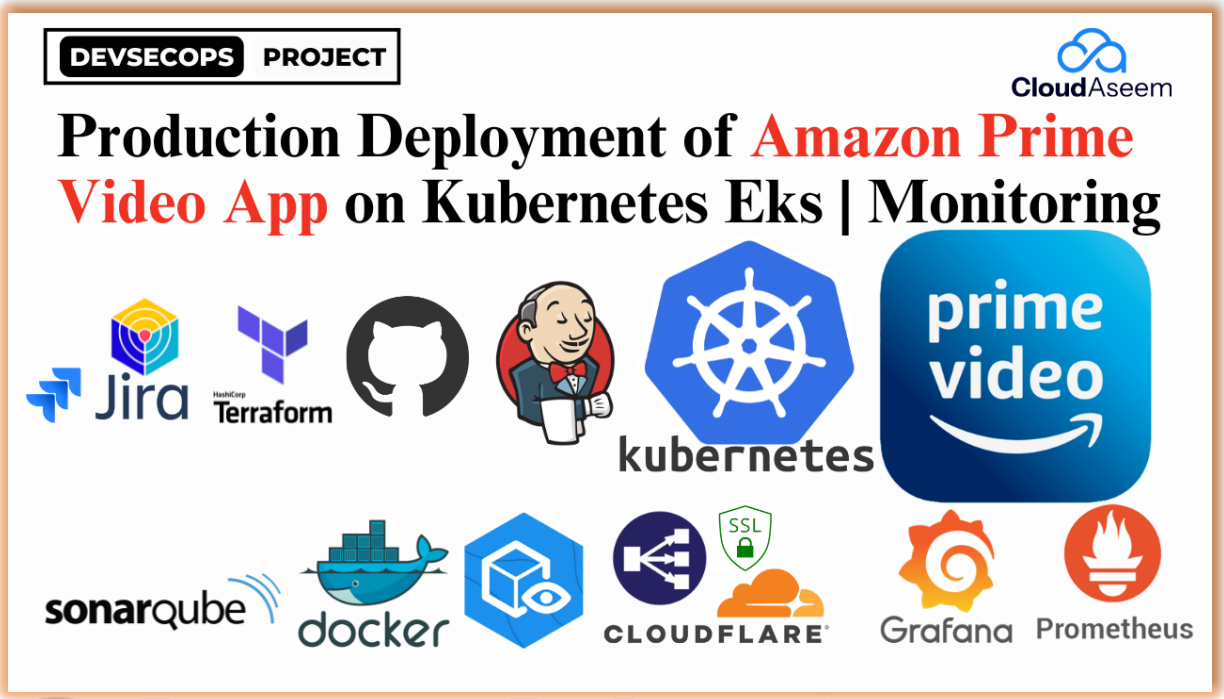
Amazon-prime-video App — End-to-End Secure & Scalable Deployment with DevSecOps & Jenkins !

Thrilled to showcase our latest project — **Amazon-prime-video App** 🔐 **End-to-End Secure &** ⚖️ **Scalable Deployment with DevSecOps** with a **DevSecOps CI/CD pipeline** and powered by **Jenkins** Parameterise Build to deliver fully automated, secure, and scalable deployments.



🌍 **Follow me for projects Links :**
🔗 **YouTube:** https://youtu.be/1FNhSda0sV4 - @clouddevopswithaseem
🔗 **Project GitHub Repo:** Aseemakram19/starbucks-kubernetes
🔗 **LinkedIn:** Mohammed Aseem Akram
🔗 **GitHub Account:** Aseemakram19

📝 Project Agenda

🔳 Infrastructure Setup

- 🧰 **Jenkins Server — For CI/CD pipeline automation**
- 🛡️ **SonarQube Server — For code quality and security checks**
- 📈 **Monitoring Server — For system observability and alerting**

2️⃣ Core Components & Toolchain

- **🐳 Docker & ☸️ Kubernetes**
  → Containerized, scalable application deployment with orchestration
- **🚀 Jenkins CI/CD Pipelines**
  → Reusable, standardized, and consistent deployment pipelines
- **🔍 SonarQube & 🧯 Trivy**
  → Code quality analysis & vulnerability scanning for secure releases
- **📊 Prometheus & 📉 Grafana**
  → Real-time metrics monitoring and visual dashboards
- **📧 Gmail Email Alerts**
  → Collaboration-driven notifications and incident alerts
- **🎛️ Parameterized Environment Orchestration**
  → On-demand infrastructure setup and teardown (Dev/Test/Prod)

Port	Protocol	Description
22	TCP	SSH (for remote access)
80	TCP	HTTP (Web traffic)
443	TCP	HTTPS (Secure web traffic)
8080	TCP	Web applications (Tomcat, etc.)
587	TCP	SMTP (Email sending)
465	TCP	SMTPS over SSL
3000	TCP	Web apps (Grafana, Node.js, etc.)
9000	TCP	SonarQube/Web apps

We are going to cover this 3 responsibilities in this project :

👷 **Roles & Responsibilities**

⚙️ **Platform Engineer**

- 🧱 Designs and builds reusable infrastructure as code (IaC)
- 🔌 Integrates CI/CD tools, secrets management, and container registries
- 🧩 Enables self-service environments and developer portals
- 🔐 Focuses on security, compliance, and platform governance

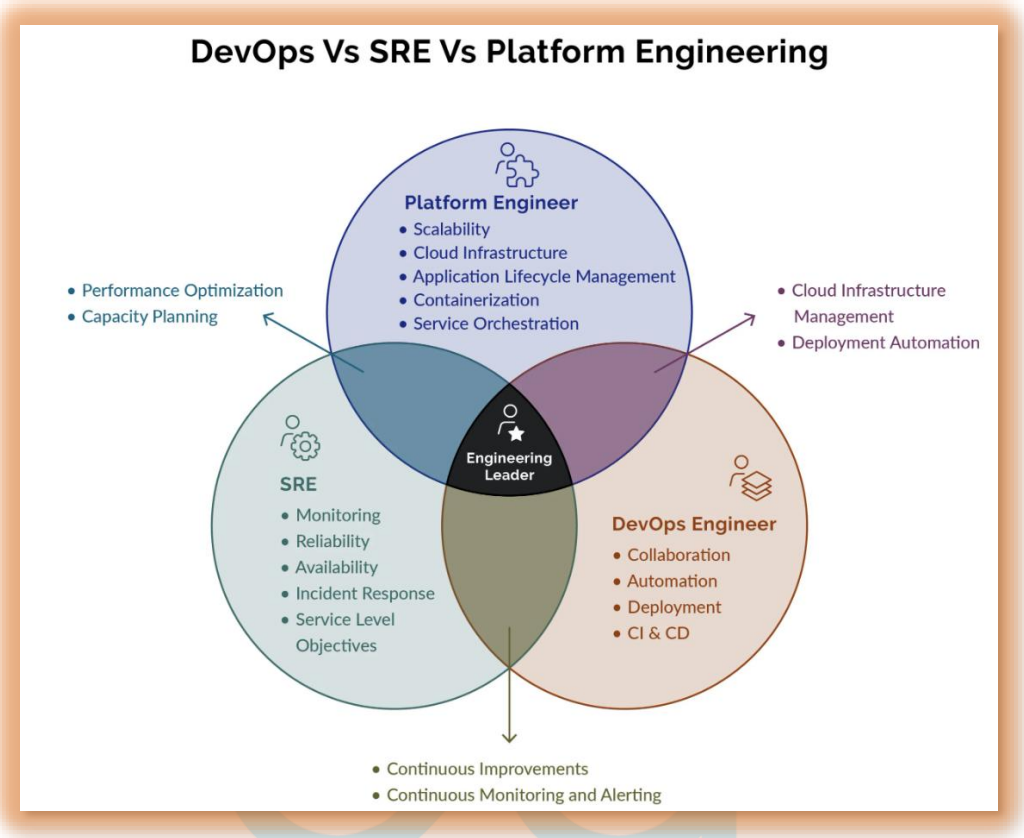
👩‍🚒 **SRE (Site Reliability Engineer)**

- 📉 Ensures system reliability, availability, and performance
- 🔁 Implements monitoring, alerting, and incident response workflows
- 📦 Automates failover, backup, and recovery strategies
- 📈 Defines SLOs/SLIs and tracks error budgets for services

👨‍💻 **DevOps Engineer**

- 🛠️ Builds and maintains CI/CD pipelines with Jenkins
- 🚀 Automates testing, security scans, and artifact deployment

- ☁️ Works on infrastructure provisioning and environment management
- 🔄 Bridges development and operations for faster delivery cycles



-

🧱 Application Software Architecture

🏛️ Monolithic Architecture

In this project, the **Amazon-prime-video App** is designed using a **Monolithic Architecture**.

🧩 What is Monolithic Architecture?

A **monolithic application** is built as a single, unified codebase that handles multiple functionalities — from UI to business logic to data access — all bundled and deployed together as one unit.

🔍 Key Characteristics

- 🧠 **Single Codebase** — All features and components are part of one project
- 🚀 **Unified Deployment** — Deployed as one artifact (e.g., a JAR, WAR, or Docker container)
- 🛠️ **Tightly Coupled Components** — Modules interact through direct function calls
- 📦 **Shared Resources** — Uses shared memory/database, often simplifying data management

---

✅ Advantages

- ⚡ **Faster Development (initially)** — Simple to build and get started
- 🔧 **Easier Testing** — No inter-service communication to mock
- 🔁 **Simplified Deployment** — One deployment package to manage

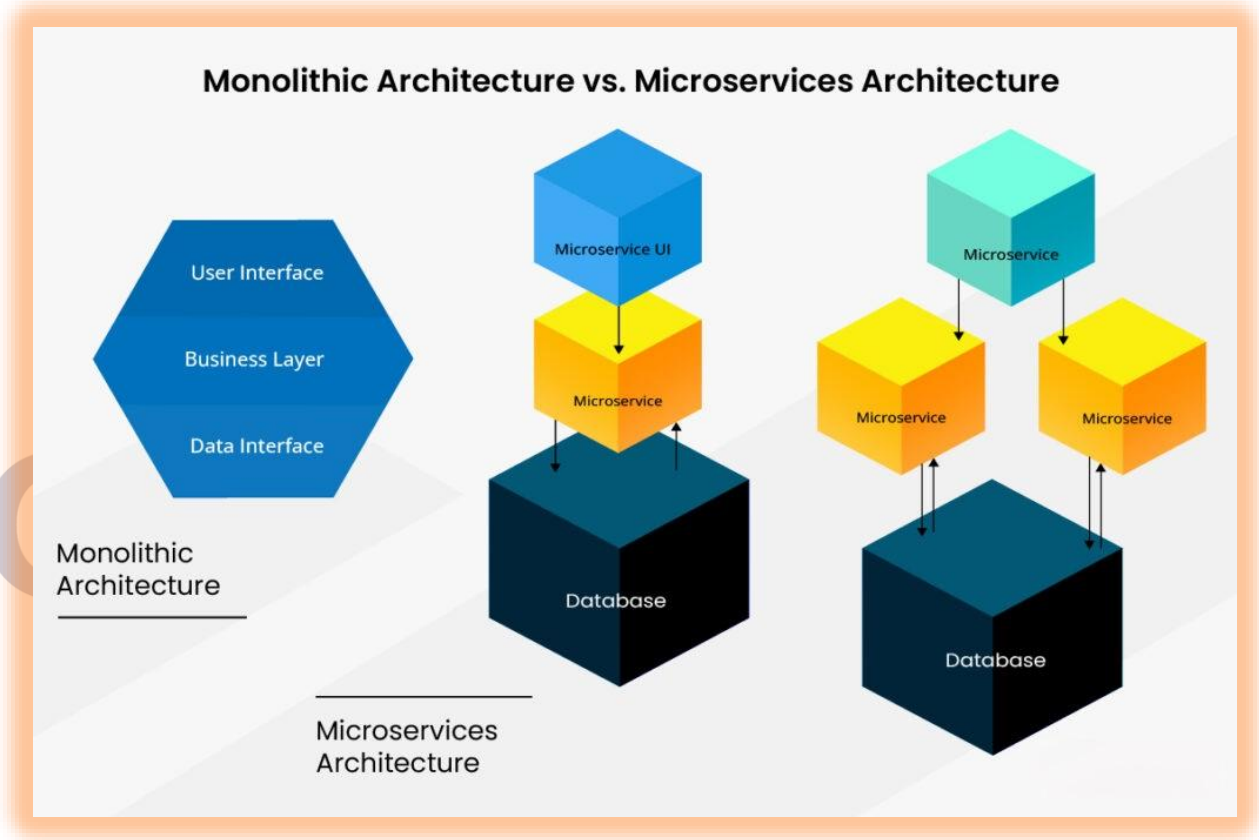
❗ **Challenges**

- 📏 **Scalability Limitations** — Can't scale individual components independently
- 🛑 **Single Point of Failure** — A bug in one part can crash the entire system
- ♻️ **Slow Development at Scale** — Harder to maintain as the codebase grows
- 🔄 **Limited Flexibility** — Difficult to adopt new tech stacks for specific modules

---

📐 **Why Monolithic for This Project?**

- 👥 Suitable for **small teams** or **early-stage MVPs**
- ⚙️ Faster initial setup and deployment using Docker and Kubernetes
- 🔒 Easier to integrate **security and monitoring tools** across a single service



CI/CD pipeline diagram

👷 Code Development starts when a developer pushes code to GitHub.

🧪 Static Code Analysis runs automatically to check code quality and security.

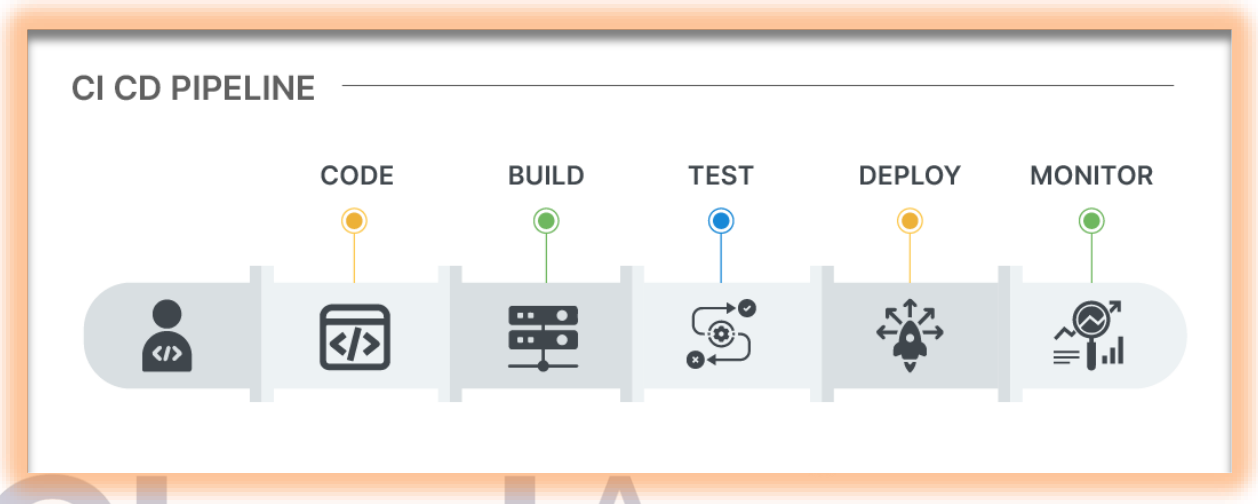
🚀 Jenkins CI gets triggered, pulls the latest code, and performs automated build and testing.

📦 Successfully built artifacts are stored in Artifactory, and a "bake" process packages them.

🧪 The baked image is deployed to a Staging/Test environment for deployment testing.

☁️ After validation, the app is automatically deployed to production clusters in the cloud infrastructure.

Pipeline :



JIRA PROJECT MANAGEMENT TOOL :

📄 **What is Jira?**

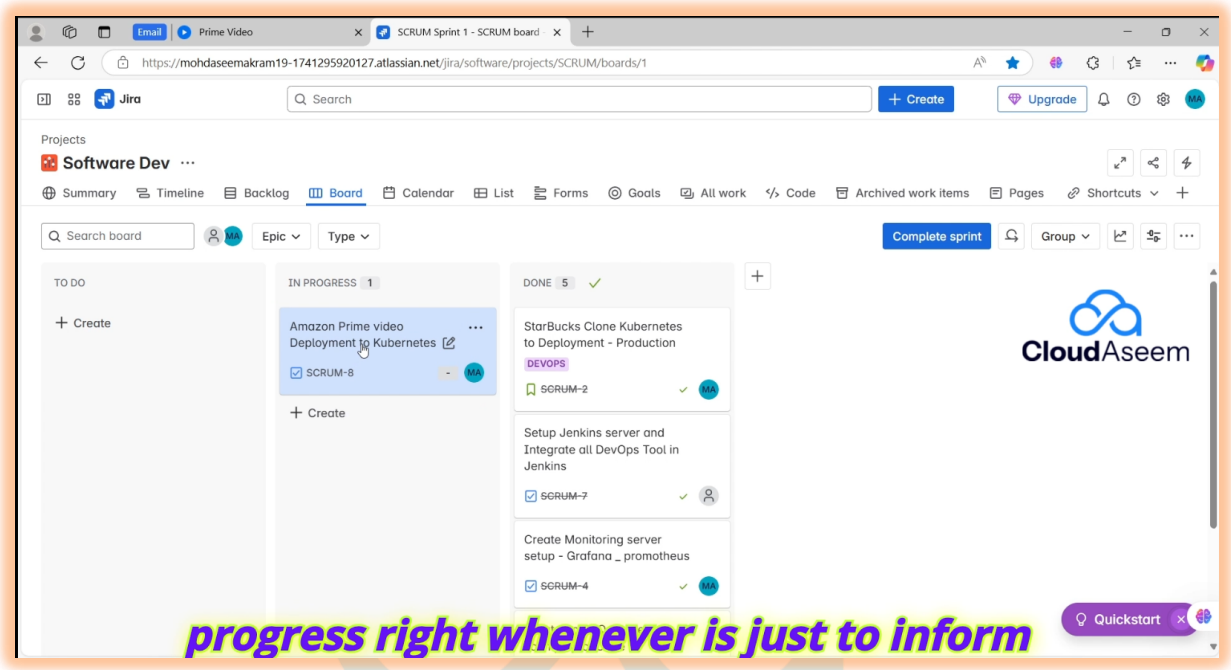
Jira is a powerful **project management** and **issue-tracking** tool developed by Atlassian. It is widely used by **software development teams** to plan, track, release, and maintain software products.

🚀 **What You Can Do with Jira:**

1. 🧠 **Plan Work**
   - Create **Epics**, **User Stories**, **Tasks**, and **Bugs**
   - Prioritize them in a **Backlog**
2. 💻 **Track Progress**
   - Use **Scrum** or **Kanban boards** to visually track work
   - Move tasks through statuses like *To Do* → *In Progress* → *Done*
3. 🔗 **Collaborate**
   - Add **comments**, attach files, and tag teammates
   - Link issues to code, builds (CI/CD), and documentation
4. 📈 **Report & Analyze**

- Use **Dashboards** for real-time updates
- Generate **burndown charts**, **velocity reports**, and more

5. 🧪 **Test & Release**
   - Integrate with tools like **Jenkins, GitHub, SonarQube, Xray, Zephyr**
   - Track releases and deployment pipelines



🧰 **Task Description:**

Implement the full CI/CD flow as per the shared architecture diagram:

1. 👨‍💻 **Code Commit & Analysis**
   - Developers push code to GitHub.
   - Integrate **Static Code Analysis** using tools like SonarQube.
2. 🛠️ **Continuous Integration (CI)**
   - Configure **Jenkins** to trigger builds on Git push.
   - Run unit and integration tests.
   - Store build artifacts in **Artifactory**.
3. 🍱 **Artifact Baking**
   - Bake Docker/DEB images post successful tests.
   - Publish them to the artifact repository.
4. 🚀 **Continuous Deployment (CD)**
   - Deploy artifacts to **Staging** using Jenkins and Kubernetes.
   - Run automated **Deployment Testing**.
   - If successful, proceed to deploy to **Production**.
5. 📈 **Monitoring & Notification**
   - Ensure all stages are monitored.
   - Configure **email alerts** and **build/test notifications** to relevant stakeholders.

Awesome! 🙌

🔧 Let's Start the Implementation of the Amazon-prime-video App 🚀

👨‍💻 👉 Proceeding to the Lab Setup…

We'll begin with:

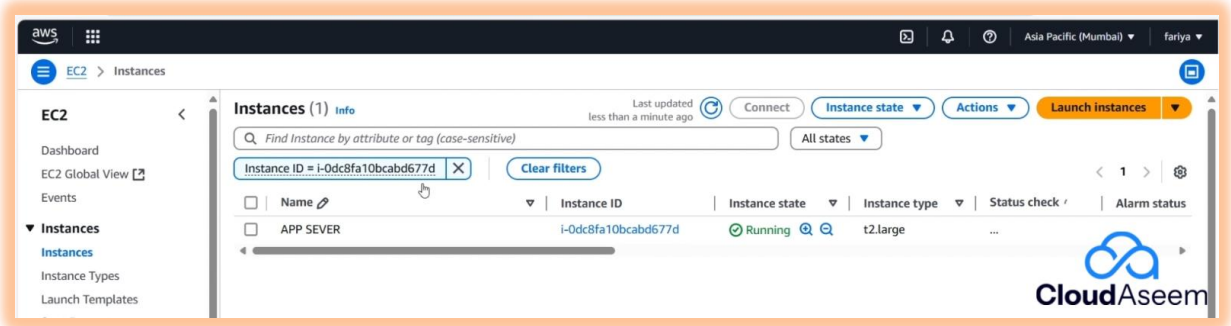
1. 🖥️ **Provisioning Jenkins Server**
2. 🛡️ **Setting up SonarQube for Code Quality & Security**
3. 📊 **Launching Monitoring Stack (Prometheus & Grafana)**
4. ☸️ **Preparing Kubernetes Cluster for Deployment**
5. 🧪 **Integrating CI/CD Pipeline with DevSecOps Practices**

Now, let's get started and dig deeper into each of these steps: -

I. Configure Infrastructure jenkins Server In AWS Cloud

1. Launch an EC2 Instance Ubuntu (22.04) T3 Large Instance

- Go to the **AWS Management Console** → **EC2** → **Instances**.
- Click **Launch Instance**.
- Set the following configurations:
  - **Name**: <Jenkins server>
  - **AMI (Amazon Machine Image)**: Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
  - **Instance Type**: t3.large (2 vCPUs, 8 GB RAM)
  - **Key Pair**: Select an existing key pair or create a new one.
  - **Storage**: Default (e.g., 20GB GP3 SSD, adjust as needed).

