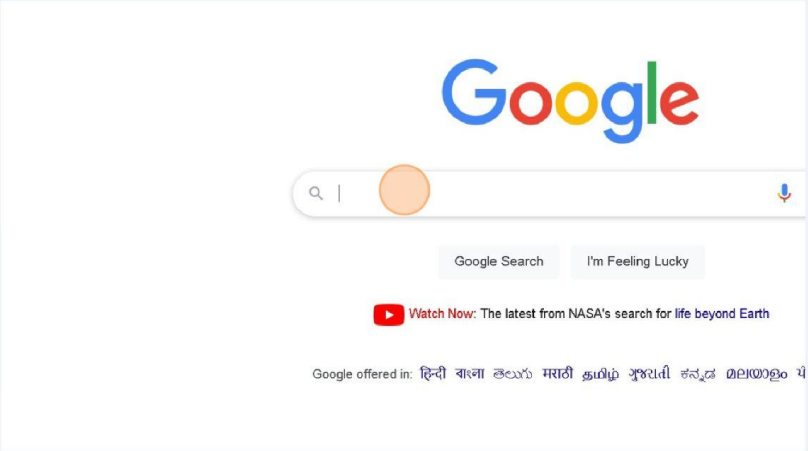
→ **Verify On Browser Using the IP Address**
→ Now Again Go to : AWS → Instance → Instance State: Stop Instance

—---------- AWS EC2 End
—---------—----------

How to Create an S3 Bucket in AWS:

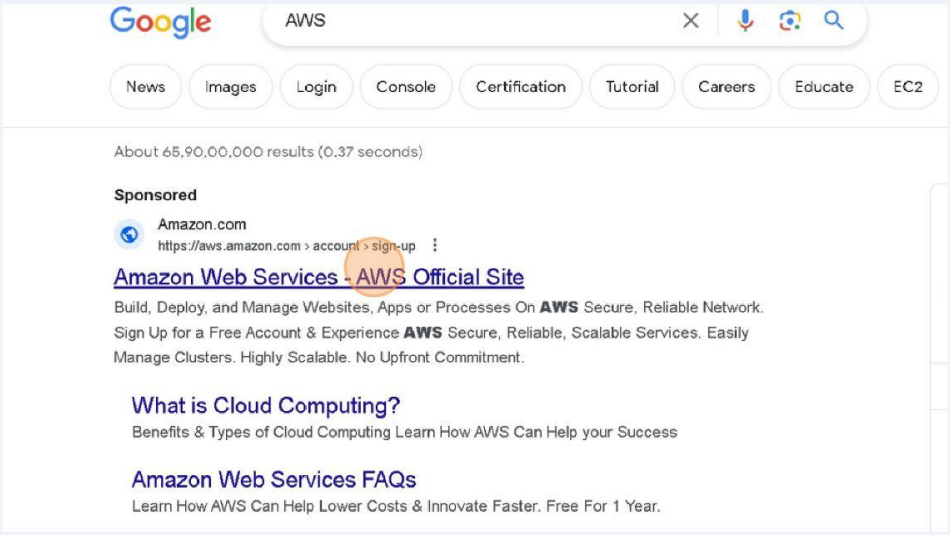
1 Navigate to www.google.com

2 Click this search field.

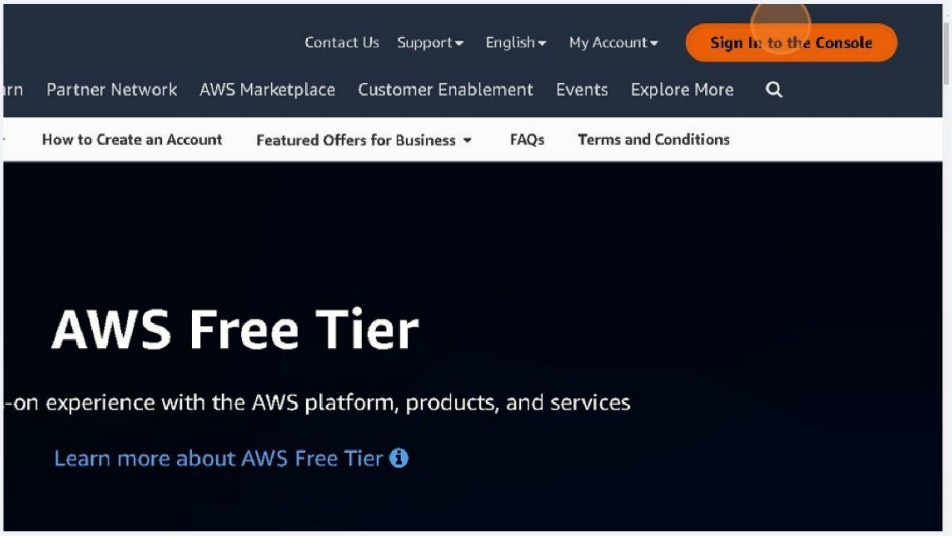


3 Type "AWS **ENTER**"

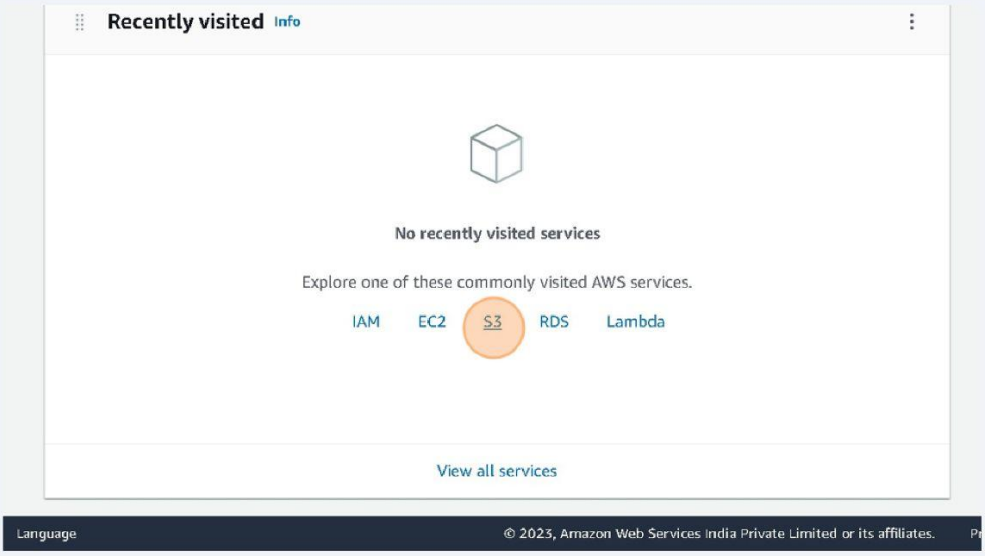
4 Click "Amazon Web Services - AWS Official Site"



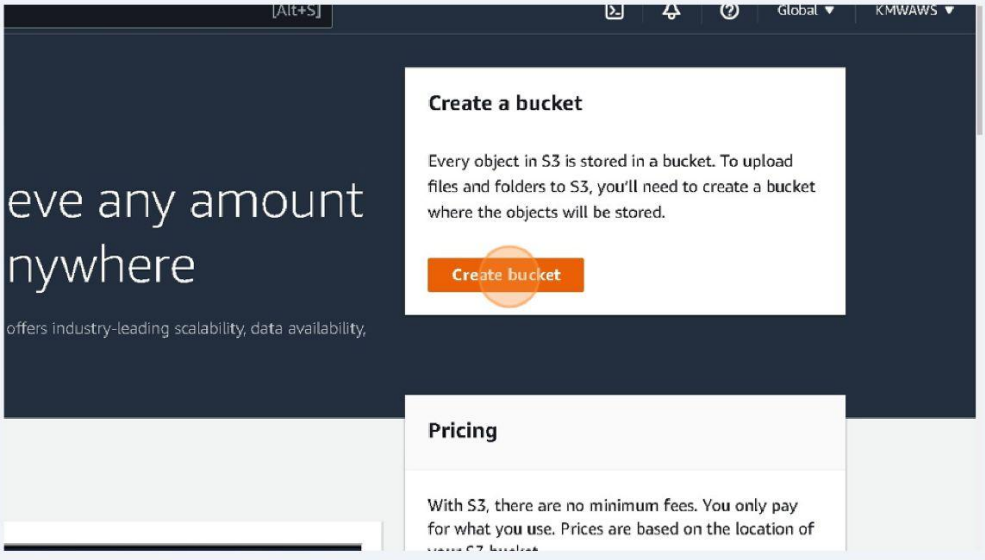
5 Click "Sign In to the Console"



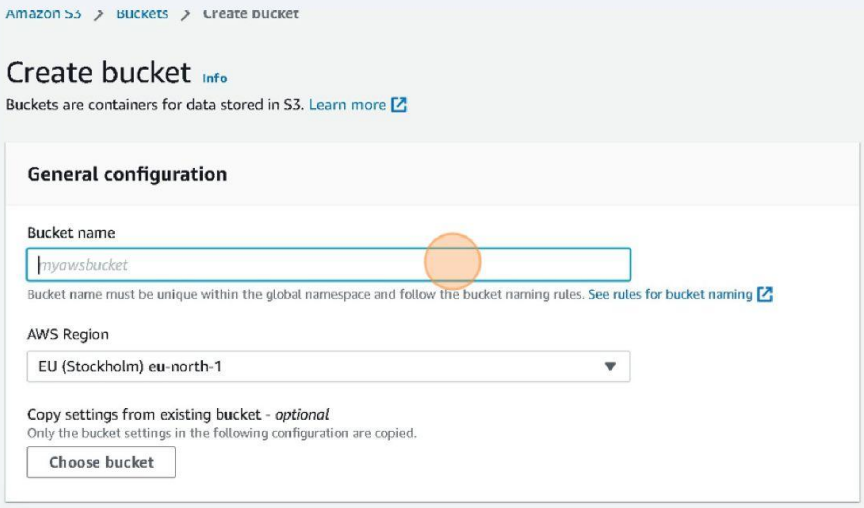
6 Click "S3"



7 Click "Create bucket"

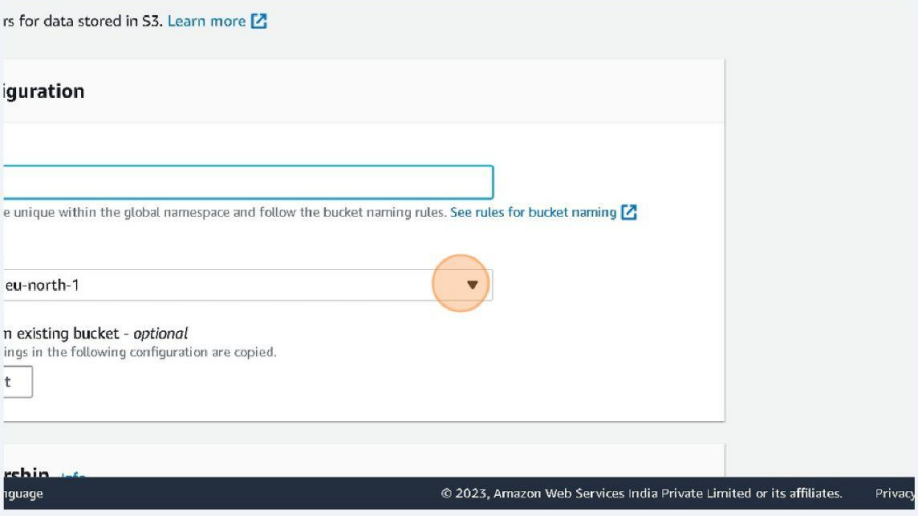


8 Click the "Bucket name" field.

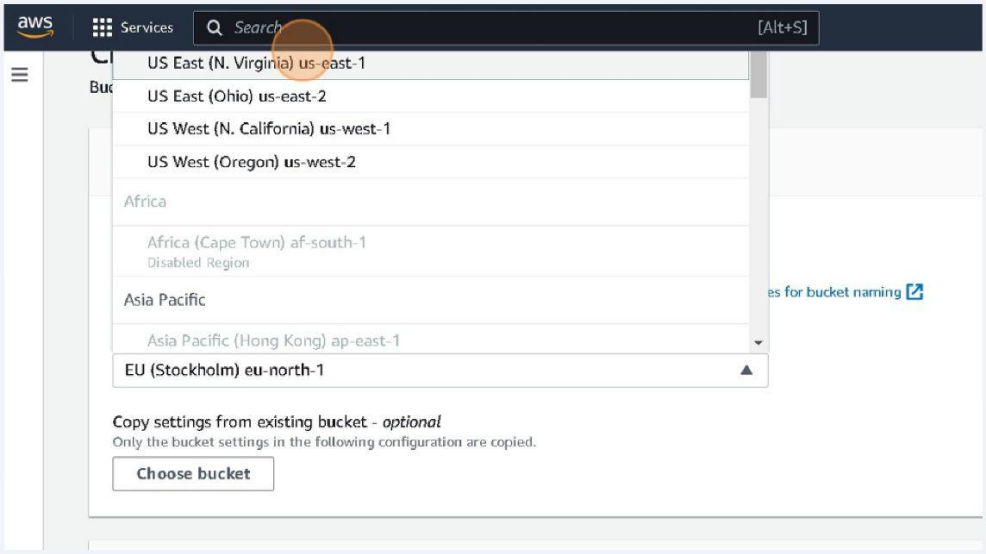


9 Type "my_bucket"

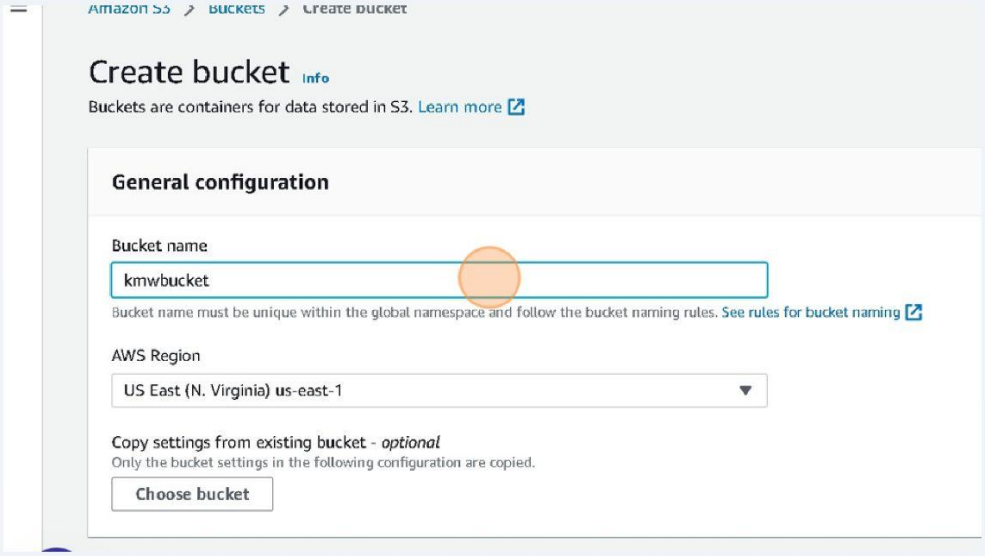
10 Click "EU (Stockholm) eu-north-1"



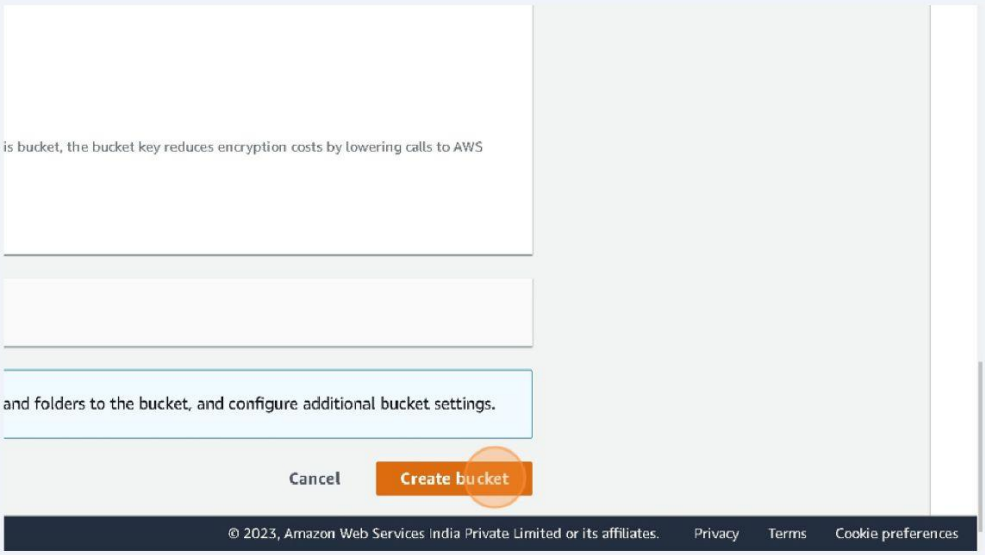
11 Click "US East (N. Virginia) us-east-1"



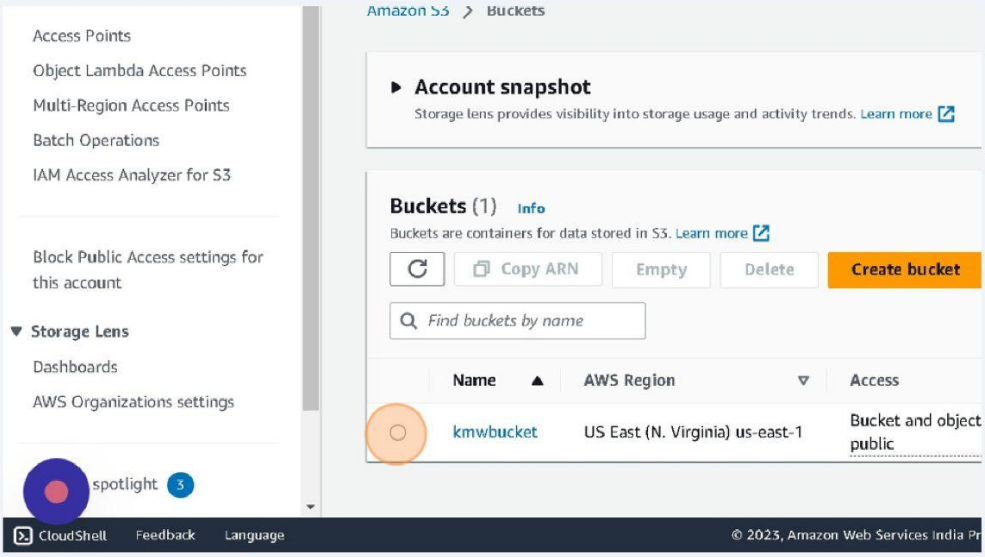
12 Click the "Bucket name" field.



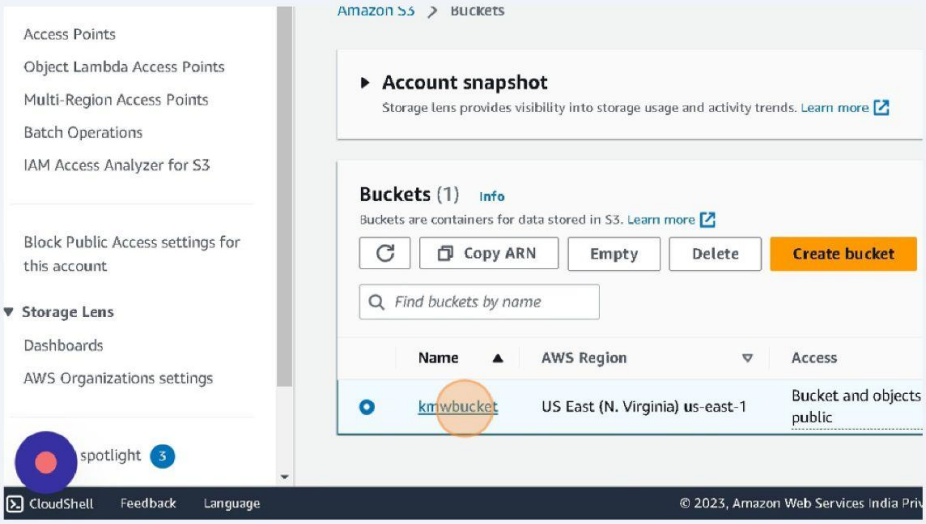
13 Click "Create bucket"



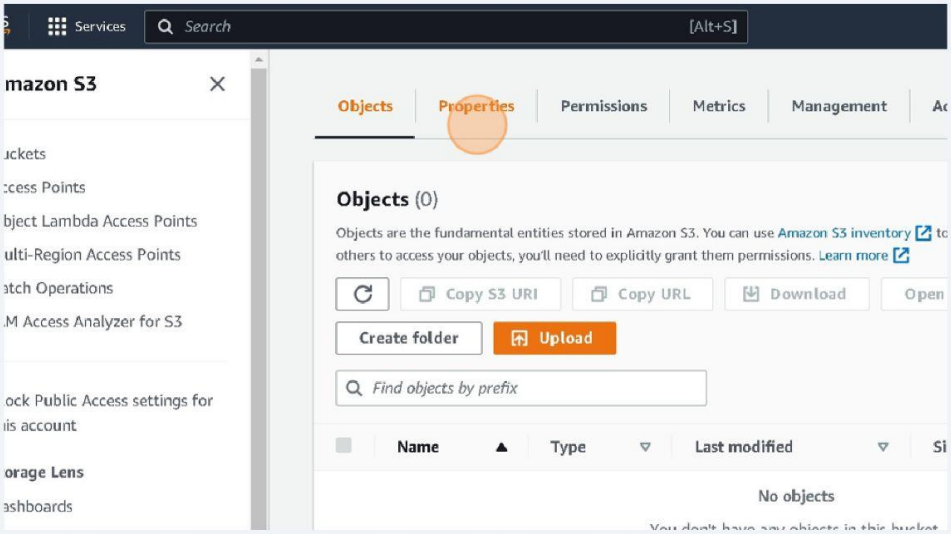
14 Click this radio button.



