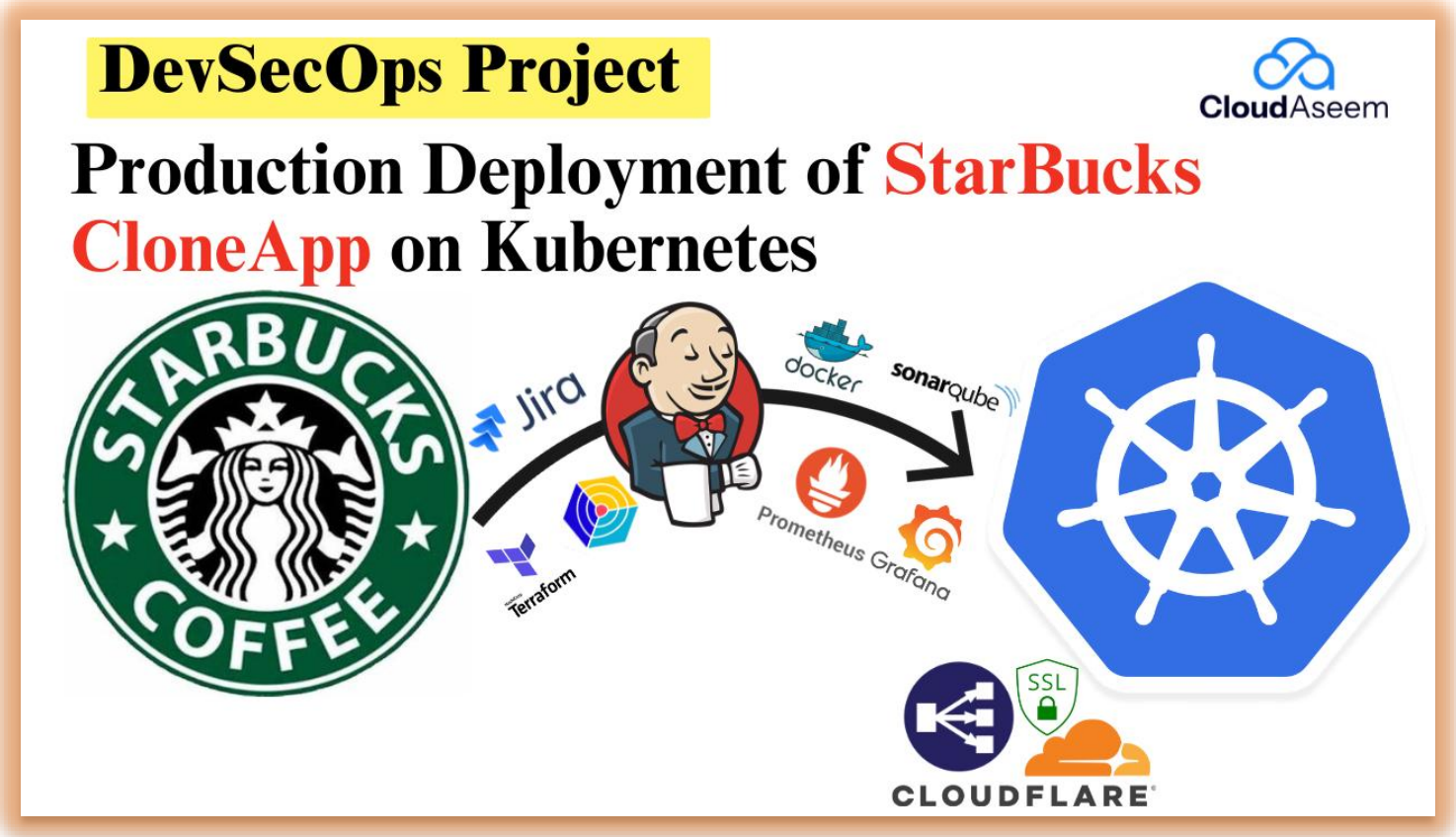
CloudAseem

CloudAseem

Mohammed Aseem Akram

StarbucksClone App — End-to-End Secure & Scalable Deployment with DevSecOps & Jenkins !

Thrilled to showcase our latest project — a Starbucks**Clone Application** engineered 🚀 **Starbucks Clone 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:** https://github.com/Aseemakram19/starbucks-kubernetes.git

🔗 **LinkedIn:** Mohammed Aseem Akram

🔗 **GitHub Account:** Aseemakram19

Project Agenda

1️⃣ Infrastructure Setup

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

---

2️⃣Core 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	SMTP 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**

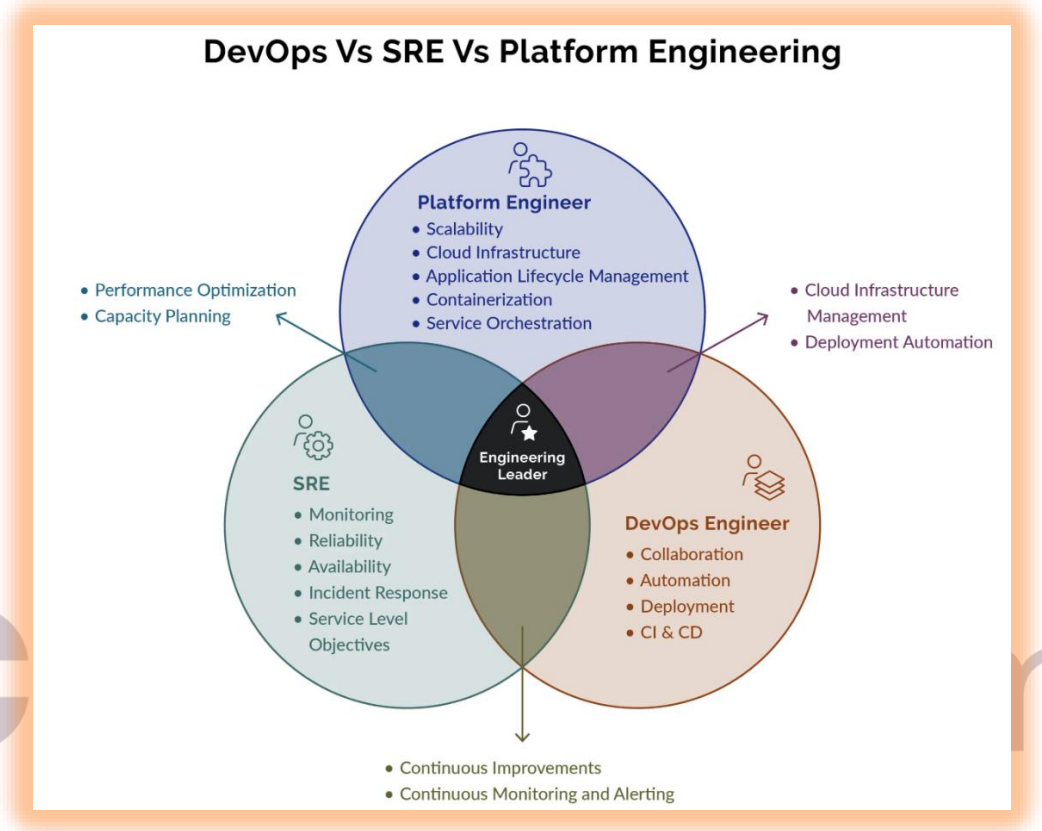
- 🧱 Designs and builds reusable infrastructure as code (IaC)
- 🔌 Integrates CI/CD tools, secrets management, and container registries
- 🧩 Enables self-service environments and developer portals
- 🔐 Focuses on security, compliance, and platform governance

💂 **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 **Starbucks Clone 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