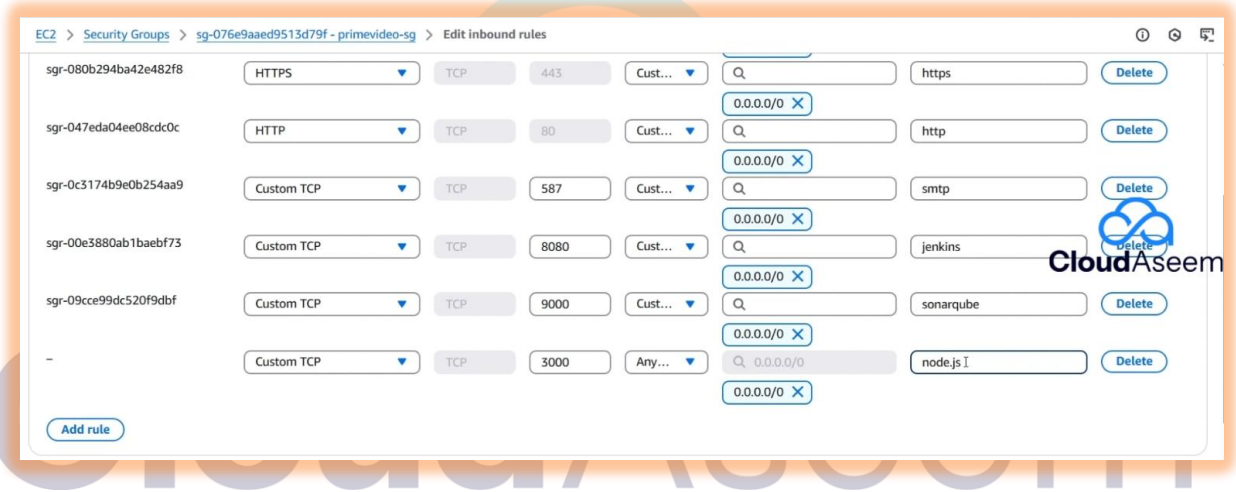


2. Configure Security Group

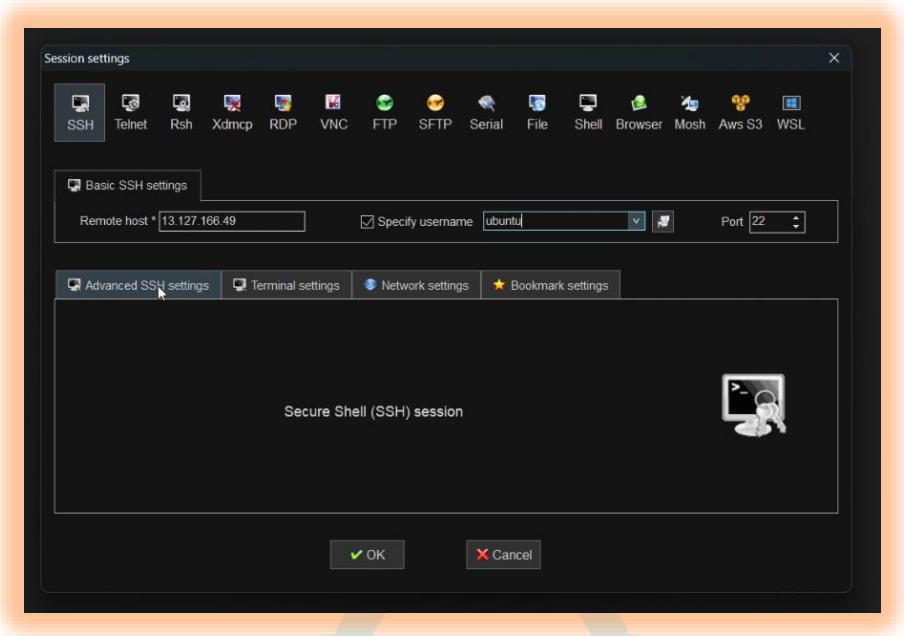
Create or modify a security group to allow the following ports:

Port	Protocol	Description
22	TCP	SSH (for remote access)
80	TCP	HTTP (Web traffic)
443	TCP	HTTPS (Secure web traffic)
8080	TCP	Jenkins
3000	TCP	Web Applications
587	TCP	SMTP (Email sending)
9000	TCP	SONARQUBE
465	TCP	SMTP over SSL

- Set the **Source** to **Anywhere (0.0.0.0/0, ::/0)** unless you want to restrict access.



- **Launch & Connect with Mobaxterm App**Click **Launch Instance**. Once the instance is running, connect using:



clone the GitHub repository

the step-by-step guide to clone the GitHub repository,:

1. **Clone the existing GitHub repository**:

git clone https://github.com/Aseemakram19/amazon-prime-video-kubernetes

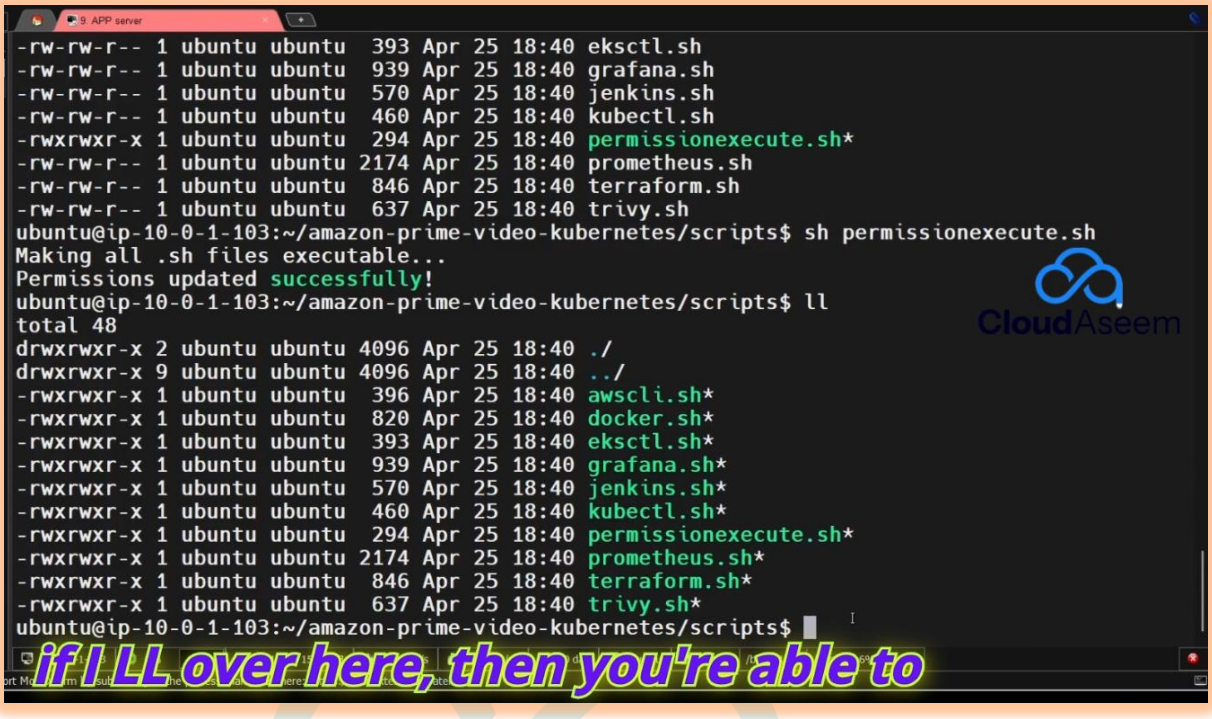
Navigate to the project directory:

cd amazon-prime-video-kubernetes/

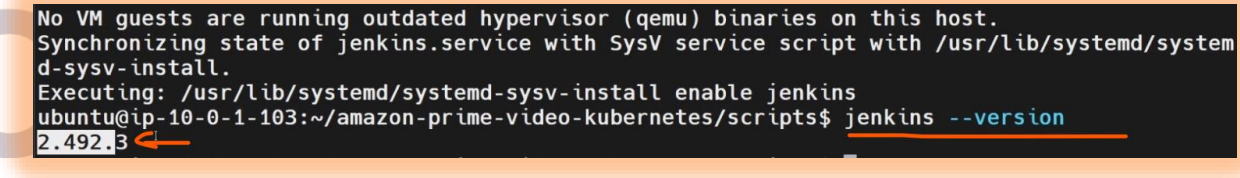
- Install the TOOLS in the VM machine via Scripts . add executable permission to shell script
chmod +x *.sh

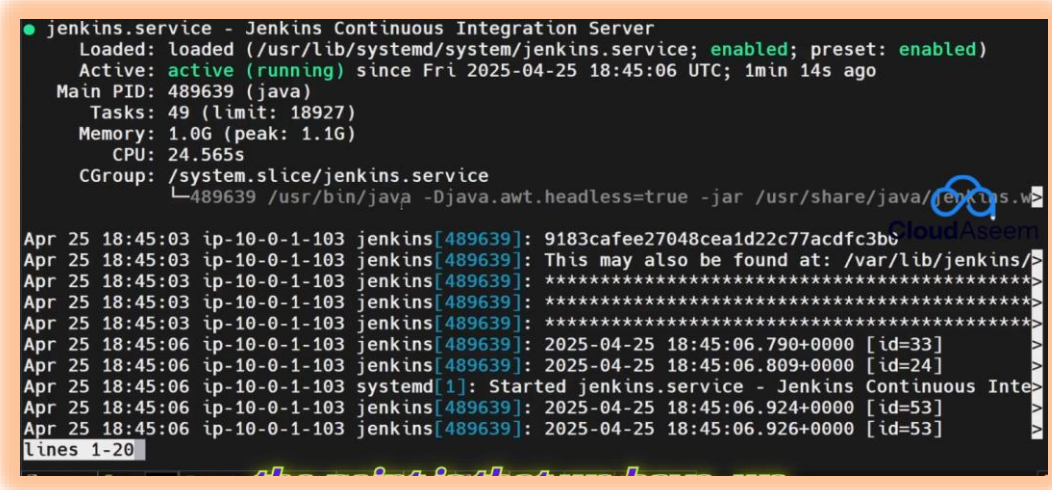
Install Tools

1. Jenkins
2. Docker
3. Sonarqube
4. Trivy
5. Terraform
6. AWS cli
7. Kubectl
8. EKSctl



Jenkins Installed



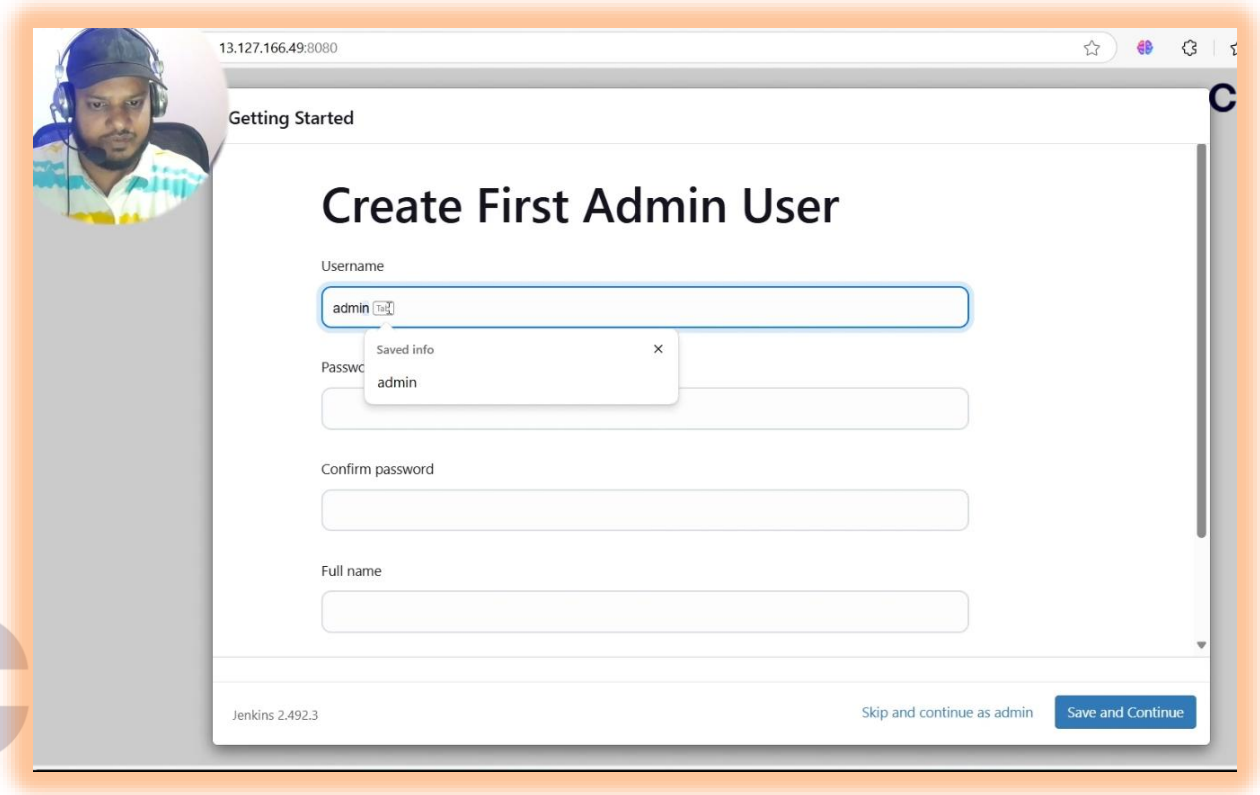


- Access Jenkins in your browser:

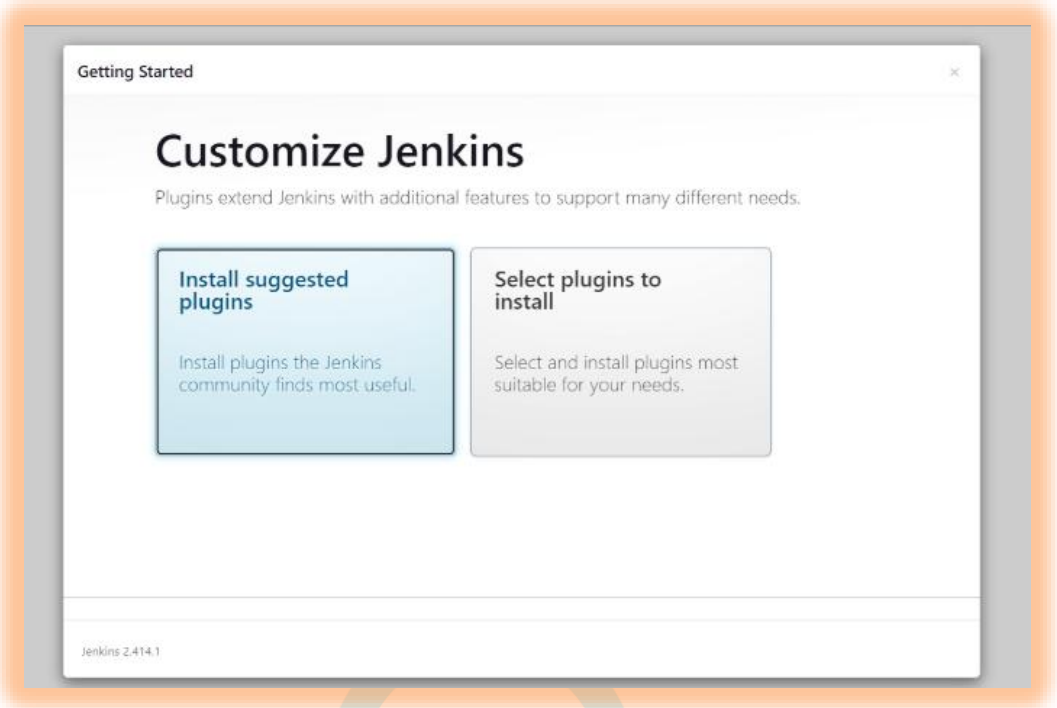
```
http://<PUBLIC_IP>:8080
```

- Unlock Jenkins using an administrative password and install the suggested plugins. Retrieve the initial admin password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```



Unlock Jenkins using an administrative password and install the suggested plugins.



Configure Jenkins Installed

- Create a user click on save and continue.
- Jenkins Getting Started Screen.
- Follow the setup wizard and install recommended plugins.

Install Plugins like JDK, SonarQube Scanner, NodeJs, OWASP Dependency Check

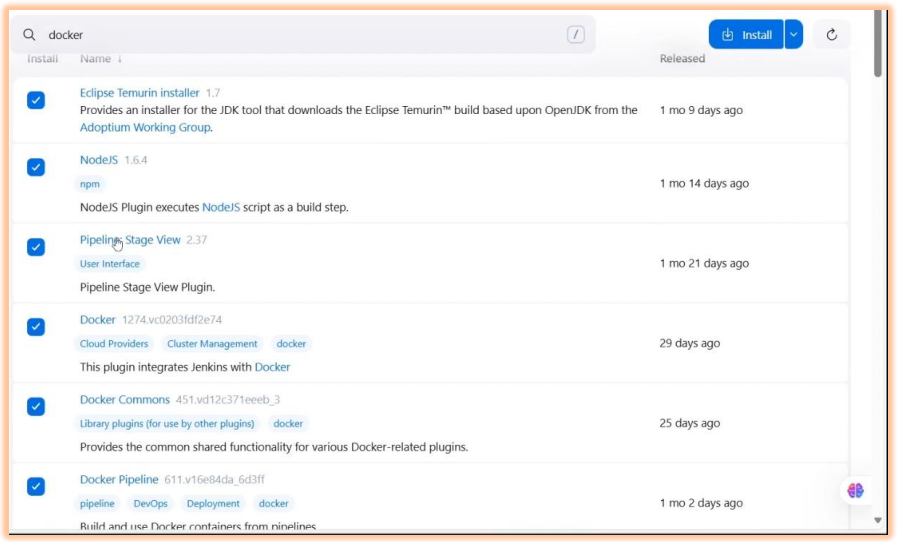
Goto Manage Jenkins →Plugins → Available Plugins →

Install below plugins

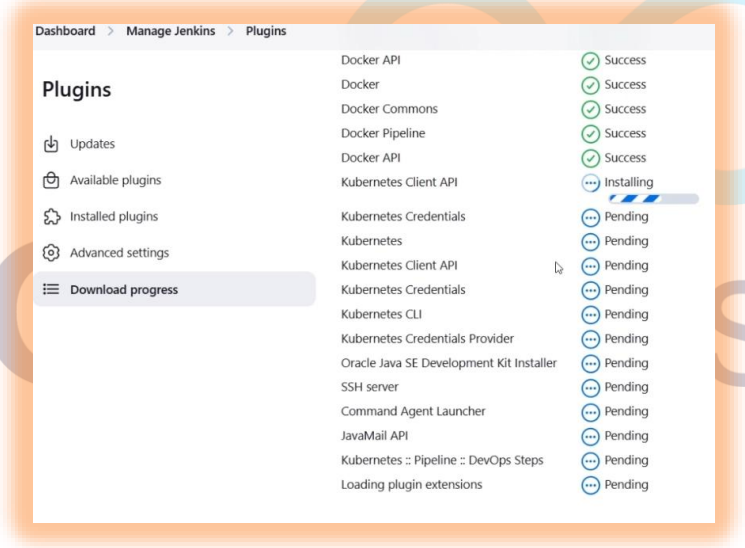
1. Eclipse Temurin Installer (Install without restart)
2. SonarQube Scanner (Install without restart)
3. NodeJs Plugin (Install Without restart) – 16.20.2
5. Stage view
6. jdk

Docker plugin

7. Docker
8. Docker Commons
9. Docker Pipeline
10. Docker API
11. docker-build-step
12. Kubernetes Client API
13. Kubernetes Credentials
14. Kubernetes CLI
15. Kubernetes



Add this too plugins :



Setup SonarQube

Docker already installed , configure SonarQube Container

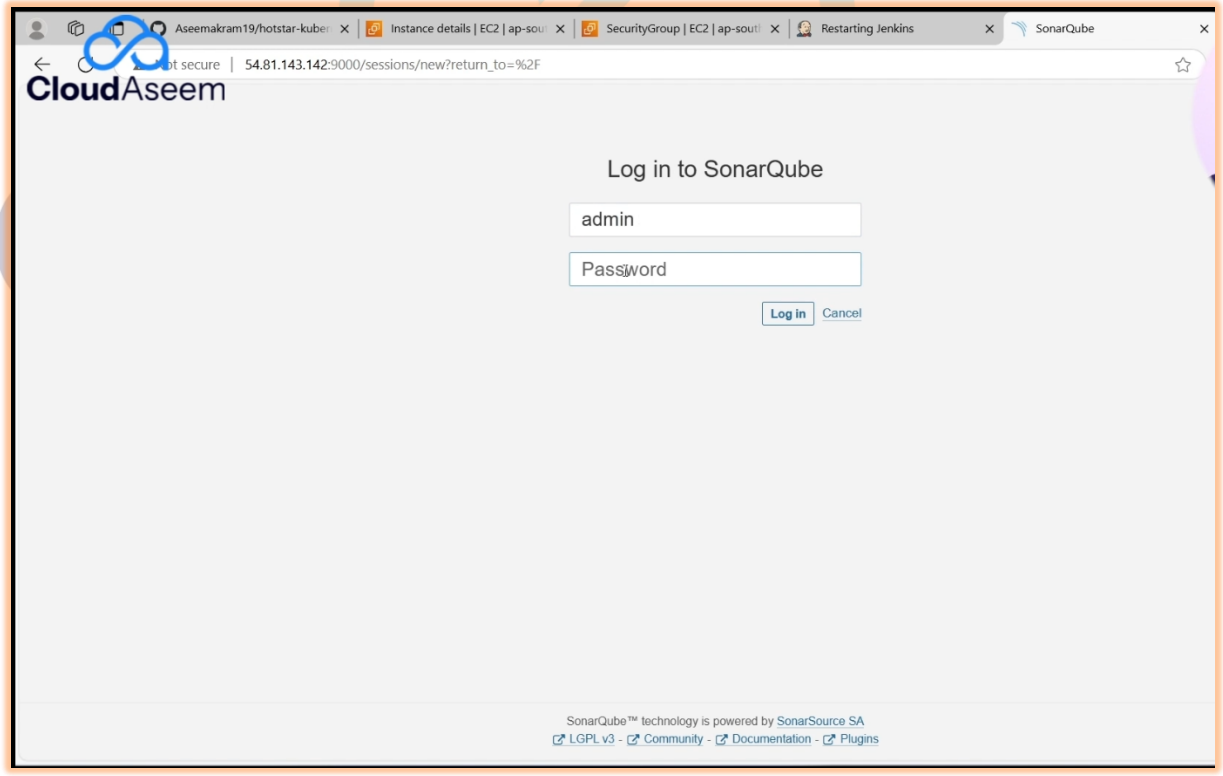
```
# Verify installation
docker -version
# Run SonarQube container in detached mode with port mapping
```

docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

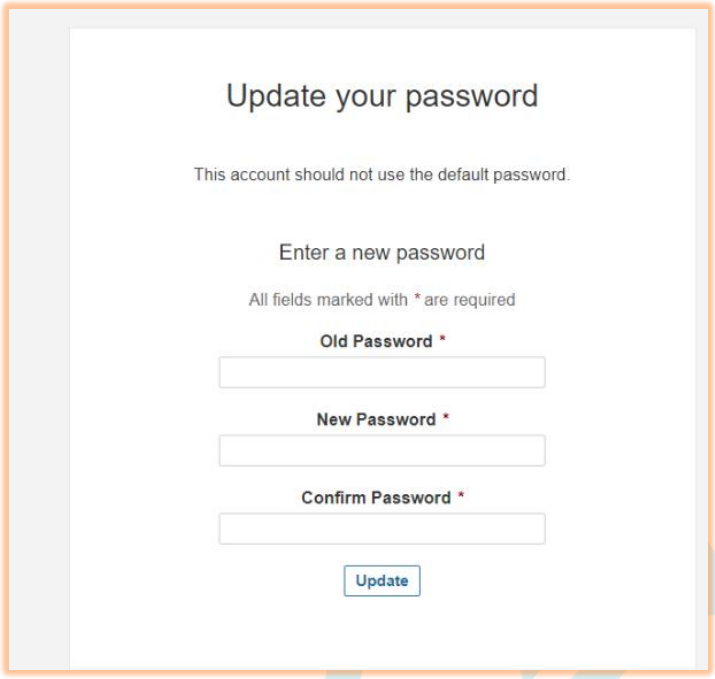
```
# Run SonarQube container in detached mode with port mapping
#docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
ubuntu@ip-10-0-1-103:~/amazon-prime-video-kubernetes/scripts$ docker run -d --name sonar -p
9000:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
30a9c22ae099: Pull complete
97781601768c: Pull complete
fa6848548472: Pull complete
8e85f04a42eb: Pull complete
0137fea4bef4: Pull complete
6a33d500eb85: Pull complete
893e2612e97d: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:3abad2e9d705543f5a24831db2fcd98edf9f24aaec864f372a97b2b0dc7a2f46
Status: Downloaded newer image for sonarqube:lts-community
4fa2d45a5edacd98d594428bc9d0e4cbaf6f3c361f16618f7145818910dca236
ubuntu@ip-10-0-1-103:~/amazon-prime-video-kubernetes/scripts$
```