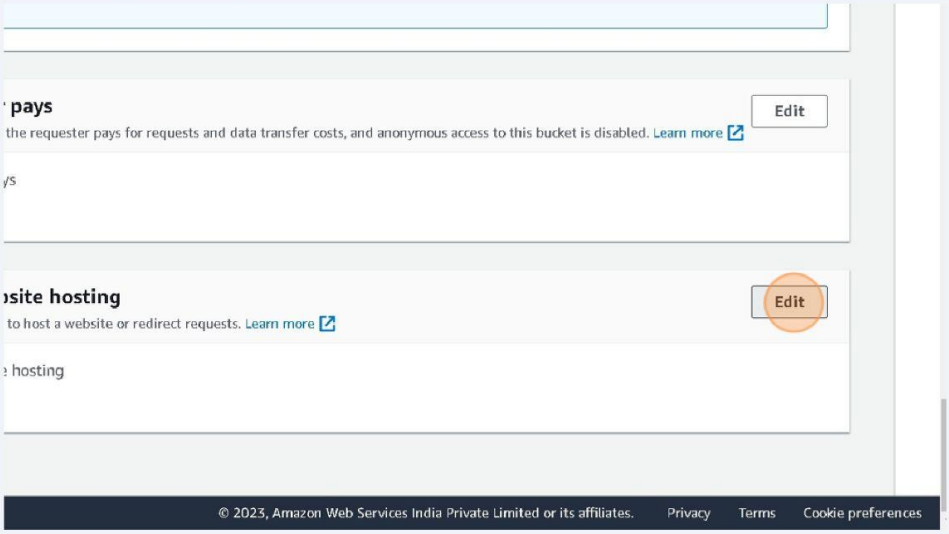
15 Click "kmwbucket"



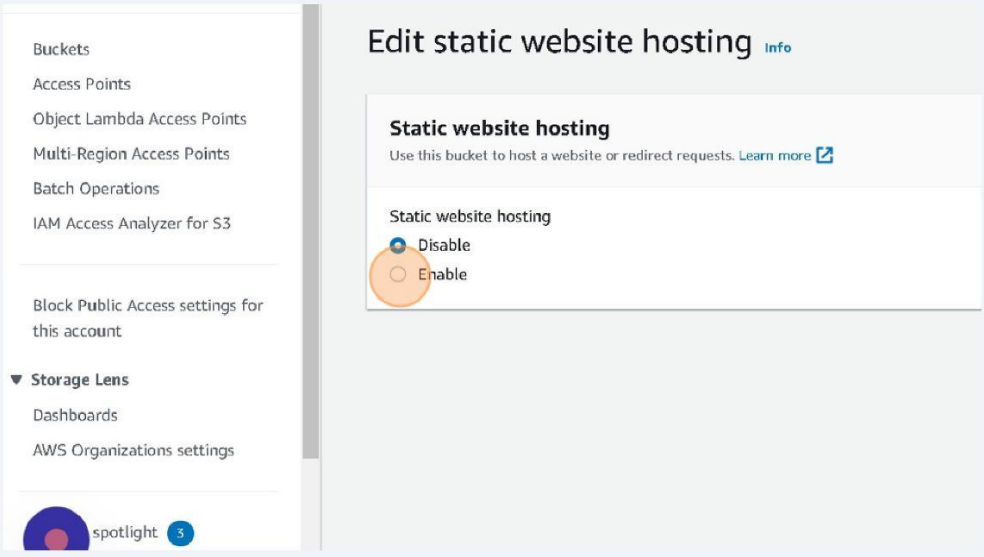
16 Click "Properties"



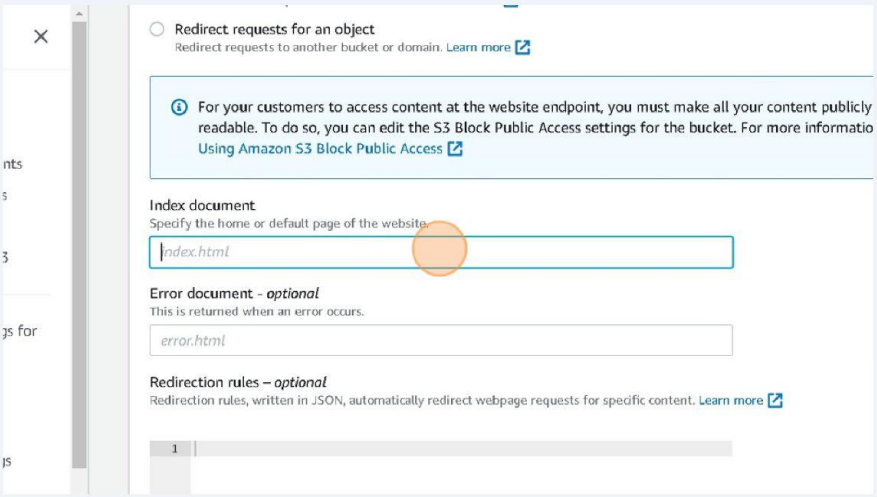
17 Click "Edit"



18 Click the "Enable" field.

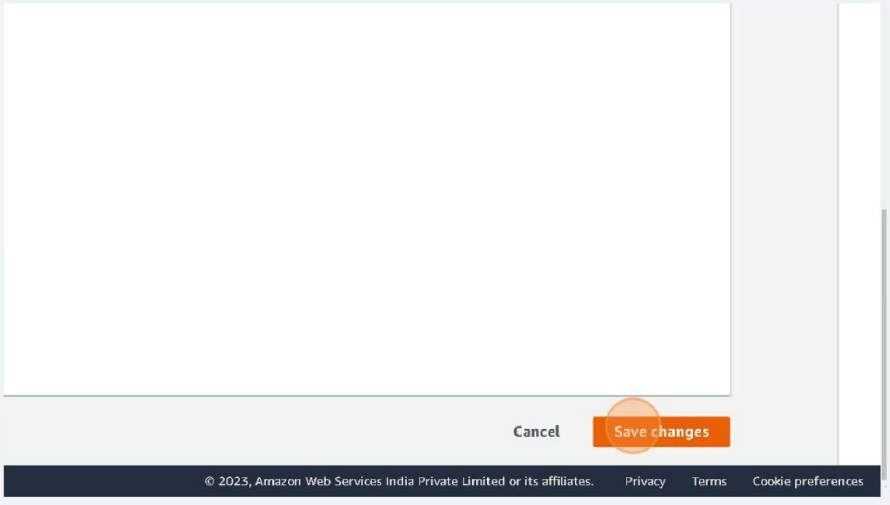


19 Click the "Index document" field.

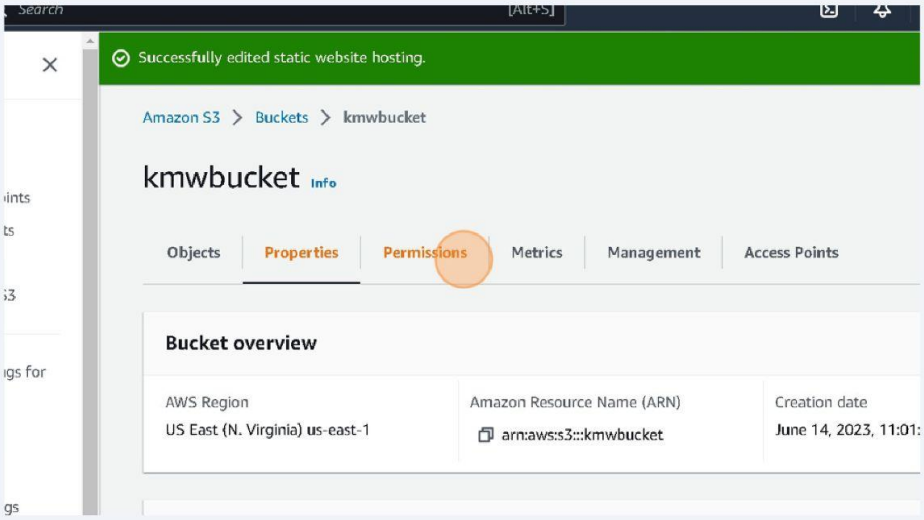


20 Type "index.html **TAB**"

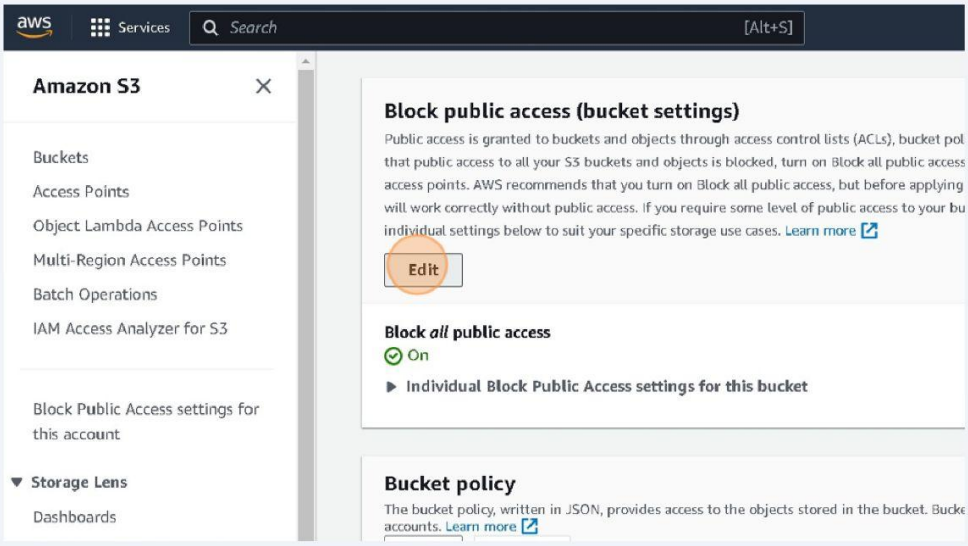
21 Click "Save changes"



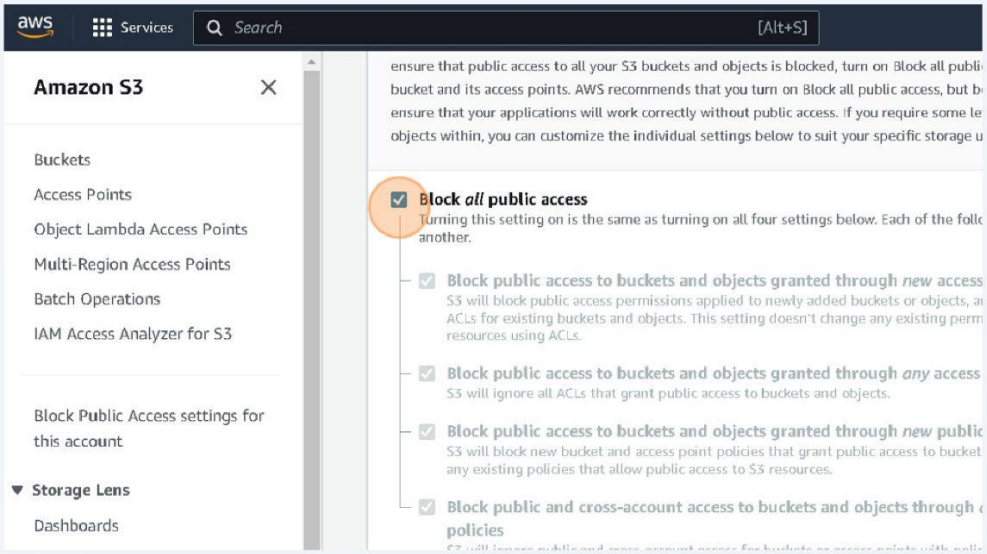
22 Click "Permissions"



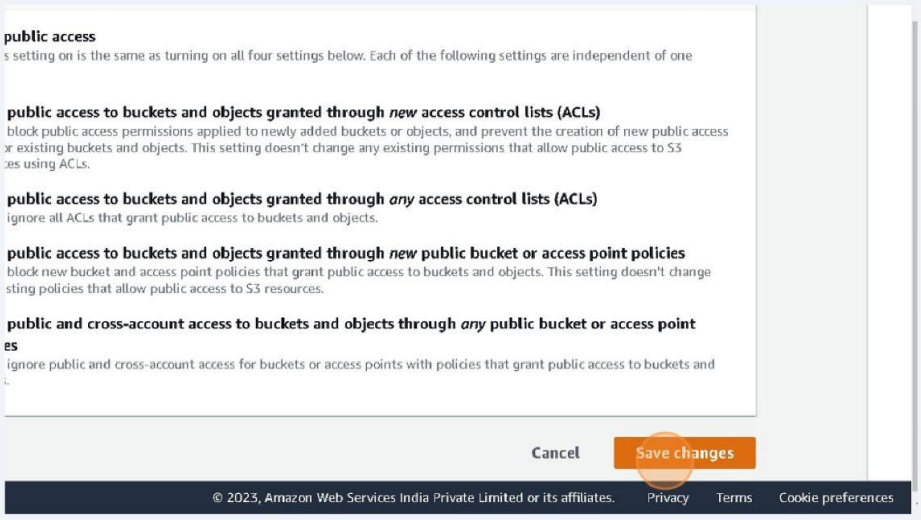
23 Click "Edit"



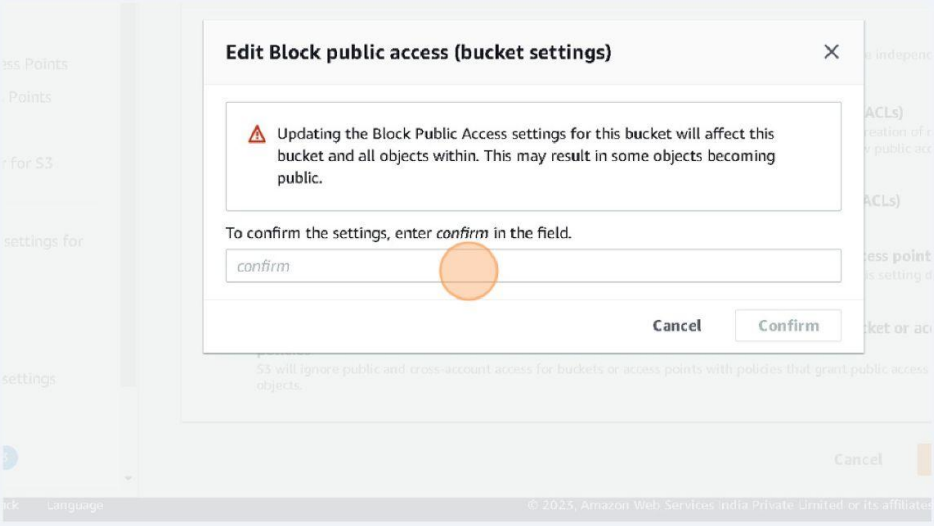
24 Click "Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independen..."



25 Click "Save changes"

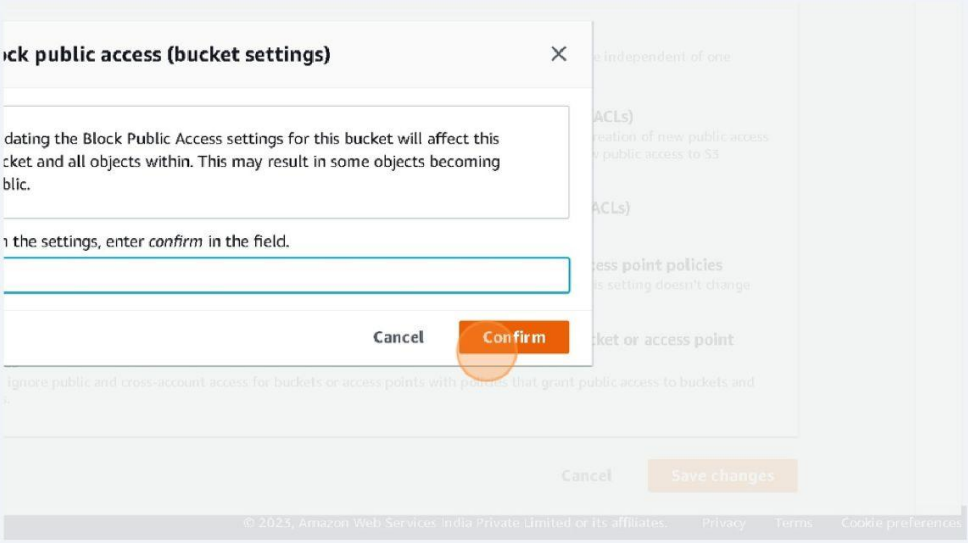


26 Click the "To confirm the settings, enter confirm in the field." field.

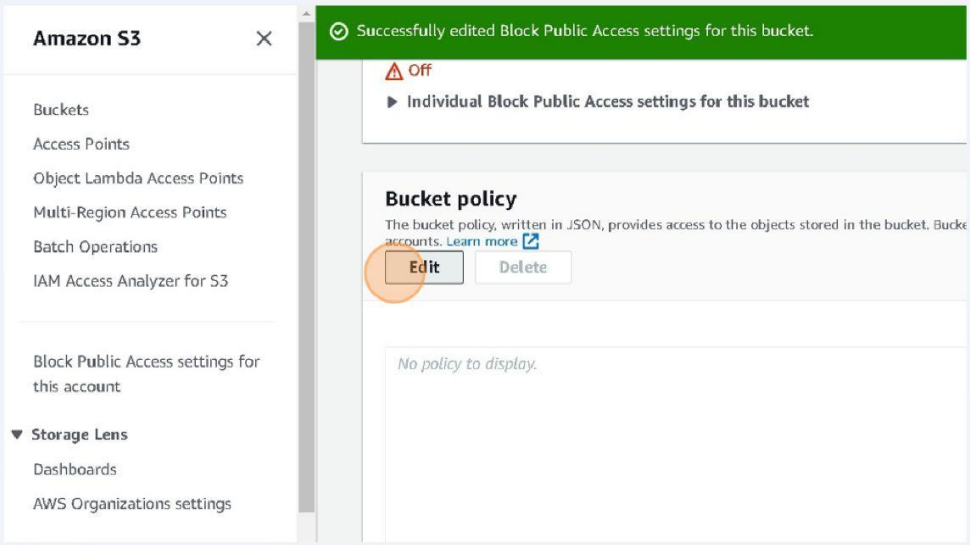


27 Type "confirm"

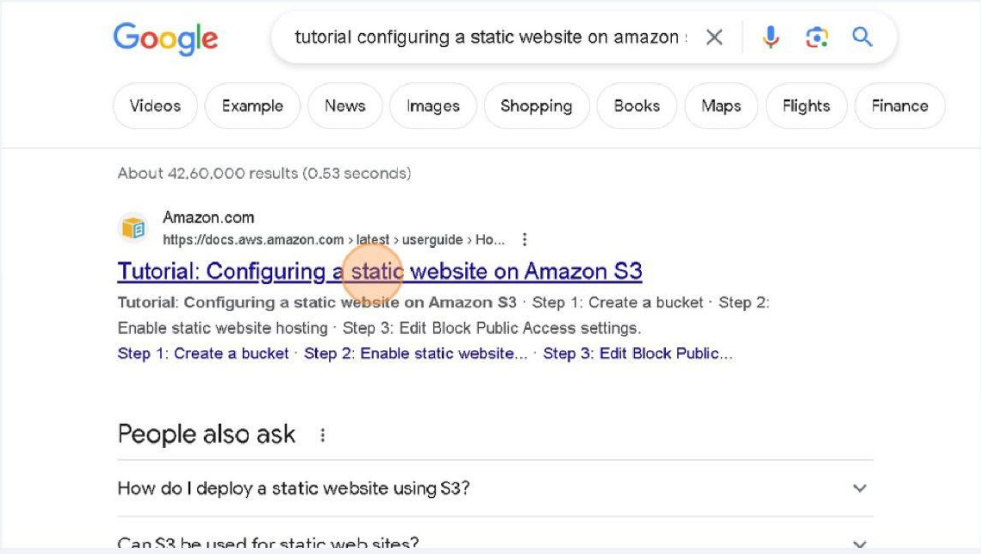
28 Click "Confirm"



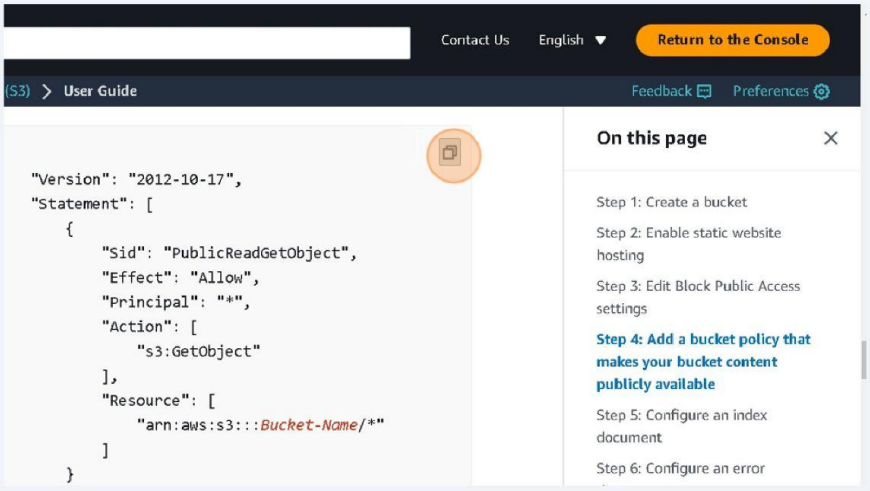
29 Click "Edit"



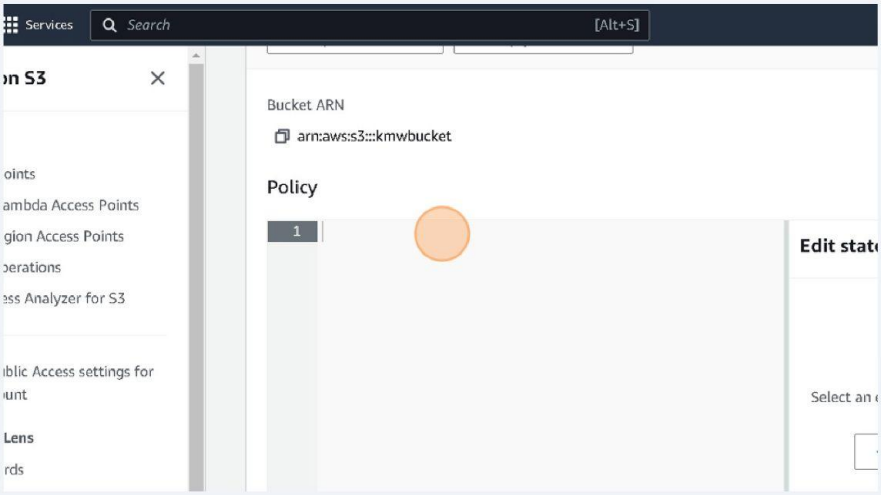
30 Click "Tutorial: Configuring a static website on Amazon S3"



31 Click here.



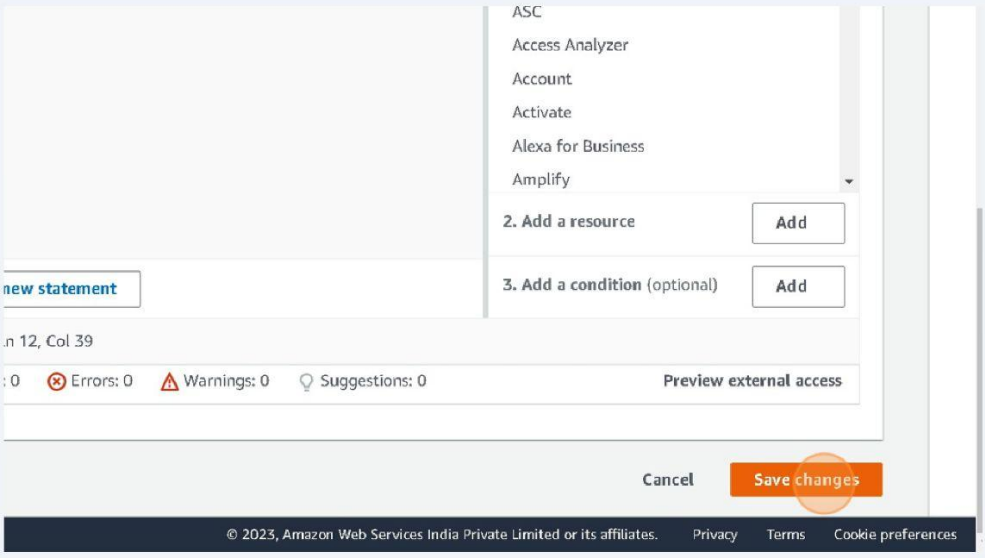
32 Click here.