---

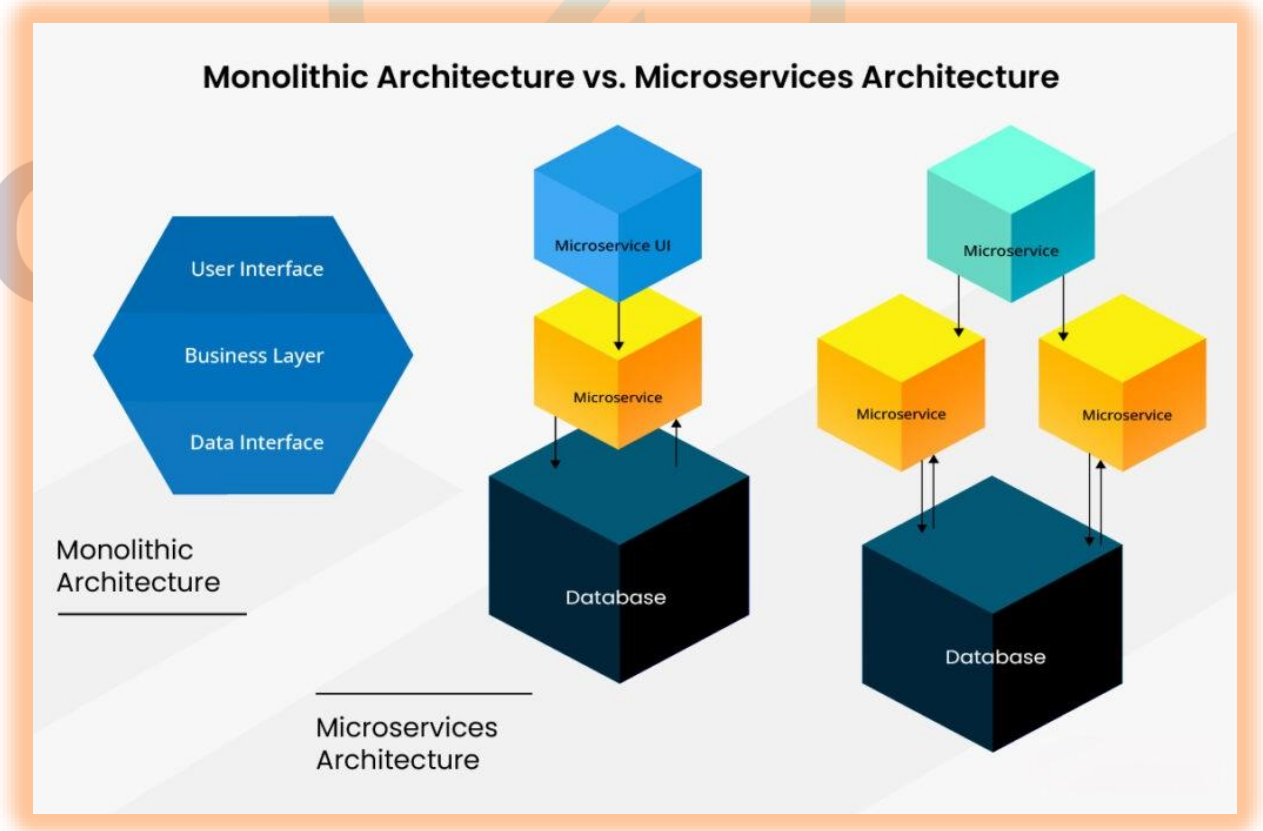
❗ **Challenges**

- 📏 **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.

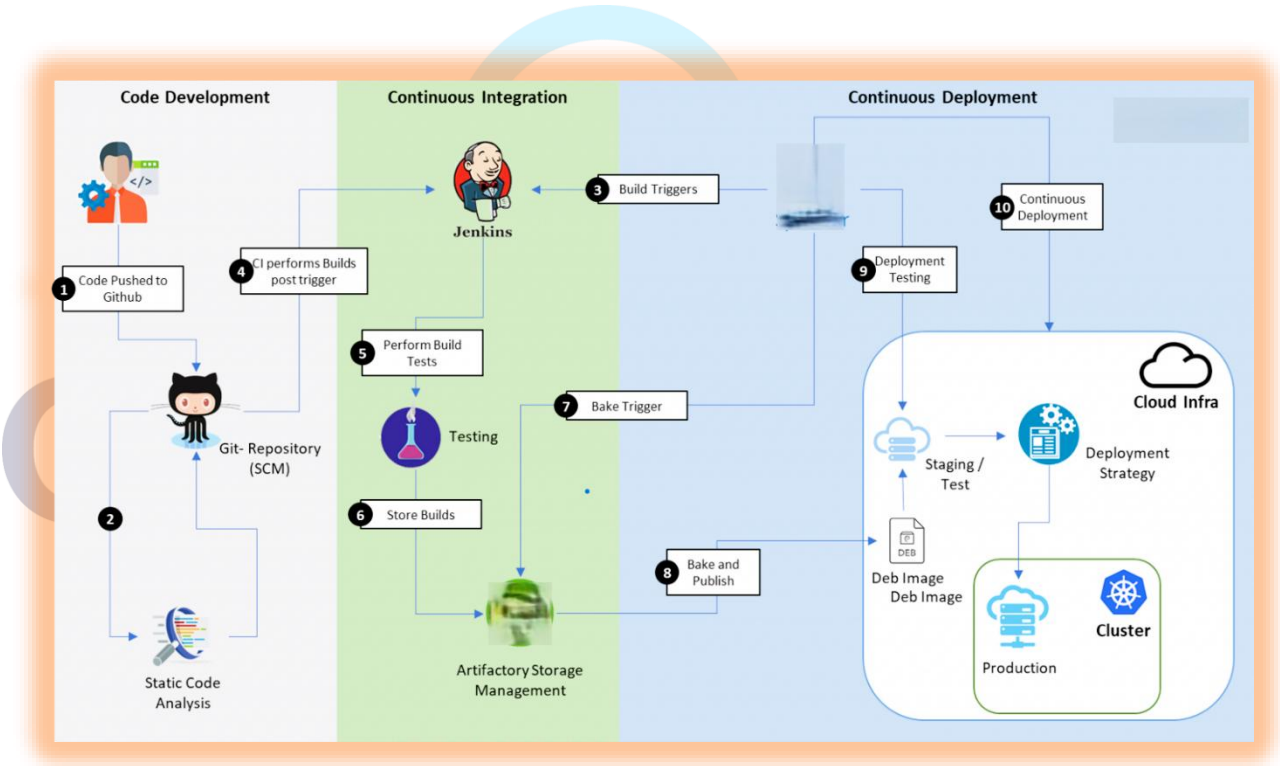
🧪 Static Code Analysis runs automatically to check code quality and security.

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

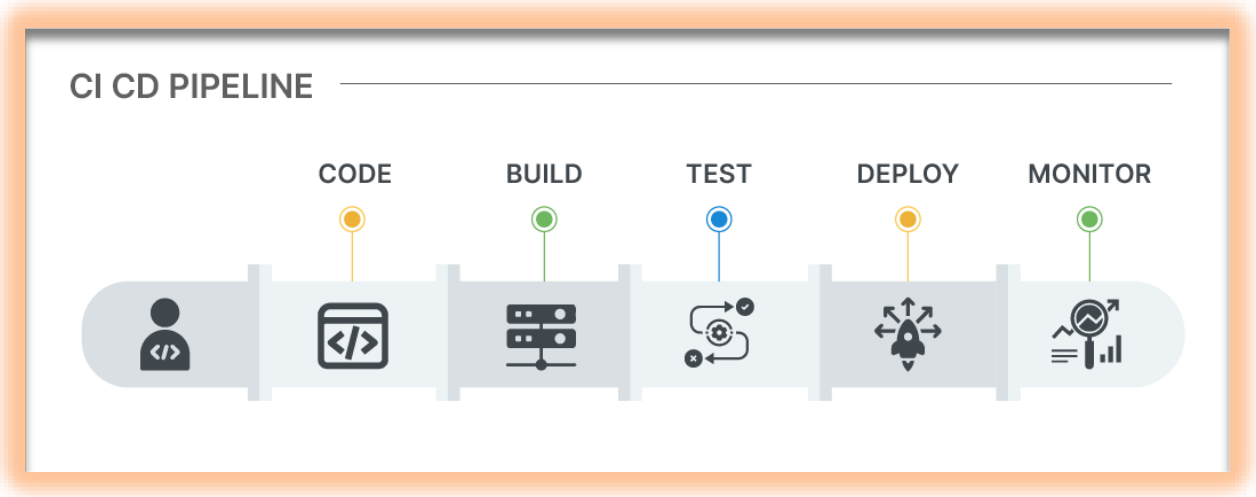
📦 Successfully built artifacts are stored in Artifactory, and a "bake" process packages them.