Now our Sonarqube is up and running

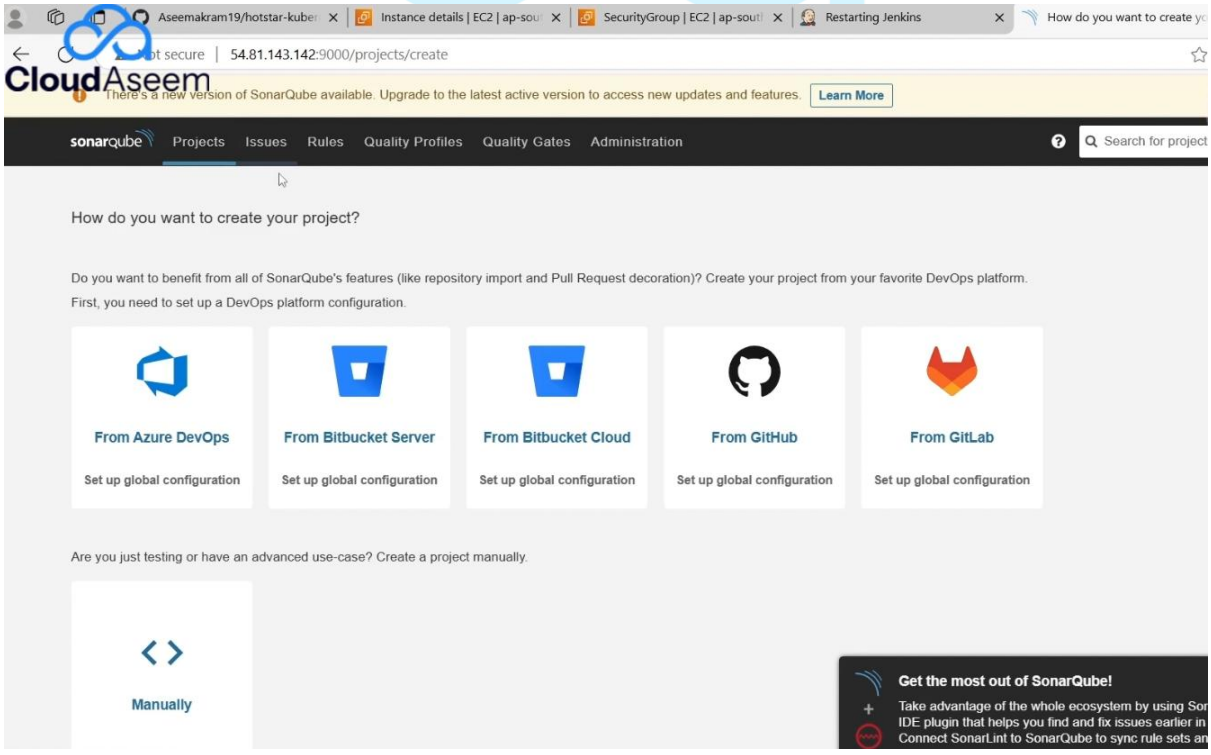


Enter username and password, click on login and change password

username admin
password admin

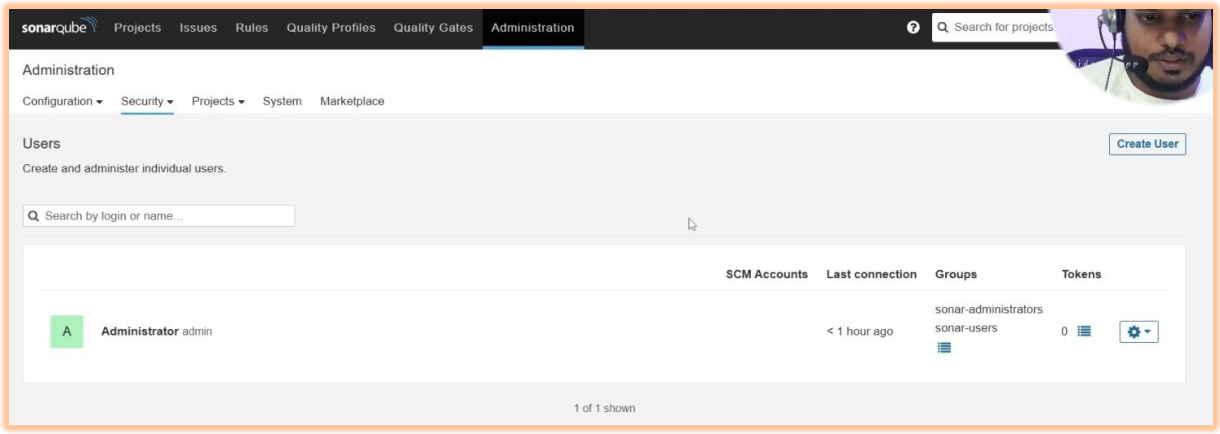


Update New password, This is Sonar Dashboard.



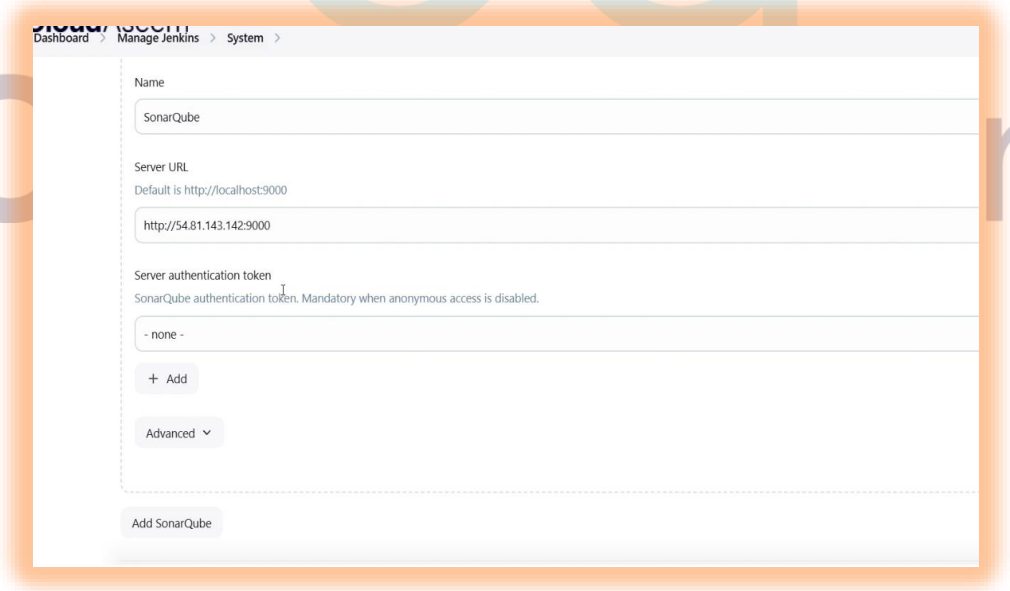
B. - Create Sonar token in order to connect with Jenkins

Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token

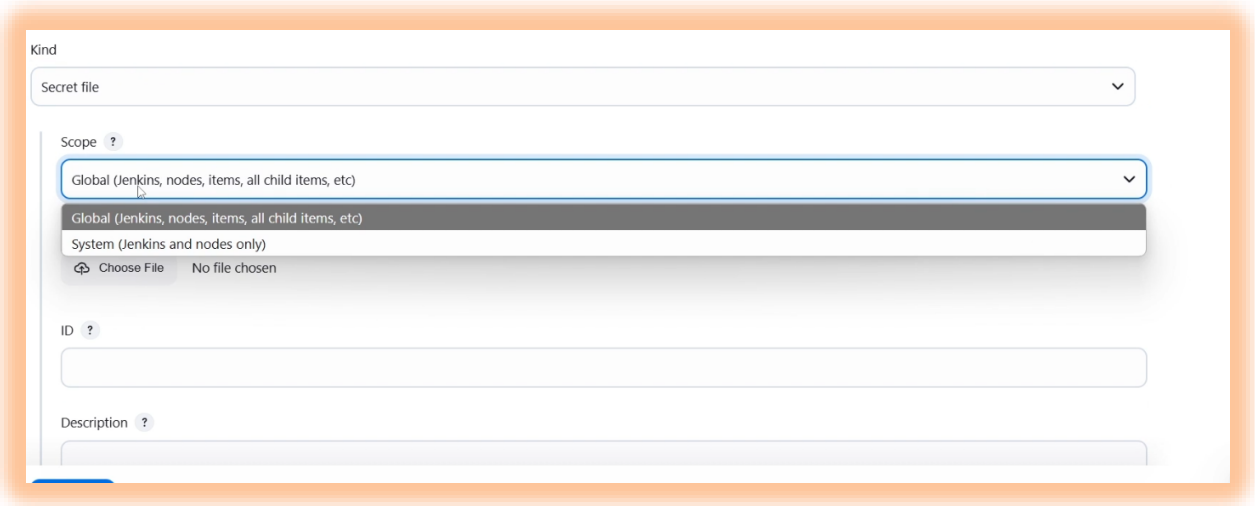


Create a token with a name and generate

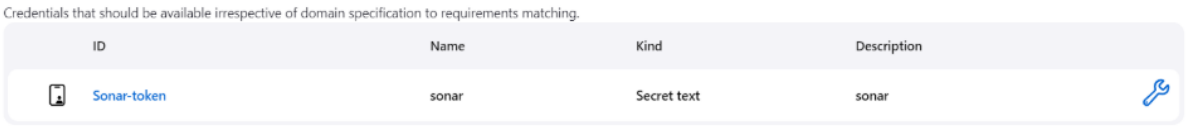
Now, go to Dashboard → Manage Jenkins → System and Add like the below image.



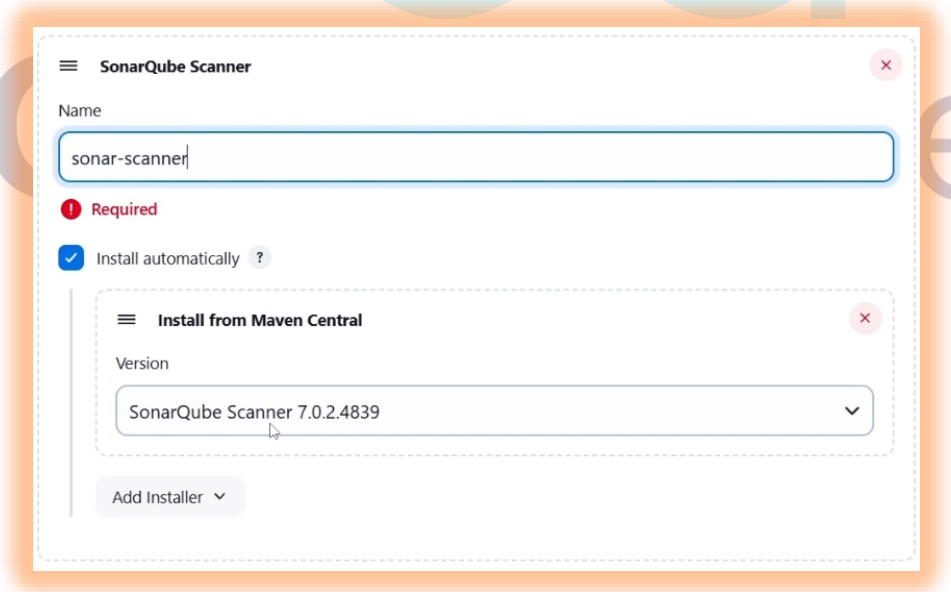
Add Credentials → Add Secret Text. It should look like this



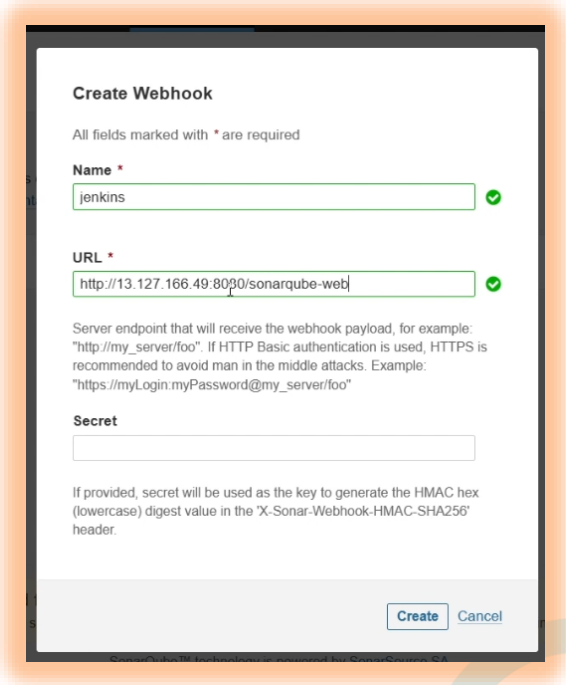
You will this page once you click on create



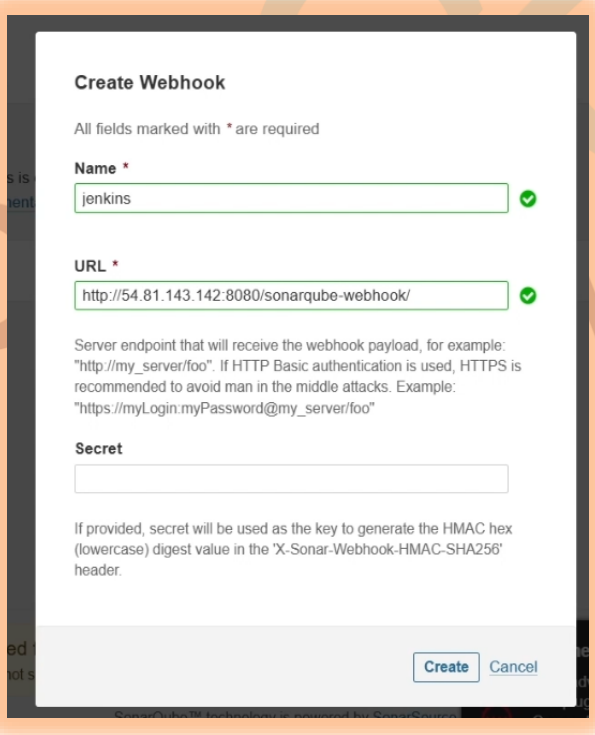
The Configure System option is used in Jenkins to configure different server
Global Tool Configuration is used to configure different tools that we install using Plugins
We will install a sonar scanner in the tools.
Manage Jenkins –> Tools –> SonarQube Scanner



In the Sonarqube Dashboard add a quality gate also
Administration–> Configuration–>Webhooks

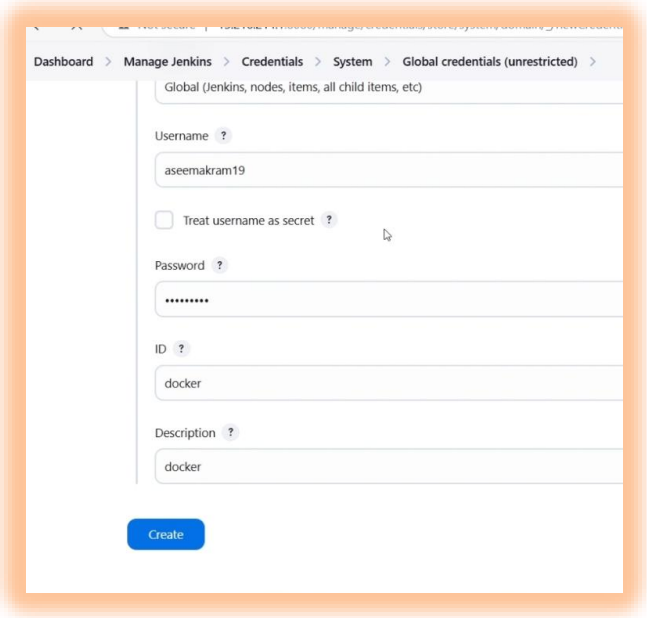


- webhook - http://13.127.166.49:8080/sonarqube-webhook/



-

Docker Hub credentials in Jenkins secret



Install Dockers scout in the app server

Install Docker Scout: docker login Give Dockerhub credentials here curl -sSfL https://raw.githubusercontent.com/docker/scout-cli/main/install.sh | sh -s -- -b /usr/local/bin

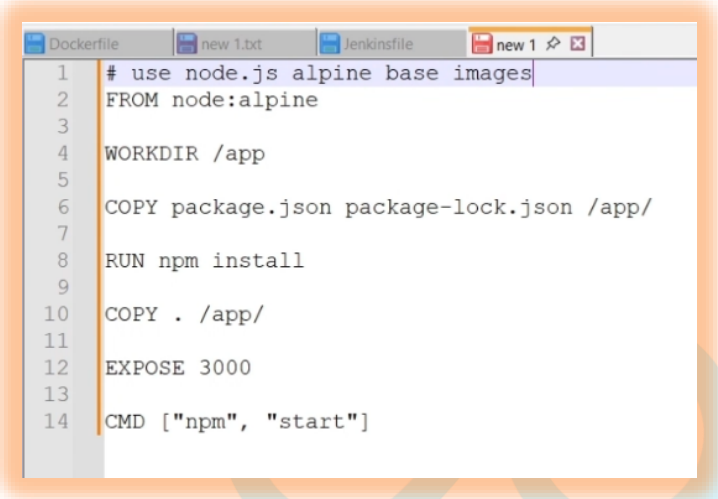
```
Username: aseemakram19
Password:
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu@ip-10-0-1-103:/var/lib/jenkins/workspace/amazon-prime-video$ curl -sSfL https://raw.g
ithubusercontent.com/docker/scout-cli/main/install.sh | sh -s -- -b /usr/local/bin
[info] fetching release script for tag='v1.17.1'
[info] using release tag='v1.17.1' version='1.17.1' os='linux' arch='amd64'
install: cannot create regular file '/usr/local/bin/docker-scout': Permission denied
[error] failed to install docker-scout
ubuntu@ip-10-0-1-103:/var/lib/jenkins/workspace/amazon-prime-video$ sudo su
root@ip-10-0-1-103:/var/lib/jenkins/workspace/amazon-prime-video# curl -sSfL https://raw.git
hubusercontent.com/docker/scout-cli/main/install.sh | sh -s -- -b /usr/local/bin
[info] fetching release script for tag='v1.17.1'
[info] using release tag='v1.17.1' version='1.17.1' os='linux' arch='amd64'
[info] installed /usr/local/bin/docker-scout
root@ip-10-0-1-103:/var/lib/jenkins/workspace/amazon-prime-video#
```

Dockerfile

Write Dockerfile from Strach

This Dockerfile:

1. Uses a lightweight Node.js image.
2. Installs your Node.js app dependencies.
3. Copies your code inside the image.
4. Sets the working directory.
5. Prepares the container to run the app on port 3000.



This Dockerfile is used to containerize a Node.js application using a lightweight image (node:alpine). Let's go line by line and explain **from scratch**, assuming you're new to Docker:

🔹 **# use node.js alpine base images**

This is a **comment** (starts with #). It helps developers understand what the next line is doing. It's **not executed**.

🔹 **FROM node:alpine**

This tells Docker to use the official **Node.js Alpine Linux image** as the **base image**.

- node:alpine is a minimal Node.js image (~5MB), great for smaller image sizes.
- It has Node.js pre-installed.

🔹 **WORKDIR /app**

Sets the **working directory** inside the container to /app.

- All the next commands (like COPY, RUN) will be run **inside** this directory.
- If /app doesn't exist, Docker will create it.

🔹 **COPY package.json package-lock.json /app/**

Copies your package.json and package-lock.json files from your **local machine** into the container's /app directory.
These files are needed to install Node.js dependencies.

🔹 **RUN npm install**

Runs npm install **inside the container**.
This installs all dependencies listed in package.json.

🔹 **COPY . /app/**

Copies the entire project folder (.) from your local machine to the /app/ folder in the container.

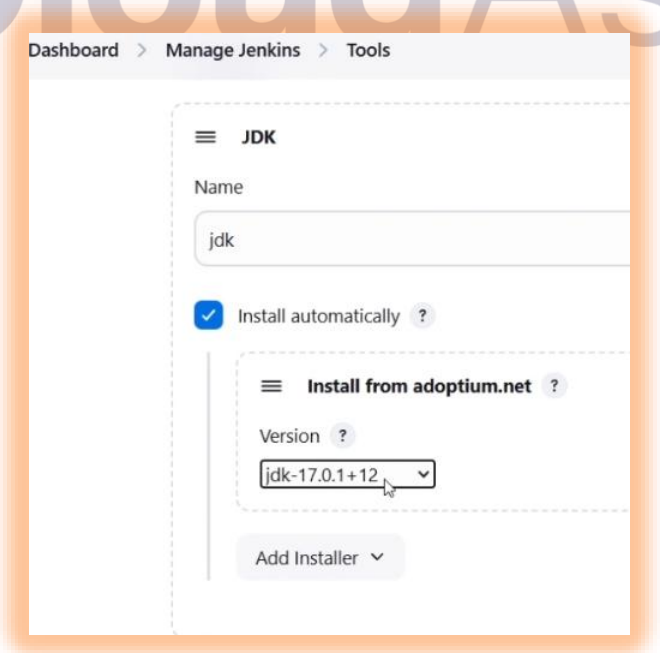
🔹 **EXPOSE 3000**

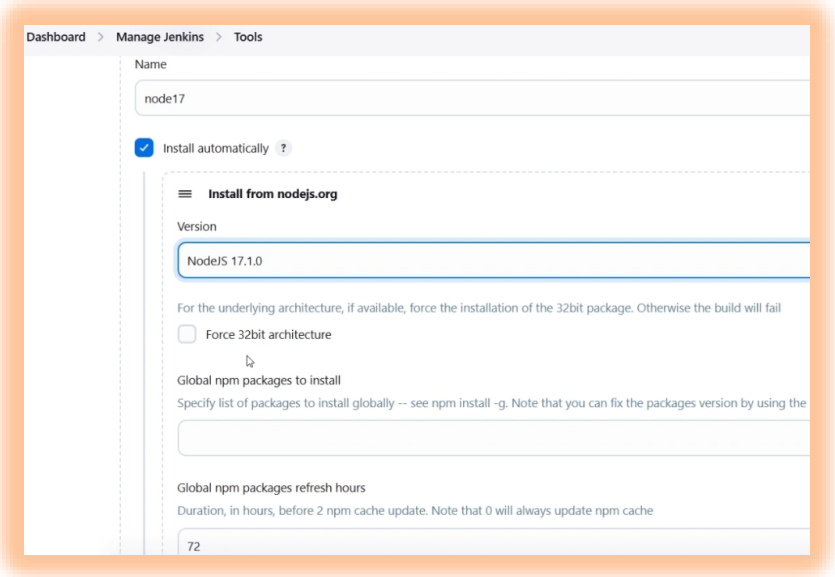
Informs Docker that the app **will listen on port 3000** inside the container.
This doesn't actually publish the port—it's just a declaration.
You'll still need to run docker run -p 3000:3000 to expose it on your host machine.