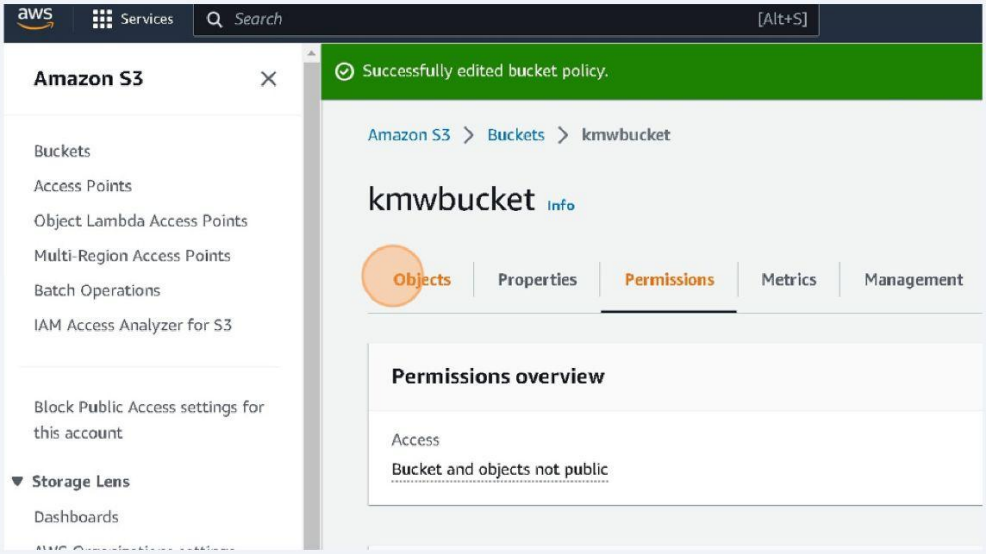
33 Press **CTRL** + **V**

34 Replace: Bucket-Name with "youbucket" name.
Eg: "arn : aws : s3 : : : my-bucket/*"

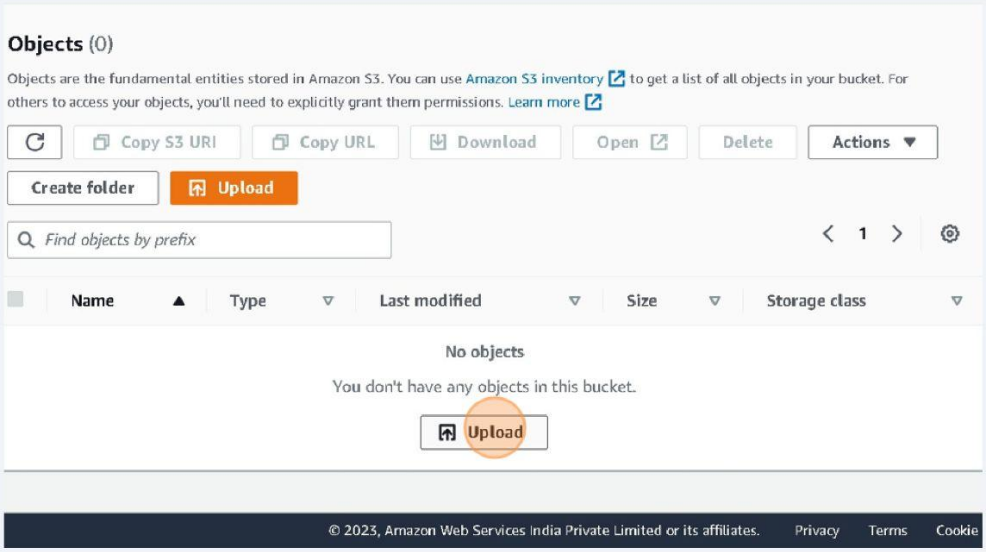
35 Click "Save changes"



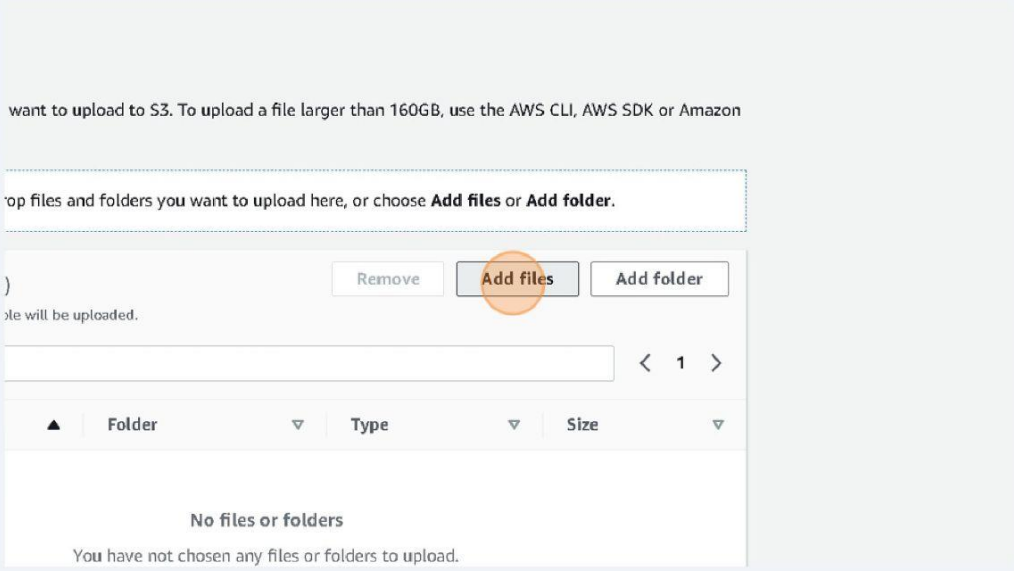
36 Click "Objects"



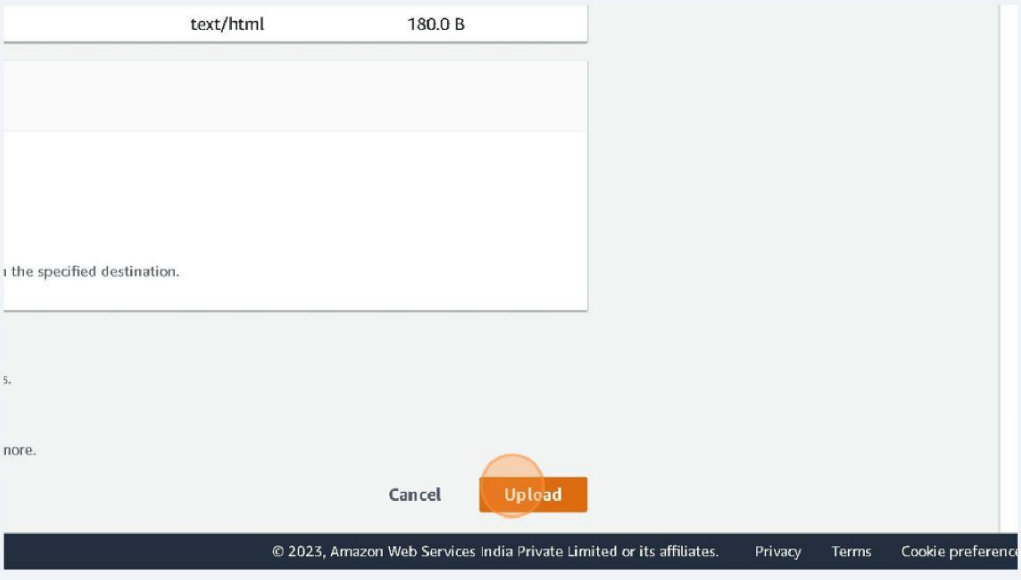
37 Click "Upload"



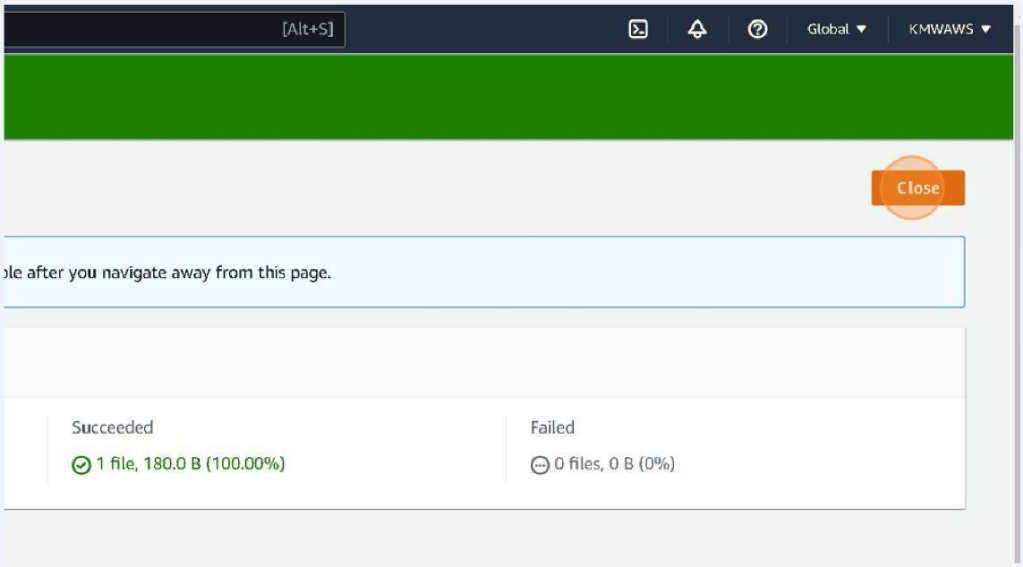
38 Click "Add files"



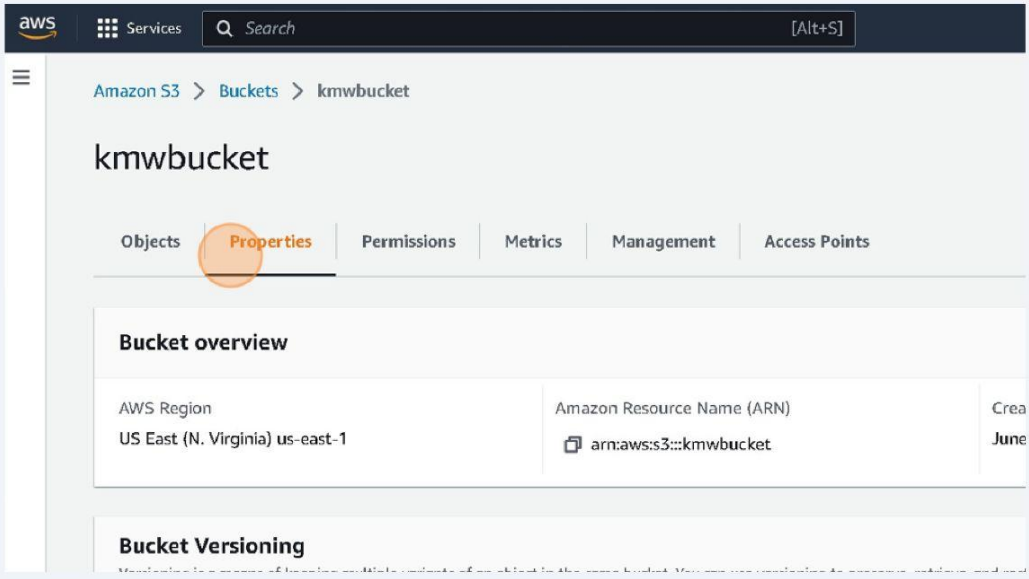
39 Click "Upload"



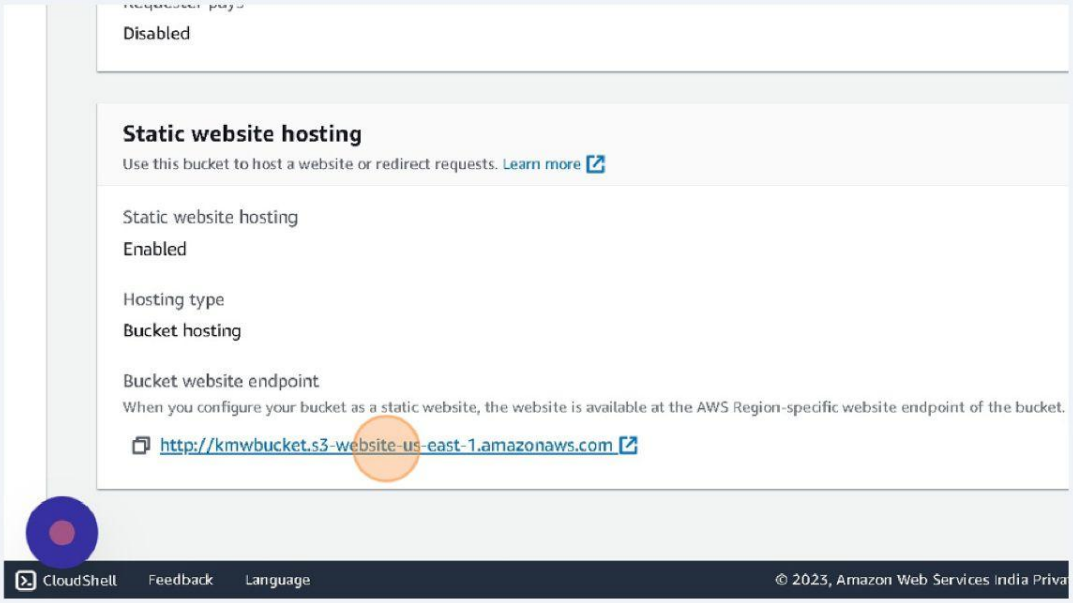
40 Click "Close"



41 Click "Properties"



42 Click "http://kmwbucket.s3-website-us-east-1.amazonaws.com"



43 AWS S3 Setup Successful.

Delete The Bucket:

Amazon S3 → Choose Bucket: iacsd.com → Click On: Empty

→ Type : (*permanently delete*) → Click on : Empty → Click on : Delete

→ Type: *bucket_name* [Eg. *iacsd-web*] → Click On: Delete

—---------- AWS S3 End
---—---------—----------

Git:

→ Take a fresh “debian Machine”
→ Configure the machine
→ Open PuTTY
→ Login from the PuTTY

```
sudo su -
apt-get update
apt-get install git
mkdir techgalena
cd techgalena
git init
ls -a
nano index.html
```

```
 GNU nano 3.2                    index.html                    Modified

<!DOCTYPE html>
<html>
<head>
    <title>TechGalena</title>
</head>
<body>
    <h1>Welcome to My TechGalena</h1>
    <p>This is a single page HTML document.</p>
    <p>Feel free to customize it as per your needs.</p>
    <ul>
        <li>List item 1</li>
        <li>List item 2</li>
        <li>List item 3</li>
    </ul>
    <p>Thank you for visiting!</p>
</body>
</html>
```

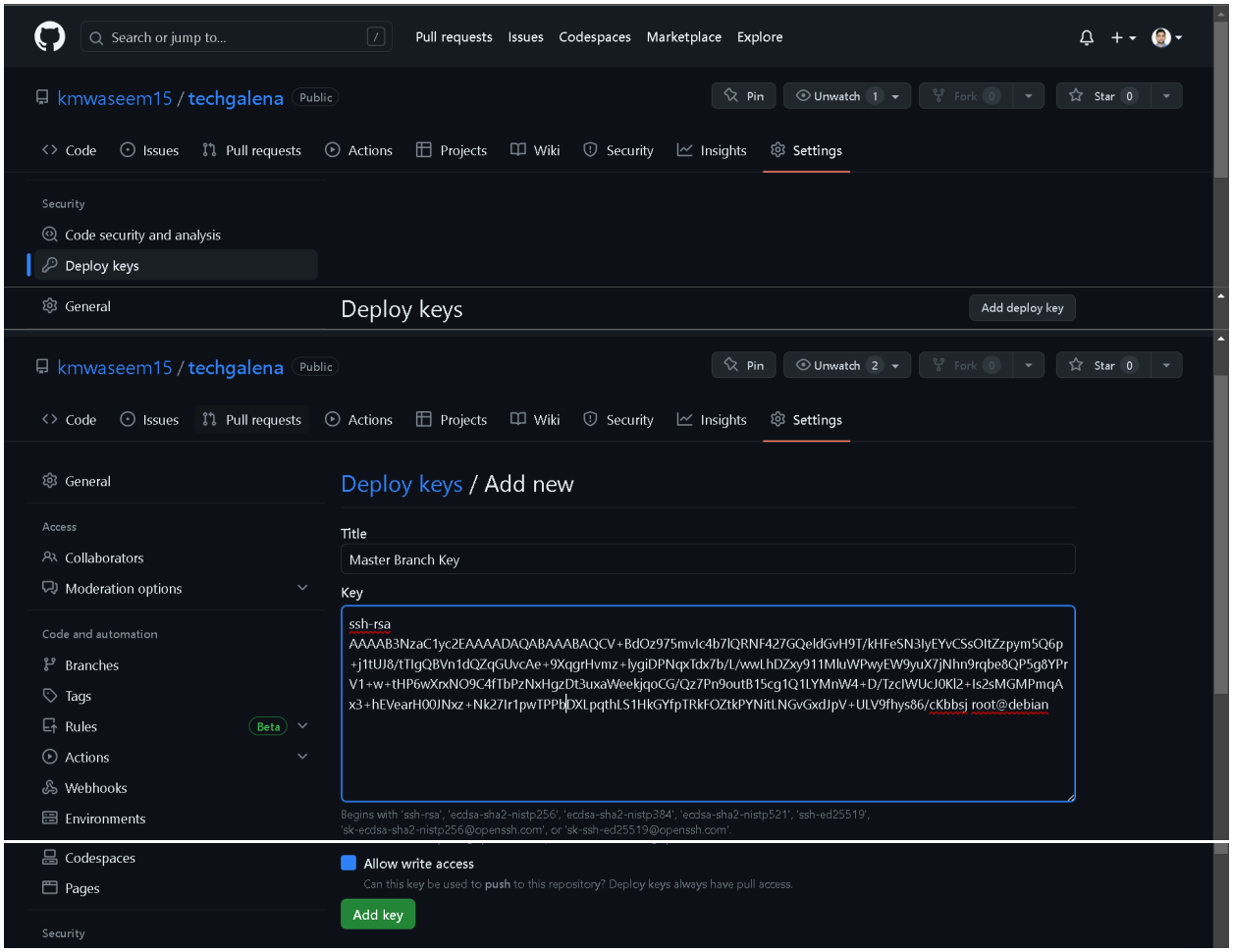
```
git status
```

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        index.html

nothing added to commit but untracked files present (use "git add" to track)
```

```
git add index.html
git status
git commit -m "This is my first commit"
git config --global user.email "example@gmail.com"
git config --global user.name "username" #username = repository name
git commit -m "This is my first commit"
```

```
git log
git push origin master
git remote add origin "git@github.com:kmwaseem15/techgalena.git"
git push origin master
ssh-keygen
cat /root/.ssh/id_rsa.pub
```

Paste this key to github repository: Settings → Deploy Keys → Click on: Add deploy key → Give key a Name Under : Title → Paste the key Under: Key → Click On: Allow Write → Click On: Add key.



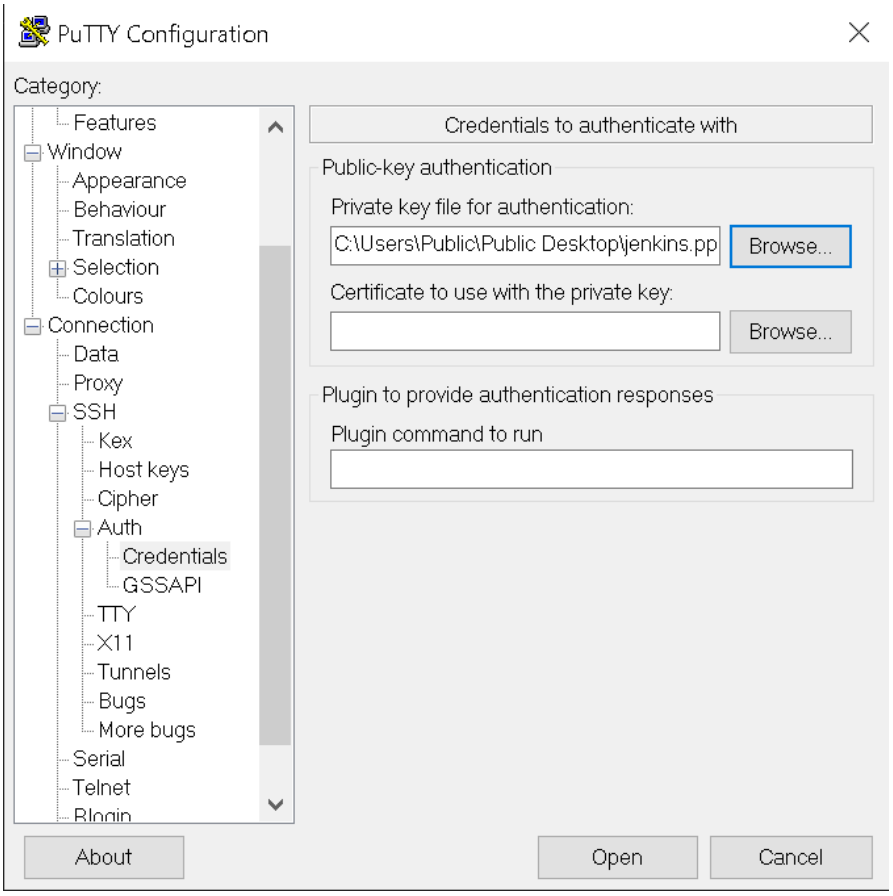

```
git config pull.rebase true
git pull origin master
git push origin master
```

—---------- Git End —---------—----------

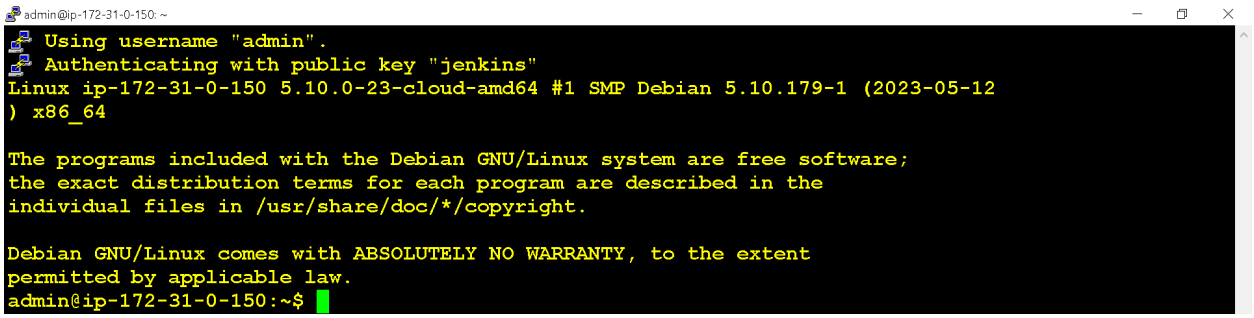
Jenkins Installation On: AWS (Deb 10)

- AWS
- Region: US East (N. Virginia)
- EC2
- Launch Instance
- Name : Jenkins
- OS: Debian
- Create Key Pair
- Networking: Auto Assign Public IP Enable
  - ☑ Allow SSH Traffic from: Anywhere
  - ☑ Allow HTTP Traffic From Internet
  - ☑ Allow HTTPS Traffic From Internet
- Click: Launch Instance

→ Open PuTTY - Enter IP Address
→ Connections → SSH Auth → Credentials → Private Key File For Auth
→ Save Settings → Open

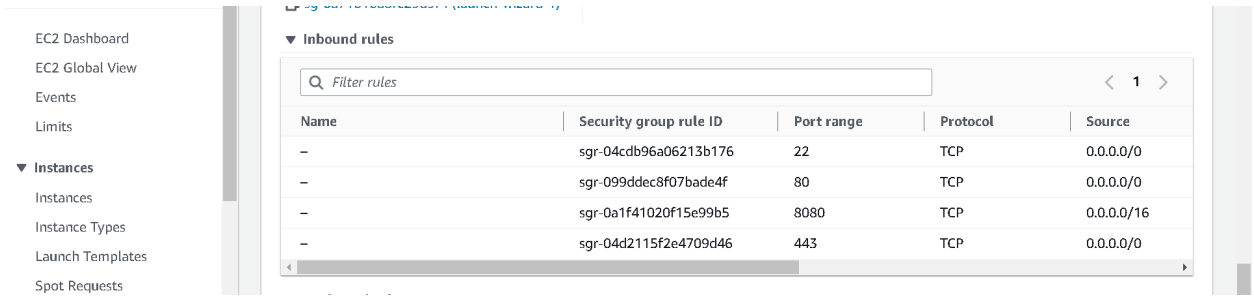


→ Login → Username: **admin** [Login Succesful]