🧪 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 :



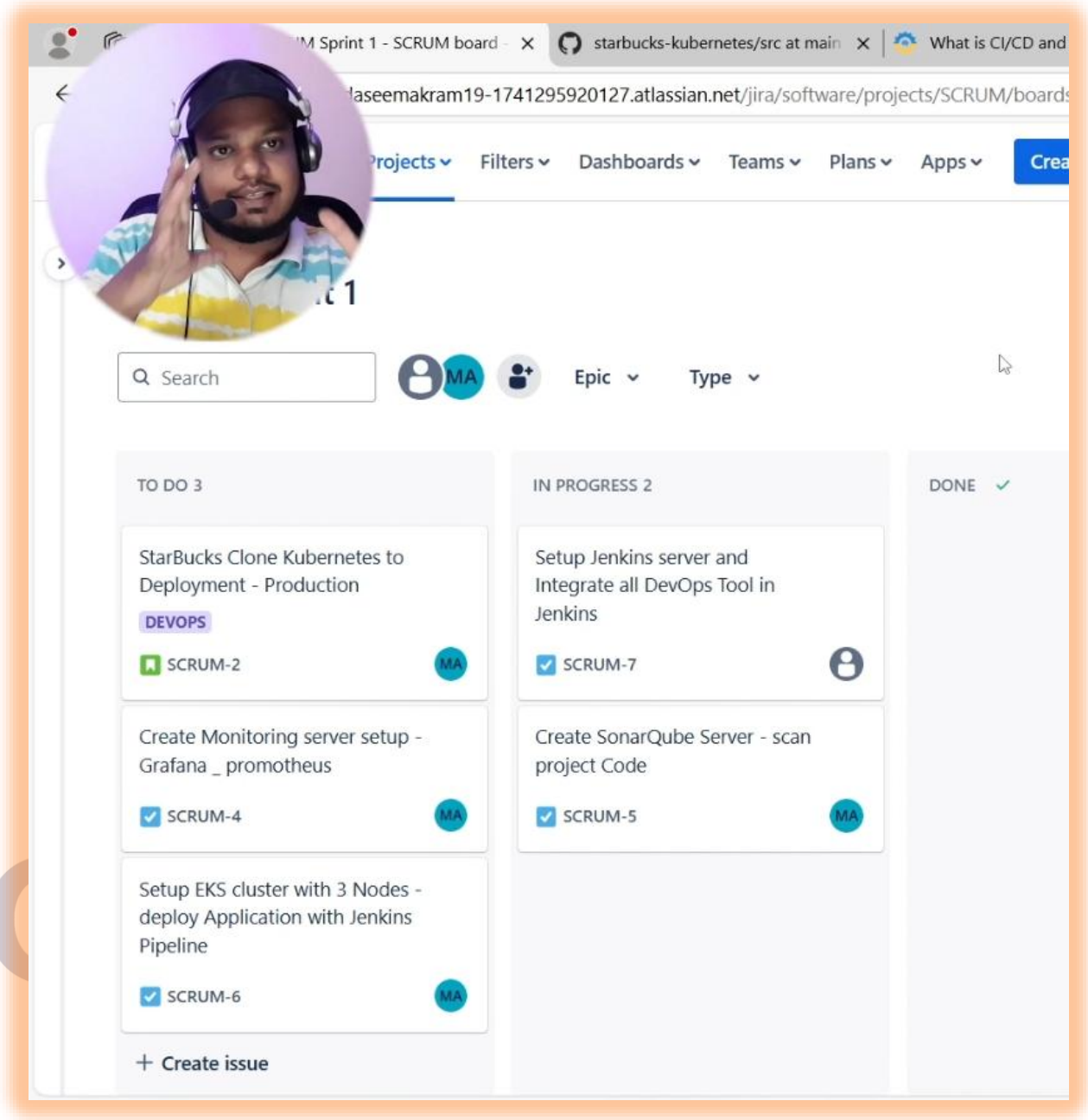
JIRA PROJECT MANAGEMENT TOOL :

📄 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 Starbucks Clone App 🚀

👩‍💻 👉 Proceeding to the Lab Setup…

We'll begin with:

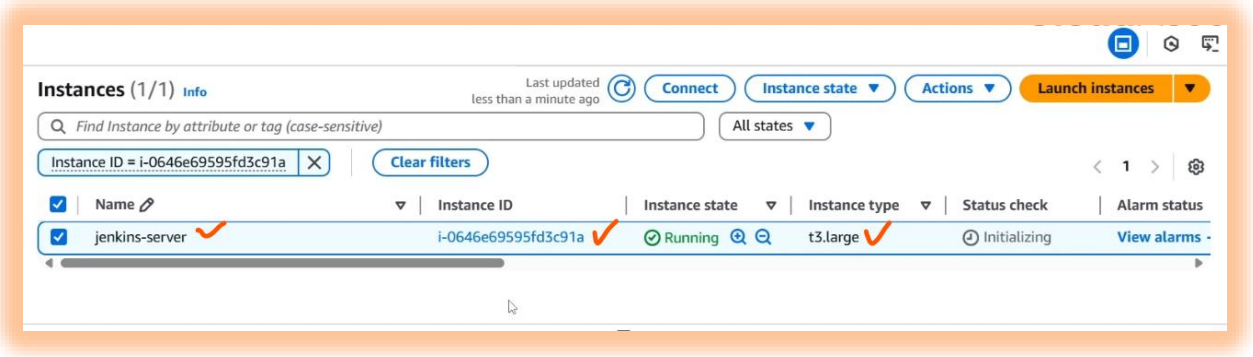
1. 🖥️ **Provisioning Jenkins Server**
2. 🛡️ **Setting up SonarQube for Code Quality & Security**
3. 📊 **Launching Monitoring Stack (Prometheus & Grafana)**
4. ☸️ **Preparing Kubernetes Cluster for Deployment**
5. 🧪 **Integrating CI/CD Pipeline with DevSecOps Practices**

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 X 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).

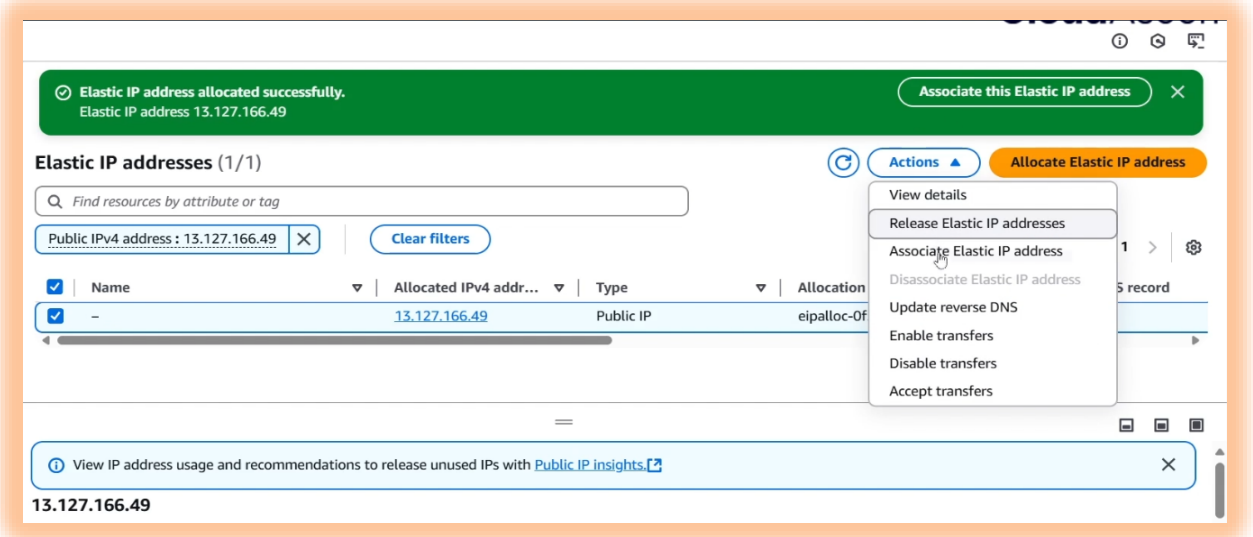


To assign an Elastic IP (EIP) to a Jenkins server on AWS, follow these steps: Timestamp : 16:09

1. **Allocate an Elastic IP**:
   - Go to the **EC2 Dashboard** in AWS Management Console.
   - Under **Network & Security**, click **Elastic IPs**.
   - Click **Allocate new address** and choose your region.
2. **Associate Elastic IP with Jenkins Server**:
   - In the **Elastic IPs** section, select the allocated EIP.
   - Click **Actions**, then **Associate Elastic IP address**.
   - Select the **Jenkins server instance** (EC2 instance) and click **Associate**.
3. **Verify Association**:
   - Check if the EIP is now associated with the Jenkins EC2 instance.
   - Ensure Jenkins is accessible via the new IP.