🔹 **CMD ["npm", "start"]**

Specifies the **default command** that runs when the container starts.
In this case, it runs npm start, which should start your Node.js app (as defined in package.json).

Configure Tools in jenkins
1. JDK
2. Nodejs





Create a Gmail SMTP App Password

An **App Password** is a 16-character password that allows third-party applications (like Jenkins) to send emails using **Gmail SMTP** securely.

Step 1: Enable 2-Step Verification

Before generating an App Password, you **must** enable **2-Step Verification** in your Google Account.

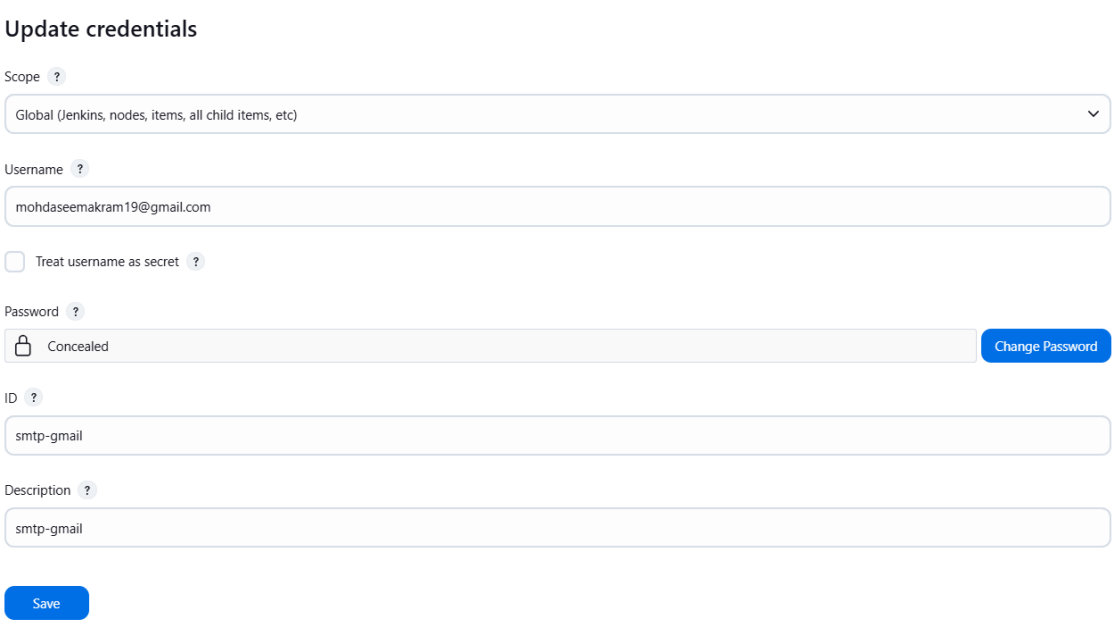
1. **Go to Google Account Security**:
   Google My Account
2. Scroll to **"Signing in to Google"**.
3. Click **"2-Step Verification"** → Click **"Get Started"**.
4. Follow the steps to set up **2-Step Verification** (via SMS or Authenticator App).

Step 2: Generate an App Password

1. **Go to App Passwords Page**:
   Google App Passwords
2. Sign in with your **Google Account**.
3. Under **"Select app"**, choose **"Mail"**.
4. Under **"Select device"**, choose **"Other (Custom Name)"** and enter **"Jenkins SMTP"**.
5. Click **"Generate"**.
6. **Copy the 16-character App Password** (e.g., abcd efgh ijkl mnop).

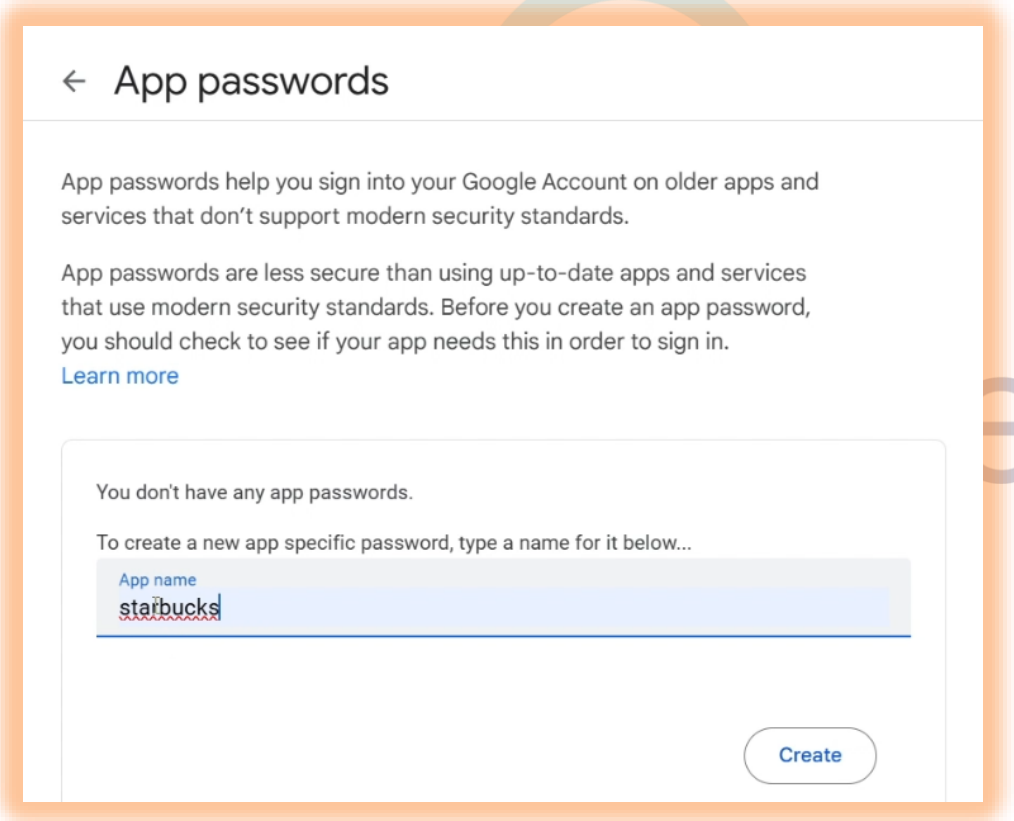
Add credentials as Username and password in jenkins

7.



Step 3: Configure Gmail SMTP in Jenkins

1. **Go to Jenkins Dashboard → Manage Jenkins → Configure System.**
2. Scroll to **"E-mail Notification"**.
3. Set the following:
   - **SMTP Server**:

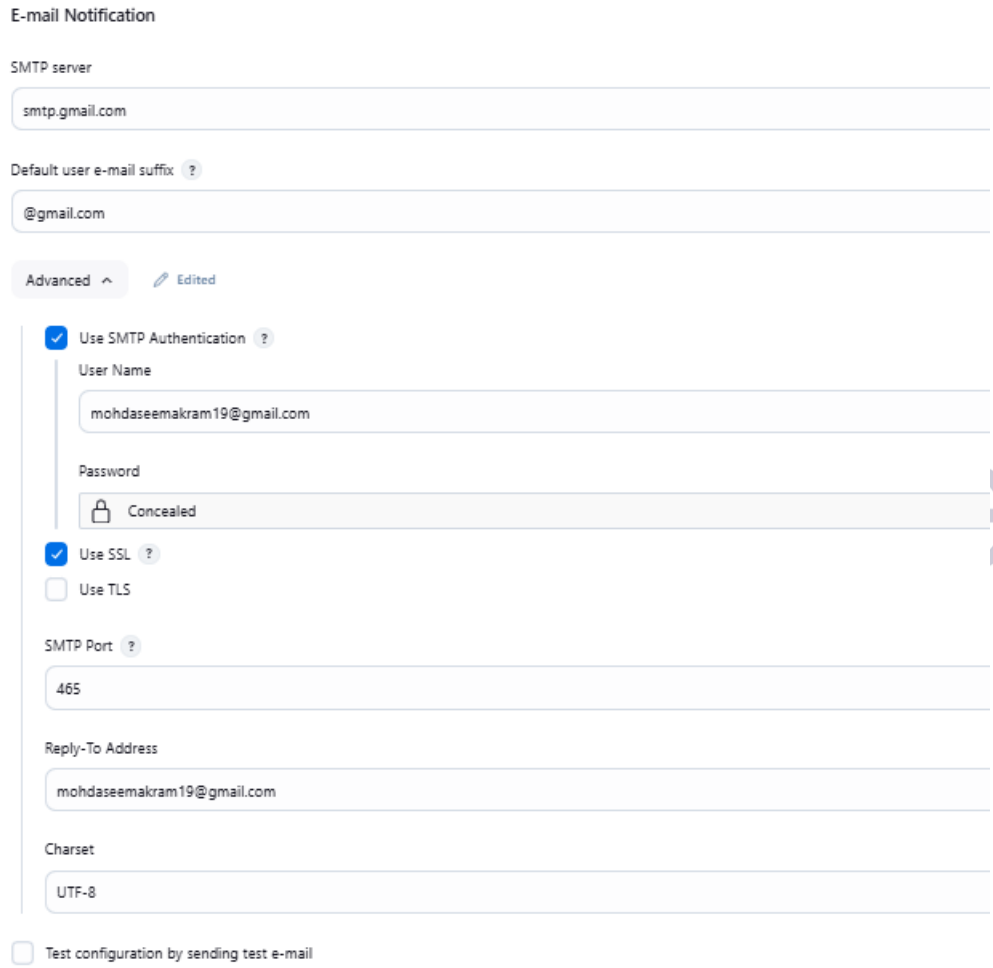


smtp.gmail.com

   - **Use SMTP Authentication**: ✅ Checked
   - **User Name**: Your Gmail ID (your-email@gmail.com)
   - **Password**: Paste the **App Password**
   - **SMTP Port**: 587
   - **Use TLS**: ✅ Checked
4. Click **Save**.

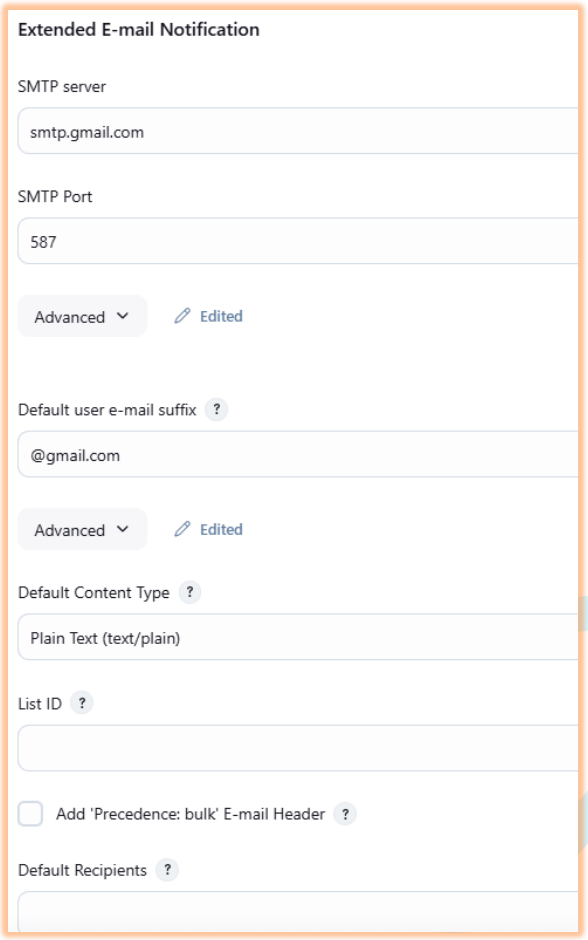
Email Extension Plugin
xsuc kxeb xcvk xqkf

1. Basic Email Notification
SMTP Server: smtp.gmail.com
Email Suffix: @gmail.com (default user email domain)
SMTP Authentication: Enabled
Username: Your Gmail address
Password: Your Gmail password or App Password (for 2-factor authentication)
Use TLS: Checked
SMTP Port: 587
Reply-To Address: Your email address
Charset: UTF-8



2. Extended Email Notification
SMTP Server: smtp.gmail.com
SMTP Port: 465 (for tls)
Use SSL: Checked

Credentials: Select from the



Lets setup our first Jenkins pipeline

Create Job for

Let's add a pipeline , to test the Github Clone stage

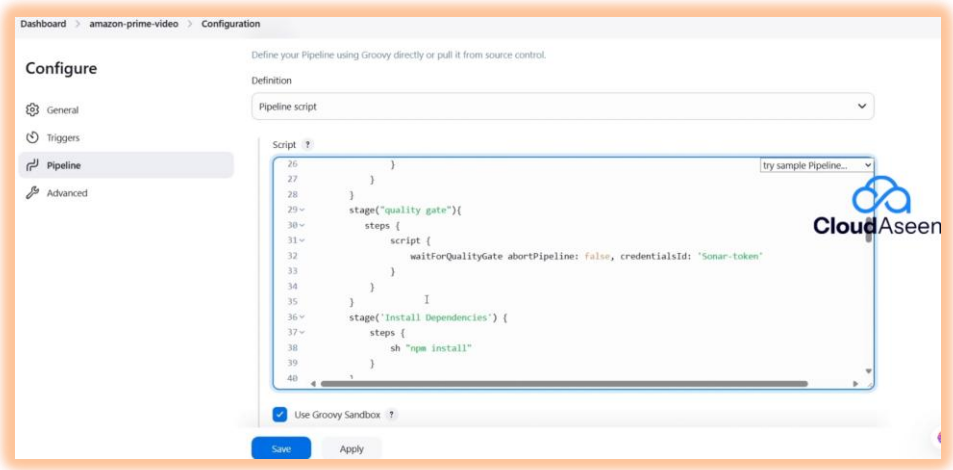
Apply and Save and click on Build

Let's add a pipeline of Project

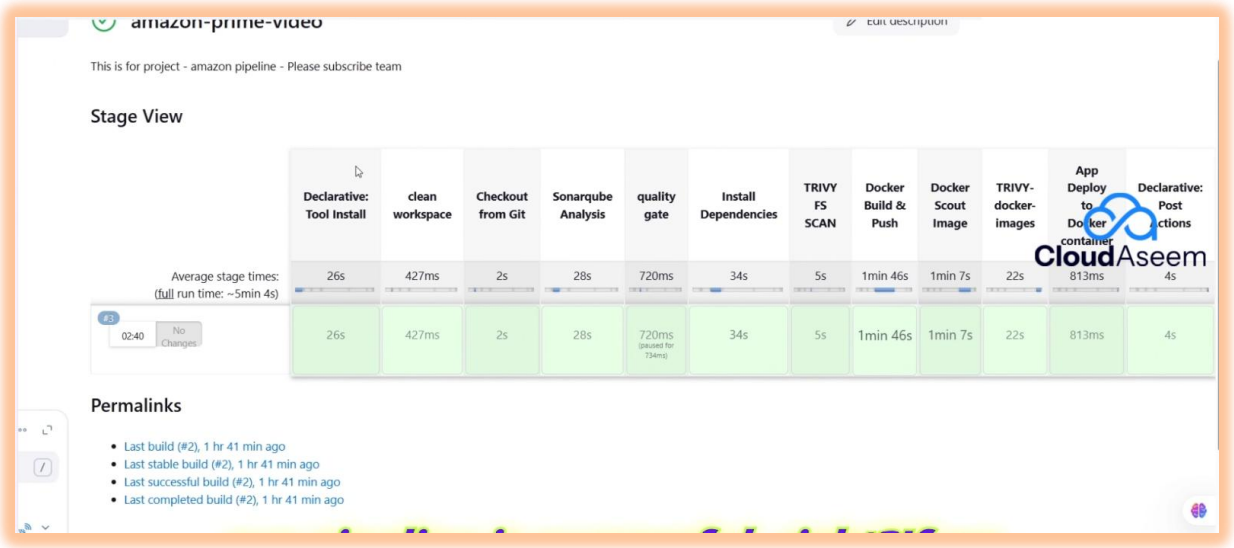
```
pipeline{
agent any
tools{
   jdk 'jdk'
   nodejs 'node17'
}
environment {
   SCANNER_HOME=tool 'sonar-scanner'
}
stages {
   stage('clean workspace'){
      steps{
         cleanWs()
      }
   }
```

```
stage('Checkout from Git'){
   steps{
      git branch: 'main', url: 'https://github.com/Aseemakram19/amazon-prime-video-kubernetes.git'
   }
}
stage("Sonarqube Analysis "){
   steps{
      withSonarQubeEnv('SonarQube') {
         sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=amazon-prime-video \
         -Dsonar.projectKey=amazon-prime-video '''
      }
   }
}
stage("quality gate"){
  steps {
      script {
         waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
      }
   }
}
stage('Install Dependencies') {
   steps {
      sh "npm install"
   }
}
stage('TRIVY FS SCAN') {
   steps {
      sh "trivy fs . > trivyfs.txt"
   }
}
stage("Docker Build & Push"){
   steps{
      script{
        withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
           sh "docker build -t amazon-prime-video ."
           sh "docker tag amazon-prime-video aseemakram19/amazon-prime-video:latest "
           sh "docker push aseemakram19/amazon-prime-video:latest "
         }
      }
   }
}
                        stage('Docker Scout Image') {
   steps {
      script{
        withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
           sh 'docker-scout quickview aseemakram19/amazon-prime-video:latest'
           sh 'docker-scout cves aseemakram19/amazon-prime-video:latest'
           sh 'docker-scout recommendations aseemakram19/amazon-prime-video:latest'
         }
      }
   }
}

stage("TRIVY-docker-images"){
   steps{
      sh "trivy image aseemakram19/amazon-prime-video:latest > trivyimage.txt"
   }
}
stage('App Deploy to Docker container'){
   steps{
      sh 'docker run -d --name amazon-prime-video -p 3000:3000 aseemakram19/amazon-prime-video:latest'
   }
}
```

```
  }
  post {
  always {
    script {
      def buildStatus = currentBuild.currentResult
      def buildUser = currentBuild.getBuildCauses('hudson.model.Cause$UserIdCause')[0]?.userId ?: 'Github User'

      emailext (
        subject: "Pipeline ${buildStatus}: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
        body: """
          <p>This is a Jenkins amazon-prime-video CICD pipeline status.</p>
          <p>Project: ${env.JOB_NAME}</p>
          <p>Build Number: ${env.BUILD_NUMBER}</p>
          <p>Build Status: ${buildStatus}</p>
          <p>Started by: ${buildUser}</p>
          <p>Build URL: <a href="${env.BUILD_URL}">${env.BUILD_URL}</a></p>
        """,
        to: 'mohdaseemakram19@gmail.com',
        from: 'mohdaseemakram19@gmail.com',
        replyTo: 'mohdaseemakram19@gmail.com',
        mimeType: 'text/html',
        attachmentsPattern: 'trivyfs.txt,trivyimage.txt'
      )
    }
  }

  }

}
```

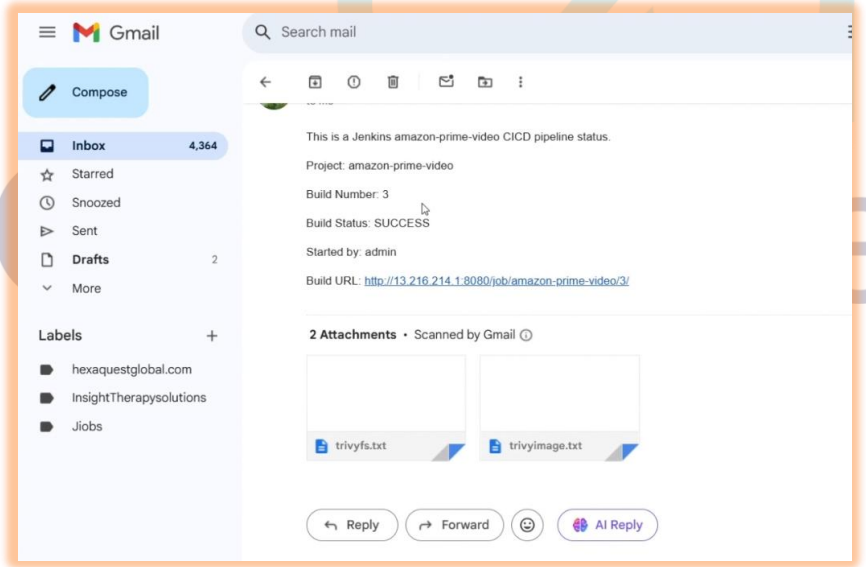


Build the PIPELINE TO EXECUTE THE STAGE OF PROJECT

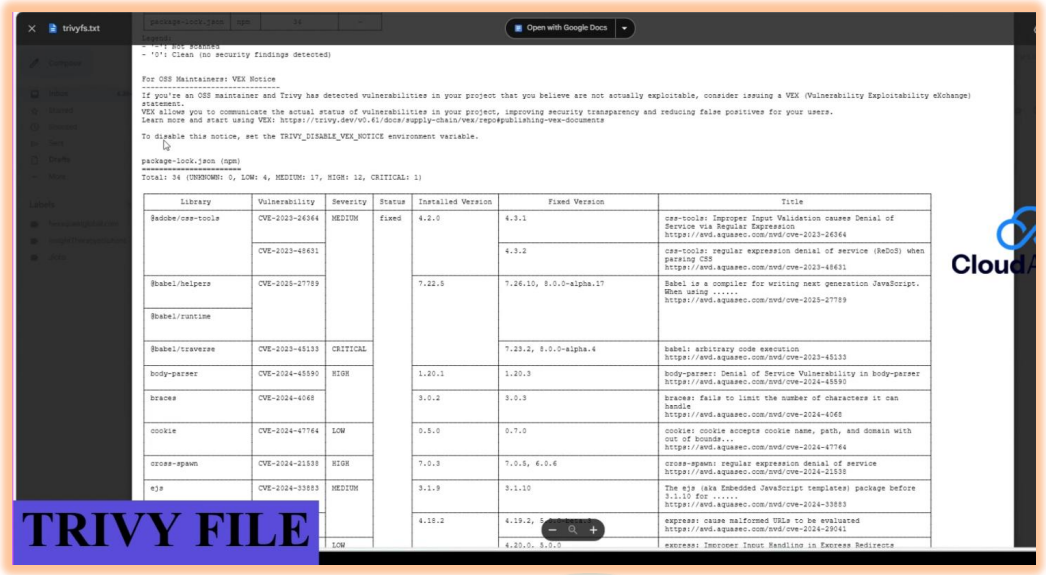


Result :