```
apt-get update -y
apt-get upgrade -y
apt-get install gnupg -y
apt-get install default-jre -y
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo
tee \
    /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
    https://pkg.jenkins.io/debian binary/ | sudo tee \
    /etc/apt/sources.list.d/jenkins.list > /dev/null
apt update -y
apt-get install jenkins -y
sudo usermod -a -G root jenkins
systemctl status jenkins
```

- Go to AWS → EC2 → Select Your Instance → Scroll Below : Security → Allow Inbound Traffic on port : 8080



- Open Jenkins Public IP in Browser: 3.238.69.100:8080
- Copy Red Path

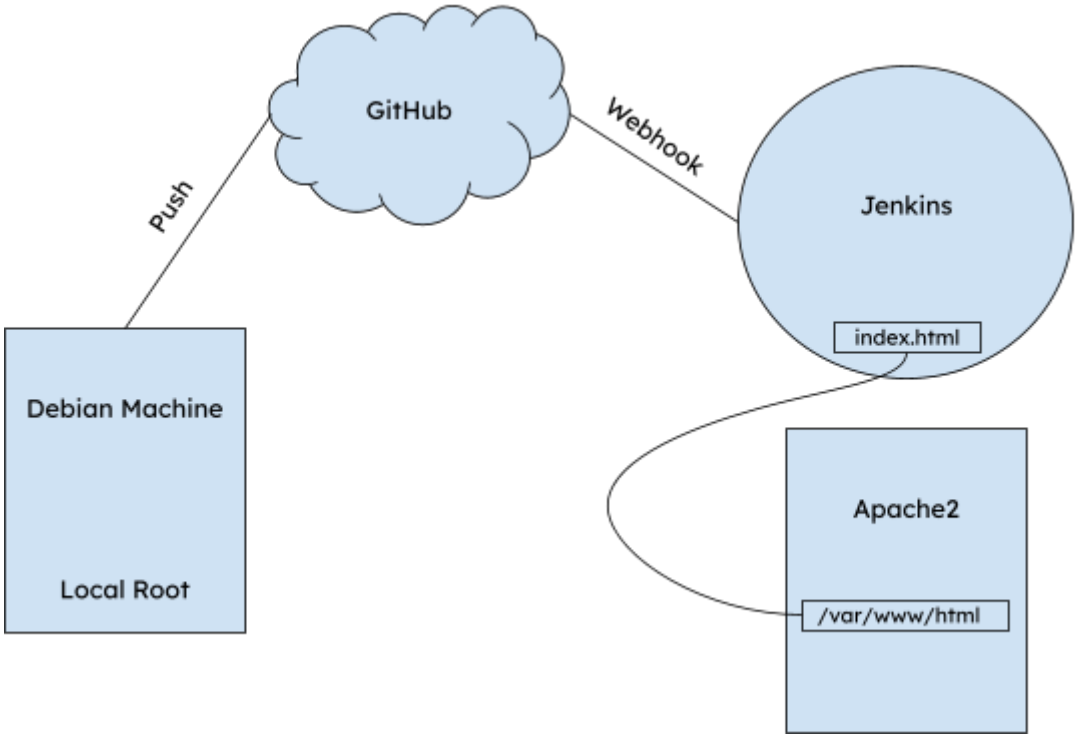
  /var/lib/jenkins/secrets/initialAdminPassword

- Cat : the path in deb 10

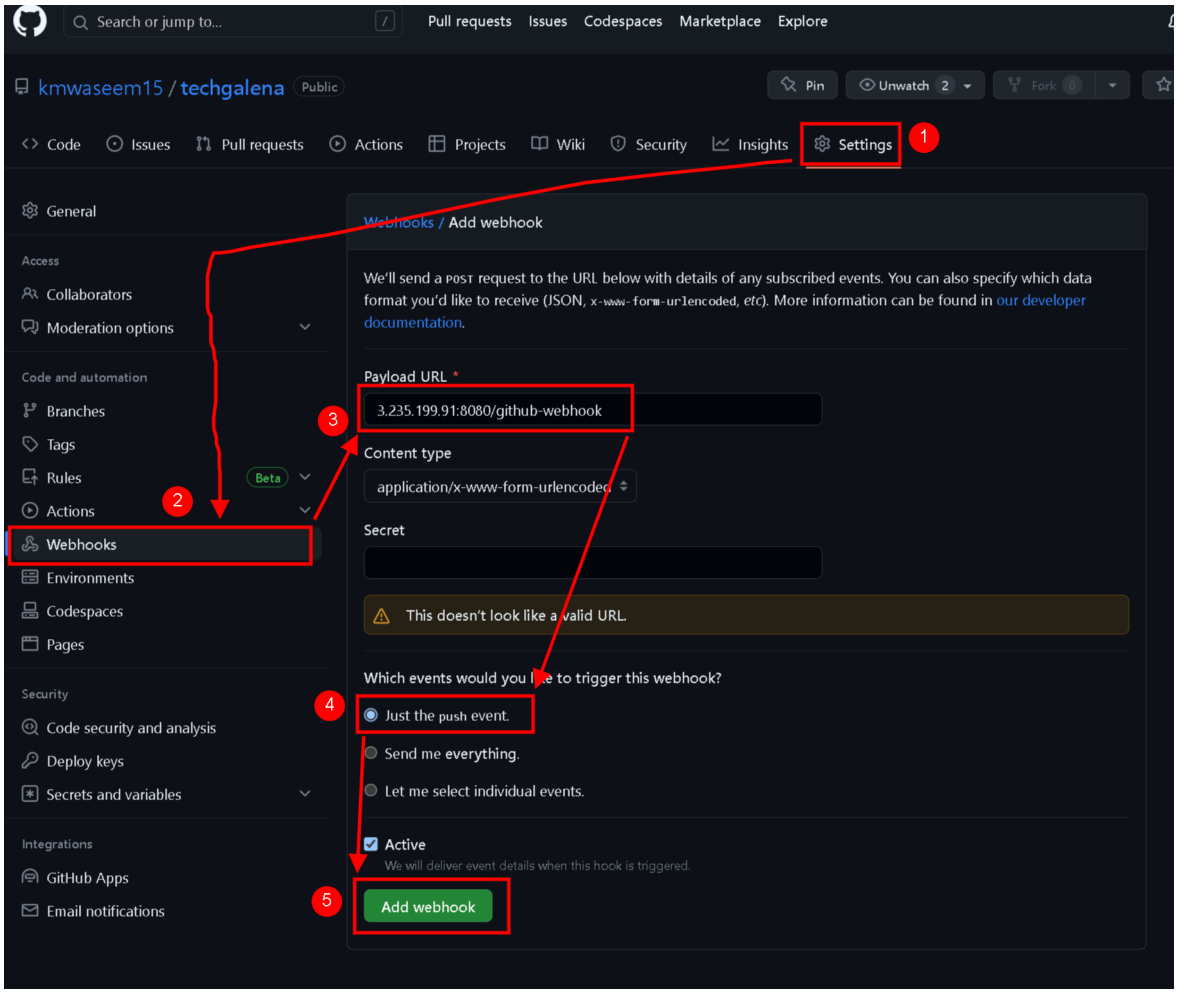
```
admin@ip-172-31-2-20:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
6801195dab6a4dc1a0b08e61fcc17c04
```

- Copy cat content
- Paste in : Admin Password Fileld
- Install: Suggested Plugins
- Create First Admin User
  - Save and Continue
- Start USing jenkins
- Logout
- Stop EC2 Instance
- Verify Stoppage on instances: EC2 → Dashboard → Instances → Instance State → No Instance Should be there.

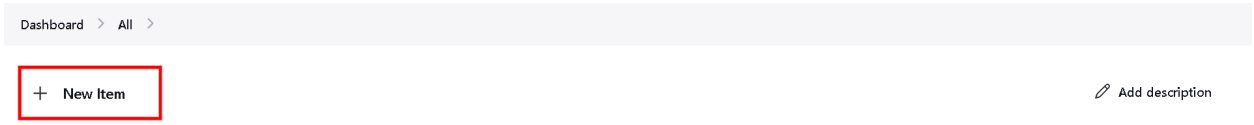
Jenkins Working With Github/Debian Machine For CI/CD :



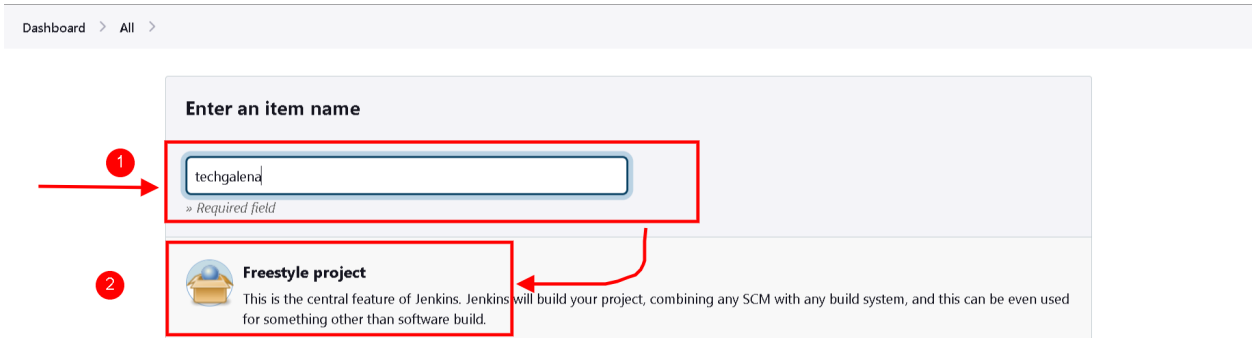
Go to: Github → Your Repository → Settings → Webhook
→ Click on: Add webhook → Edit Payload URL: jenkins ip:808/github-webhook
→ Scroll Below & Click on: Add webhook



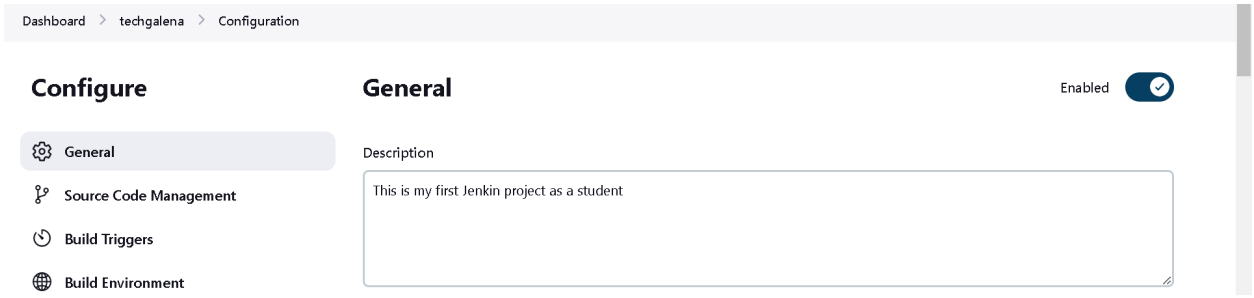
Now go to: Jenkins Web Page → New Item



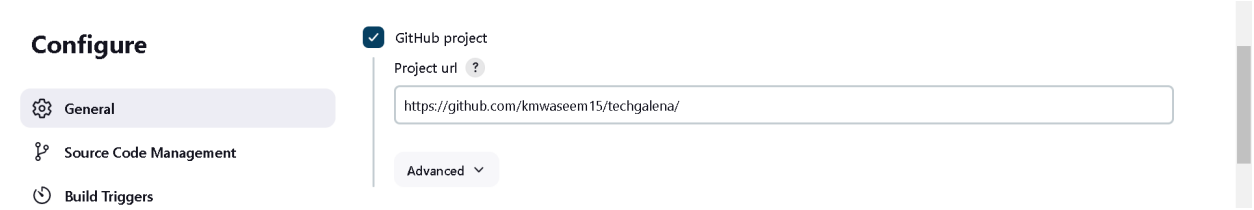
Enter : item name → Click on: Freestyle project → Click: OK



Enter: Description



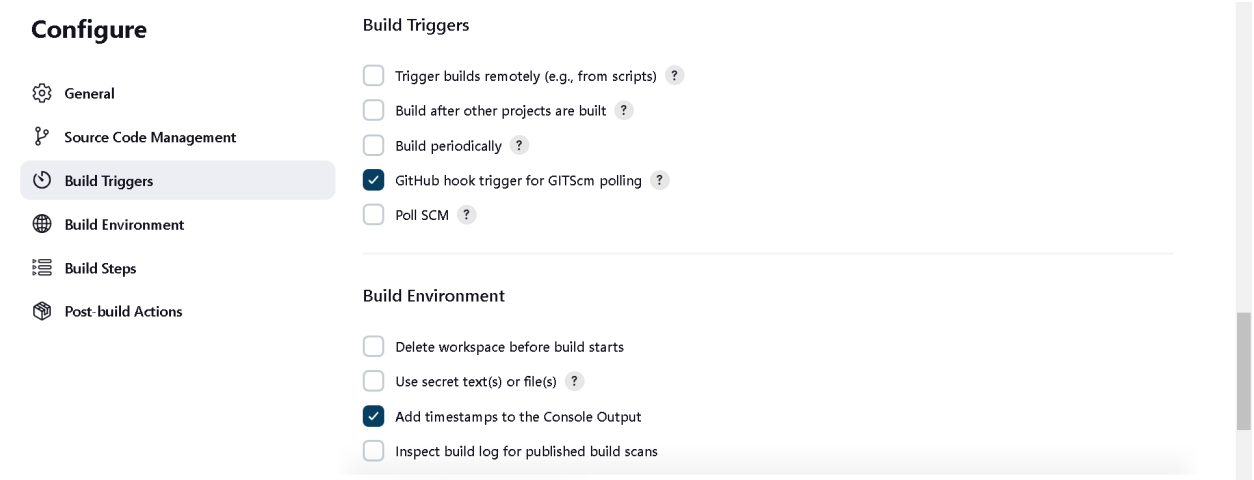
Check: Github Project Enetr Github Repository Link



Install: git on Jenkins Machine Running on AWS

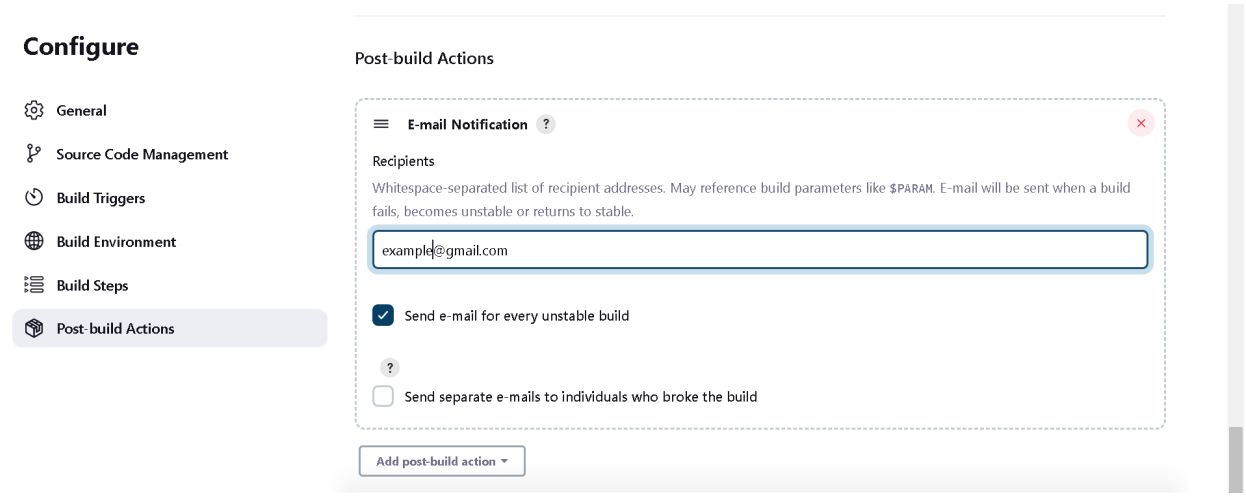
Branch Name: */master

Check: GitHub hook triggers | Check: Add timestamp to the Console Output



Go to: Build Steps → Expand Below Tab → Check: Execute Shell → Type Something

Post build actions: → Expand Add post-build action: Select: Send e-mail Notification → Type your email



Click on: Save

CI/CD in Jenkins

Go to: Jenkins Machine running on AWS
Change “jenkins” user’s password

```
admin@ip-172-31-2-20:~$ sudo passwd jenkins
```

Switch to “jenkins” user

```
admin@ip-172-31-2-20:~$ sudo su jenkins
```

Generate “key-pair” for jenkins user

```
jenkins@ip-172-31-2-20:/home/admin$ ssh-keygen
```

Cat the generated “key-pair”

```
jenkins@ip-172-31-2-20:~$ cat /var/lib/jenkins/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDJiSQZYxRiAIvZzRGThJiSVCaE0Cj7lm7O8O7DDwMCN8o8WRXumnSf7wTFyJV
L+Cv3uFjlHmxLd8uS4rKh06aB9Se6n/BWBvPgPwqINafTKngEm52YGg1531Vyd2pDlk0v0z1DLqFq9USwyqCDPZC4ZTHalQ1Xmv
Xrzua3DKqU1J/7FVZ1i5e09ZfFEcoePV0S7ISqLpol8AhO6btDAFGJnBY71nIbz4x7hlG+BXu/1JNAr2Fm5UKVqcY/7goJdR5ZM
mHWx3vMqz2U/pV4S8VWKaocIHtrJfy85GVingZT0Fd/DNUwdv/1xynQG7cUKB5EsrRuNjR3zYONmDu03+Ndt9J0CV+U0F5fkDhk
ZgPt9dNVtBzlmu/kw2WHWt/9x7CtO/kbmtf7Ce9vb4LZAOrhwy2B8klBljB4m85BjLrRm3Nd3pqJcaaqk0lp5+1YuYSaVgJpy+m
y3XJTBsyU/vJNFbkJwL2ZCq5XRKRkTirFigDLyy1H2Wohs3/NMK7d4GM= jenkins@ip-172-31-2-20
```

Now copy this public key to: Web server’s :
“/home/admin/.ssh/authorized_keys” path

```
  GNU nano 5.4                        /home/admin/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC9gH9422DFiUtg7ZSCtkLEuG8cWS6zcaBoNGdpd4N+iEJ1Ihek53SyFdXBew>
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDJiSQZYxRiAIvZzRGThJiSVCaE0Cj7lm7O8O7DDwMCN8o8WRXumnSf7wTFyJ>
```

Change permission of /var/www/ in web Server machine.

```
admin@ip-172-31-15-56:~$ sudo chmod -R 7777 /var/www/
```

Go to jenkins Your Project Configure :
Add : scp index.html admin@172.31.15.56:/var/www/html/

Click on: Save

Now Come to: Jenkins Machine running on AWS (git)
Go to your git directory: Eg. techgalena

```
admin@ip-172-31-2-20:~$ cd techgalena/
```

Make a new: index.html file

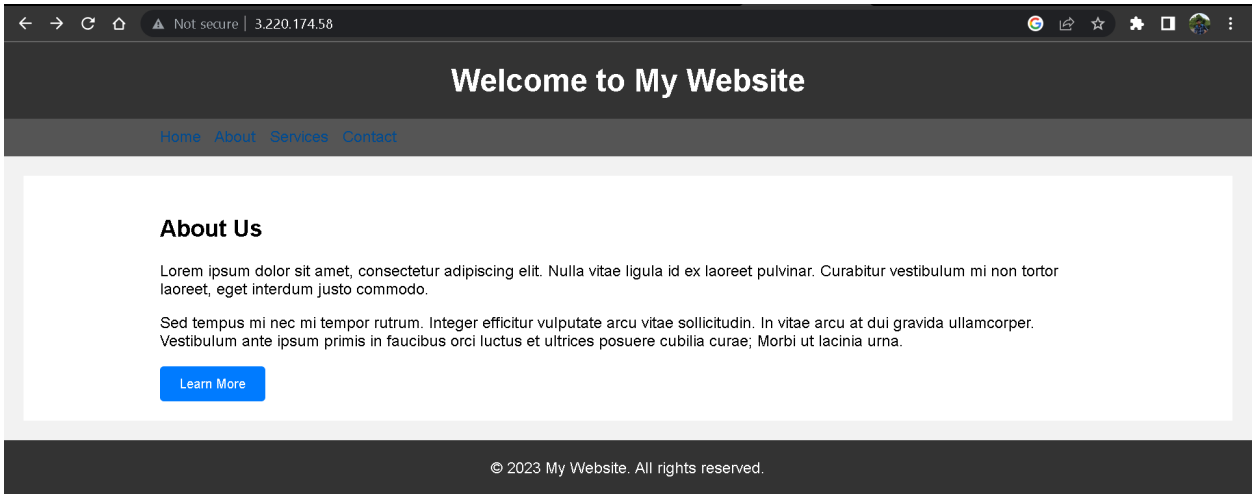
```
admin@ip-172-31-2-20:~/techgalena$ sudo nano index.html
```

```
admin@ip-172-31-2-20: ~/techgalena
  GNU nano 5.4                                index.html *
<!DOCTYPE html>
<html>
<head>
  <title>My Website</title>
  <style>
```

```
admin@ip-172-31-2-20:~/techgalena$ git add index.html
```

```
admin@ip-172-31-2-20:~/techgalena$ git commit -m "New html commit"
[master a0dd261] New html commit
 1 file changed, 112 insertions(+), 3 deletions(-)
 rewrite index.html (100%)
```

```
admin@ip-172-31-2-20:~/techgalena$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.12 KiB | 1.12 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:kmwaseem15/techgalena.git
   3b0059b..a0dd261  master -> master
```



—---------- Jenkins End

—---------—----------

Copy “index.html” file from Jenkins Machine To New Debian Based AWS EC2 Machine/var/www/html/

Create New EC2 Instance: Login on PuTTY & Install: Apache2

```
admin@ip-172-31-15-56:~$ sudo apt-get update
```

```
admin@ip-172-31-15-56:~$ sudo apt-get install apache2
```

Now: Remove the default “index.html” file to the path /var/www/html

```
admin@ip-172-31-15-56:~$ sudo rm /var/www/html/index.html
```

—---------- End —---------—----------

Virtual Private Cloud (AWS):

1. AWS → Name: ditiss-lab → Auto Generate → IP 172.30.0.0/16
   → No IPv6 → No. Of Availability Zones [Single Zone(1)]
   → No. ofpublic subnets: 1 → No. of private subnets: 1
   → NAT Gateway: 1AZ → Click on: Create VPC → View VPC

   Now Create Two EC2 Instance:

   1. EC2 - 01
      → Launch Instance → Name: Private
      → OS: Debian
      → Create a new key-pair (.pem): Name - private
      → RSA → .pem
      → Networking: Auto Assign Public IP Enable
      - ☑ Allow SSH Traffic from: Anywhere
      - ☑ Allow HTTP Traffic From Internet
      - ☑ Allow HTTPS Traffic From Internet
      - Subnets: private

      → Launch Instance

   2. EC2 - 02
      → Launch Instance → Name: Public
      → OS: Debian
      → Create a new key-pair (.pem): Name - public
      → RSA → .ppk
      → Networking: ditiss-vpc
      - ☑ Allow SSH Traffic from: Anywhere
      - Subnets: public
      - Auto Assign IP: Enable

      → Launch Instance
      → Copy IP → Paste in PuTTY
      → Login as passwordless SSH

```
nano key:
     Paste private Key Of EC2 - 01 (i.e private.pem)
chmod 400 key
ssh -i key admin@Private IP of EC2 - 01
sudo apt-get install update
```