Benefits:

- **Static IP**: Provides a fixed IP address that won't change even if you stop/start your EC2 instance.
- **Easy Access**: Direct and consistent access to Jenkins from external networks.
- **High Availability**: In case of instance failure, the EIP can be reassigned to a new instance quickly.

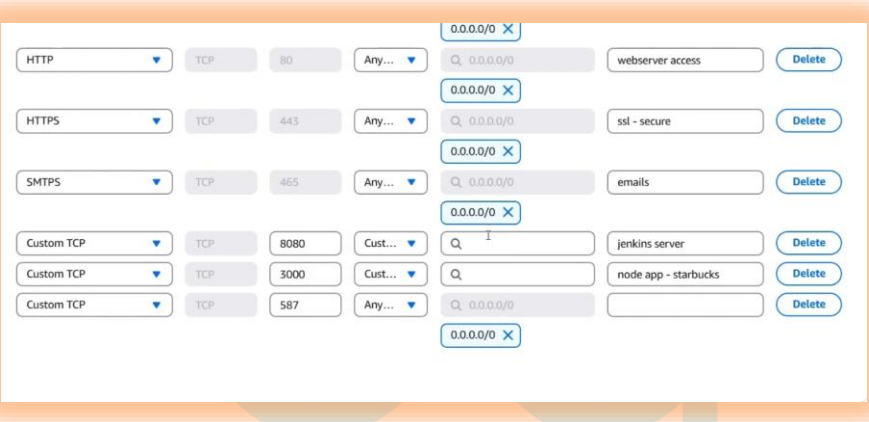


2. Configure Security Group

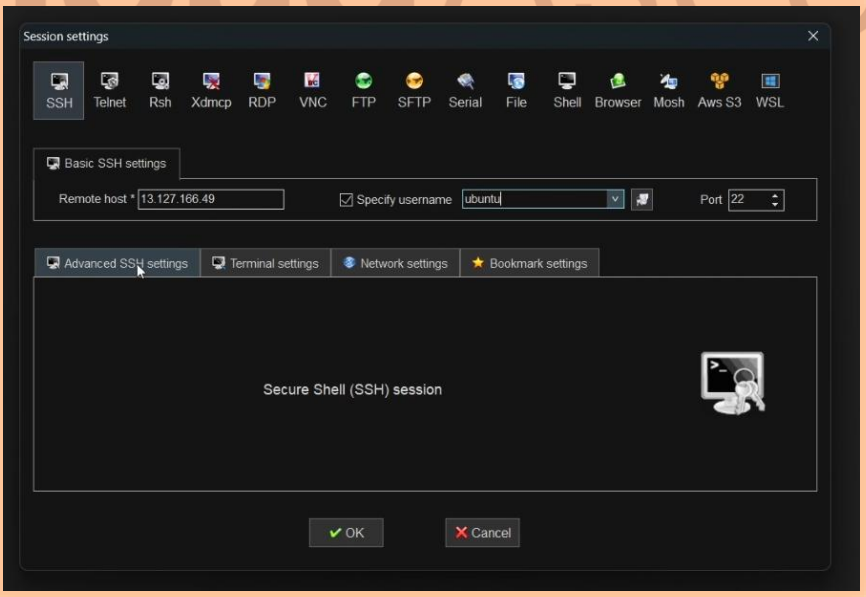
Create or modify a security group to allow the following ports:

Port	Protocol	Description
22	TCP	SSH (for remote access)
80	TCP	HTTP (Web traffic)
443	TCP	HTTPS (Secure web traffic)
8080	TCP	Web applications (Tomcat, etc.)
587	TCP	SMTP (Email sending)
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:



Create GITHUB Repo and Push code

the step-by-step guide to clone the GitHub repository, make changes, and push it to your own GitHub account, ensuring it's private:

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

git clone https://github.com/Aseemakram19/starbucks-kubernetes.git

2. **Navigate to the project directory**:

cd starbucks-kubernetes/

3. **Remove the current origin** (this is necessary to unlink the original repository):

git remote remove origin

4. **Verify the remote is removed** (optional):

git remote -v

5. **Create a new repository on GitHub**:
    - Go to your GitHub account.
    - Create a new repository with the desired name (e.g., starbucks-kubernetes-project).
    - Ensure to make it **Private** when creating the repository.
6. **Add your new GitHub repository as the origin**:

git remote add origin https://github.com/youraccount/starbucks-kubernetes-project.git

7. **Push the repository to your GitHub account**:

git push -u origin main

8. **Enter GitHub credentials**:
    - When prompted, enter your GitHub **username**.
    - Use your **GitHub Personal Access Token** as the password (GitHub now uses tokens instead of passwords for enhanced security).

{ Steps to Create a GitHub Personal Access Token (PAT):

1. **Log in to GitHub**:
    - Open GitHub and log in to your account.
2. **Access Your Settings**:
    - Click on your **profile icon** in the top right corner.
    - Select **Settings** from the dropdown menu.
3. **Navigate to Developer Settings**:
    - On the left sidebar, scroll down and click on **Developer settings**.
4. **Go to Personal Access Tokens**:
    - In the Developer settings section, click on **Personal access tokens**.
    - Choose **Tokens (classic)**.
5. **Generate New Token**:
    - Click on **Generate new token** (or **Generate new token (classic)**, depending on the UI).
    - Give the token a **descriptive name**, e.g., "Jenkins access token" or "CLI operations".
    - Set an **expiration date** (optional), but you can choose **No expiration** if you prefer the token to remain valid indefinitely.
6. **Select Scopes**:
    - Choose the required **scopes** for your token, based on the permissions you need. For example:
        - repo: Full control of private repositories (push, pull, delete, etc.).

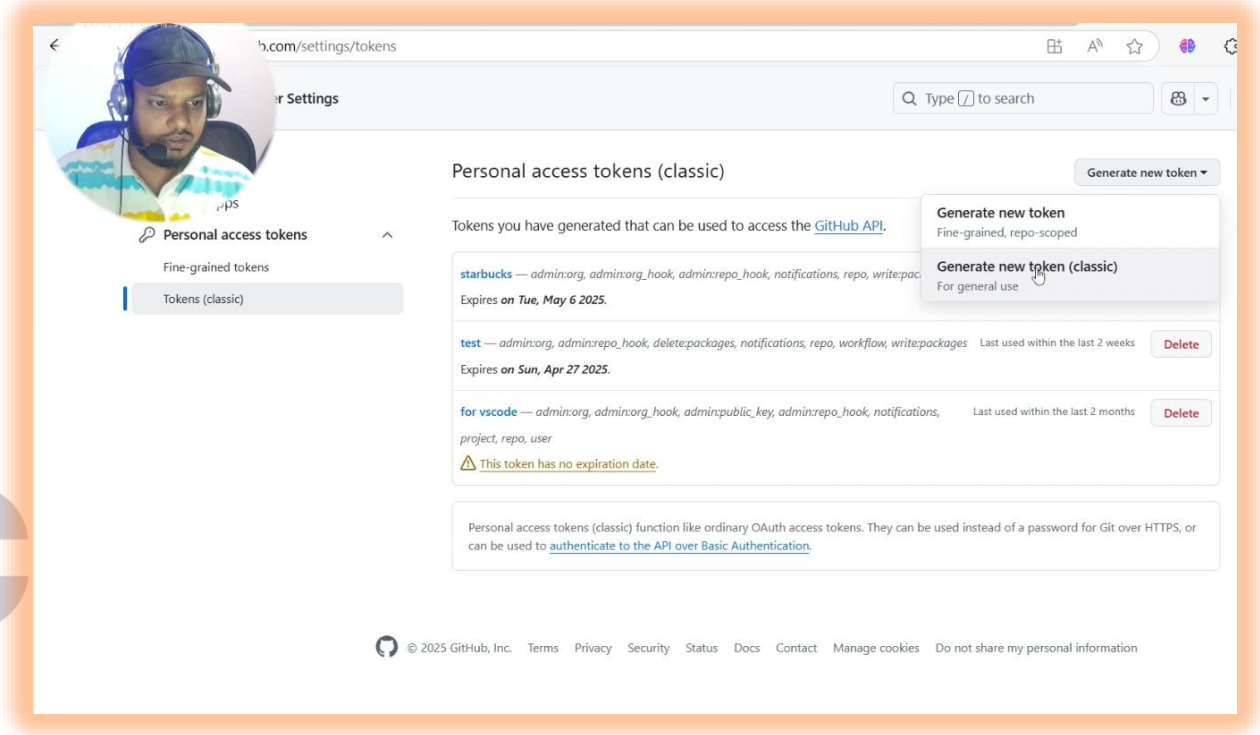
- workflow: For managing GitHub Actions workflows.
- admin:org: For managing organization settings.
- read:user: For accessing public user data, etc.