- Emails received

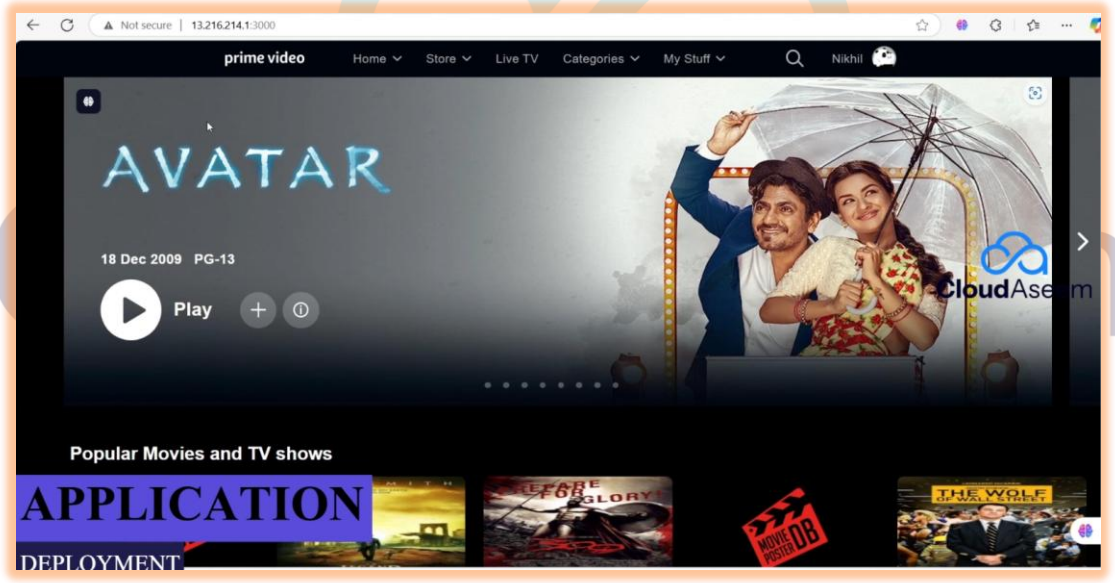


Report attached in emails

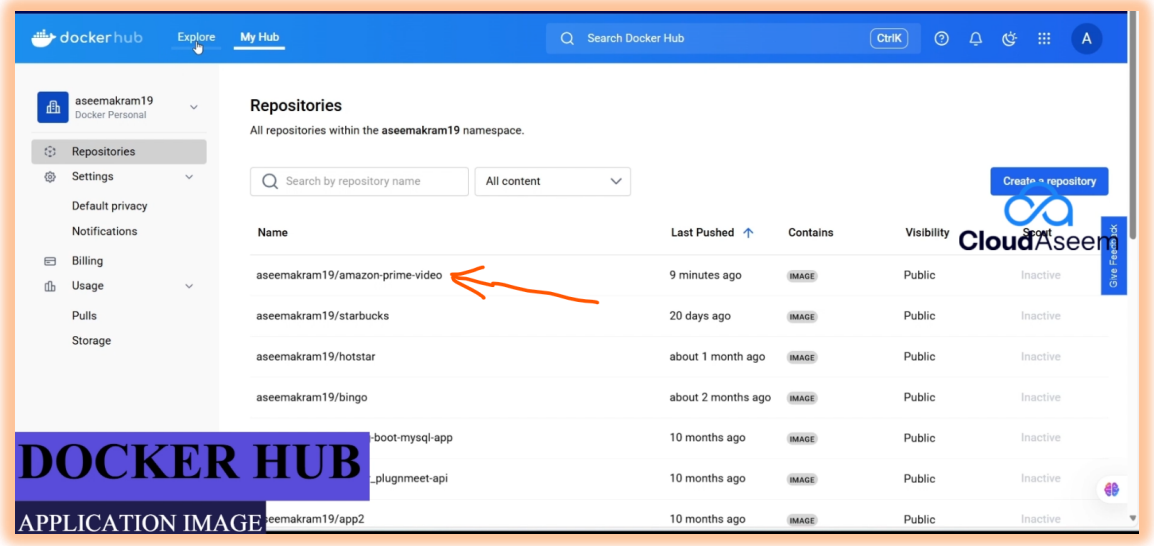


1. Verify Application Deployment as Docker container <public-ip:3000>

Our Application is live with this output

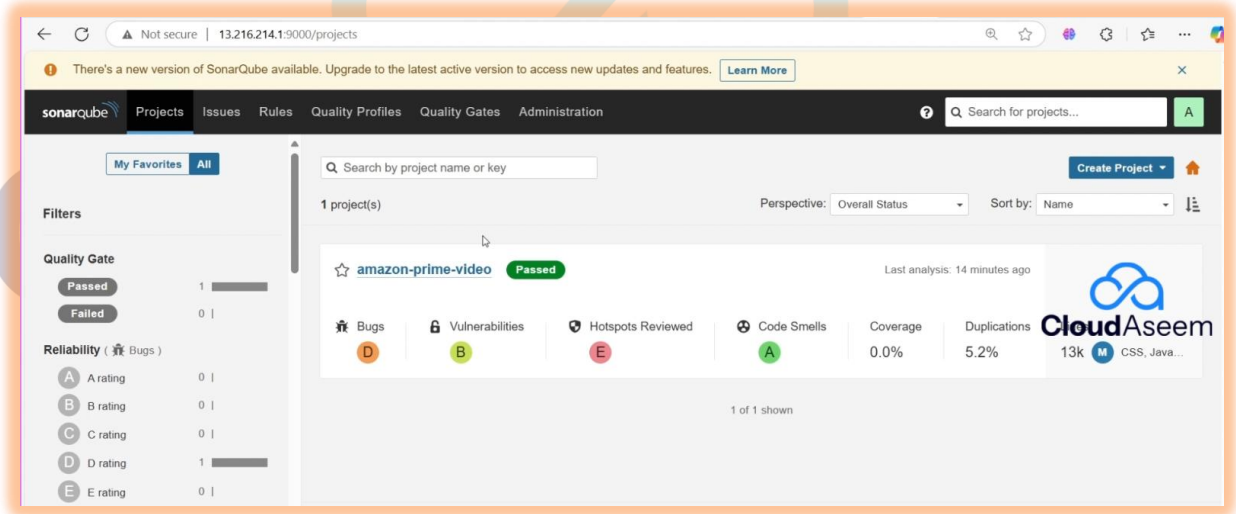


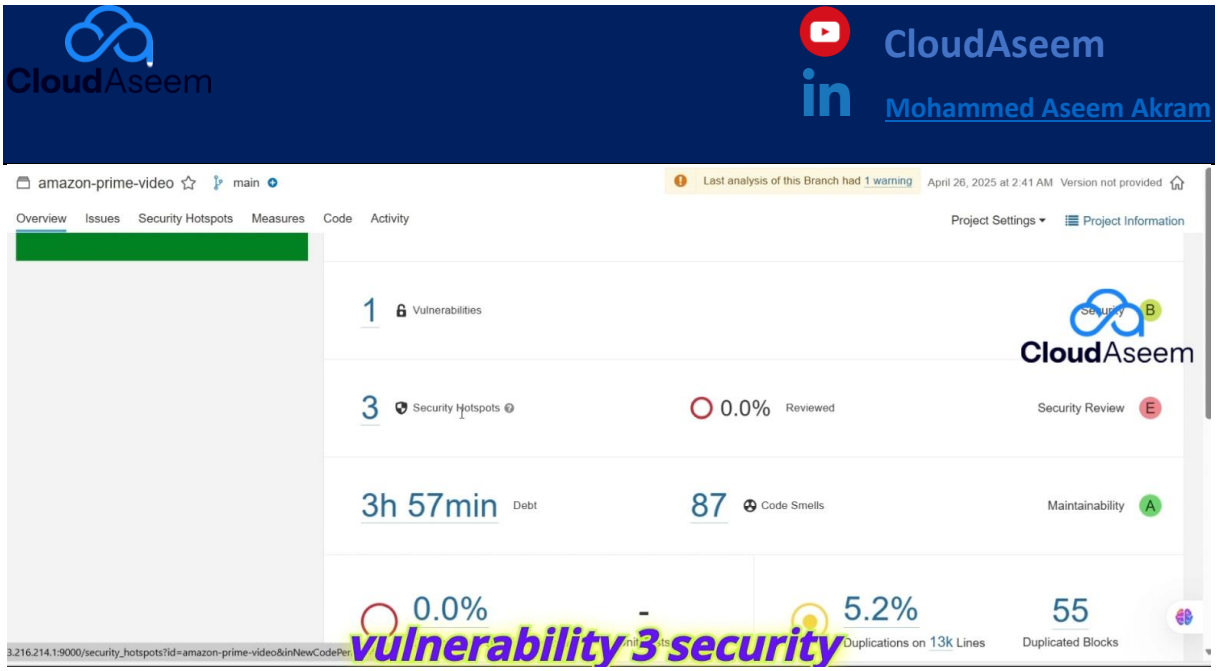
2. Verify Docker Images in Dockerhub



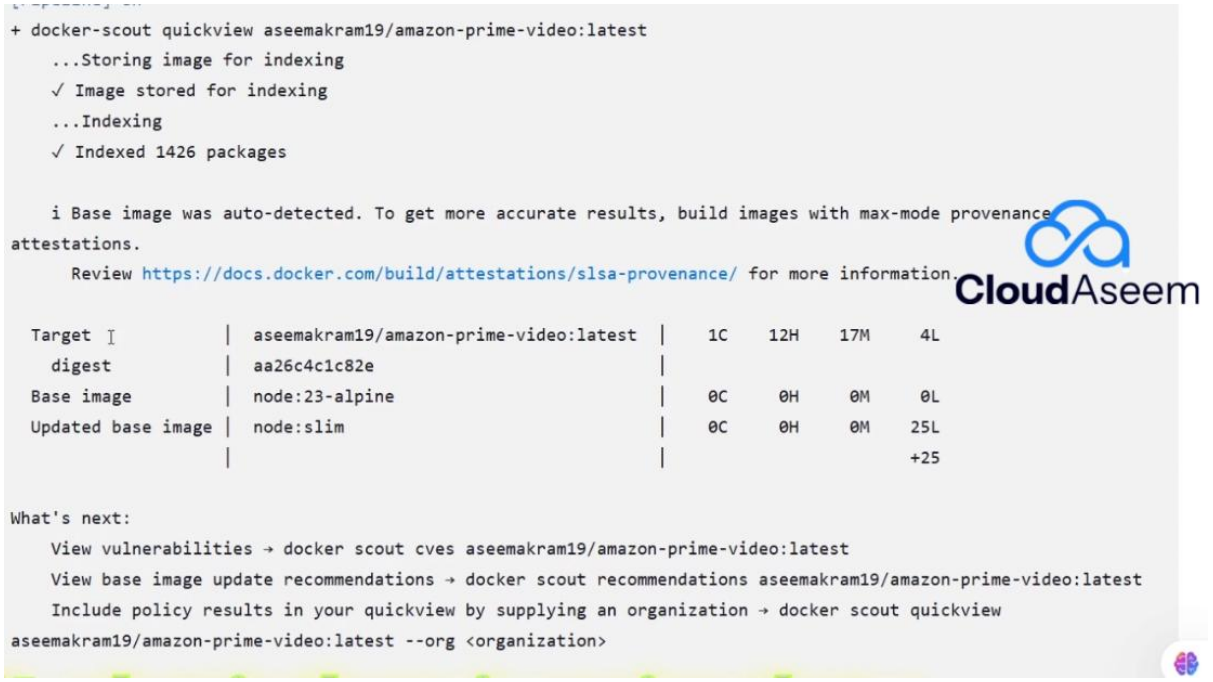
2. SonarQube Project scanned :

You can see the report has been generated and the status shows as passed. You can see that there are 13k lines it scanned. To see a detailed report, you can go to issues.
You will see that in status, a graph will also be generated and Vulnerabilities.





Docker Scout review

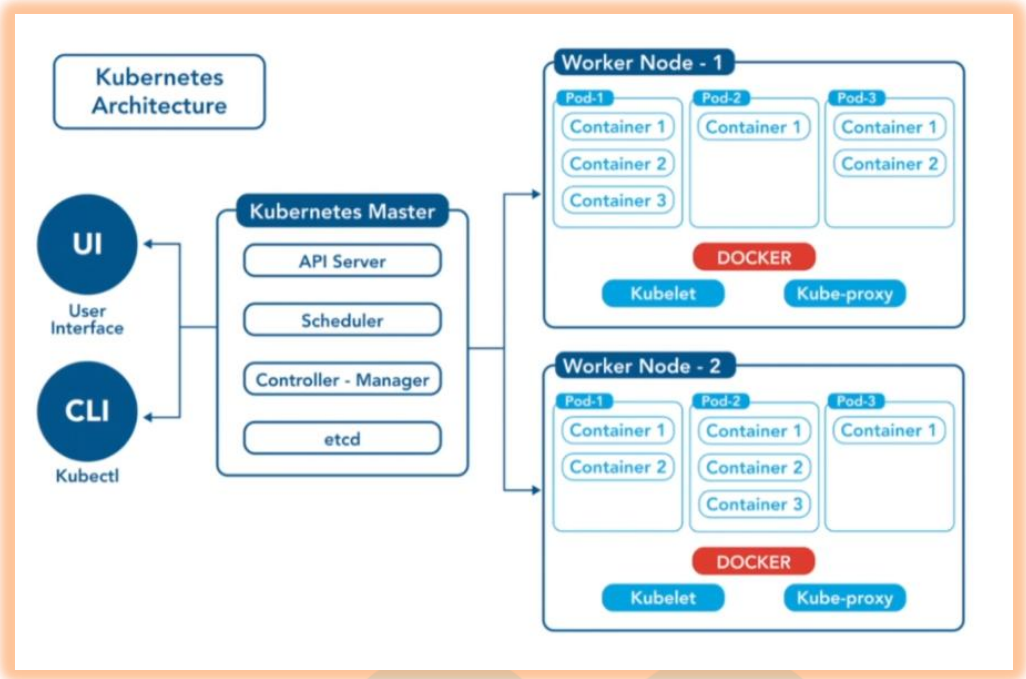


Stage 2

Kubernetes

Deploy app to Kubernetes

Kubernetes architecture – EKS



🔸 Kubernetes Components (in General and in EKS)

✅ 1. Control Plane (Managed by AWS in EKS)

Responsible for maintaining the desired state of the cluster:

API Server: Frontend of Kubernetes; receives commands (kubectl, REST API).

etcd: Key-value store that holds cluster state and config.

Controller Manager: Ensures the cluster state matches the desired state (e.g., if a pod dies, it restarts it).

Scheduler: Assigns workloads (pods) to worker nodes based on resource availability.

🔹 In EKS: AWS manages the control plane (you don't have to worry about installing, updating, or scaling it).

✅ 2. Worker Nodes (Data Plane)

These are EC2 instances (or Fargate pods) that run your containerized applications:

Kubelet: Agent that runs on each node, communicates with the control plane.

Kube-proxy: Handles networking and forwards traffic inside the cluster.

Container runtime: Like Docker or containerd — runs containers.

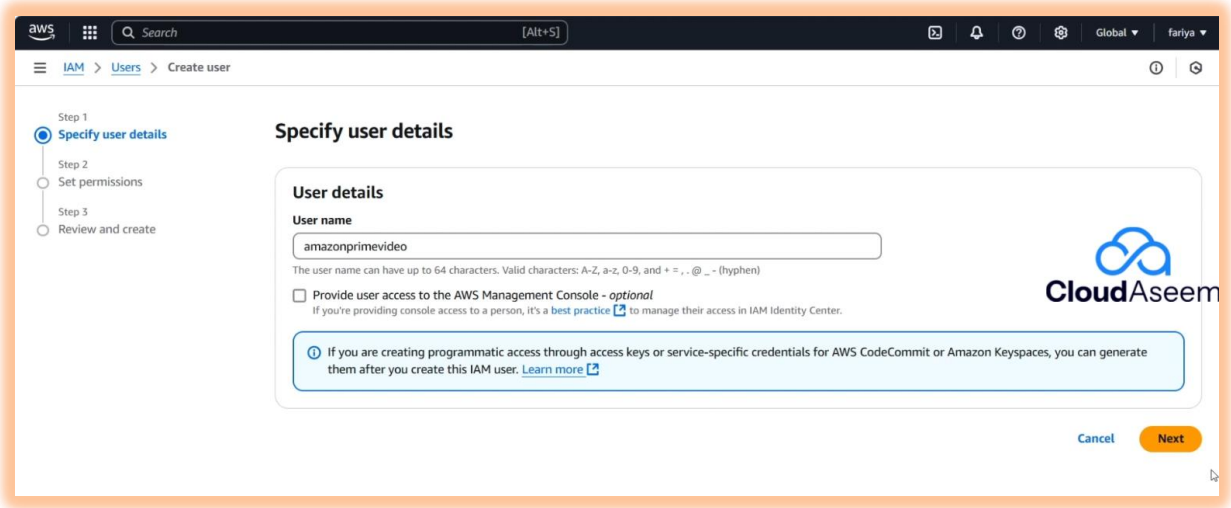
- In EKS: You manage the worker nodes. You can use:

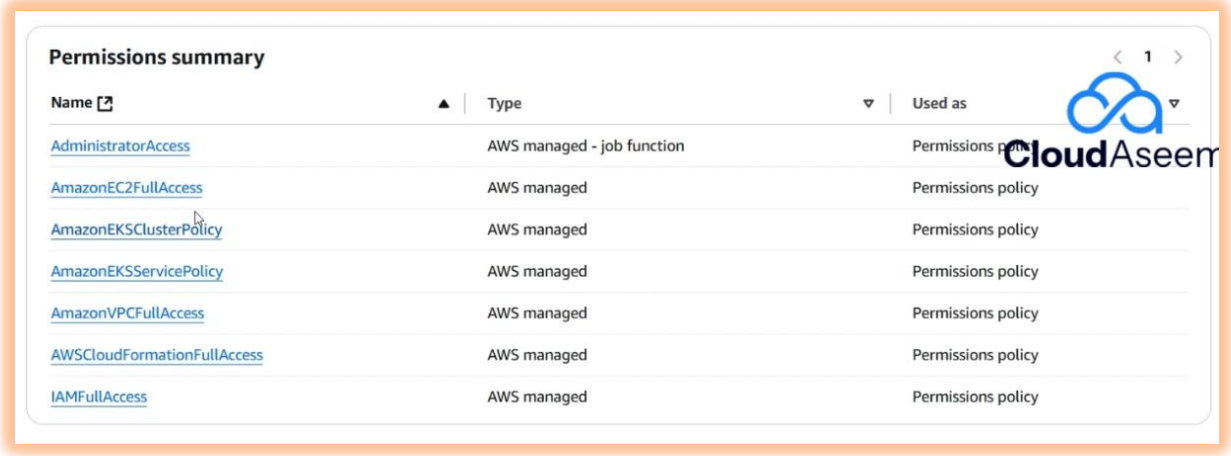
Managed Node Groups (recommended)
To set up the **AWS Access Key** and **Secret Access Key** for configuring AWS EKS from an EC2 instance, follow these steps:

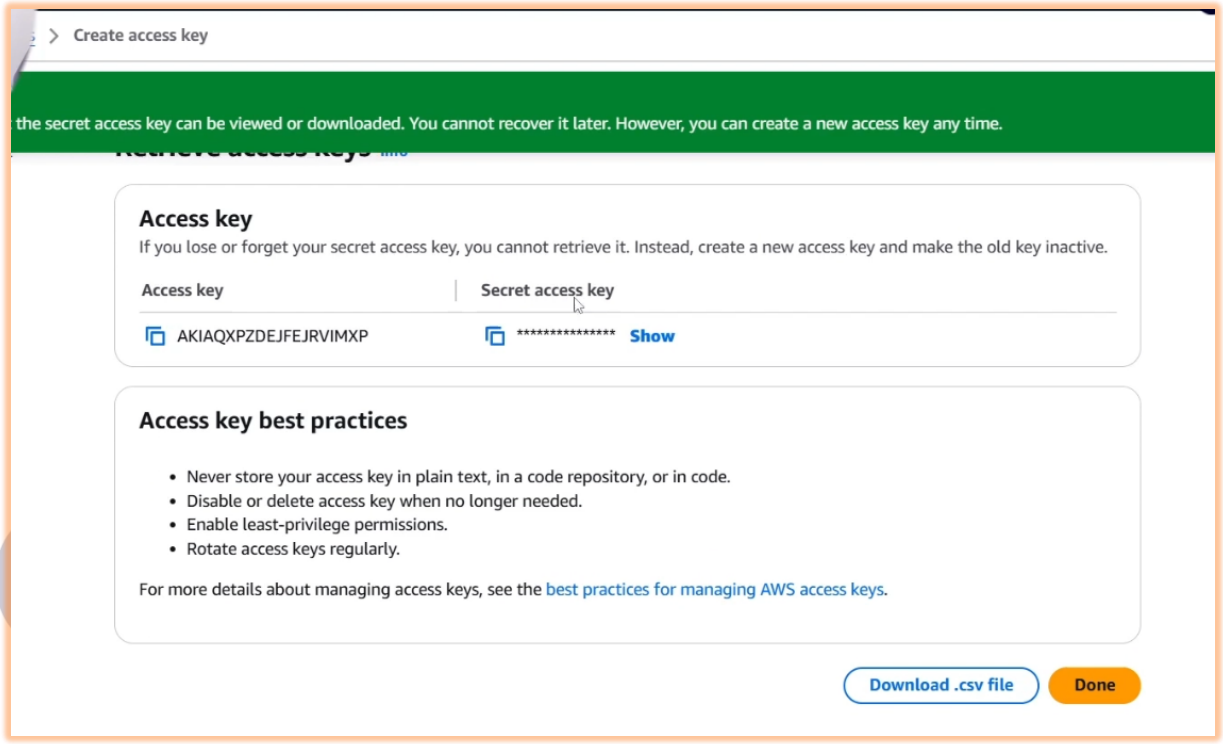
1. Create IAM User with EKS Permissions

1. **Log in to the AWS Management Console**.
2. **Go to IAM (Identity and Access Management)**.
3. **Create a new IAM user**:
   - Click on **Users** from the left sidebar and then click **Add user**.
   - Provide a username (e.g., `eks-setup-user`).
   - Choose **Programmatic access** to allow the user to access AWS via CLI or SDK.
4. **Attach permissions**:
   - Attach the **AmazonEKSClusterPolicy** and **AmazonEKSServicePolicy** for managing EKS clusters.
   - Optionally, you can attach other permissions such as `AmazonEC2FullAccess` if your user needs access to EC2 instances for EKS worker nodes.
5. **Review and create the user**.
6. **Save the Access Key ID and Secret Access Key**:
   - Once the user is created, you'll be provided with an **Access Key ID** and **Secret Access Key**. Save these credentials securely.