```
sudo apt-get install apache2
sudo rm /var/www/html/index.html
sudo nano /var/www/html/index.html
sudo apt-get install curl
curl (Private IP) → We will see private machine's browser
sudo apt-get install links
links (Private IP) → We will see private machine's browser
```

Remove VPC:

Terminate both Instances

Then Delete VPC Created

Will show 3 error red marks:

1. Remove NAT gateway At left side Actions Delete
2. EC2 Instance → Elastic IP's → Select all
   → Release Elastic IP address

There should be nothing in: 1. Route Table 2. Elastic IP 3. VPC 4. EC2 Instances
5. Total No. Of Active Regions: Not more than 1

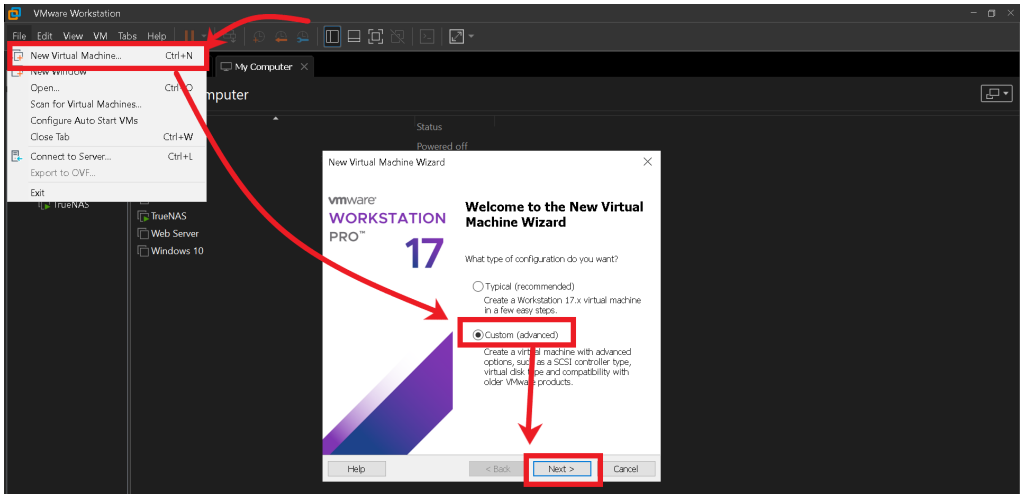
—---------- VPC End

—-----—----------

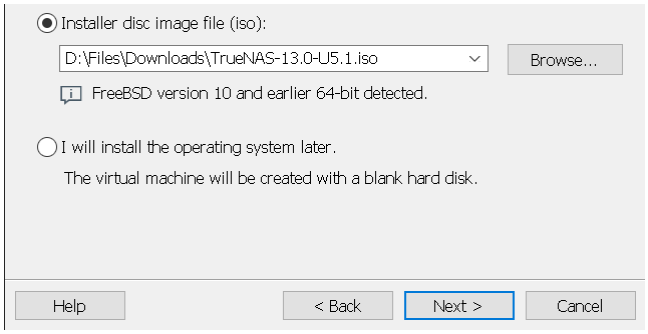
Set Up ZFS Volume on TrueNAS:

1. Download TrueNAS .iso file from the Official Site.

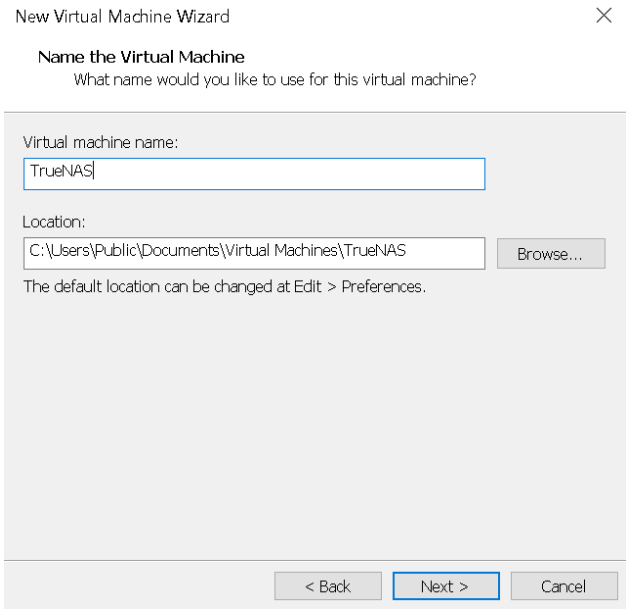
2. Open VMWare → Click on: New Virtual Machine → Custom → Next



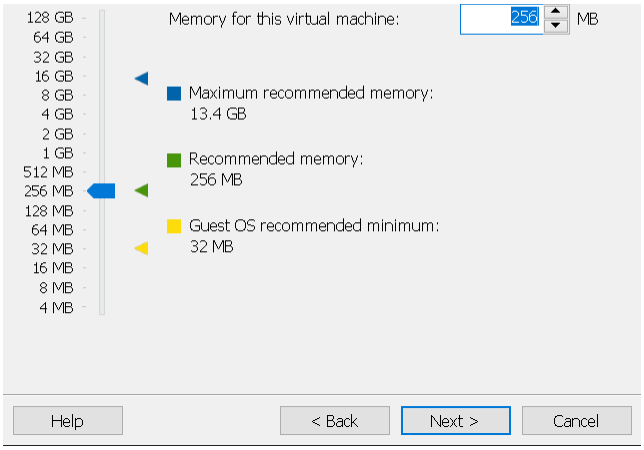
3. Next
4. Select: TrueNAS.iso file. → Next



5. Virtual Machine Name: TrueNAS Location: C:\TrueNAS → Next

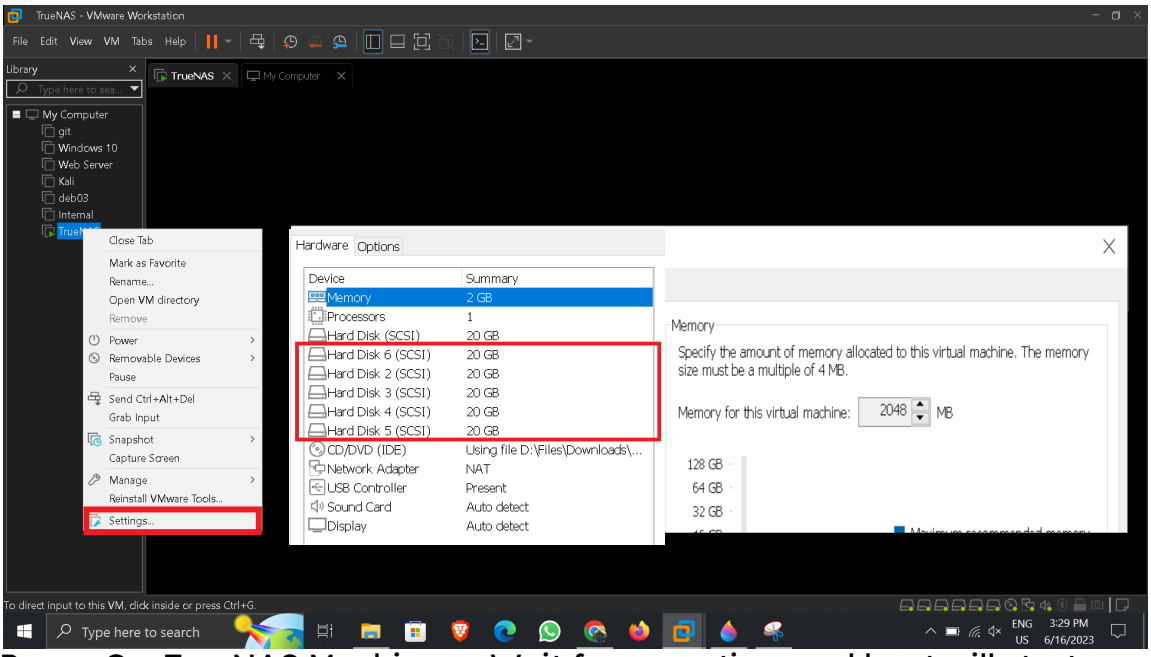


6. Next → RAM:2048 → Next



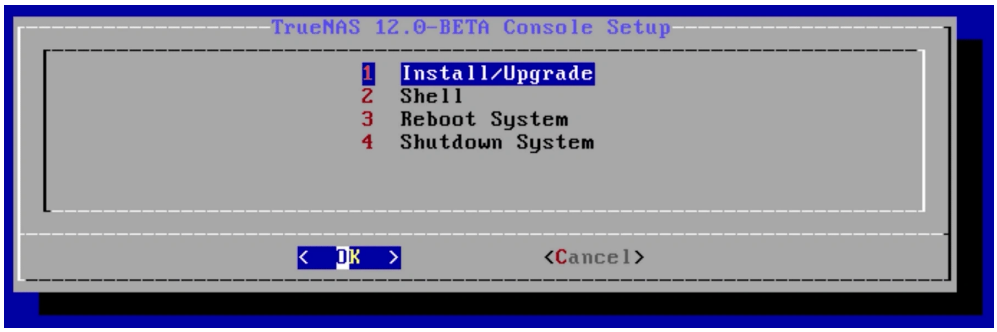
7. Next → Next → Next → Next → Next → Next → Finish

8. Right Click on: TrueNAS Settings Add 5 Disks (Size: > 5GB)

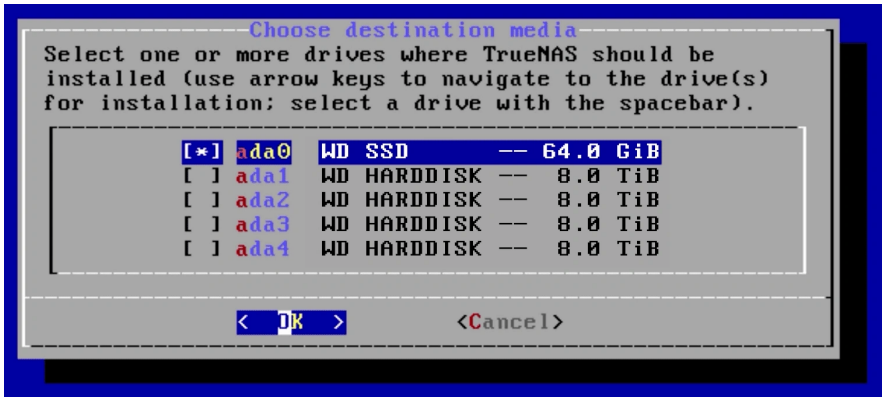


9. Power On: TrueNAS Machine → Wait for some time and boot will start

10. Select Option 1 Click: **OK**

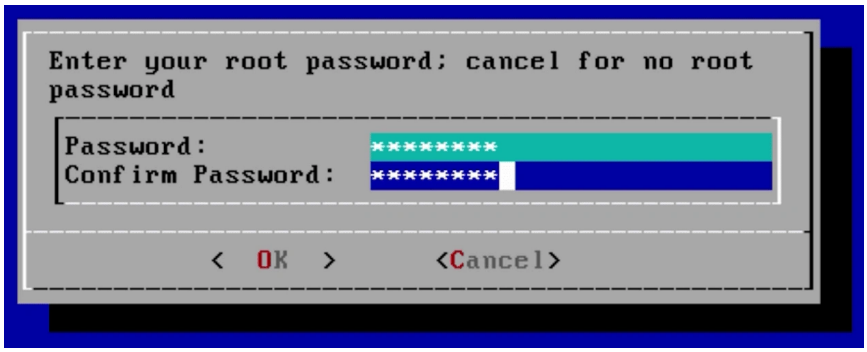


11. Select Disk [***Use Space Key To Select***] → Click: **OK**

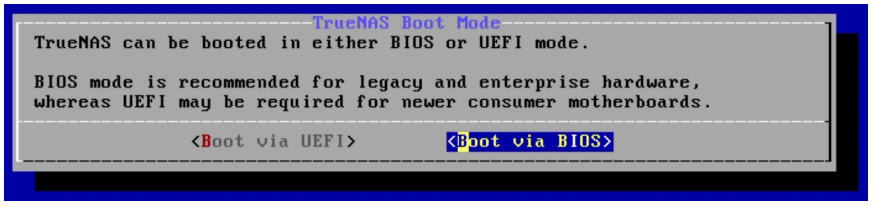


12. Select: **Yes**

13. Assign Password. Click : OK

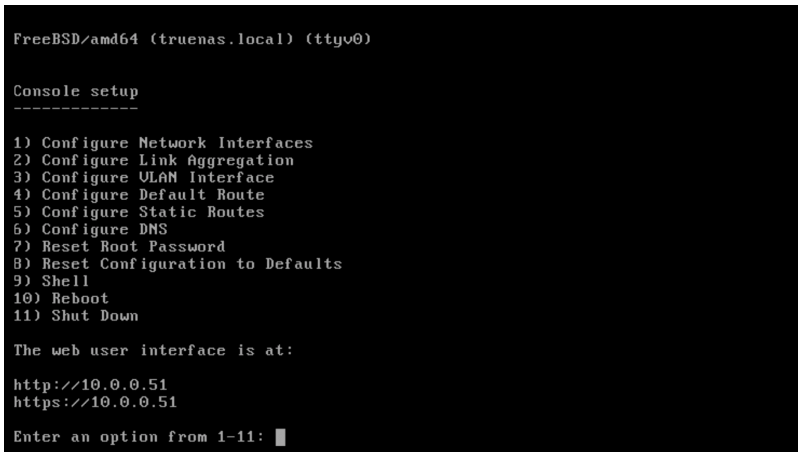


14. Select: **Boot vis BIOS**

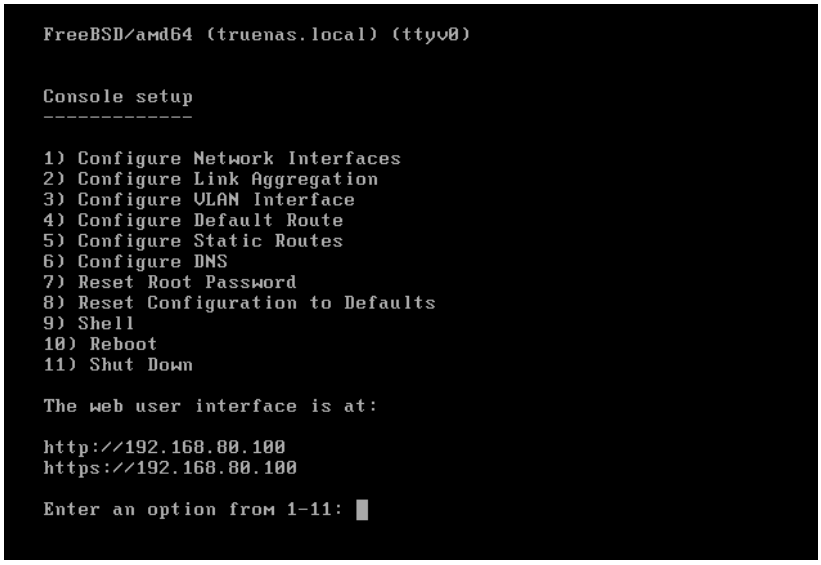


15. Wait for Installation & To Reboot Click: OK

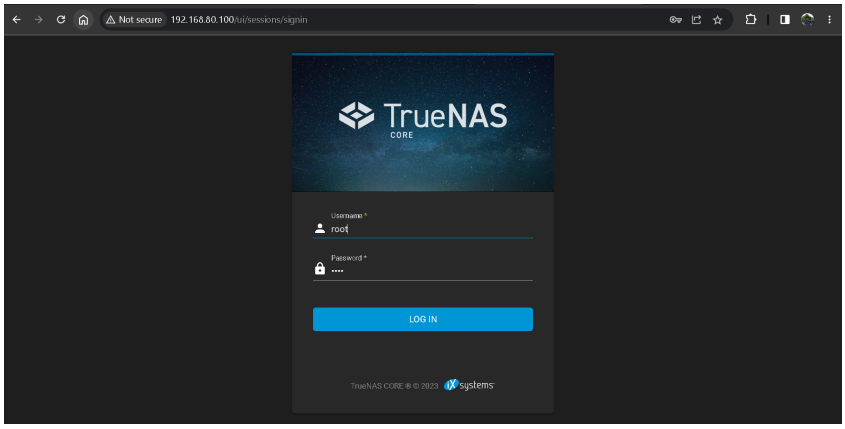
16. After Reboot Configure IP.



17. IP configured.

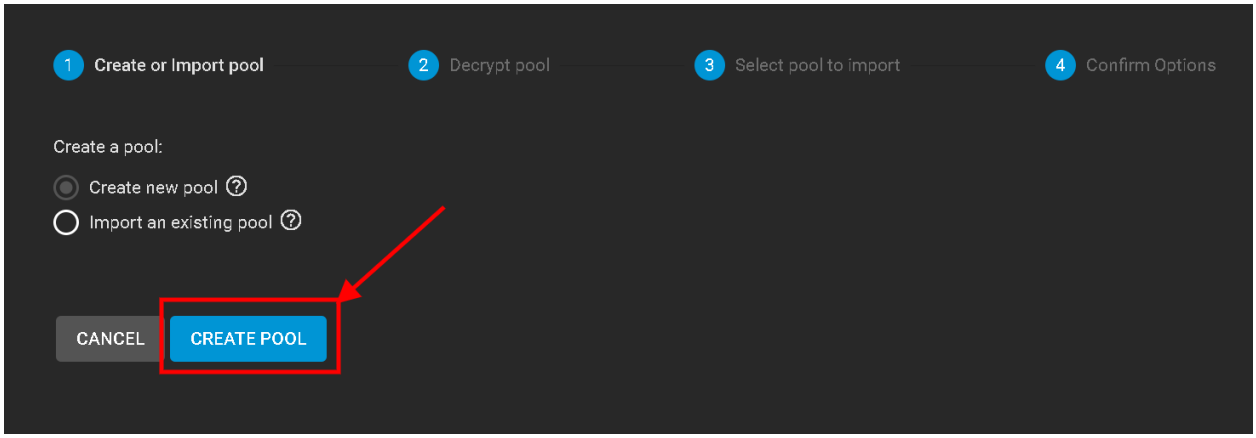


18. Now go to browser: Eneter: Configured IP [Eg. **http://192.168.80.100**]
    Username: **root** | Password : What you configured on stage: 13 [Eg. **toor**]
    Press: Enter

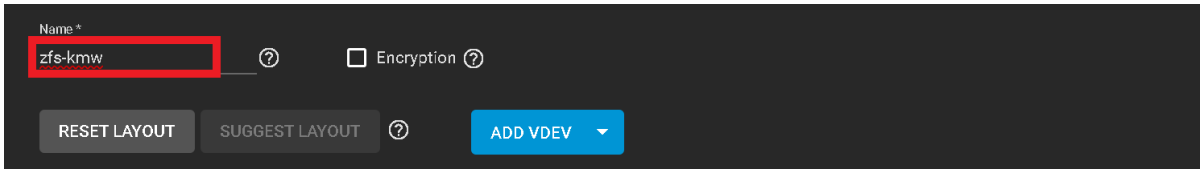


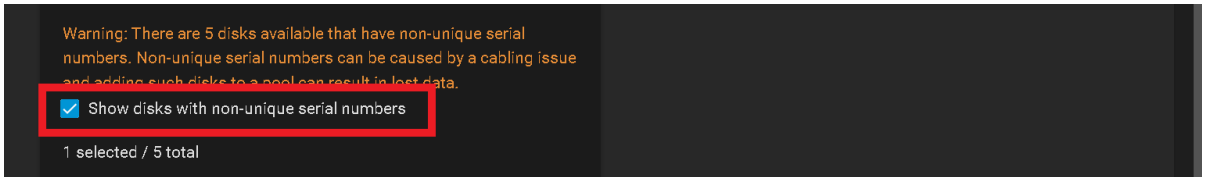
19. Storage → Pool → ADD

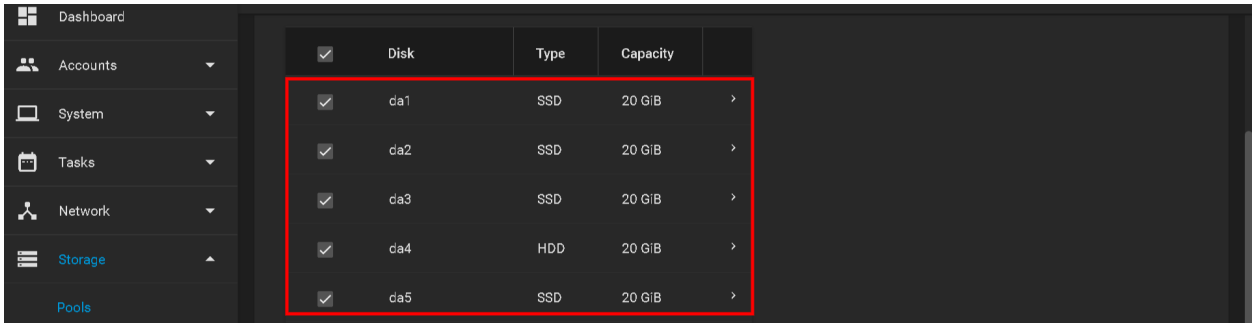
20. Click: Create Pool

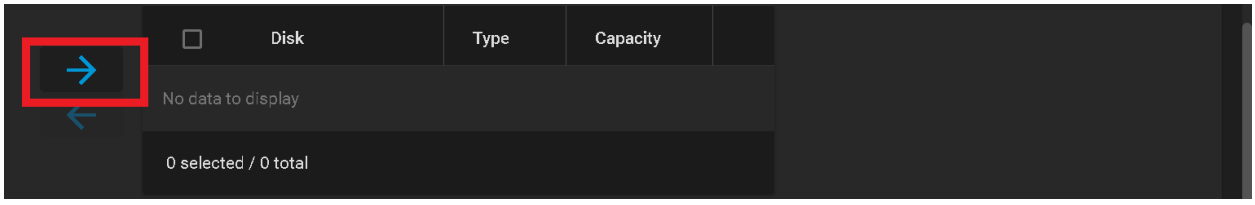


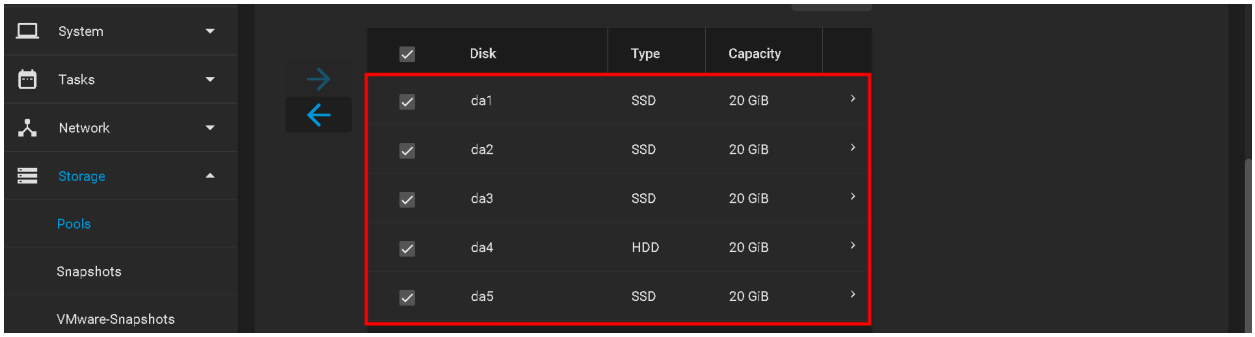
21. Provide a name for the pool, select the disk or disks you want to use for the pool, and choose the RAID level or redundancy mechanism (e.g., RAIDZ1, RAIDZ2, etc.).

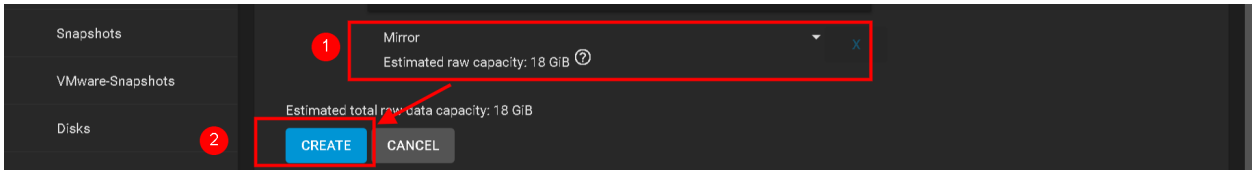


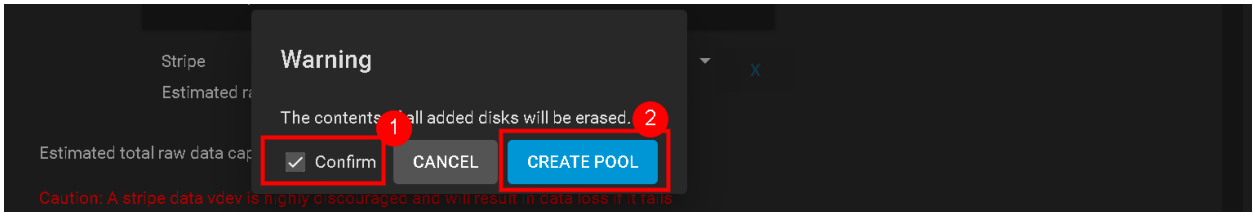


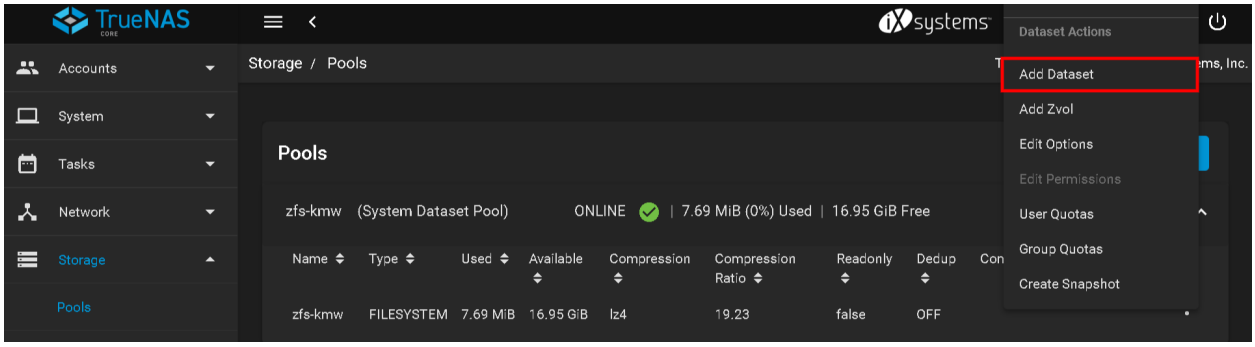




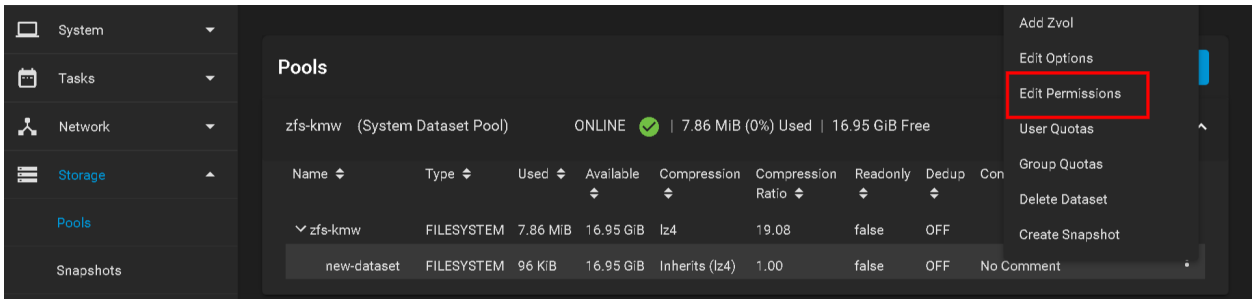

System
Tasks
Network
Storage
Pools
Snapshots
VMware-Snapshots
Disk
Type
Capacity
da1 SSD 20 GiB
da2 SSD 20 GiB
da3 SSD 20 GiB
da4 HDD 20 GiB
da5 SSD 20 GiB

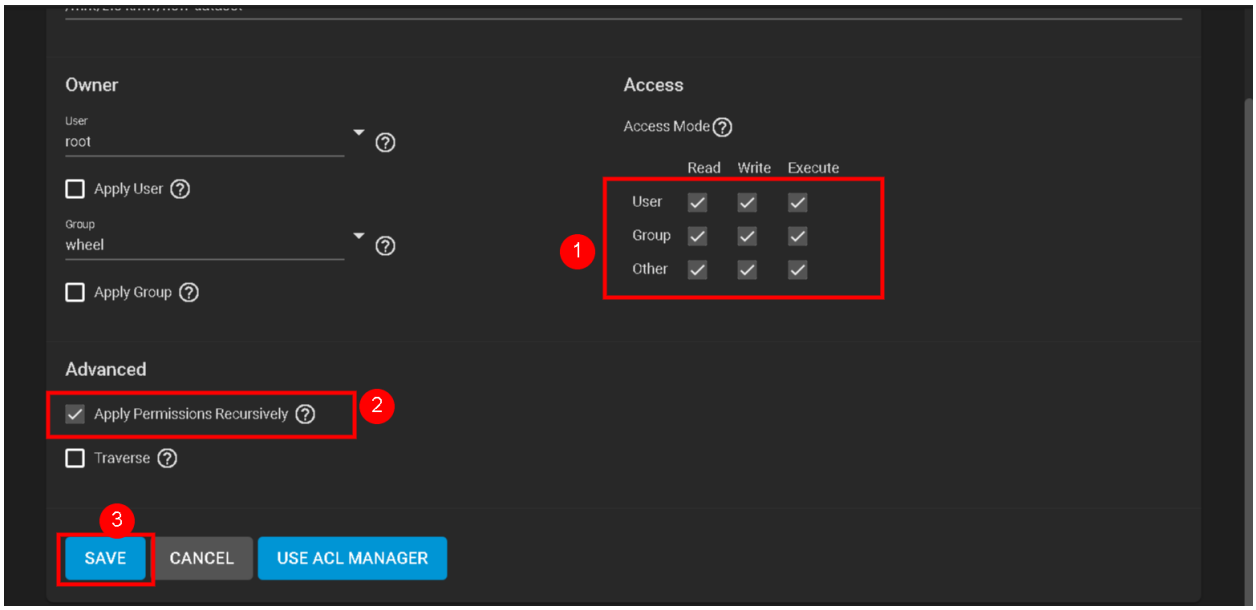

Snapshots
VMware-Snapshots
Disks
1
Mirror
Estimated raw capacity: 18 GiB