For general Git operations (clone, push, pull, etc.), the repo scope is usually sufficient.

7. **Generate Token**:
   - Once you have selected the necessary scopes, click **Generate token** at the bottom.
   - Copy the **token** shown on the screen. **This is the only time it will be visible**.

Important Notes:

- **Save your token securely**, as you won't be able to see it again after this step.
- **Use the token** in place of your GitHub password when performing Git operations, such as cloning or pushing to a repository.
- If you use the token in a CI/CD pipeline or for Jenkins, you can store it securely in the credentials store or environment variables to avoid using it directly in scripts.



-

}

Enhancements:

- **Set a default branch** (if using main instead of master):

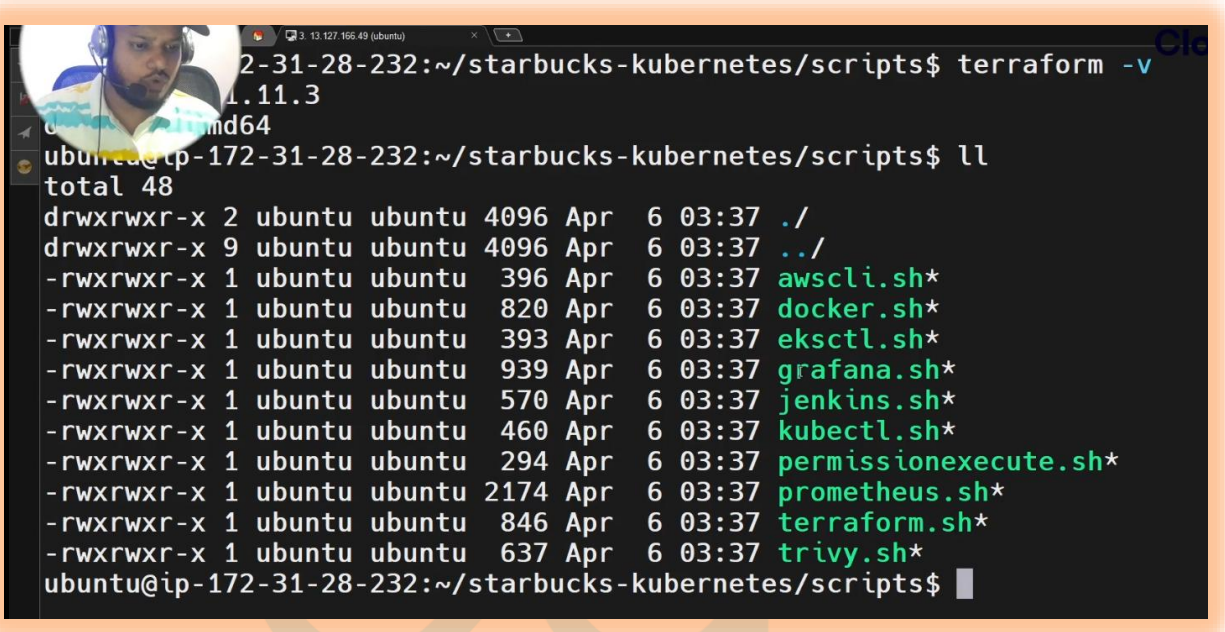
git branch -M main

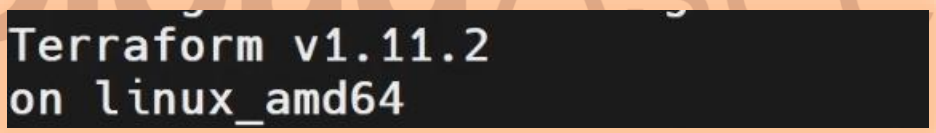
- **Set the repository to private**:
  - When creating the repository on GitHub, ensure you select the **Private** option. If the repository is public and you want to change it, you can adjust it in the repository settings on GitHub.

cd starbucks-kubernetes/**scripts/**

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

Install Tools



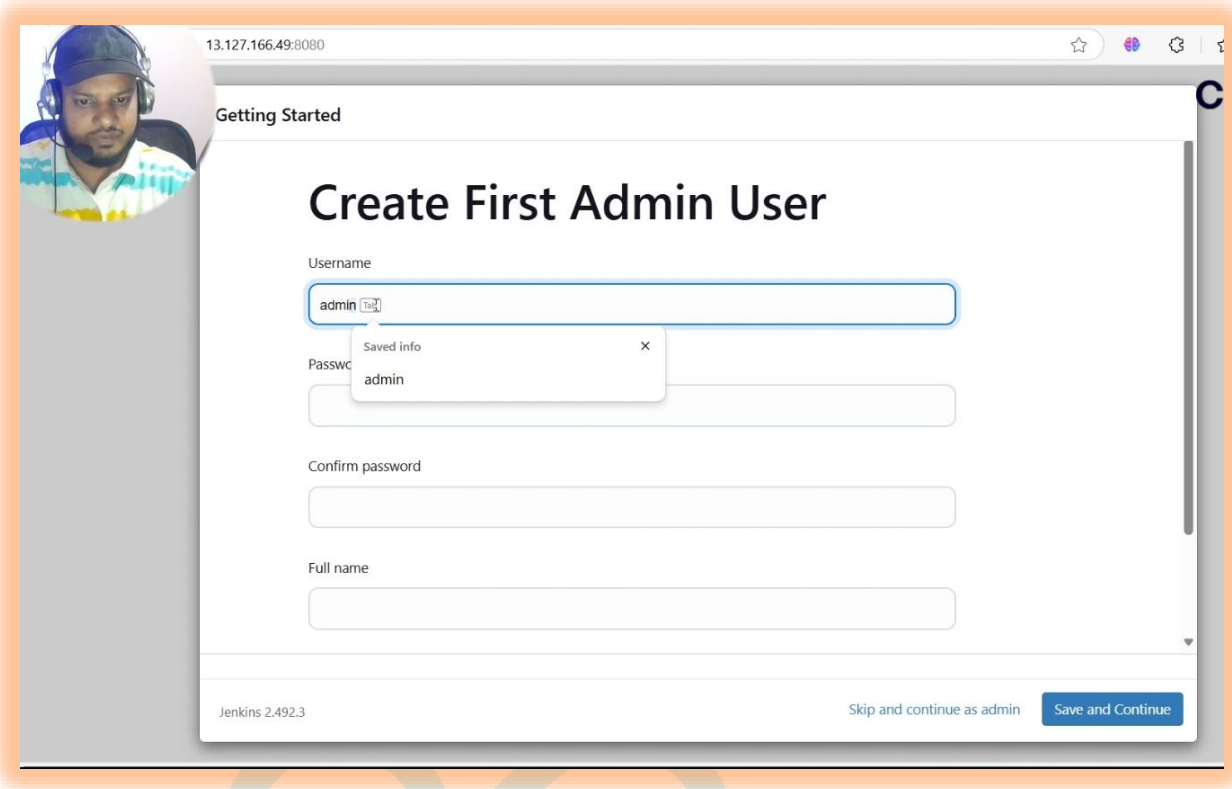


- Access Jenkins in your browser:

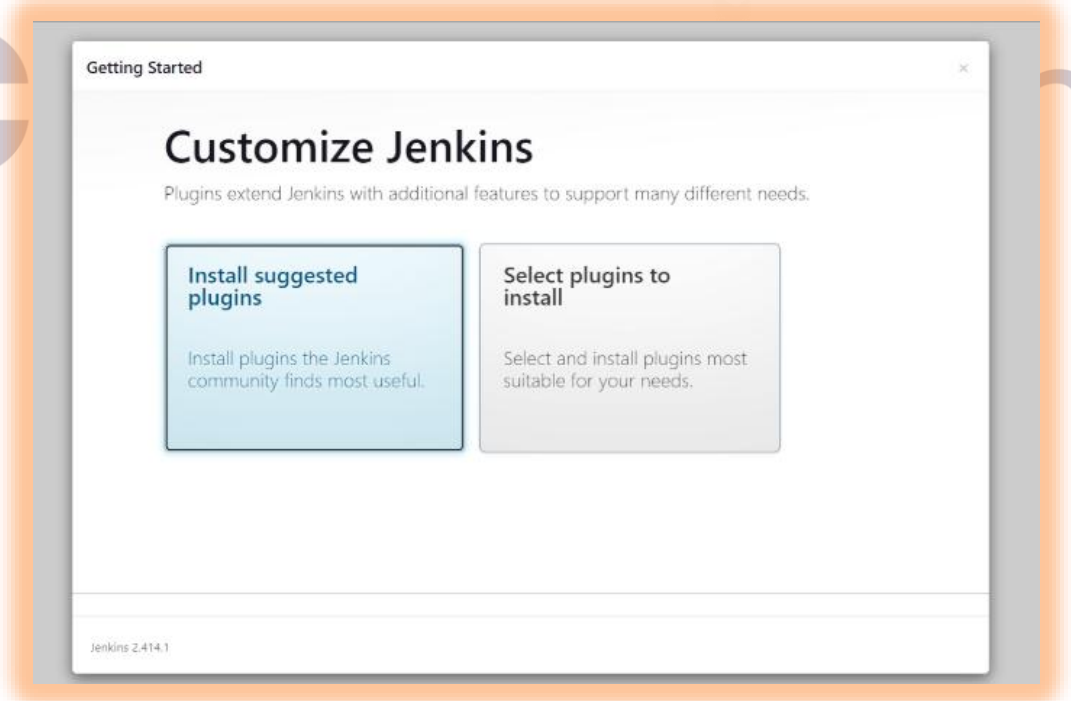
```
http://<PUBLIC_IP>:8080
```

- 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.



- 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

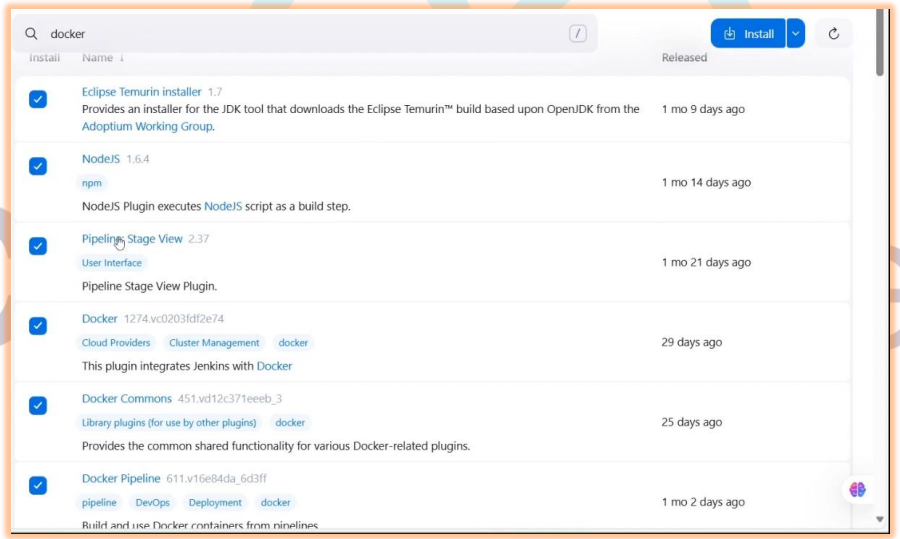
Goto Manage Jenkins →Plugins → Available Plugins →

Install below plugins

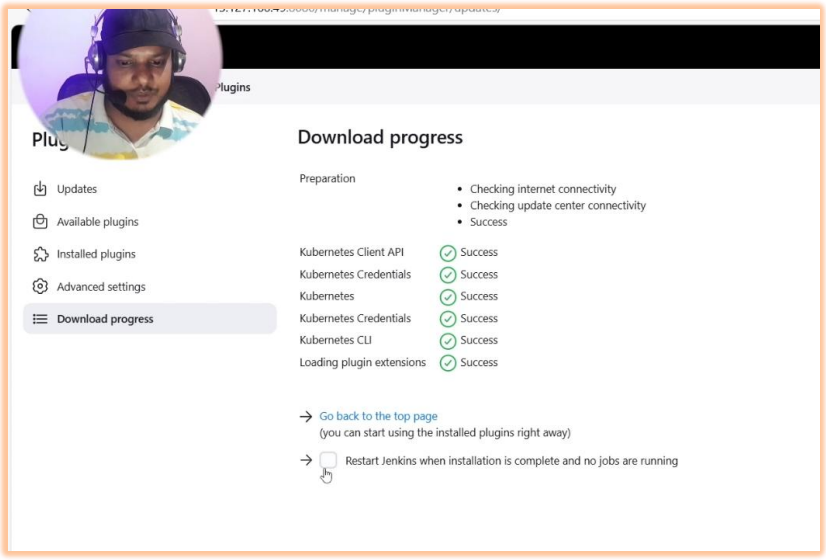
1. Eclipse Temurin Installer (Install without restart)
2. SonarQube Scanner (Install without restart)
3. NodeJs Plugin (Install Without restart) – 16.20.2
5. Stage view
6. jdk

Docker plugin

7. Docker
8. Docker Commons
9. Docker Pipeline
10. Docker API
11. docker-build-step



Add this too plugins :



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.

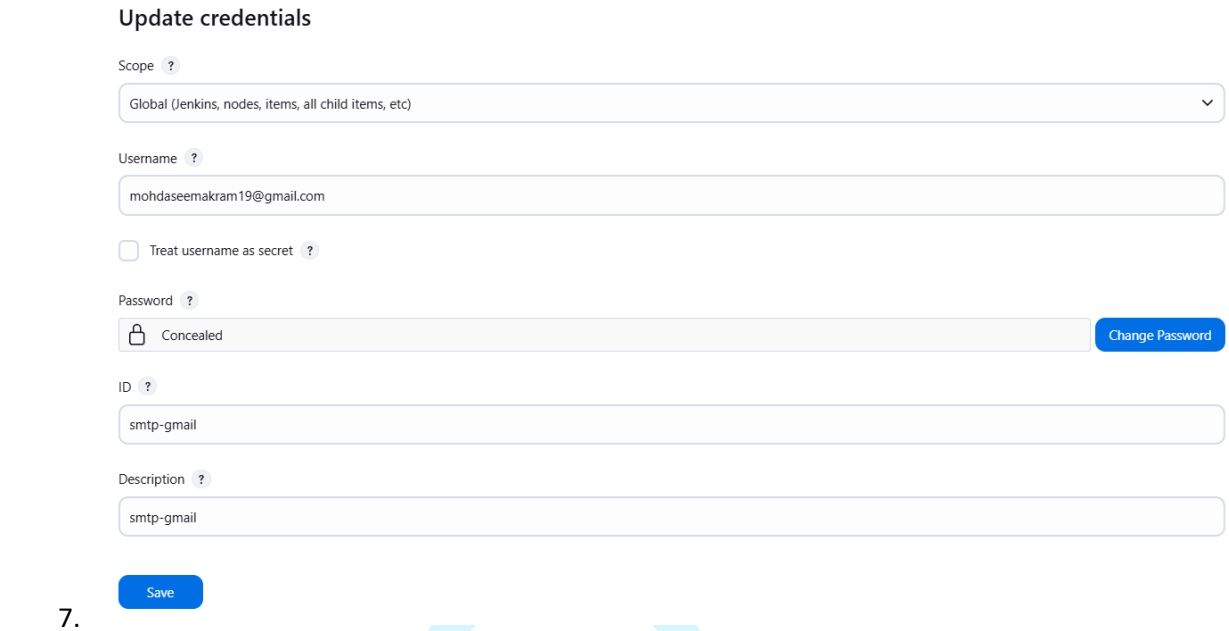
1. **Go to Google Account Security**:
   Google My Account
2. Scroll to **"Signing in to Google"**.
3. Click **"2-Step Verification"** → Click **"Get Started"**.
4. Follow the steps to set up **2-Step Verification** (via SMS or Authenticator App).