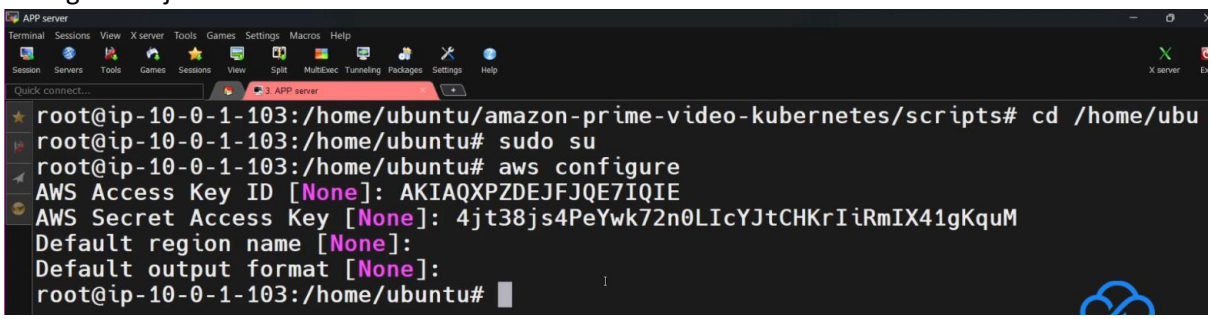
Create IAM user for AWS credentials to auth for EKS cluster setup







Configure on jenkins server



Create EKS Cluster from Jenkins server

Execute the below commands as separate set

(a)

```
eksctl create cluster --name=Cloudaseem \
                --region=ap-south-1 \
                --zones=ap-south-1a,ap-south-1b \
                --version=1.30 \
                --without-nodegroup
```

It will take 5-10 minutes to create the cluster
Goto EKS Console and verify the cluster.

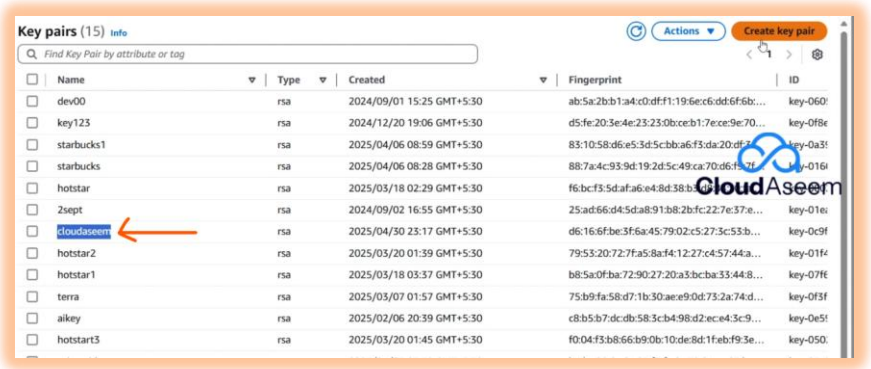
(b)

```
eksctl utils associate-iam-oidc-provider \
   --region ap-south-1 \
   --cluster Cloudaseem \
   --approve
```

The above command is crucial when setting up an EKS cluster because it enables IAM roles for service accounts (IRSA)
Amazon EKS uses OpenID Connect (OIDC) to authenticate Kubernetes service accounts with IAM roles.
Associating the IAM OIDC provider allows Kubernetes workloads (Pods) running in the cluster to assume IAM roles securely.
Without this, Pods in EKS clusters would require node-level IAM roles, which grant permissions to all Pods on a node.
Without this, these services will not be able to access AWS resources securely.

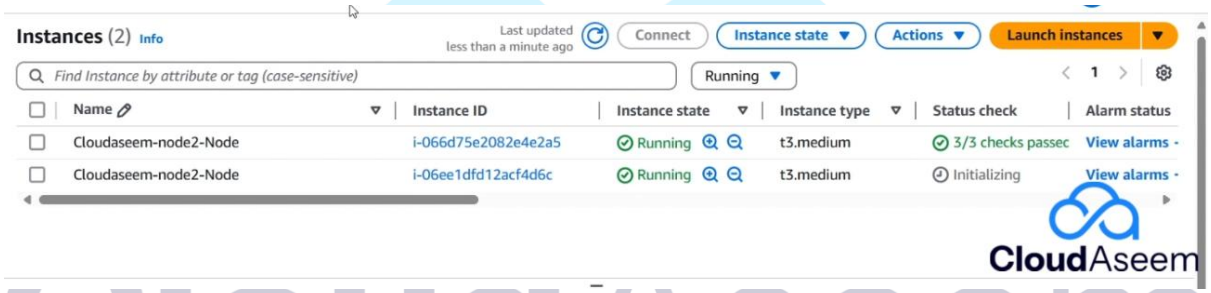
(c)

Before executing the below command, in the 'ssh-public-key' keep the '<PEM FILE NAME>' (dont give .pem. Just give the pem file name) which was used to create Jenkins Server



```
eksctl create nodegroup --cluster=Cloudaseem \
                --region=ap-south-1 \
                --name=node2 \
                --node-type=t3.medium \
                --nodes=2 \
                --nodes-min=2 \
                --nodes-max=4 \
                --node-volume-size=20 \
                --ssh-access \
                --ssh-public-key=cloudaseem \
                --managed \
                --asg-access \
                --external-dns-access \
                --full-ecr-access \
                --appmesh-access \
                --alb-ingress-access
```

It will take 5-10 minutes



Jenkins Pipeline 2 deploy to EKS cluster

Why add AWS credentials for the `jenkins` user to deploy to EKS?

When you deploy to **Amazon EKS from a Jenkins pipeline**, Jenkins needs **authenticated access** to your AWS account and **authorization to interact with the EKS cluster** — just like any AWS CLI user.

🔑 Here's why credentials are needed under the `jenkins` user:

✅ 1. Jenkins Runs Pipelines as the `jenkins` User

- When Jenkins executes a pipeline, it runs all commands as the `jenkins` user (not `ubuntu`, not `root`).

- So, AWS CLI commands inside the pipeline (`aws eks update-kubeconfig`, `kubectl apply`, etc.) will **fail** unless the `jenkins` user has access to AWS credentials.

✅ 2. AWS CLI Requires `~/.aws/credentials` for Auth

- AWS CLI looks for credentials in:

  `/home/jenkins/.aws/credentials`

- If not present, you'll get errors like:

  `Unable to locate credentials`

✅ 3. Required for `aws eks update-kubeconfig`

This command is used to authenticate to an EKS cluster:

```
aws eks --region us-east-1 update-kubeconfig --name my-eks-cluster
```

It:

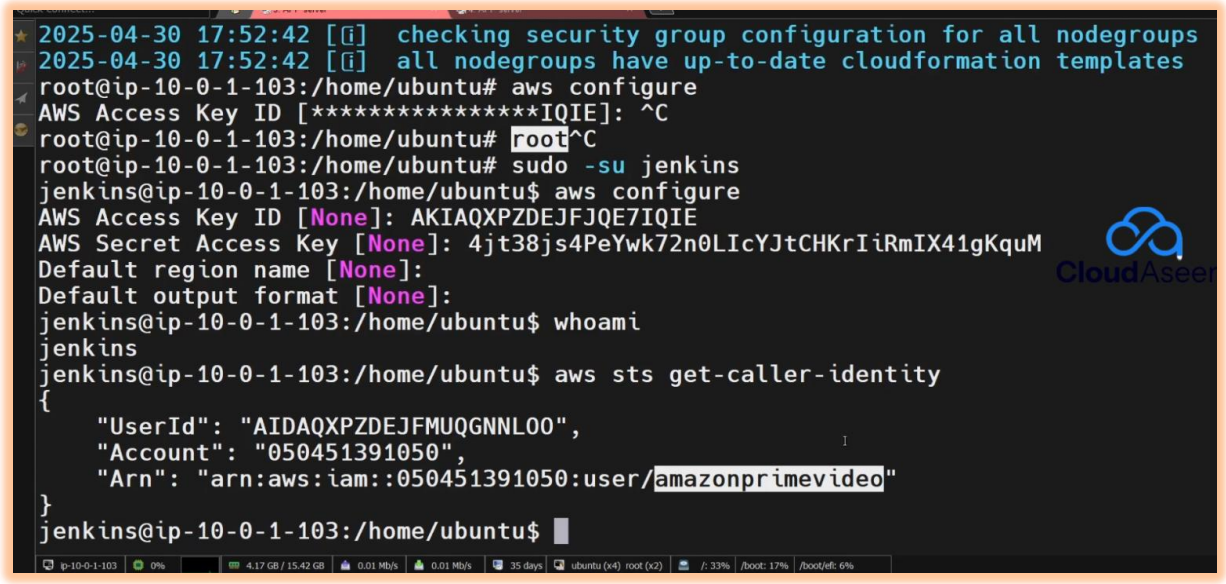
- Uses AWS credentials to call EKS APIs
- Writes a kubeconfig file for `kubectl` in `/var/lib/jenkins/.kube/config`

Without AWS credentials, **this step will fail** and `kubectl` won't be able to talk to the cluster.

Switch to the jenkins user
sudo -su jenkins
pwd ---- /home/ubuntu
whoami ---- jenkins

Configure AWS credentials:
aws configure ---> Configure with access and secret access keys
This will create the AWS credentials file at "/var/lib/jenkins/.aws/credentials"

Verify the credentials



aws sts get-caller-identity
If the credentials are valid, you should see output like this:

```
{
   "UserId": "EXAMPLEUSERID",
   "Account": "123456789012",
   "Arn": "arn:aws:iam::123456789012:user/example-user"
}
```

Comeout of the Jenkins user to Restart Jenkins
exit
sudo systemctl restart jenkins

Switch to Jenkins user
sudo -su jenkins
aws eks update-kubeconfig --region ap-south-1 --name Cloudaseem

add this New stage in jenkins file

```
pipeline{
   agent any
   stages {
      stage('clean workspace'){
         steps{
            cleanWs()
         }
      }
      stage('Checkout from Git'){
         steps{
            git branch: 'main', url: 'https://github.com/Aseemakram19/amazon-prime-video-kubernetes.git'
```

```
        }
      }
      stage('Deploy to EKS Cluster') {
        steps {
          dir('kubernetes') {
          script {
            sh '''
            echo "Verifying AWS credentials..."
            aws sts get-caller-identity

            echo "Configuring kubectl for EKS cluster..."
            aws eks update-kubeconfig --region ap-south-1 --name Cloudaseem

            echo "Verifying kubeconfig..."
            kubectl config view

            echo "Deploying application to EKS..."
            kubectl apply -f manifest.yml

            echo "Verifying deployment..."
            kubectl get pods
            kubectl get svc
            '''
          }
        }
       }

     }

   }
   post {
   always {
     script {
       def buildStatus = currentBuild.currentResult
       def buildUser = currentBuild.getBuildCauses('hudson.model.Cause$UserIdCause')[0]?.userId
?: 'Github User'

       emailext (
         subject: "Pipeline ${buildStatus}: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
         body: """
           <p>This is a Jenkins amazon-prime-video CICD pipeline status.</p>
           <p>Project: ${env.JOB_NAME}</p>
           <p>Build Number: ${env.BUILD_NUMBER}</p>
           <p>Build Status: ${buildStatus}</p>
           <p>Started by: ${buildUser}</p>
           <p>Build URL: <a href="${env.BUILD_URL}">${env.BUILD_URL}</a></p>
```

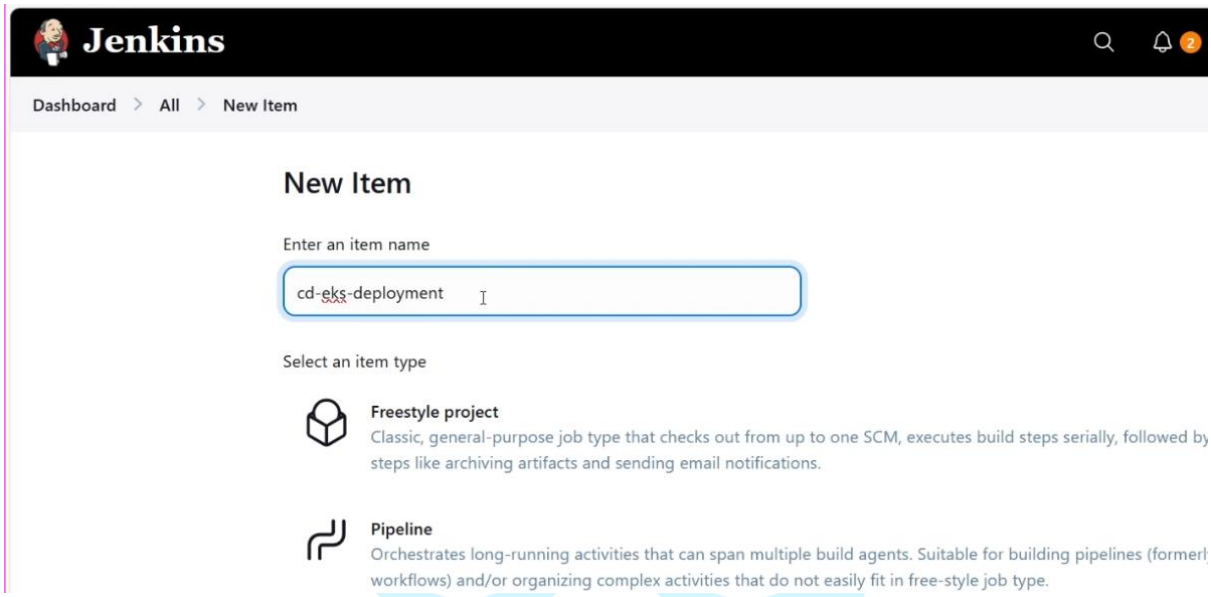
```
            """,
            to: 'mohdaseemakram19@gmail.com',
            from: 'mohdaseemakram19@gmail.com',
            replyTo: 'mohdaseemakram19@gmail.com',
            mimeType: 'text/html',
            attachmentsPattern: 'trivyfs.txt,trivyimage.txt'
          )
        }
      }

    }

}
```

File exist ,
Note to update this manifest.yml file with your Docker images name and apply

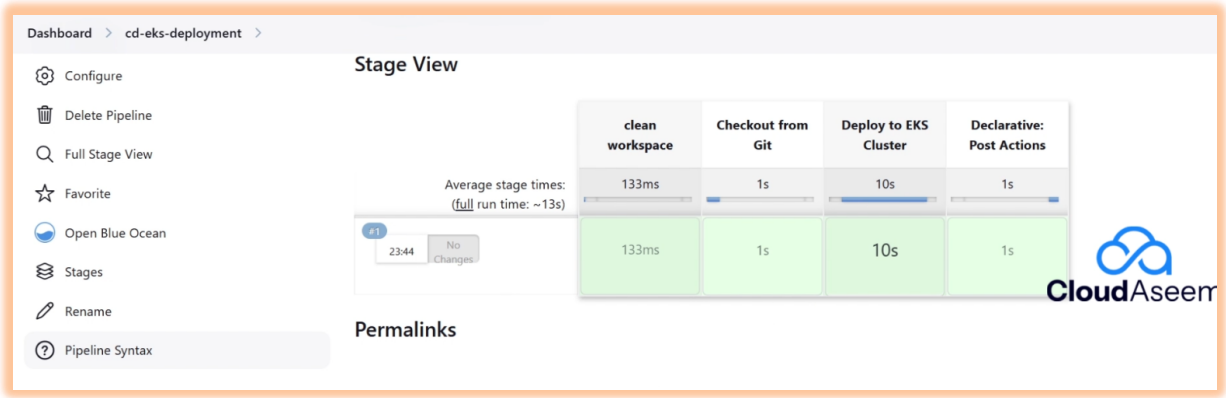
manifest.yml file

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: amazon-prime-video-deployment
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: amazon-prime-video
  template:
    metadata:
      labels:
        app: amazon-prime-video
    spec:
      containers:
      - name: amazon-prime-video-container
        image: aseemakram19/amazon-prime-video
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
```

```
  name: amazon-prime-video-service
spec:
  type: LoadBalancer
  selector:
    app: amazon-prime-video
  ports:
  - port: 80
    targetPort: 3000
```



and execute the pipeline now



kubectl get all

```
679b03a4cec8be37d-602414080.ap-south-1.elb.amazonaws.com   80:31908/TCP   2m33s
service/kubernetes                  ClusterIP      10.100.0.1      <none>
                                                   443/TCP         39m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AG
E
deployment.apps/amazon-prime-video-deployment   2/2     2            2           2m
34s

NAME                                                    DESIRED   CURRENT   READ
Y   AGE
replicaset.apps/amazon-prime-video-deployment-58dc6fc55b   2         2         2
    2m34s
root@ip-10-0-1-103:/home/ubuntu# kubectl get all
NAME                                             READY   STATUS    RESTARTS   AGE
pod/amazon-prime-video-deployment-58dc6fc55b-bhr85   1/1     Running   0          2m40s
pod/amazon-prime-video-deployment-58dc6fc55b-c5rr6   1/1     Running   0          2m40s

NAME                                  TYPE           CLUSTER-IP      EXTERNAL-IP
                PORT(S)        AGE
service/amazon-prime-video-service    LoadBalancer   10.100.93.189   aa031e28de87d48579b03a4cec8be37d-602414080.ap-south-1.e
lb.amazonaws.com   80:31908/TCP   2m41s
service/kubernetes                    ClusterIP      10.100.0.1      <none>
                443/TCP        39m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/amazon-prime-video-deployment   2/2     2            2           2m41s

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/amazon-prime-video-deployment-58dc6fc55b   2         2         2       2m41s
root@ip-10-0-1-103:/home/ubuntu#
```

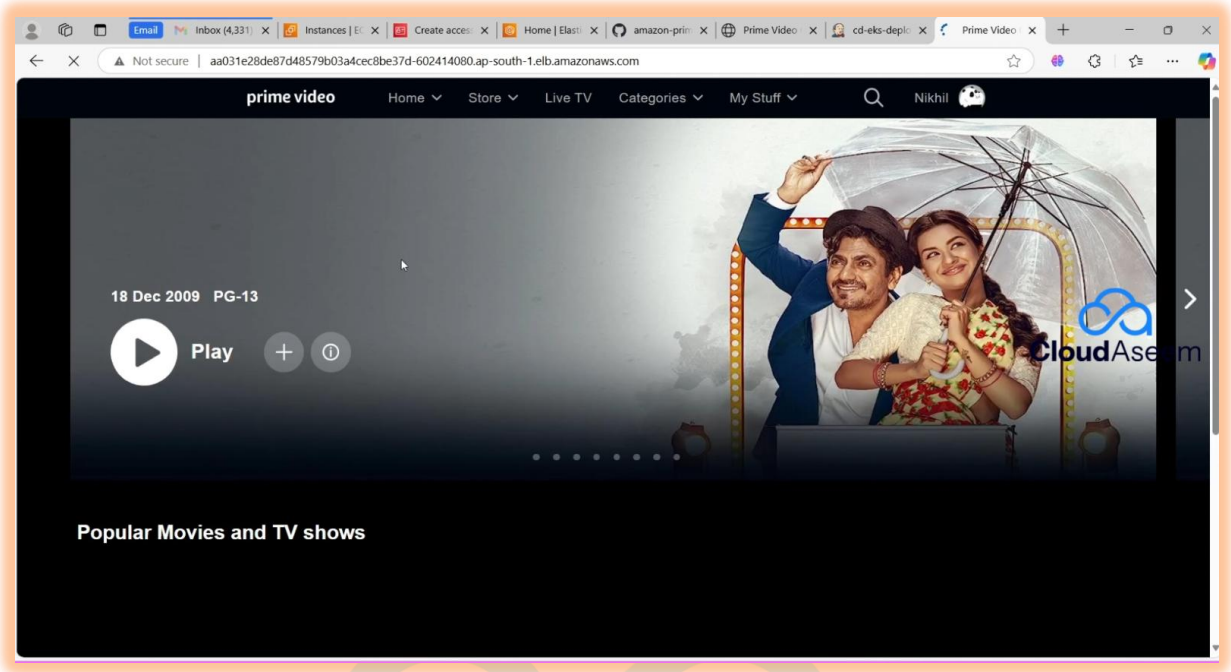
Test Kubernetes Auto Healing feature

Manually delete the Pod:

Observe auto-healing:
Kubernetes will automatically recreate the pod to maintain the desired state.
You can verify with:
kubectl get pods -w