Estimated total raw data capacity: 18 GiB
2
CREATE
CANCEL


Stripe
Estimated ra
Estimated total raw data cap
Warning
The contents of all added disks will be erased.
1
2
Confirm
CANCEL
CREATE POOL
Caution: A stripe data vdev is highly discouraged and will result in data loss if it fails

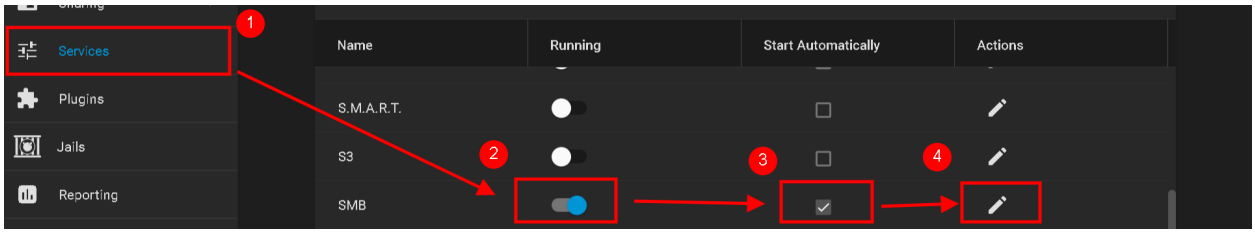

TrueNAS
iXsystems
Accounts
System
Tasks
Network
Storage
Pools
Storage / Pools
Dataset Actions
Add Dataset
Add Zvol
Edit Options
Edit Permissions
User Quotas
Group Quotas
Create Snapshot
Pools
zfs-kmw (System Dataset Pool) ONLINE | 7.69 MiB (0%) Used | 16.95 GiB Free
Name Type Used Available Compression Compression Ratio Readonly Dedup
zfs-kmw FILESYSTEM 7.69 MiB 16.95 GiB lz4 19.23 false OFF


Name and Options
Name *
new-dataset

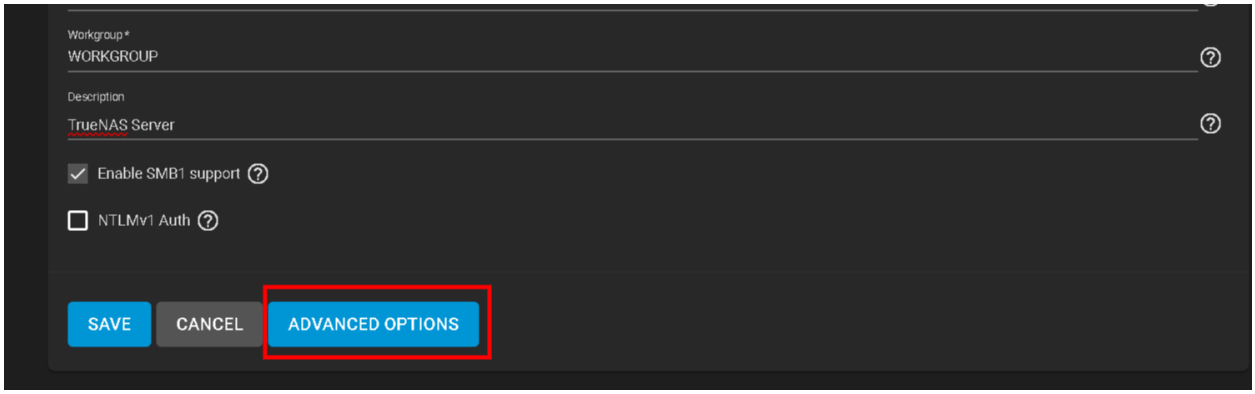

System
Tasks
Network
Storage
Pools
Snapshots
Pools
zfs-kmw (System Dataset Pool) ONLINE | 7.86 MiB (0%) Used | 16.95 GiB Free
Name Type Used Available Compression Compression Ratio Readonly Dedup
zfs-kmw FILESYSTEM 7.86 MiB 16.95 GiB lz4 19.08 false OFF
new-dataset FILESYSTEM 96 KiB 16.95 GiB Inherits (lz4) 1.00 false OFF No Comment
Add Zvol
Edit Options
Edit Permissions
User Quotas
Group Quotas
Delete Dataset
Create Snapshot

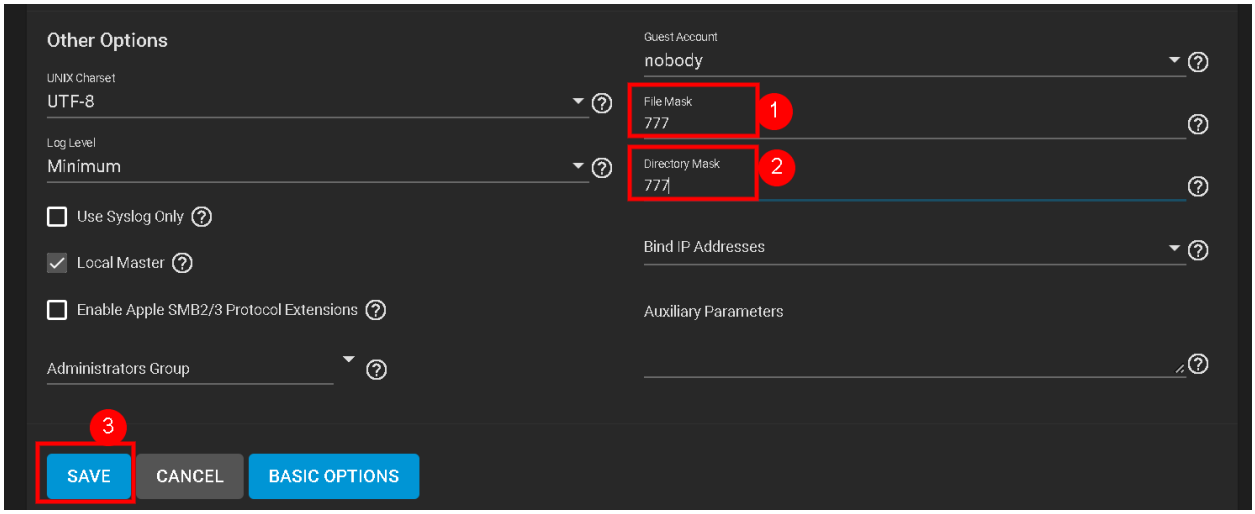


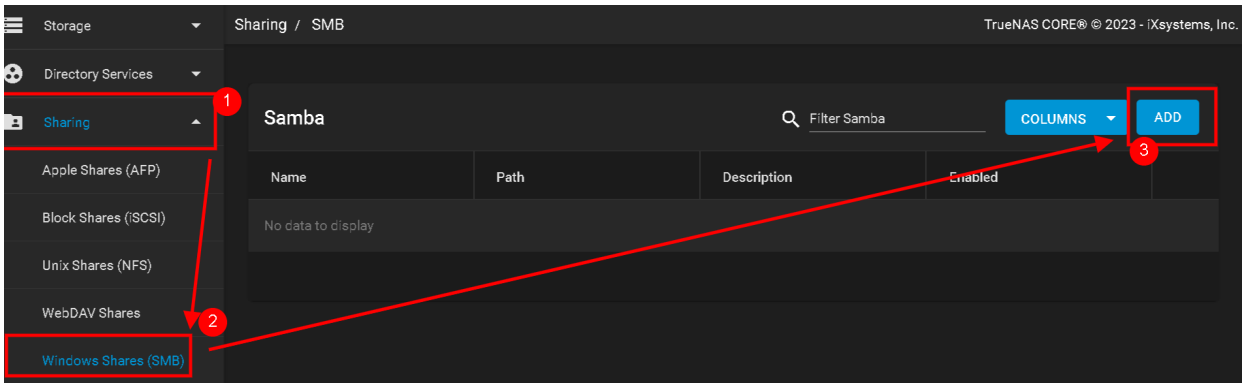
22. Now Go To Services: Enable SMB

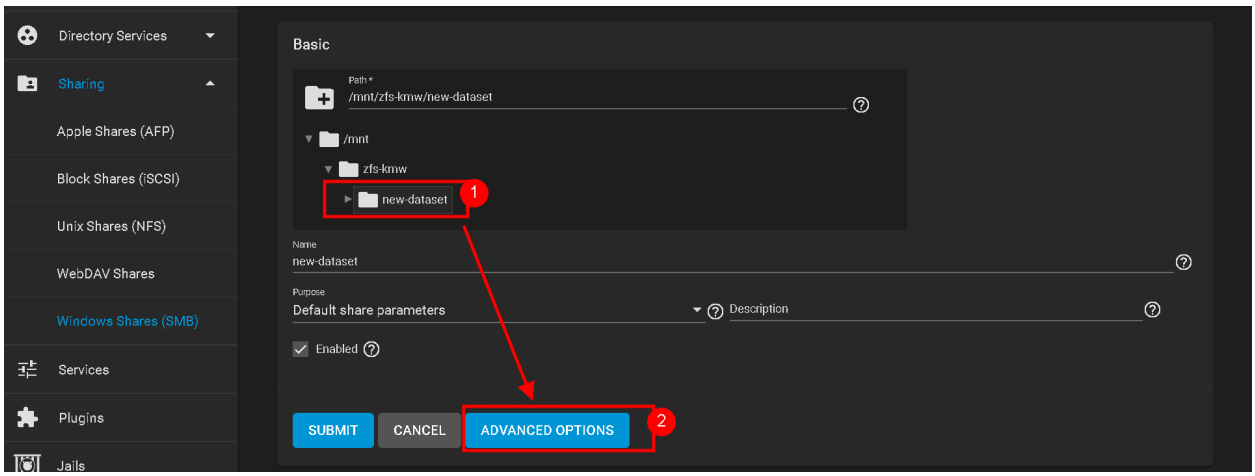


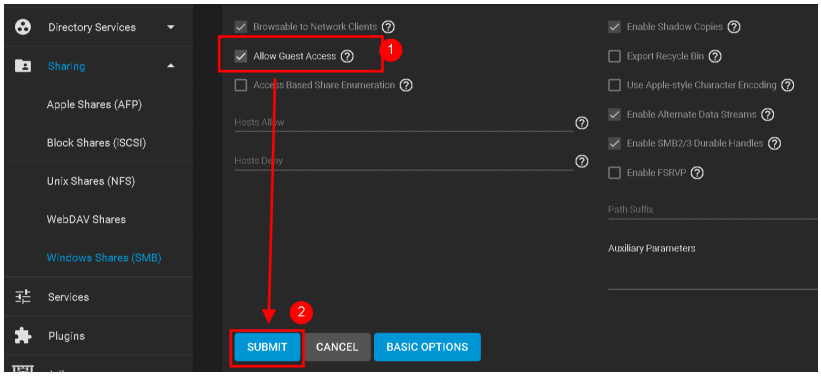
23. Here: Click on “Advance Option”



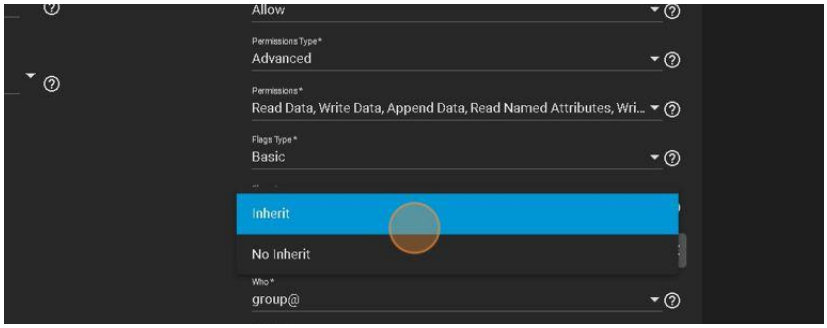

Other Options
UNIX Charset
UTF-8
Log Level
Minimum
Use Syslog Only
Local Master
Enable Apple SMB2/3 Protocol Extensions
Administrators Group
Guest Account
nobody
File Mask
777
1
Directory Mask
777
2
Bind IP Addresses
Auxiliary Parameters
3
SAVE
CANCEL
BASIC OPTIONS

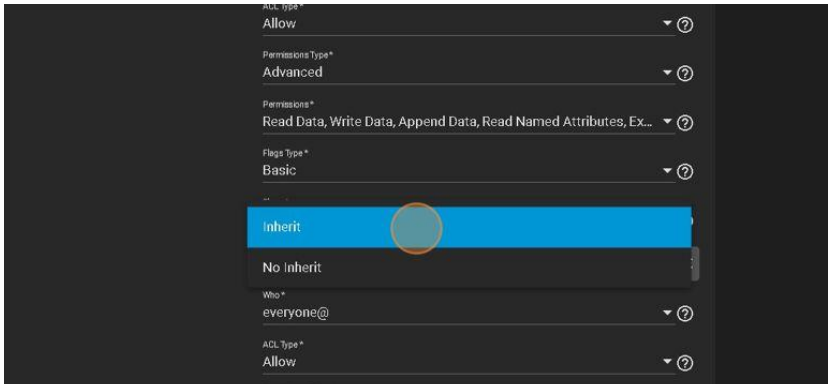

Storage
Directory Services
Sharing
Apple Shares (AFP)
Block Shares (iSCSI)
Unix Shares (NFS)
WebDAV Shares
Windows Shares (SMB)
Sharing / SMB
TrueNAS CORE® © 2023 - iXsystems, Inc.
Samba
Filter Samba
COLUMNS
ADD
Name
Path
Description
Enabled
No data to display
1
2
3

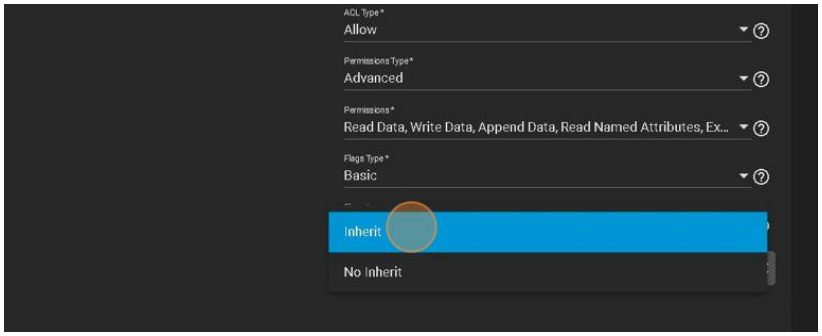

Directory Services
Sharing
Apple Shares (AFP)
Block Shares (iSCSI)
Unix Shares (NFS)
WebDAV Shares
Windows Shares (SMB)
Services
Plugins
Jails
Basic
Path *
/mnt/zfs-kmw/new-dataset
/mnt
zfs-kmw
new-dataset
1
Name
new-dataset
Purpose
Default share parameters
Description
Enabled
SUBMIT
CANCEL
ADVANCED OPTIONS
2

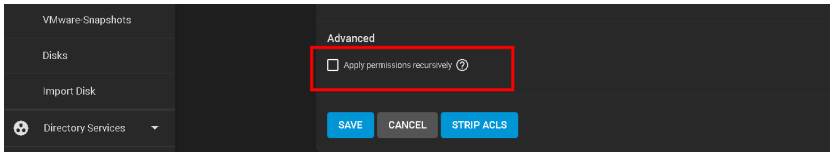


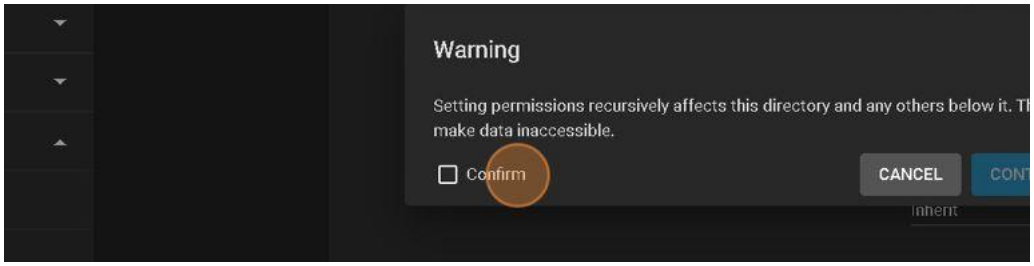
After this it will ask you to go to the ACL: Check : Confirm

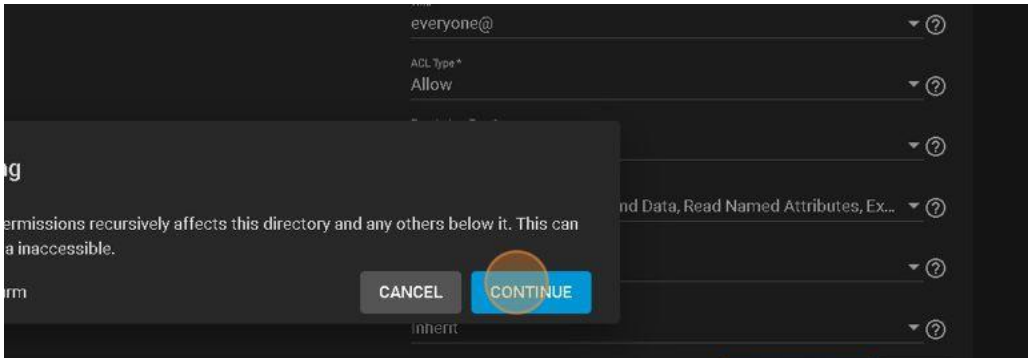


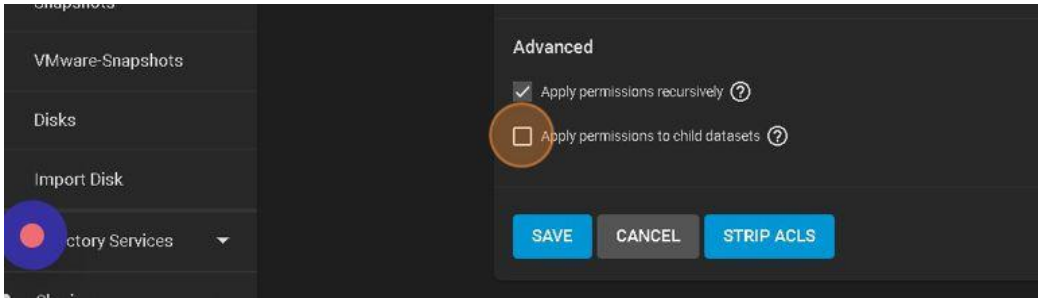


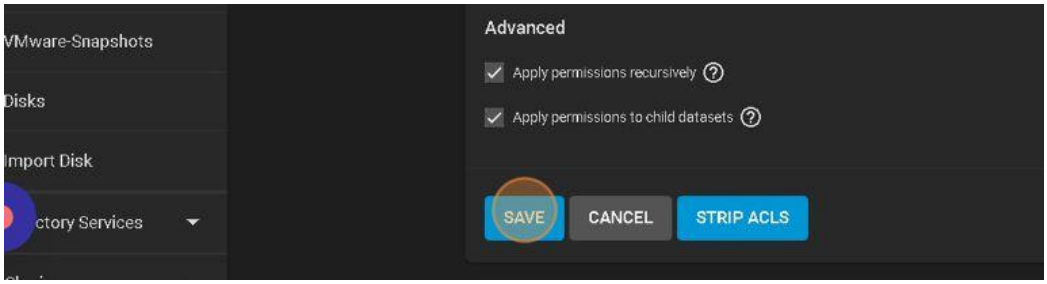












3. Remove NAT gateway At left side Actions Delete
4. EC2 Instance → Elastic IP's → Select all
   → Release Elastic IP address

—---------- TrueNAS End
–—----------—----------

Docker Installation:

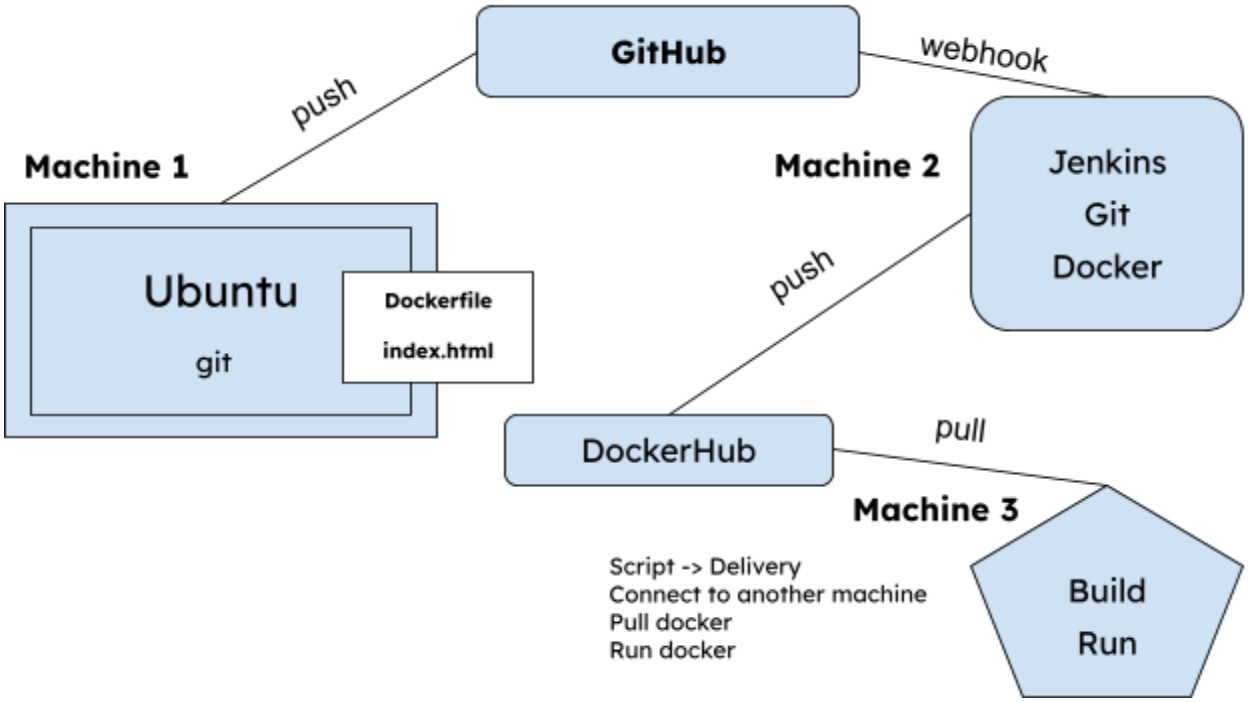
```
sudo apt-get update
sudo apt-get install docker.io