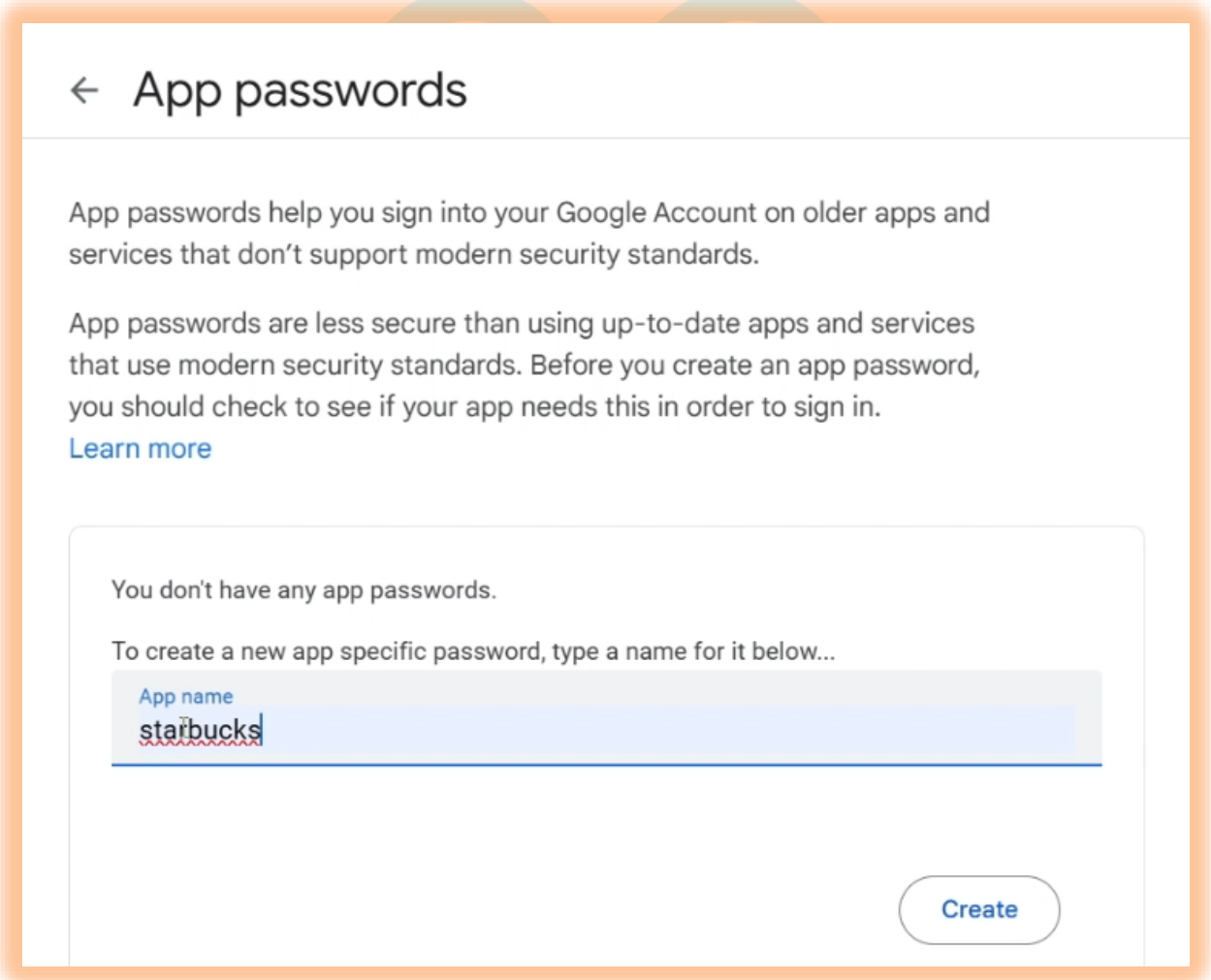
Step 2: Generate an App Password

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.



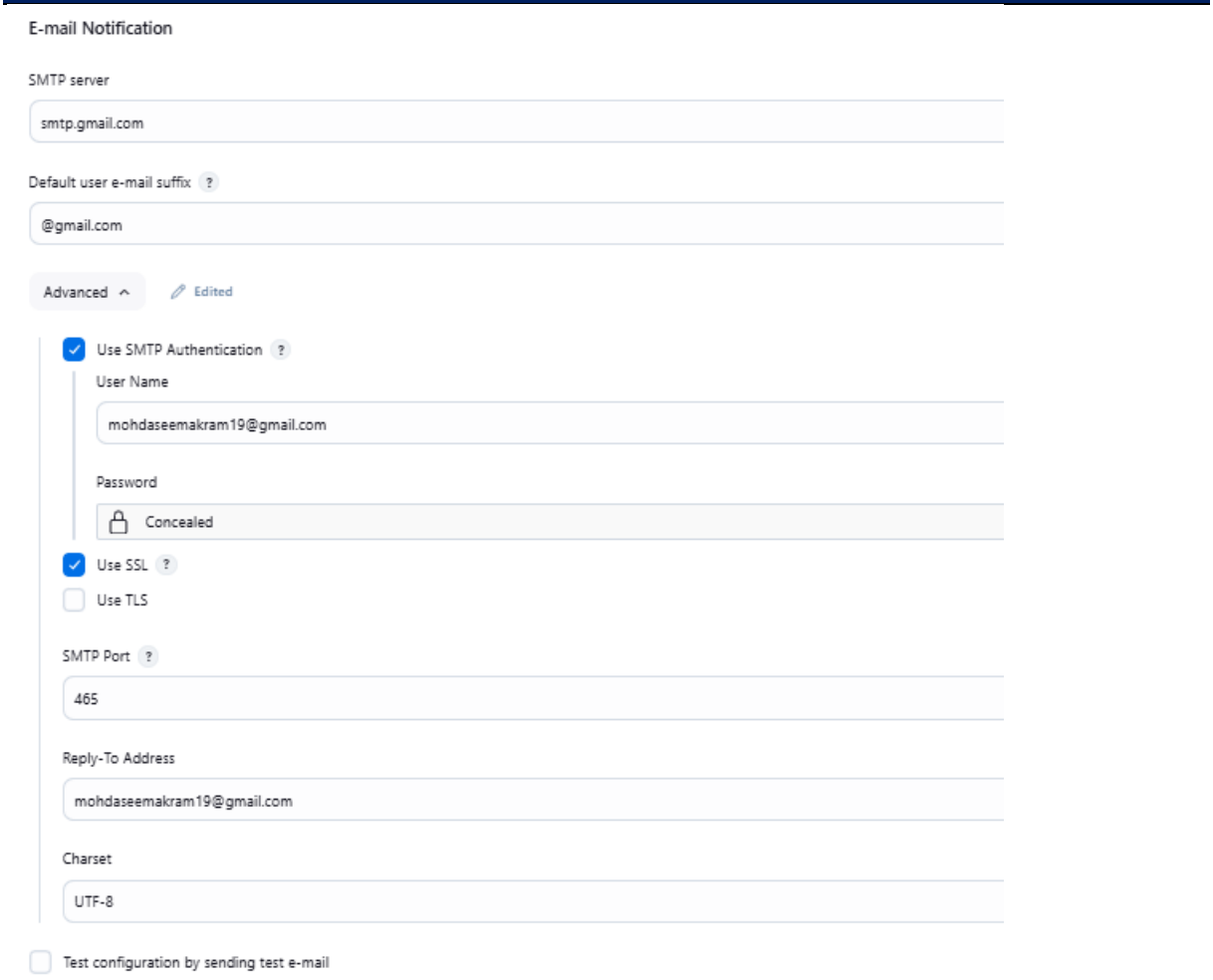


Step 3: Configure Gmail SMTP in Jenkins

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**.