```
root@ip-10-0-1-103:/home/ubuntu# kubectl get pods
NAME                                             READY   STATUS    RESTARTS   AGE
amazon-prime-video-deployment-58dc6fc55b-bhr85   1/1     Running   0          3m35s
amazon-prime-video-deployment-58dc6fc55b-c5rr6   1/1     Running   0          3m35s
root@ip-10-0-1-103:/home/ubuntu# kubectl delete amazon-prime-video-deployment-58dc6fc55b-bhr85
error: the server doesn't have a resource type "amazon-prime-video-deployment-58dc6fc55b-bhr85"
root@ip-10-0-1-103:/home/ubuntu# kubectl delete pod  amazon-prime-video-deployment-58dc6fc55b-bhr85
pod "amazon-prime-video-deployment-58dc6fc55b-bhr85" deleted
root@ip-10-0-1-103:/home/ubuntu# kubectl get pods
NAME                                             READY   STATUS    RESTARTS   AGE
amazon-prime-video-deployment-58dc6fc55b-c5rr6   1/1     Running   0          4m51s
amazon-prime-video-deployment-58dc6fc55b-jfdjj   1/1     Running   0          25s
root@ip-10-0-1-103:/home/ubuntu#
```

Access app with Loadbalancer

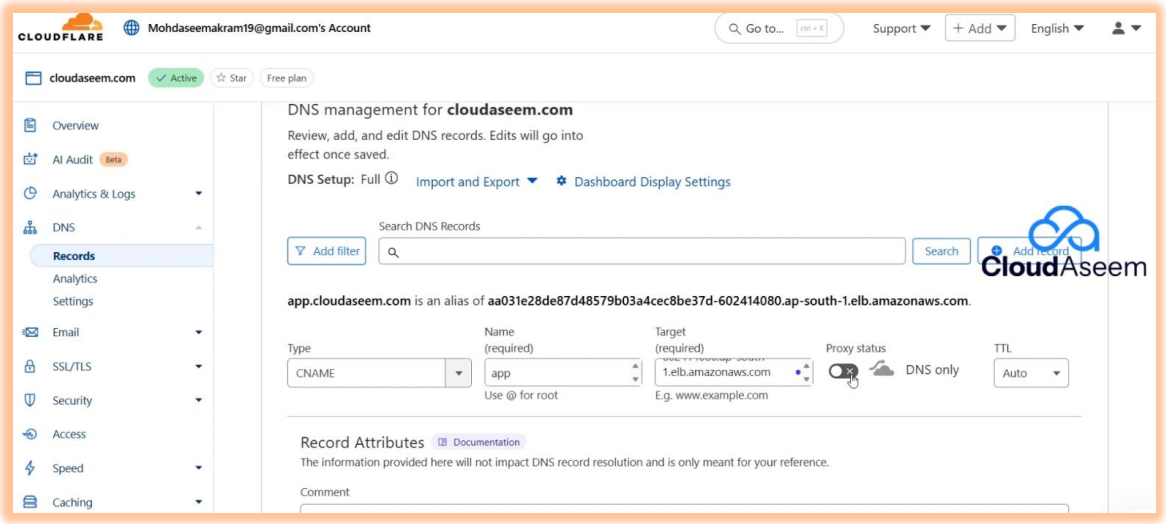


Add Domain to Our application

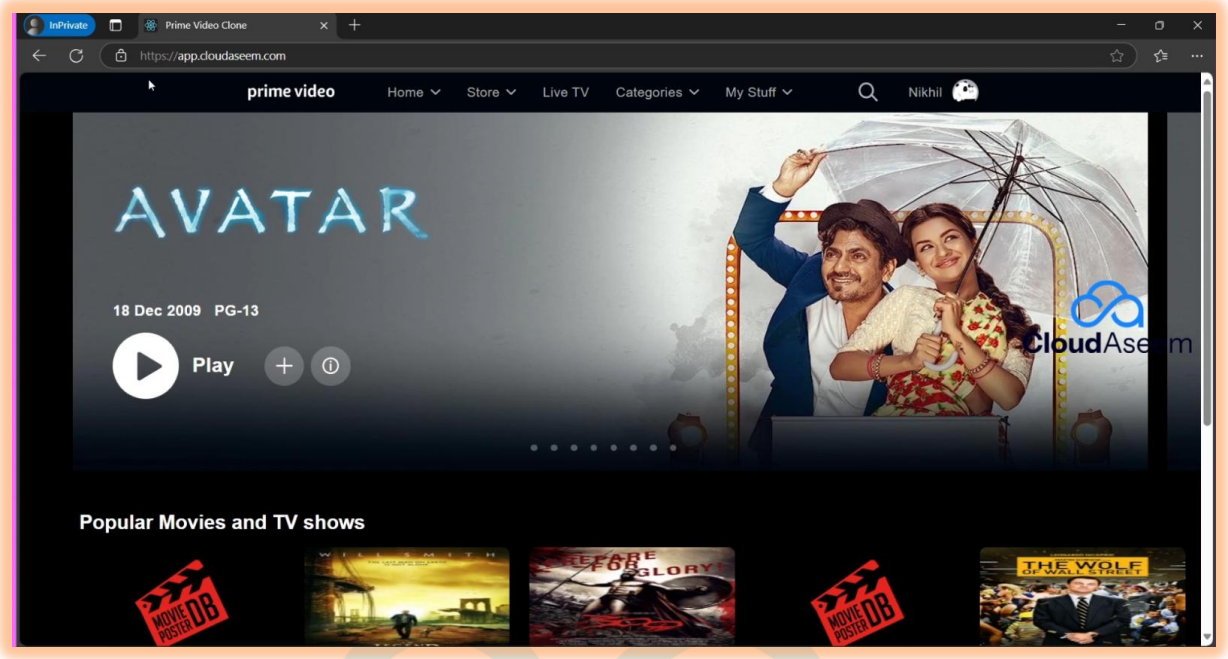
Add Loadbalance in Cloudflare to apply domain and ssl to access by client with cname entry

Configure CNAME for Client Access

1. **Go to** Cloudflare **DNS Settings**.
2. Add a **CNAME Record**:
   - **Name**: app.example.com
   - **Target**: Load Balancer domain (e.g., lb.example.com)
   - **Proxy Status**: **Proxied (Orange Cloud)** for Cloudflare SSL.
3. Save and Test.



You have successfully Deployed a Starbucks Kubernetes with Loadbalancer Enabled and SSL certificated

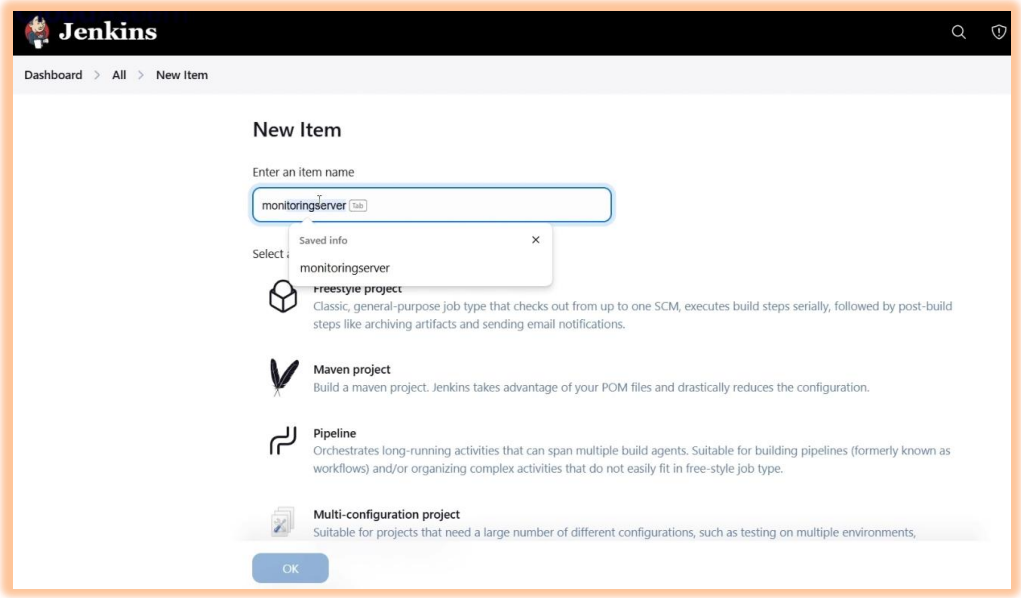


Part – 03

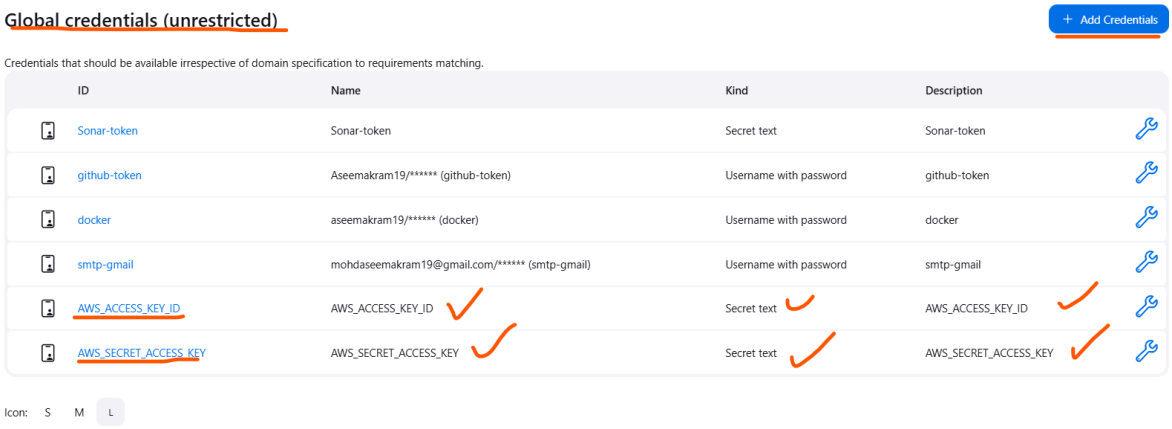
monitoring server with JENKINS + TERRAFORM = MONITORING SERVER SETUP

Create a new pipeline as below :

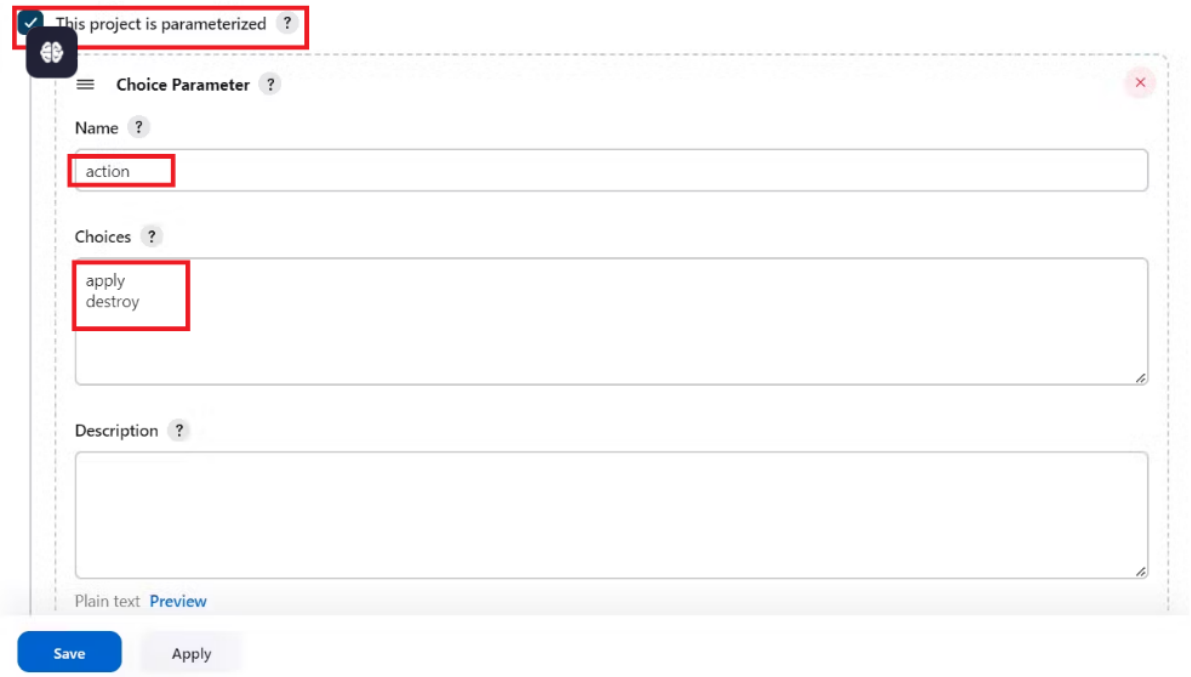
Monitoring Server setup with Jenkins + Terraform



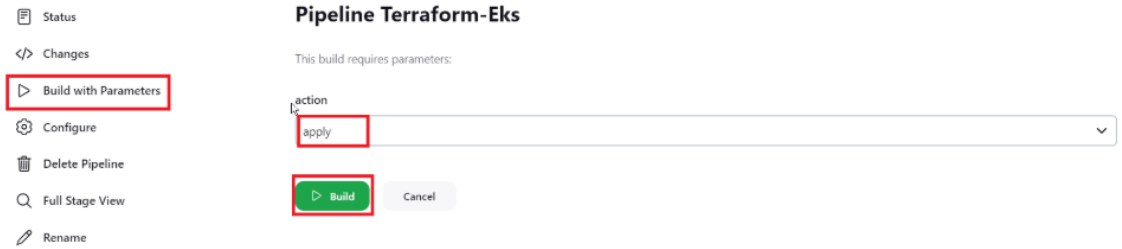
add Secret in AWS credentials



I want to do this with build parameters to apply and destroy while building only.
you have to add this inside job like the below image



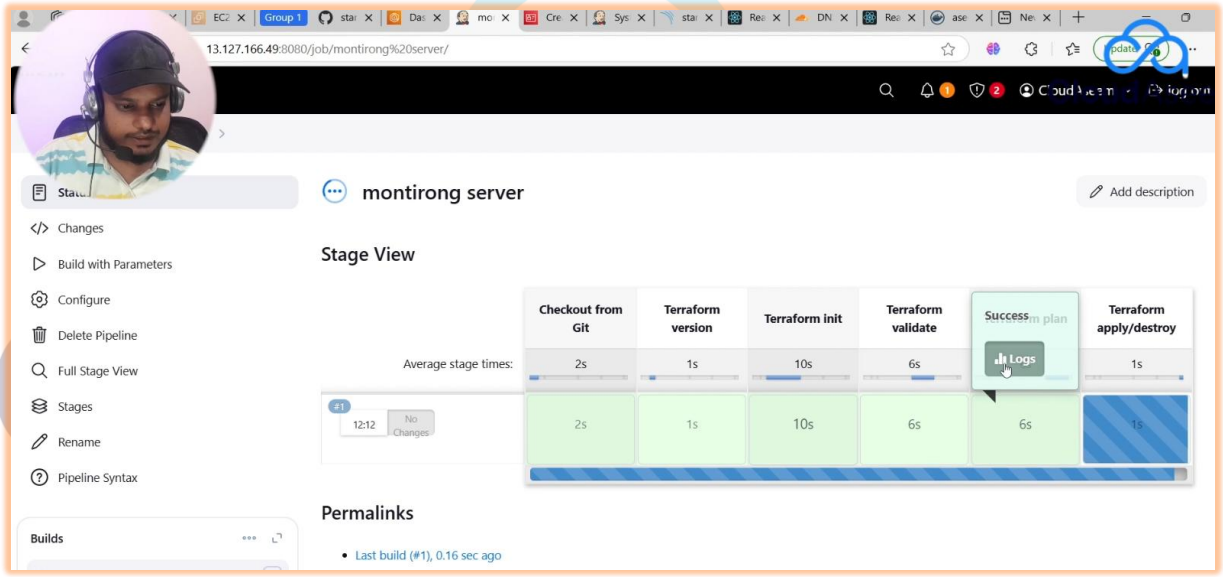
Let's apply and save and Build with parameters and select action as apply



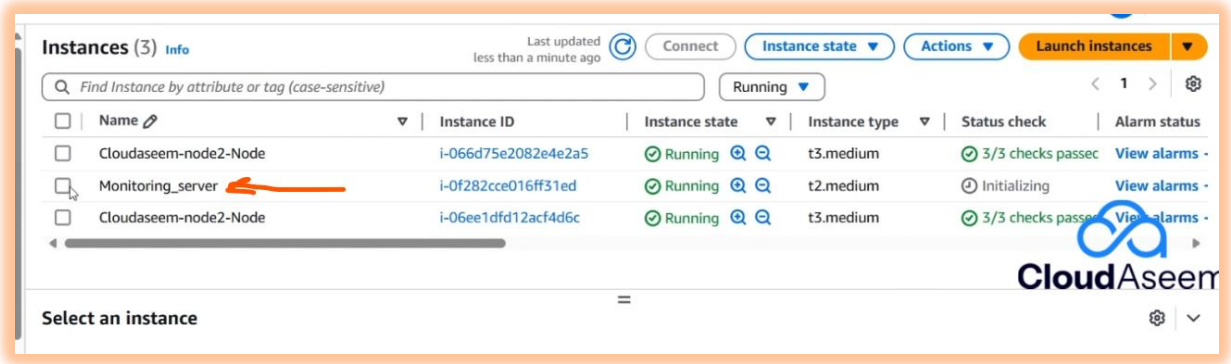
```
pipeline {
  agent any

  environment {
    AWS_ACCESS_KEY_ID     = credentials('AWS_ACCESS_KEY_ID')
    AWS_SECRET_ACCESS_KEY = credentials('AWS_SECRET_ACCESS_KEY')
  }

  parameters {
    string(name: 'action', defaultValue: 'apply', description: 'terraform action: apply or destroy')
  }

  stages {
    stage('Checkout from Git') {
      steps {
        git branch: 'main', credentialsId: 'github-token', url: 'https://github.com/Aseemakram19/amazon-prime-video-kubernetes.git'
      }
    }

    stage('terraform version') {
      steps {
        sh 'terraform --version'
      }
    }

    stage('terraform init') {
      steps {
        dir('terraform') {
          sh '''
          terraform init \
          -backend-config="access_key=$AWS_ACCESS_KEY_ID" \
          -backend-config="secret_key=$AWS_SECRET_ACCESS_KEY"
          '''
        }
      }
    }

    stage('terraform validate') {
      steps {
        dir('terraform') {
          sh 'terraform validate'
        }
      }
    }

    stage('terraform plan') {
      steps {
        dir('terraform') {
          sh '''
          terraform plan \
          -var="access_key=$AWS_ACCESS_KEY_ID" \
          -var="secret_key=$AWS_SECRET_ACCESS_KEY"
          '''
        }
      }
    }

    stage('terraform apply/destroy') {
      steps {
        dir('terraform') {
          sh '''
          terraform ${action} --auto-approve \
```

```
            -var="access_key=$AWS_ACCESS_KEY_ID" \
            -var="secret_key=$AWS_SECRET_ACCESS_KEY"
            '''
          }
        }
      }
    }

    post {
      success {
        echo '✅ terraform execution completed successfully!'
      }
      failure {
        echo '❌ terraform execution failed! Check the logs.'
      }
    }
}
```

Note: Store aws secret and secret access key in jenkins crednetials s below



Note: Use same Key pair to access monitoring server
Verify the Monitoring server



Installing Grafana and Prometheus for Monitoring

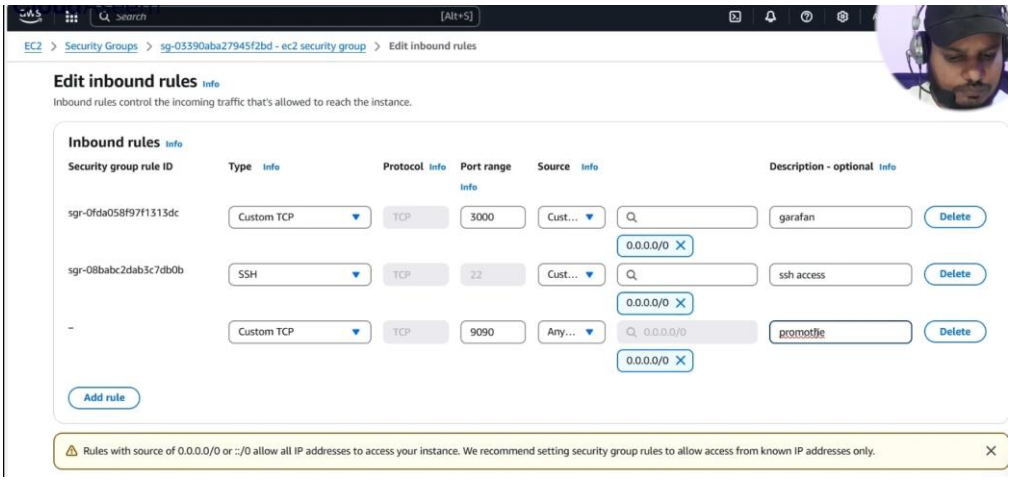
Grafana and Prometheus are commonly used for monitoring Kubernetes clusters, EC2 instances, and other infrastructure components. Follow these steps to install them on an **Ubuntu** server.

3. **Grafana installation**

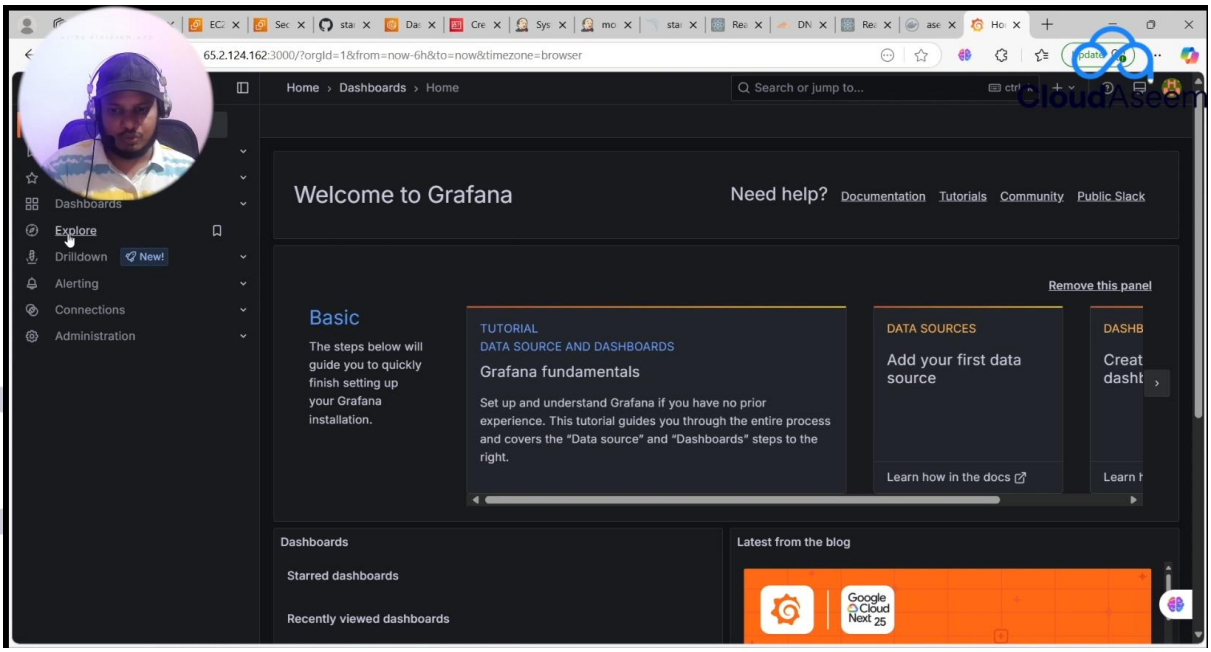
   Access and create a garafan.sh , add permission , and xecute it

```
#!/bin/bash
# Script to install Grafana on a Linux instance

# Update package list and install dependencies
sudo apt-get install -y apt-transport-https software-properties-common wget

# Create a directory for Grafana's GPG key
sudo mkdir -p /etc/apt/keyrings/

# Add Grafana's GPG key
wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee
/etc/apt/keyrings/grafana.gpg > /dev/null

# Add Grafana's repository to the sources list
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" |
sudo tee -a /etc/apt/sources.list.d/grafana.list

# Update package lists
sudo apt-get update -y
# Install the latest OSS release of Grafana
sudo apt-get install grafana -y

# Start and enable Grafana service
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

   Open ports in SG group



After installation, you can access Grafana at:
http://your-server-ip:3000 (default user: admin, password: admin)



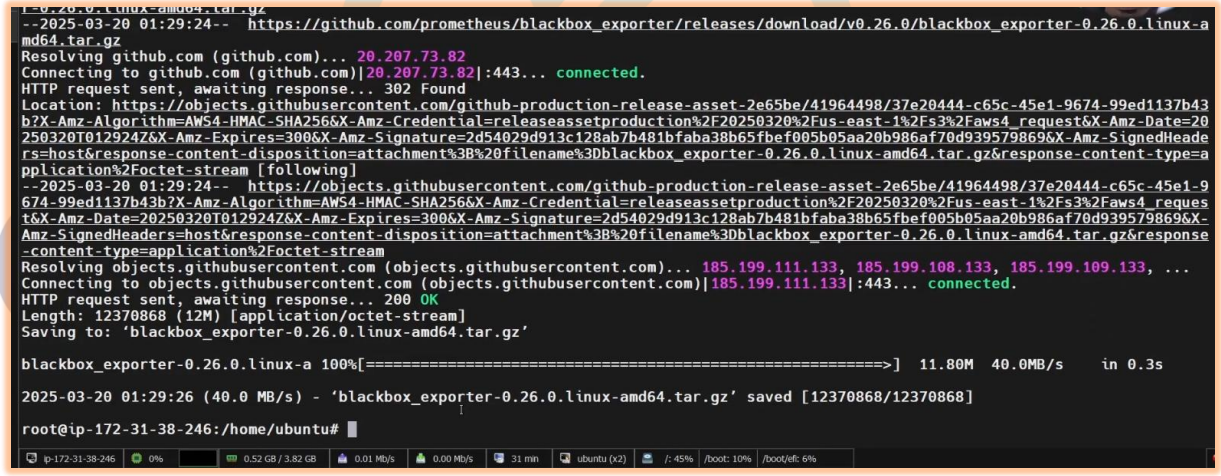
4. **Install Prometheus**

```
#!/bin/bash
#this script belong to "CLOUDASEEM" YOUTUBE CHANNEL

# Define Prometheus version
PROMETHEUS_VERSION="2.51.2"

# Update system and install necessary packages
echo "Updating system and installing dependencies..."
sudo apt update -y
```