sudo docker --version

sudo docker run hello-world
sudo docker

sudo docker images
sudo docker ps -a

sudo docker pull centos
sudo docker run -it centos

sudo docker rmi -f  hello-world
```

```
sudo docker pull httpd

mkdir web
cd web
nano index.html
       <h1>Welcome To My Webpage</h1>
       <h3>Thank You..!</h3>

nano Dockerfile
      FROM httpd:latest
      COPY index.html /usr/local/apache2/htdocs
      EXPOSE 80

docker build -t httpd .
docker run -d -p 8080:80 httpd
```

—---------- Docker End

–—----------—----------

Local → GitHub → Jenkins on EC2 → DockerHub → EC2:



Local Machine 1:

```
apt-get update
apt-get install git
mkdir techgalena
cd techgalena
root@debian:~/techgalena# ls -l
total 0

nano index.html
     |- <h1>   !..Nginx Via Jenkins Through GitHub & Docker...!</h1>

nano Dockerfile
     |- FROM nginx:latest
     |- COPY index.html /usr/share/nginx/html/

root@debian:~/techgalena# ls -l
total 8
-rw-r--r-- 1 root root 58 Jun 20 06:55 Dockerfile
```

```
-rw-r--r-- 1 root root 61 Jun 20 06:55 index.html
git add .
git commit -m “message”
git push origin master
```

Make a webhook between GitHub & Jenkins

JENKINS, Machine 2:

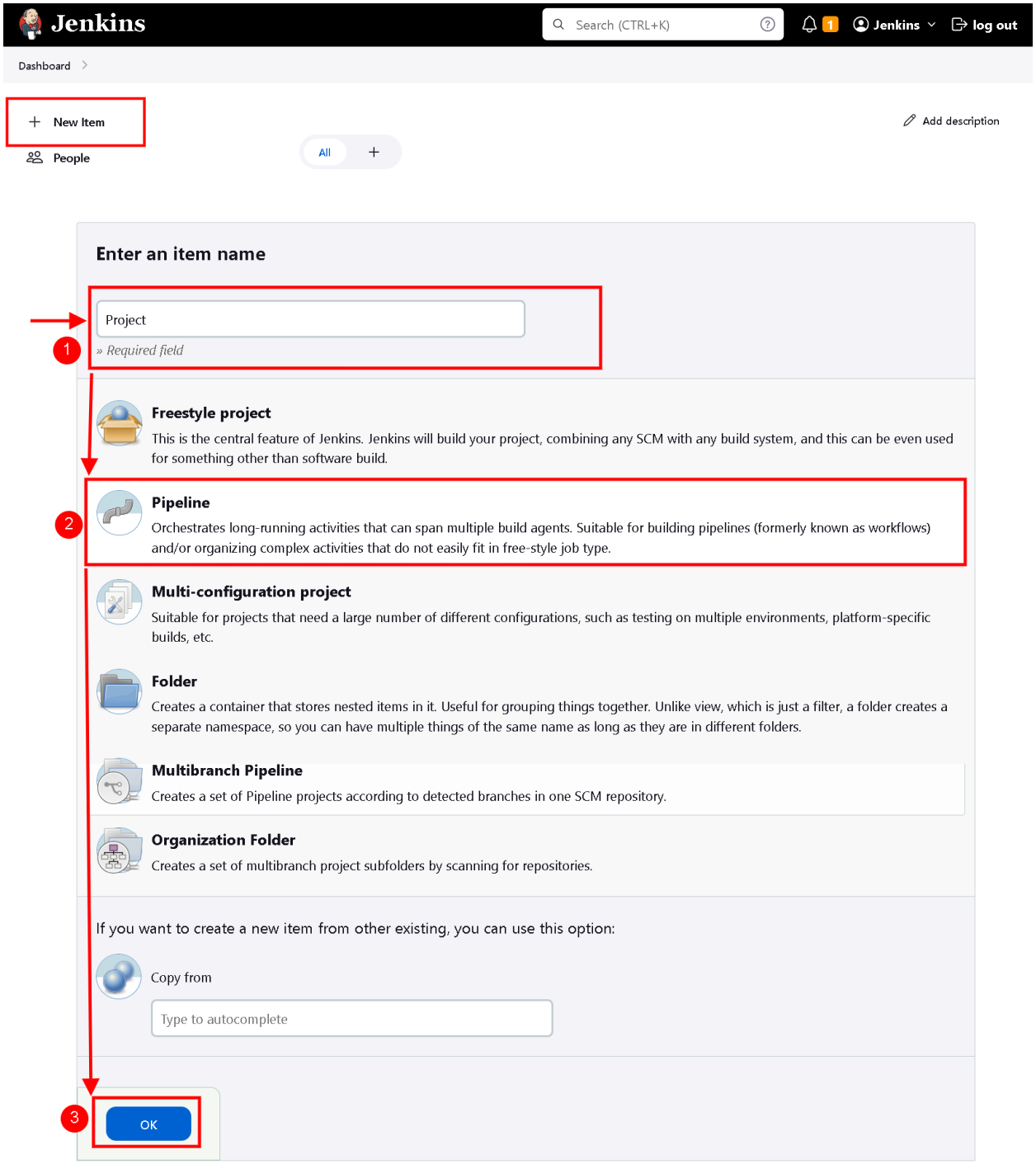
```
apt-get update
apt-get install git
apt-get install docker.io
apt-get install jenkins

ssh-keygen
cat /root/.ssh/id_rsa.pub

// Copy this key and paste it in Machine 3
   at “~/.ssh/authorized_keys” //

sudo usermod -aG docker admin
sudo usermod -aG docker jenkins
sudo chmod 666 /var/run/docker.sock
sudo chown root:docker /var/run/docker.sock
sudo service jenkins restart
sudo service docker restart
```

Machine 3:

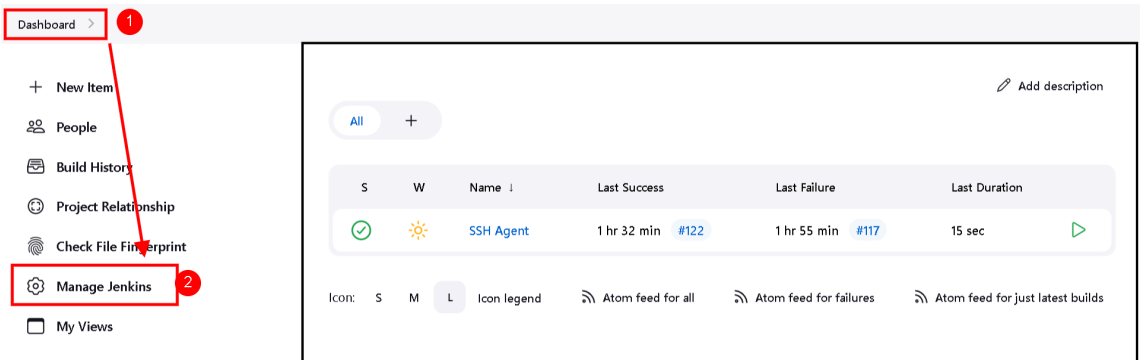
Install: docker.io

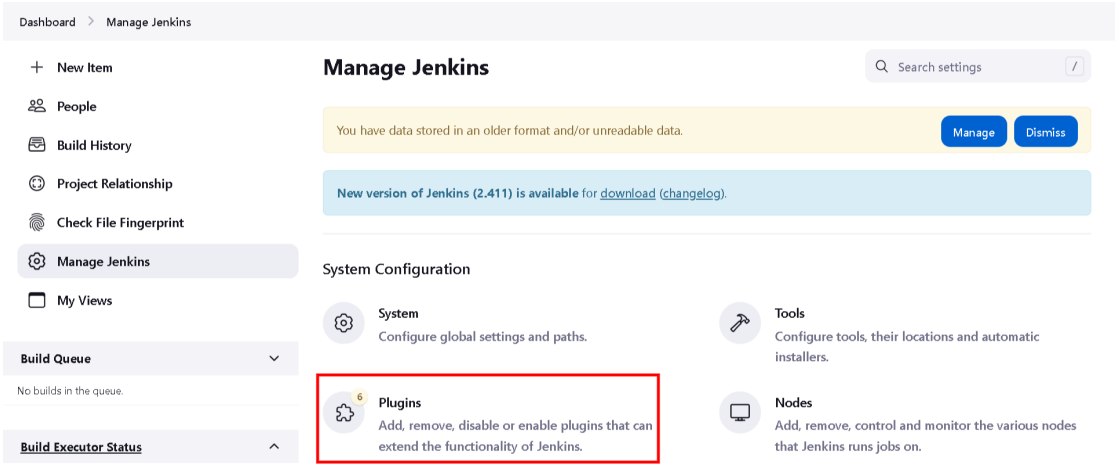
```
sudo chmod 666 /var/run/docker.sock
sudo chown nobody:nogroup /var/run/docker.sock
sudo chmod 700 ~/.ssh
sudo chmod 600 ~/.ssh/authorized_keys

sudo usermod -aG docker admin
sudo usermod -aG docker root
```

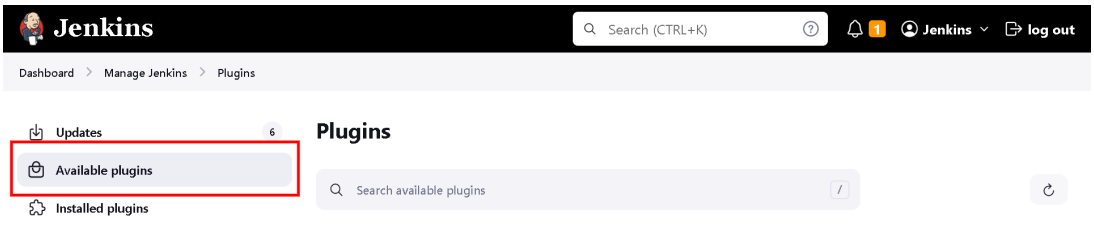
Open Jenkins in Browser on port 8080:

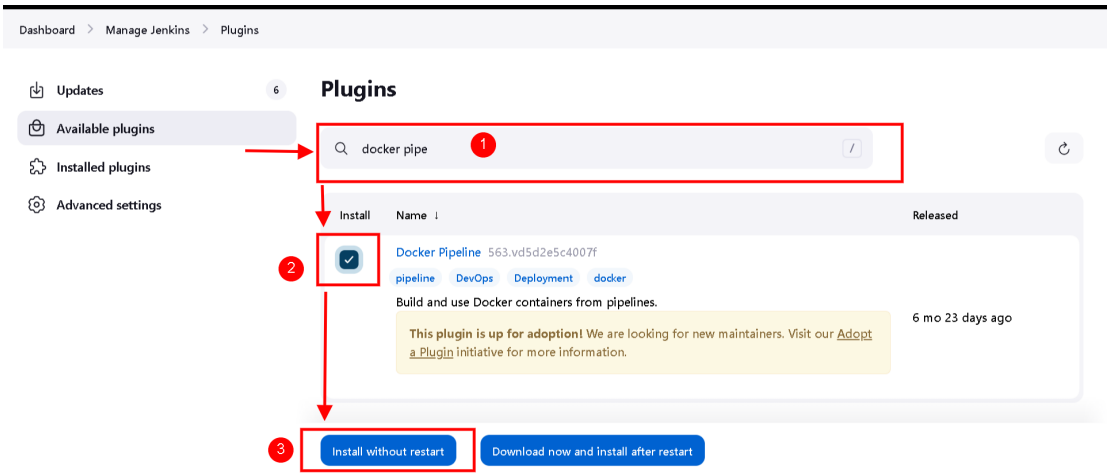


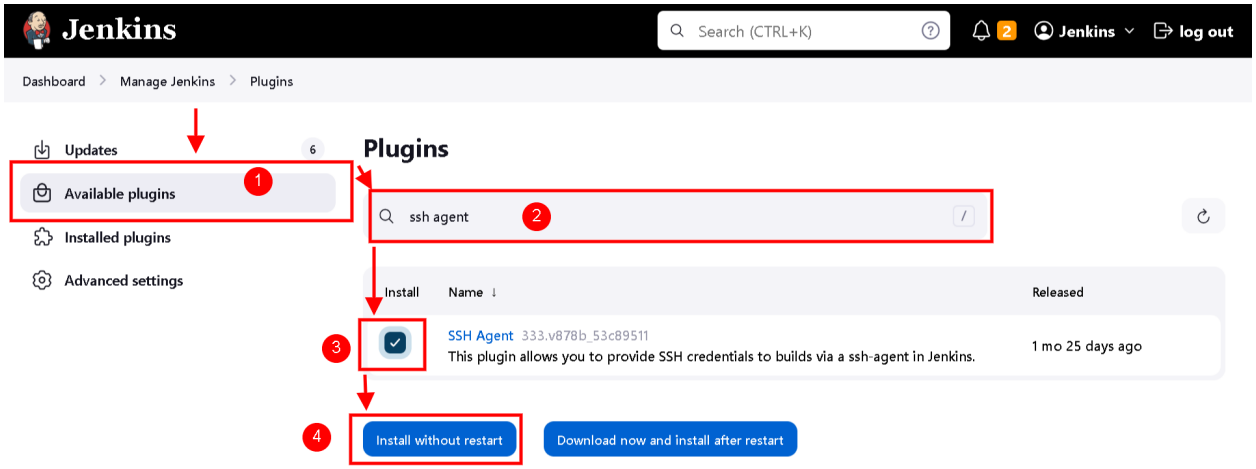
Install 2 Plugins: 1. Docker Pipeline 2. SSH Agent


Dashboard
1
New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
2
My Views
Add description
All
S
W
Name
Last Success
Last Failure
Last Duration
SSH Agent
1 hr 32 min #122
1 hr 55 min #117
15 sec
Icon: S M L Icon legend
Atom feed for all
Atom feed for failures
Atom feed for just latest builds

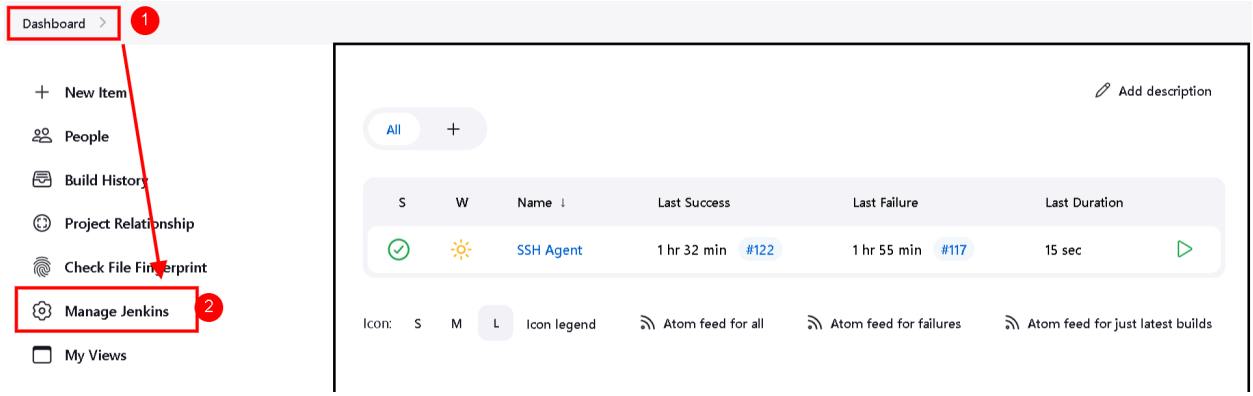

Dashboard > Manage Jenkins
New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Build Queue
No builds in the queue.
Build Executor Status
Manage Jenkins
Search settings
You have data stored in an older format and/or unreadable data.
Manage
Dismiss
New version of Jenkins (2.411) is available for download (changelog).
System Configuration
System
Configure global settings and paths.
Tools
Configure tools, their locations and automatic installers.
Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
Nodes
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

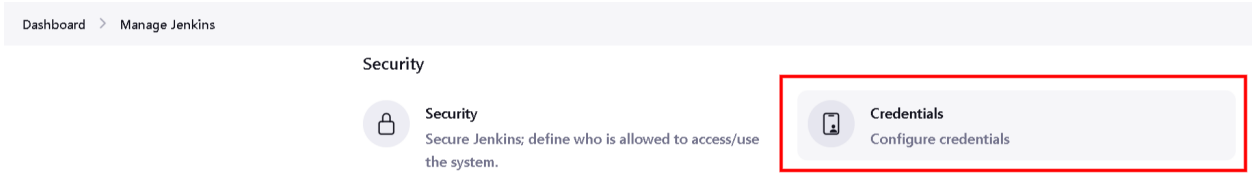

Jenkins
Search (CTRL+K)
Jenkins
log out
Dashboard > Manage Jenkins > Plugins
Updates
Available plugins
Installed plugins
Plugins
Search available plugins

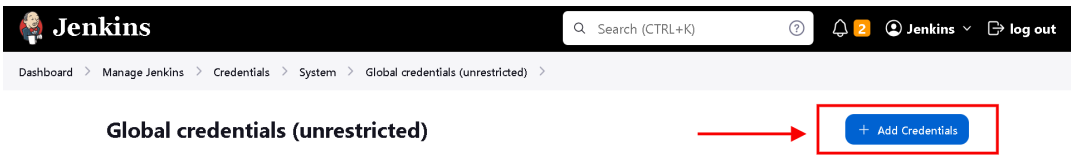

Dashboard > Manage Jenkins > Plugins
Updates
Available plugins
Installed plugins
Advanced settings
Plugins
docker pipe
1
Install
Name
Released
2
Docker Pipeline 563.vd5d2e5c4007f
pipeline
DevOps
Deployment
docker
Build and use Docker containers from pipelines.
This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.
6 mo 23 days ago
3
Install without restart
Download now and install after restart



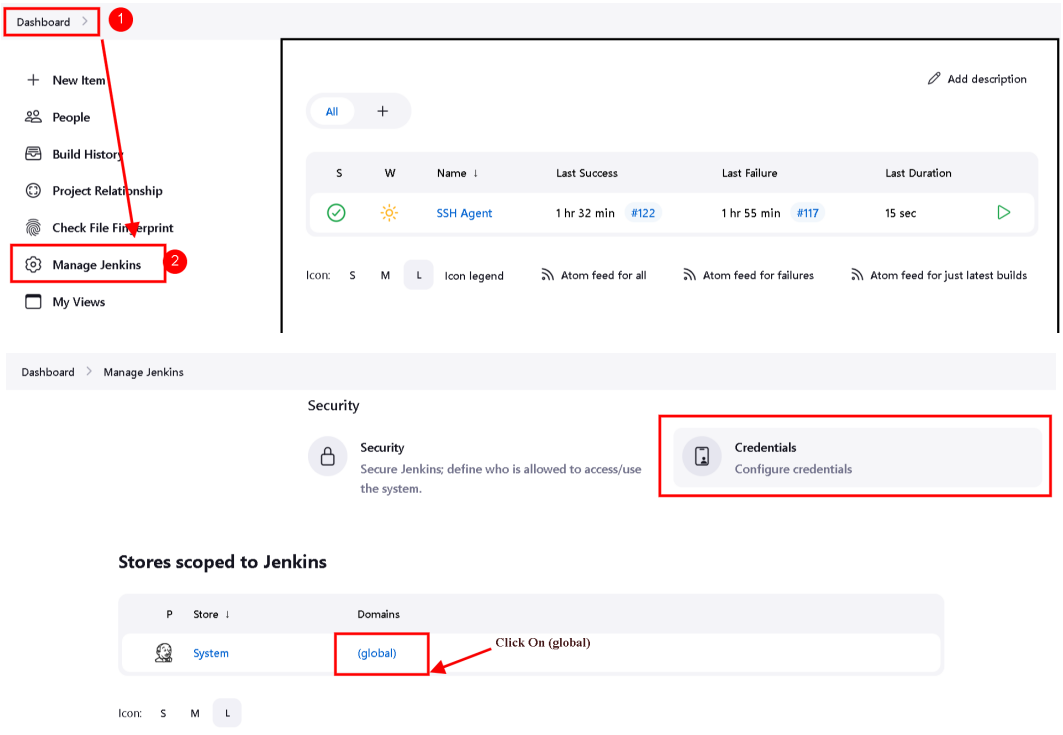
Configure Credentials for: Docker Login

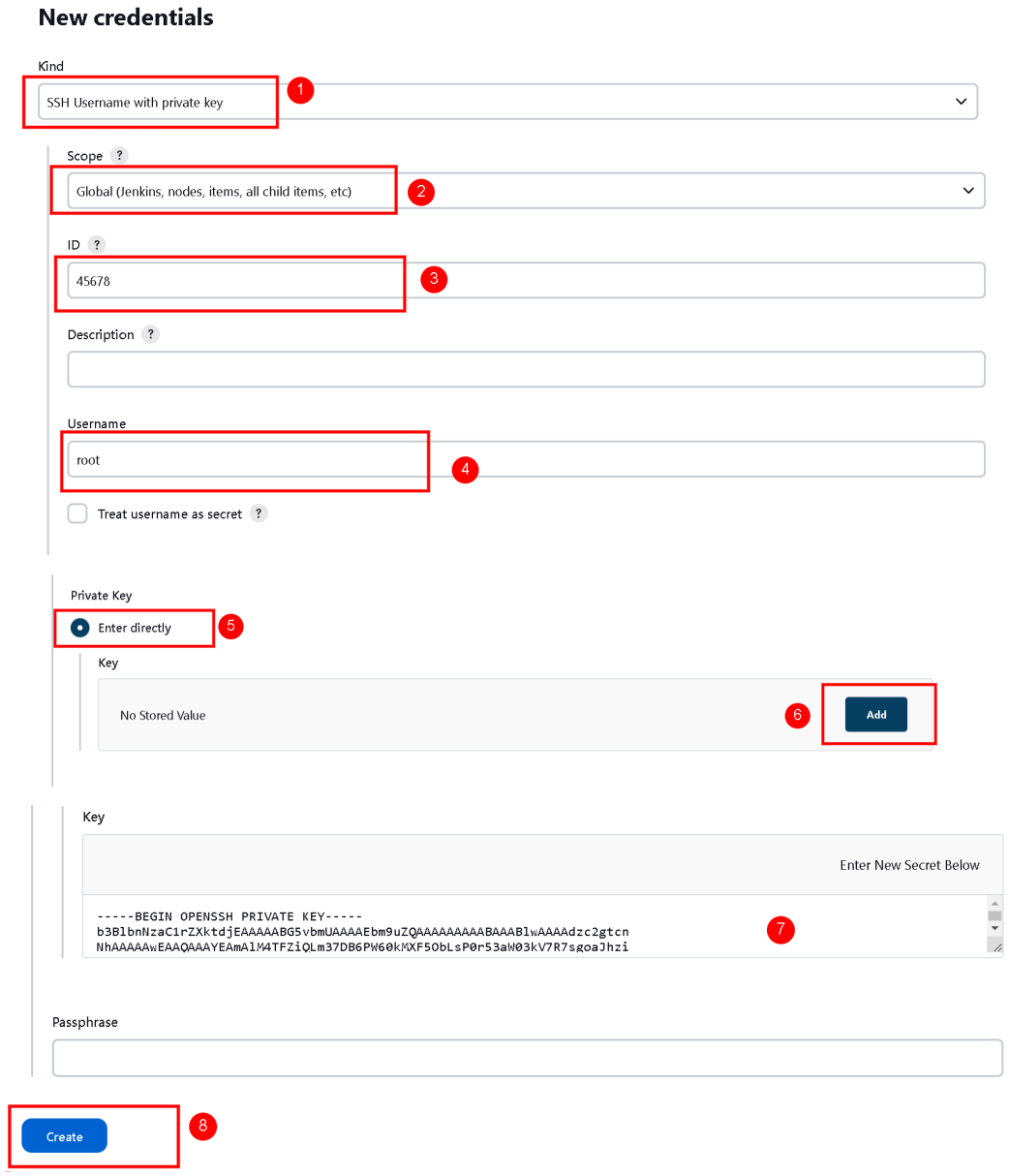






Configure Credentials for: SSH




New credentials
Kind
SSH Username with private key
1
Scope
Global (Jenkins, nodes, items, all child items, etc)
2
ID
45678
3
Description
Username
root
4
Treat username as secret
Private Key
Enter directly
5
Key
No Stored Value
6
Add
Key
Enter New Secret Below
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAmAlM4TFZiQLm37DB6PW60kMXF5ObLsP0r53aW03kV7RZsgoaJhzi
7
Passphrase
Create
8

General

Enabled

Description

nginx project

Plain text Preview

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [x] GitHub project

Project url

https://github.com/kmwaseem15/techgalena.git/

Advanced

- [ ] Pipeline speed/durability override
- [ ] Preserve stashes from completed builds
- [ ] This project is parameterized
- [ ] Throttle builds

Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [x] GitHub hook trigger for GITScm polling
- [ ] Poll SCM

Pipeline

Definition

Pipeline script

Script

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                echo "Performing Checkout"
                checkout([$class: 'GitSCM',
                          branches: [[name: 'master']],
                          userRemoteConfigs: [[url: 'https://github.com/kmwaseem15/techgalena.git']]])
            }
        }

        stage('Build Docker Image') {
            steps {
                echo "Building Docker Image"
```

- [x] Use Groovy Sandbox

Pipeline Syntax

Save Apply

JENKINS Pipeline Stages:

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                echo "Performing Checkout"
                checkout([$class: 'GitSCM',
                          branches: [[name: 'master']],
                          userRemoteConfigs: [[url:
'https://github.com/onkgithub/iacsd.git']]])
            }
        }

        stage('Build and Push to Docker Hub') {
            steps {
                script {
                    // Build Docker image
                    sh 'docker build -t 9766791539/nginx .'

                    // Log in to Docker Hub using credentials
 // Here credentialsID: ‘ID You Gave During Docker Login Configuration (992040)’
                    withDockerRegistry(credentialsId: '992010') {
                        // Push Docker image to Docker Hub
                        sh 'docker push 9766791539/nginx'
                    }
                }
            }
        }

        stage('Pull on Remote Machine') {
            steps {
                sshagent(['992040']) {
                    // SSH into the remote machine and pull the Docker
image
 // Here credentialsID: ‘ID You Gave SSH Configuration (45678)’
                    sh 'ssh admin@54.147.9.237 "docker pull
9766791539/nginx"'
                }
            }
```

```
        }
        stage('Kill Running Containers') {
                    steps {
// Here sshagent(['ID You Gave During SSH Configuration (45678)'])
                        sshagent(['992040']) {
                            // SSH into the remote machine and kill all
running containers
                            sh 'ssh admin@54.147.9.237 "docker kill \$(ssh
admin@54.147.9.237 docker ps -q)"'
                        }
                    }
                }

        stage('Build and Run on Remote Machine') {
            steps {
// Here sshagent(['ID You Gave During SSH Configuration (45678)'])
                sshagent(['992040']) {
                    // Copy the Dockerfile to the remote machine
                    sh 'ssh admin@54.147.9.237 "cd /home/admin/docker"'
                    sh 'scp /var/lib/jenkins/workspace/docker/Dockerfile
admin@54.147.9.237:/home/admin/docker'
                    sh 'scp /var/lib/jenkins/workspace/docker/index.html
admin@54.147.9.237:/home/admin/docker'
                    // SSH into the remote machine and build/run the Docker
image/container
                    sh 'ssh admin@54.147.9.237 "cd /home/admin/docker &&
docker build -t myapp . && docker run -d -p 8080:80 myapp"'
                }
            }
        }
    }
}
```

After setting up all the machines: Got to Machine 1:
Push all files for first time.

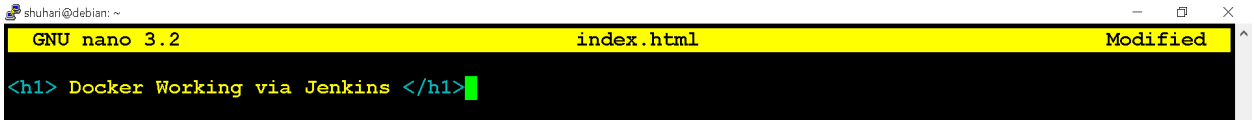
```
git add .
git commit -m "message"
git push origin master
```

From Second times onward you will only have to push index.html only.

```
git add index.html
git commit -m "message"
git push origin master
```

After Doing Above Things on Machine 1
Your index.html content should be vibile on Machine 3's IP in browser on port 8080

Input:



Output:

Docker Working via Jenkins

—---------- JENKINS CICD, GitHub, DockerHub End
–—----------

Docker Swarm :

Docker Swarm is a container orchestration tool for managing multiple containers across multiple hosts. Its key benefit is high scalability and fault tolerance. It enables automatic load balancing, service discovery, and self-healing capabilities, ensuring reliable and efficient deployment of containerized applications. With Swarm, users can easily scale their applications up or down based on demand, providing flexibility and resilience in containerized environments.

```
docker swarm init
docker swarm join --token //Run the command in node machines
docker node ls
docker service create –name cdac -p 80:80 –replicas 3 iacsd
docker service
```

```
docker service rm cdac
```

Services Of AWS:

➔ **EC2:**

Amazon Elastic Compute Cloud (Amazon EC2) is a web service provided by Amazon Web Services (AWS) that allows you to create and manage virtual servers in the cloud. EC2 provides scalable computing capacity, enabling you to quickly provision instances and scale them up or down based on your requirements.

Here are some key features and concepts related to EC2:

1. **Instances:** EC2 instances are virtual servers in the cloud. You can choose from a variety of instance types based on your workload requirements, such as compute-optimized, memory-optimized, storage-optimized, GPU instances, etc.
2. **Regions and Availability Zones:** EC2 is available in multiple geographic regions worldwide. Each region consists of multiple isolated locations called Availability Zones. You can choose the region and availability zone(s) where you want to deploy your EC2 instances.
3. **Amazon Machine Images (AMIs):** An AMI is a pre-configured template that contains the necessary software, operating system, and configuration settings to launch an EC2 instance. You can select a public AMI provided by AWS or create your own custom AMI.
4. **Instance Types:** EC2 provides various instance types optimized for different use cases. These instance types differ in terms of CPU, memory, storage, and networking capacity. You can choose the most suitable instance type based on your workload requirements.
5. **Security Groups:** Security groups act as virtual firewalls for your EC2 instances. They control inbound and outbound traffic by specifying rules that allow or deny traffic based on protocols, ports, and IP addresses.
6. **Elastic IP Addresses:** An Elastic IP address is a static, public IPv4 address that you can allocate and associate with your EC2 instances. It allows you to retain the same IP address even if you stop or terminate the instance.

7. **Elastic Block Store (EBS):** EBS provides persistent block-level storage volumes for EC2 instances. You can attach EBS volumes to your instances and use them as primary storage or additional storage for your data.
8. **Auto Scaling:** Auto Scaling allows you to automatically adjust the number of EC2 instances based on application demand. It helps you maintain performance, availability, and cost optimization by dynamically scaling your instances.
9. **Load Balancing:** EC2 offers load balancing services such as Elastic Load Balancing (ELB) to distribute incoming traffic across multiple EC2 instances. This enhances fault tolerance and ensures high availability for your applications.
10. **Integration with Other AWS Services:** EC2 integrates with various other AWS services, such as Amazon S3 for object storage, Amazon RDS for managed databases, Amazon VPC for networking, AWS CloudFormation for infrastructure as code, etc.

→ AWS S3:

AWS S3 or Amazon Simple Storage Service is a highly scalable and durable cloud storage service provided by Amazon Web Services (AWS). It allows you to store and retrieve large amounts of data, such as files, documents, images, videos, and backups, over the internet. S3 is designed to provide 99.999999999% (11 nines) durability for objects and 99.99% availability.

Key Features of AWS S3:

1. **Storage Classes:** S3 offers multiple storage classes optimized for different use cases, such as Standard, Intelligent-Tiering, Glacier, and Glacier Deep Archive. Each storage class has its own pricing and retrieval characteristics, allowing you to choose the appropriate class based on your data access requirements and cost considerations.
2. **Scalability and Durability:** S3 is designed to scale to accommodate virtually unlimited amounts of data. It automatically replicates data across multiple availability zones within a region to ensure durability. This redundancy provides high data durability and protects against hardware failures.

3. **Security and Access Control:** S3 allows you to manage access to your data through various mechanisms. You can use AWS Identity and Access Management (IAM) to control user permissions, create access policies, and manage encryption keys. S3 also supports server-side encryption for data at rest.
4. **Versioning and Lifecycle Policies:** S3 provides versioning capabilities, allowing you to preserve, retrieve, and restore every version of every object in your bucket. Lifecycle policies enable you to define rules to automatically transition or expire objects based on their age, which helps optimize storage costs.
5. **Data Transfer and Transfer Acceleration:** AWS offers multiple options to transfer data into and out of S3. You can use direct uploads, AWS Snowball devices for large-scale offline data transfer, or AWS DataSync for automated data transfer. Additionally, S3 Transfer Acceleration uses CloudFront's global network to speed up data transfers.
6. **Event Notifications and Logging:** S3 can generate event notifications to trigger actions or workflows when certain events occur, such as object creation or deletion. You can configure event notifications to integrate with other AWS services like AWS Lambda, Amazon SNS, or Amazon SQS. S3 also provides server access logs for auditing and troubleshooting.
7. **Integration with AWS Ecosystem:** S3 integrates with various AWS services, making it a foundation for many cloud-based solutions. For example, you can use it with AWS Lambda for serverless computing, Amazon Athena for querying data using SQL, or Amazon Redshift for data warehousing.

AWS S3 is widely used by individuals and organizations for various purposes, including website hosting, data backups, media storage and distribution, big data analytics, content delivery, and archival storage. Its flexibility, scalability, and robustness make it a popular choice for managing and storing data in the cloud.

➔ **AWS Lambda:**

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). It allows you to run your code without provisioning or managing servers. With Lambda, you can execute your code in

response to events such as changes to data in an Amazon S3 bucket, updates to a DynamoDB table, or even HTTP requests.

Key Features of AWS Lambda:

1. **Serverless Execution:** Lambda abstracts the underlying infrastructure, allowing you to focus on writing and deploying code without worrying about server management.

2. **Event-Driven Model:** Lambda functions are triggered by events such as changes to data, time-based schedules, or custom events generated by other AWS services.

3. **Scalability:** AWS Lambda automatically scales your code in response to incoming requests. It can handle a few requests per day to thousands of requests per second, based on demand.

4. **Pay-per-Use Pricing:** With Lambda, you only pay for the compute time consumed by your code. Billing is based on the number of requests and the time it takes to execute your code.

5. **Language Support:** AWS Lambda supports multiple programming languages, including JavaScript (Node.js), Python, Java, C#, PowerShell, and Ruby. You can choose the language that best suits your needs.

6. **Integration with AWS Services:** Lambda can integrate with other AWS services, such as Amazon S3, DynamoDB, API Gateway, SNS, SQS, and more. This allows you to build serverless architectures and microservices easily.

7. **Versioning and Aliases:** Lambda allows you to version your functions, enabling you to deploy updates without impacting the existing production environment. Aliases let you point to specific versions of your function for testing or canary deployments.

8. **Monitoring and Logging:** AWS provides CloudWatch for monitoring and logging Lambda functions. You can view metrics, set up alarms, and monitor the performance and resource usage of your functions.

Lambda is widely used for various use cases, such as real-time file processing, data transformations, web and mobile backends, IoT data processing, chatbots, and more. It offers a highly scalable and cost-effective way to build serverless applications.

Docker Hub:

Docker Hub is a cloud-based registry service provided by Docker, the popular containerization platform. It serves as a central repository for Docker images, allowing developers to store, manage, and distribute container images. Docker Hub offers both free and paid plans, providing a platform for sharing containerized applications and collaborating with other developers.

What is Docker?

Docker is an Open Platform for developing, shipping and running the applications.

Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage applications.

By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Key features and functionalities of Docker Hub include:

1. **Image Hosting:** Docker Hub allows users to store and host Docker images securely in the cloud. It provides storage space for public and private repositories, making it easy to share and distribute container images.

2. **Versioning and Tagging:** Docker images can have multiple versions and tags associated with them. Docker Hub allows developers to tag their images with meaningful labels, making it easier to manage different versions and variations of container images.

3. **Repository Organization:** Docker Hub provides a repository-centric organization, allowing users to group related images together within repositories. This helps to maintain a structured and organized collection of container images.

4. **Collaboration and Sharing:** Docker Hub allows developers to share their Docker images with others by making them public or sharing them privately with specific users or teams. This enables collaboration and simplifies the process of sharing containerized applications.

5. **Automated Builds:** Docker Hub supports automated builds, which means it can automatically build Docker images based on the source code or Dockerfile

provided by the user. This feature is useful for continuous integration and deployment workflows.

6. **Webhooks and Triggers:** Docker Hub provides webhooks and triggers that can be used to trigger actions or notifications based on events like image pushes, new tags, or repository updates. This allows for integration with other systems and automation of workflows.

7. **Integration with Docker CLI:** Docker Hub seamlessly integrates with the Docker command-line interface (CLI), making it easy to push and pull images to and from Docker Hub using familiar Docker commands.

8. **Official Images and Certification:** Docker Hub hosts a collection of official images maintained by Docker and its partners. These images are carefully curated, ensuring they are secure, up-to-date, and well-documented. Docker Hub also offers certification programs for images, indicating they meet certain quality standards and best practices.

Overall, Docker Hub plays a crucial role in the Docker ecosystem by providing a centralized platform for storing, sharing, and distributing Docker images. It simplifies the process of managing containerized applications and enables collaboration among developers and teams.

Docker Platform:

Docker provides the ability to package and run an application in a loosely isolated environmnet called container. The isolation and security allows you to run many containers simultaneously on a given host. Container are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easi;y share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

Develop your application and its supporting components using containers. The container becomes the unit for distributing and testing your application.
When you're ready, deploy your application into your production environment.

Commands in Docker:

1. **__docker run__**: Create and start a new container from an image.

```
docker run [options] [image] [command]
```

2. **docker build**: Build a new Docker image from a Dockerfile.

```
docker build [options] [path]
```

3. **docker pull**: Download an image from a registry.

```
docker pull [image]
```

4. **docker push**: Upload an image to a registry.

```
docker push [image]
```

5. **docker images**: List available Docker images on the host.

```
docker images [options]
```

6. **docker ps**: List running containers.

```
docker ps [options]
```

7. **docker start**: Start a stopped container.

```
docker start [container]
```

8. **docker stop**: Stop a running container.

```
docker rmi [image]
```

9. **docker rm**: Remove one or more containers.

```
docker rm [container]
```

10. **docker rmi**: Remove one or more images.

```
docker rmi [image]
```

11. **docker exec**: Run a command inside a running container.

```
docker exec [options] [container] [command]
```

12. **docker logs**: Fetch the logs of a container.

```
docker logs [options] [container]
```

13. ****docker inspect****: Display detailed information about a container or image.

```
docker inspect [options] [object]
```

14. ****docker network****: Manage Docker networks.

```
docker network [options]
```

15. ****docker-compose****: Manage multi-container Docker applications with a YAML file.

```
docker-compose [options] [command]
```

These are just a few examples of the many Docker commands available. You can find more detailed information about these commands and additional ones in the Docker documentation (https://docs.docker.com/).

GitHub:

GitHub is a web-based platform for version control and collaboration that allows developers to store and manage their code repositories. It provides a hosting service for Git repositories, which is a distributed version control system used by many developers worldwide.

Here are some key features and functionalities of GitHub:

1. **Version Control:** GitHub uses Git, a widely used version control system, to track changes in code repositories. It allows developers to create branches, make changes, and merge them back into the main codebase.
2. **Code Collaboration:** GitHub enables developers to work collaboratively on projects. Multiple developers can contribute to a project by creating branches, submitting pull requests, and reviewing each other's code.
3. **Issue Tracking:** GitHub provides a built-in issue tracking system that allows users to create and manage issues, bug reports, feature requests, and other tasks related to a project. This helps teams to organize and prioritize their work effectively.

4. **Project Management:** GitHub offers project management features like project boards, milestones, and labels to help teams track the progress of their work and manage tasks efficiently.
5. **Continuous Integration and Deployment:** GitHub integrates with various continuous integration and deployment tools, such as Travis CI, Jenkins, and GitHub Actions. These integrations enable developers to automate testing, build processes, and deployment pipelines.
6. **Community and Social Features:** GitHub fosters a vibrant developer community by providing features like social interactions, such as following other developers, starring repositories, and contributing to open-source projects.
7. **Documentation and Wikis:** GitHub allows developers to create documentation and wikis for their projects, making it easier to share knowledge and information with others.
8. **Integration with Other Tools:** GitHub integrates with a wide range of development tools and services, such as IDEs, code editors, project management tools, and chat platforms, enhancing the overall development workflow.

GitHub is widely used by individual developers, open-source projects, and organizations of all sizes. It provides a platform for collaboration, code sharing, and community building, making it an essential tool in the software development ecosystem.

Git:

Git is a distributed version control system (VCS) widely used for managing source code and project files. It was created by Linus Torvalds in 2005 to help with the development of the Linux kernel, but it has since become one of the most popular version control systems for software development.

Git allows multiple developers to work on the same project simultaneously and independently, and it tracks changes made to files over time. It provides features such as branching, merging, and version history tracking, which are essential for collaborative software development.

Here are some key concepts and commands related to Git:

1. **Repository:** A repository, or repo, is a collection of files and their revision history. It can be either a local repository on your computer or a remote repository hosted on a server.

2. **Clone:** To clone a repository means to create a local copy of a remote repository on your computer. The command to clone a repository is:

```
git clone <repository-url>
```

3. **Commit:** A commit represents a snapshot of changes made to the files in a repository. It records the changes and a descriptive message. The command to commit changes is:

```
git branch <branch-name>
```

4. **Branch:** A branch is a parallel version of a repository that allows developers to work on different features or bug fixes independently. The main branch is typically called "master" or "main," and additional branches can be created and merged later. To create a new branch, use:

```
git branch <branch-name>
```

5. **Checkout:** To switch between branches or to restore files to a previous commit, you can use the checkout command. For example, to switch to a different branch, use:

```
git checkout <branch-name>
```

6. **Merge:** Merging combines changes from different branches into one branch. This is typically done when a feature or bug fix is complete. To merge a branch into the current branch, use:

```
git push origin <branch-name>
```

7. **Push:** Pushing refers to sending local commits to a remote repository, making them accessible to other team members. The command to push changes is:

```
   git push origin <branch-name>
```

8. **Pull:** Pulling is used to update the local repository with the latest changes from a remote repository. The command to pull changes is:

```
   git pull origin <branch-name>
```

9. **Status:** Shows the current state of the repository, including modified files, untracked files, and the current branch.

```
   git status
```

10. **Log:** Displays the commit history, including the commit hash, author, date, and commit message.

```
   git log
```

11. **Diff:** Shows the differences between the current changes and the last commit or between different branches.

```
git diff
```

12. **Remote:** Manages remote repositories. It allows you to add, remove, and list remote repositories.

```
git remote add <name> <url>
   git remote remove <name>
   git remote -v
```

13. **Fetch:** Retrieves changes from a remote repository but does not merge them into the local branch.

```
git remote add <name> <url>
git remote remove <name>
git remote -v
```

14. **Reset:** Discards changes in the working directory and staging area to a specific commit. It can be used to undo commits.

```
git revert <commit>
```

15. **Revert:** Creates a new commit that undoes the changes made in a previous commit.

```
git revert <commit>
```

16. **Tag:** Marks a specific commit with a tag to give it a meaningful name, often used to indicate versions or releases.

```
git tag <tag-name>
```

17. **Stash:** Temporarily saves changes that are not ready to be committed, allowing you to switch branches or apply the changes later.

```
git stash
git stash apply
git stash pop
```

18. **Remote Branches:** Working with branches in a remote repository.

```
  ```
    git push origin --delete <branch-name> # Deletes a remote branch
    git checkout --track origin/<branch-name> # Creates a local
branch that tracks a remote branch
    ```
```

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                echo "Performing Checkout"
                checkout([$class: 'GitSCM',
                          branches: [[name: 'master']],
                          userRemoteConfigs: [[url:
'https://github.com/kmwaseem15/techgalena.git']]])
            }
        }

        stage('Build Docker Image') {
            steps {
                echo "Building Docker Image"
                sh 'docker build -t kmwaseem15/nginx .'
            }
        }

        stage('Push to DockerHub') {
            steps {
                echo "Pushing to DockerHub"
                sh 'docker push kmwaseem15/nginx'
            }
        }

        stage('Pull on EC2') {
            steps {
                echo "Pulling on EC2"
                sshagent(['992030']) {
                    sh "ssh -o StrictHostKeyChecking=no -l root
18.207.244.248 'docker pull kmwaseem15/nginx'"
```

```
                }
            }
        }

        stage('Run Docker Container on EC2') {
            steps {
                echo "Running Docker Container on EC2"
                sshagent(['992030']) {
                    script {
                        // Kill all running Docker containers
                        def killCommand = "ssh -o
StrictHostKeyChecking=no -l root 18.207.244.248 'docker kill
\$(docker ps -q)'"
                        sh(killCommand)

                        // Run the Docker container
                        def runCommand = "ssh -o
StrictHostKeyChecking=no -l root 18.207.244.248 'docker run -d -p
8080:80 kmwaseem15/nginx'"
                        try {
                            sh(runCommand)
                        } catch (Exception e) {
                            echo "Failed to start the container:
${e.getMessage()}"
                            error "Failed to start the container"
                        }
                    }
                }
            }
        }
    }
}
```