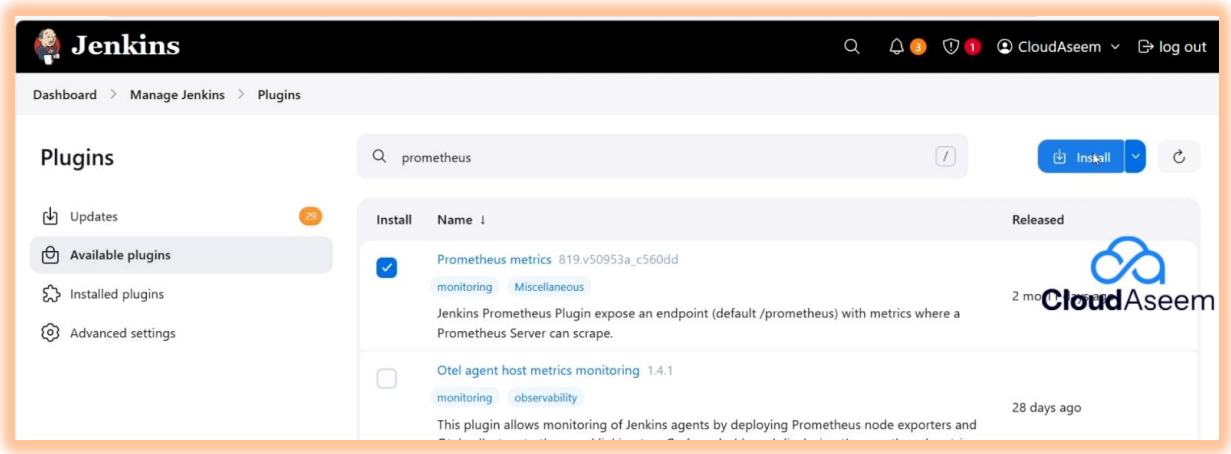
```
sudo apt install -y wget tar

# Create Prometheus user
echo "Creating Prometheus user..."
sudo useradd --no-create-home --shell /bin/false prometheus

# Create Prometheus directory
echo "Creating /etc/prometheus directory..."
sudo mkdir -p /etc/prometheus

# Download and extract Prometheus
echo "Downloading and extracting Prometheus..."
cd /tmp
wget
https://github.com/prometheus/prometheus/releases/download/v${PROMETHEUS_VERSION}/pr
ometheus-${PROMETHEUS_VERSION}.linux-amd64.tar.gz
tar -xvzf prometheus-${PROMETHEUS_VERSION}.linux-amd64.tar.gz

# Move all extracted files to /etc/prometheus
echo "Moving Prometheus files to /etc/prometheus..."
sudo mv prometheus-${PROMETHEUS_VERSION}.linux-amd64/* /etc/prometheus/

# Set ownership and permissions
echo "Setting permissions..."
sudo chown -R prometheus:prometheus /etc/prometheus

# Create symbolic links for binaries
echo "Creating symlinks for Prometheus binaries..."
sudo ln -s /etc/prometheus/prometheus /usr/local/bin/prometheus
sudo ln -s /etc/prometheus/promtool /usr/local/bin/promtool

# Create Prometheus systemd service
echo "Creating systemd service..."
cat <<EOF | sudo tee /etc/systemd/system/prometheus.service
[Unit]
Description=Prometheus Monitoring System
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \\
  --config.file=/etc/prometheus/prometheus.yml \\
  --storage.tsdb.path=/etc/prometheus/data \\
```

```
 --web.console.templates=/etc/prometheus/consoles \\
 --web.console.libraries=/etc/prometheus/console_libraries

Restart=always

[Install]
WantedBy=multi-user.target
EOF

# Reload systemd, enable and start Prometheus
echo "Starting Prometheus..."
sudo systemctl daemon-reload
sudo systemctl enable prometheus
sudo systemctl start prometheus

# Check Prometheus status
echo "Prometheus installation completed!"
sudo systemctl status prometheus --no-pager
```

Install Blackbox exporter



Note:
install Prometheus plugin in Jenkins server



Step 1: Edit prometheus.yml
Open the **Prometheus configuration file**:

Add the following **scrape jobs** at the end of the file:

```
- job_name: 'blackbox'
  metrics_path: /probe
  params:
    module: [http_2xx]  # Look for a HTTP 200 response.
  static_configs:
    - targets:
        - http://prometheus.io     # Target to probe with HTTP.
        - http://IP:3000 # Target to probe with HTTPS.
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: 13.232.214.2:9115  # The blackbox exporter's real hostname.

- job_name: 'jenkins'
  metrics_path: '/prometheus'
  static_configs:
    - targets:
        - 'ip:8080'


- job_name: node_exporter
  static_configs:
    - targets:
        - 'IP:9100'
```

Save the file (CTRL + X, then Y, and Enter).

```
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]

  - job_name: 'blackbox'
    metrics_path: /probe
    params:
      module: [http_2xx]  # Look for a HTTP 200 response.
    static_configs:
      - targets:
          - http://prometheus.io      # Target to probe with HTTP.
          - http://54.81.143.142:3000 # Target to probe with HTTPS.
          - https://hotstar.cloudaseem.com:443 # Target to probe with HTTPS.
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: 13.233.124.65:9115  # The blackbox exporter's real hostname.
```

Step 2: Restart Prometheus to Apply Changes

pgrep Prometheus and kill PID

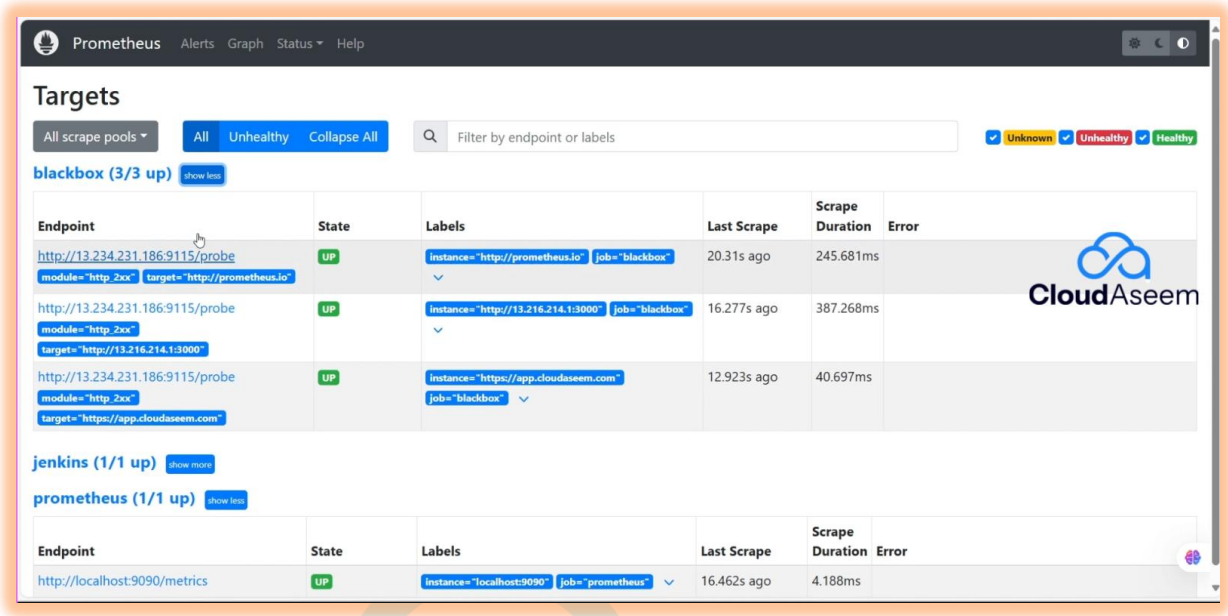
```
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# nano prometheus.yml
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# pgrep prometheus
6166
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# kill 6166
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64# time=2025-03-20T01:35:15.401Z level=WARN source=main.go:1015 msg="
Received an OS signal, exiting gracefully..." signal=terminated
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1040 msg="Stopping scrape discovery manager..."
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1054 msg="Stopping notify discovery manager..."
time=2025-03-20T01:35:15.401Z level=INFO source=manager.go:189 msg="Stopping rule manager..." component="rule manager"
time=2025-03-20T01:35:15.401Z level=INFO source=manager.go:205 msg="Rule manager stopped" component="rule manager"
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1091 msg="Stopping scrape manager..."
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1036 msg="Scrape discovery manager stopped"
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1050 msg="Notify discovery manager stopped"
time=2025-03-20T01:35:15.401Z level=INFO source=main.go:1083 msg="Scrape manager stopped"
time=2025-03-20T01:35:15.405Z level=INFO source=notifier.go:702 msg="Stopping notification manager..." component=notifier
time=2025-03-20T01:35:15.405Z level=INFO source=notifier.go:409 msg="Draining any remaining notifications..." component=notifier
time=2025-03-20T01:35:15.405Z level=INFO source=notifier.go:415 msg="Remaining notifications drained" component=notifier
time=2025-03-20T01:35:15.405Z level=INFO source=notifier.go:345 msg="Notification manager stopped" component=notifier
time=2025-03-20T01:35:15.405Z level=INFO source=main.go:1361 msg="Notifier manager stopped"
time=2025-03-20T01:35:15.406Z level=INFO source=main.go:1375 msg="See you next time!"

[1]-  Done                    ./prometheus
root@ip-172-31-38-246:/home/ubuntu/prometheus-3.2.1.linux-amd64#
```

restart
./Prometheus &

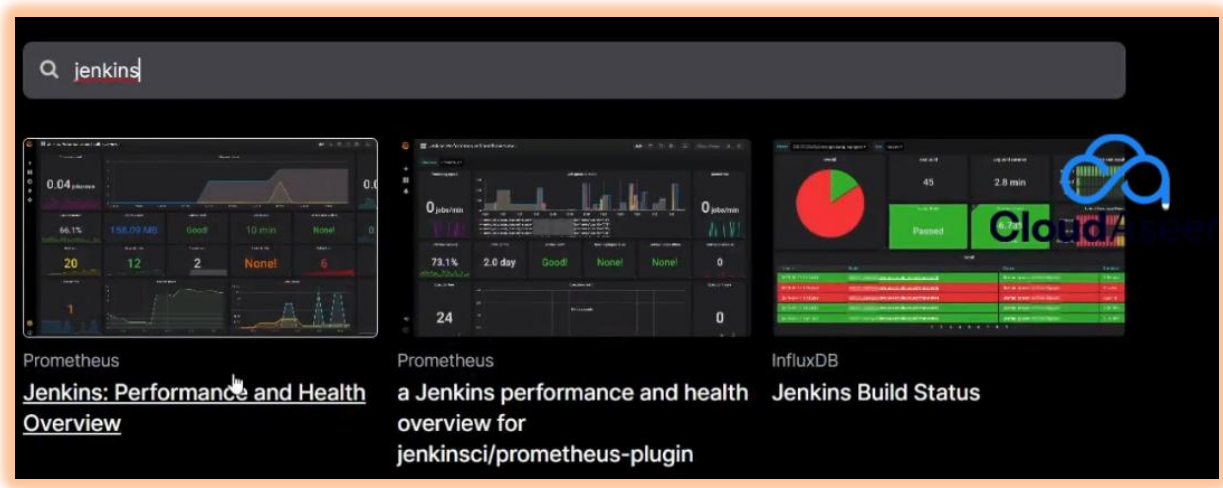
Step 3: Connect Prometheus to Grafana

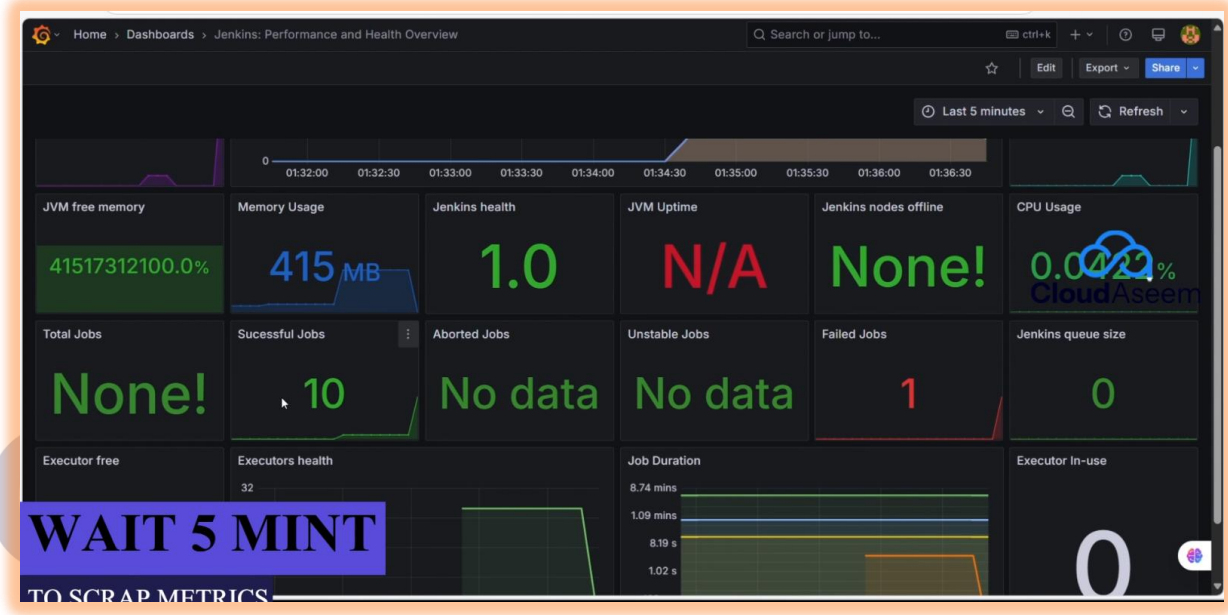
1. **Login to Grafana (http://<your-server-ip>:3000).**
2. **Go to Configuration → Data Sources → Add Data Source.**
3. **Select Prometheus.**
4. **Set Prometheus URL: http://localhost:9090.**

5. **Click Save & Test.**



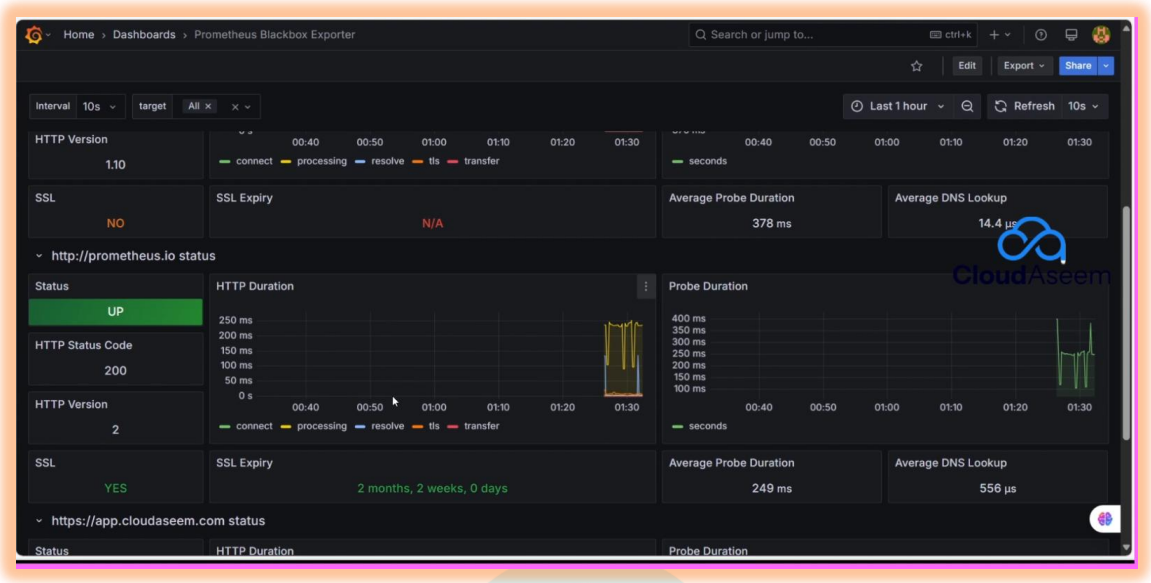
Add Jenkins Dashboard
Go to Dashboards → Import.
Enter Dashboard ID: 9964
Click Load.
Select Prometheus as the Data Source.





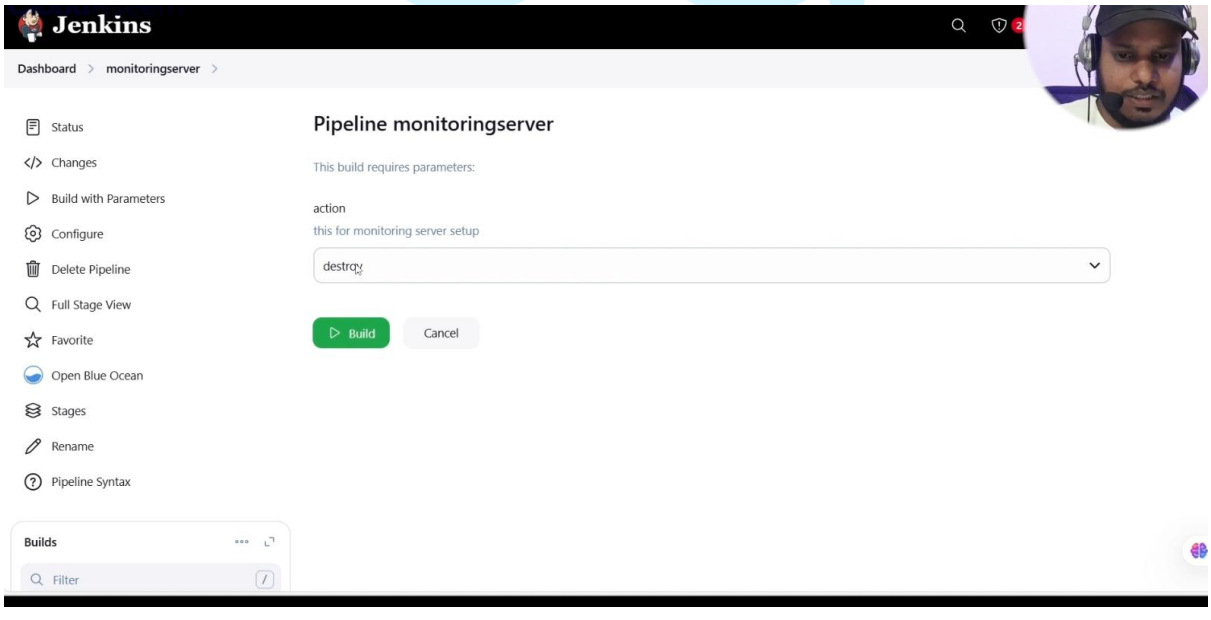
Add
Go to **Dashboards** → **Import**.
Enter **Dashboard ID: 13659** (Prometheus Blackbox Exporter).
Click **Load**.

Select **Prometheus** as the **Data Source**. Click **Import**.



Step - 4 :

1. Delete Monitoring Server with Jenkins pipeline with action as destroy



2. delete Cluster and other resources we have used in AWS Cloud to avoid billing ##

```
eksctl delete nodegroup --cluster=Cloudaseem --name=node2 --region=ap-south-1 --wait
```

eksctl delete cluster --name=Cloudaseem --region=ap-south-1 --wait

```
2025-03-20 01:56:08 [ℹ]  will drain 0 unmanaged nodegroup(s) in cluster "cloudaseem-clu
ster4"
2025-03-20 01:56:08 [ℹ]  starting parallel draining, max in-flight of 1
2025-03-20 01:56:10 [ℹ]  deleted 0 Fargate profile(s)
2025-03-20 01:56:11 [✔]  kubeconfig has been updated
2025-03-20 01:56:11 [ℹ]  cleaning up AWS load balancers created by Kubernetes objects o
f Kind Service or Ingress
```

```
For detailed docs go to https://eksctl.io/

Error: unknown resource type "disassociate-iam-oidc-provider"
2025-04-06 08:00:44 [ℹ]  deleting EKS cluster "Cloudaseem"
2025-04-06 08:00:45 [ℹ]  will drain 0 unmanaged nodegroup(s) in cluster "Cloudaseem"
2025-04-06 08:00:45 [ℹ]  starting parallel draining, max in-flight of 1
2025-04-06 08:00:45 [ℹ]  deleted 0 Fargate profile(s)
2025-04-06 08:00:45 [✔]  kubeconfig has been updated
2025-04-06 08:00:45 [ℹ]  cleaning up AWS load balancers created by Kubernetes objects of Kind Service or Ing
ress
2025-04-06 08:01:32 [ℹ]
3 sequential tasks: { delete nodegroup "node2", delete IAM OIDC provider, delete cluster control plane "Clou
daseem"
}
2025-04-06 08:01:32 [ℹ]  will delete stack "eksctl-Cloudaseem-nodegroup-node2"
2025-04-06 08:01:32 [ℹ]  waiting for stack "eksctl-Cloudaseem-nodegroup-node2" to get deleted
2025-04-06 08:01:32 [ℹ]  waiting for CloudFormation stack "eksctl-Cloudaseem-nodegroup-node2"
2025-04-06 08:02:02 [ℹ]  waiting for CloudFormation stack "eksctl-Cloudaseem-nodegroup-node2"
2025-04-06 08:02:59 [ℹ]  waiting for CloudFormation stack "eksctl-Cloudaseem-nodegroup-node2"
2025-04-06 08:04:30 [ℹ]  waiting for CloudFormation stack "eksctl-Cloudaseem-nodegroup-node2"
2025-04-06 08:04:31 [ℹ]  will delete stack "eksctl-Cloudaseem-cluster"
2025-04-06 08:04:31 [ℹ]  waiting for stack "eksctl-Cloudaseem-cluster" to get deleted
2025-04-06 08:04:31 [ℹ]  waiting for CloudFormation stack "eksctl-Cloudaseem-cluster"
2025-04-06 08:05:01 [ℹ]  waiting for CloudFormation stack "eksctl-Cloudaseem-cluster"
```

✅ **Congratulations!** You have successfully deployed **Starbucks Kubernetes** with:

- 🔹 **Load Balancer Enabled**
- 🔹 **SSL Certificates Configured**
- 🔹 **Monitoring Setup Complete**

📢 **If you found this tutorial helpful, don't forget to:**

👍 **Like** the video

💬 **Comment** your thoughts and questions

🔔 **Subscribe** to stay updated on **Cloud & DevOps tutorials!**

🌍 **Follow me for more updates on Cloud & DevOps:**

🔗 **YouTube:** Cloud DevOps with Aseem - https://www.youtube.com/@clouddevopswithaseem

🔗 **LinkedIn:** Mohammed Aseem Akram - www.linkedin.com/in/mohammed-aseem-akram

🔗 **GitHub:** Aseemakram19

📌 **Stay tuned for more DevOps insights!** 🚀
#Cloud #DevOps #Technology #Kubernetes