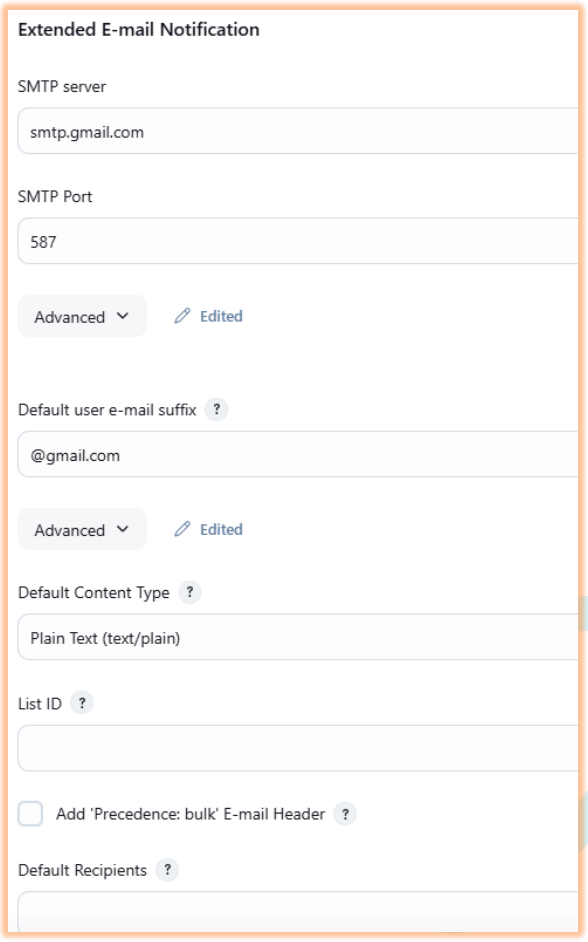
Email Extension Plugin
xsuc kxeb xcvk xqkf

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

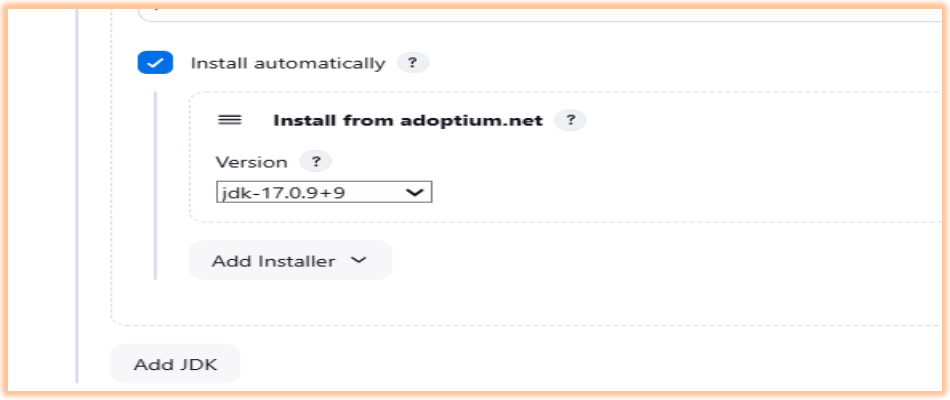


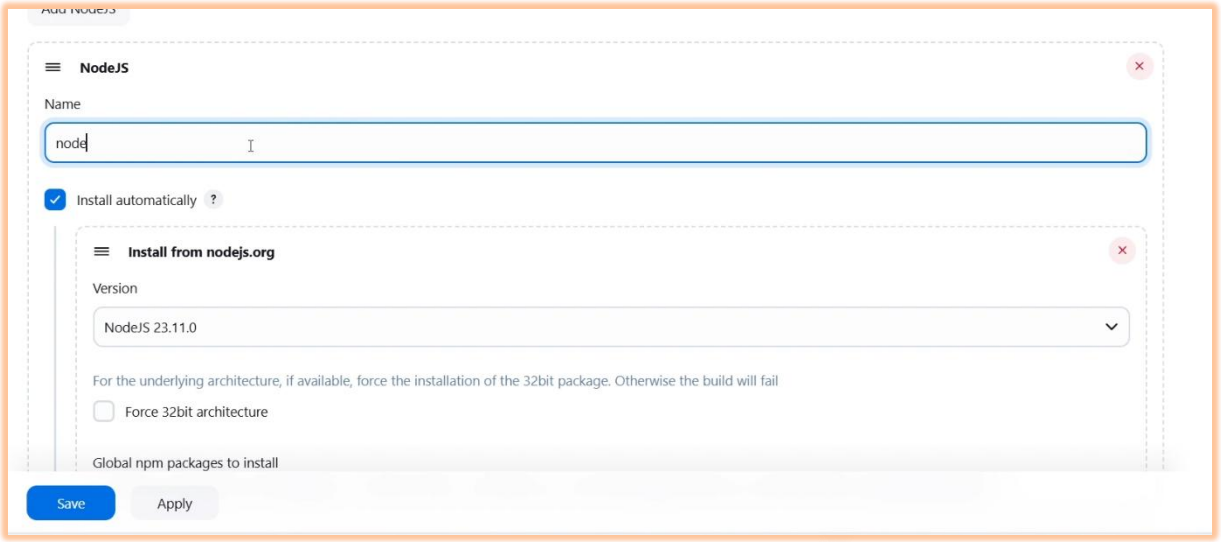
2. Extended Email Notification
SMTP Server: smtp.gmail.com
SMTP Port: 465 (for tls)
Use SSL: Checked

Credentials: Select from the

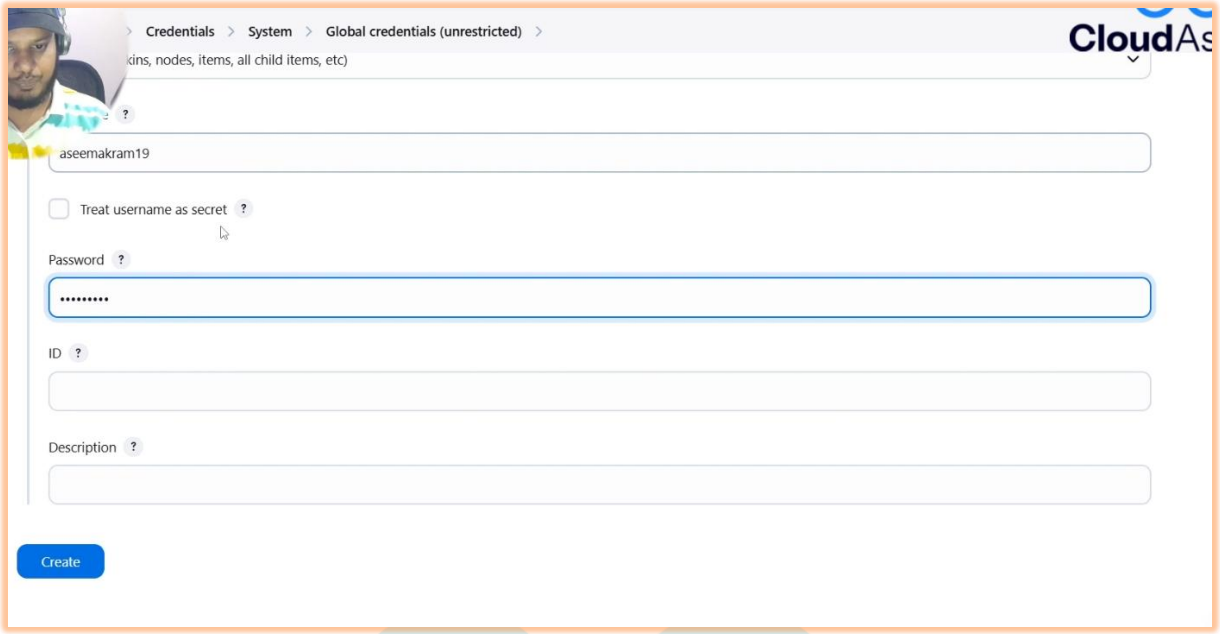


Setup the Tools form Jenkins Dashboard .


Install automatically
Install from adoptium.net
Version
jdk-17.0.9+9
Add Installer
Add JDK

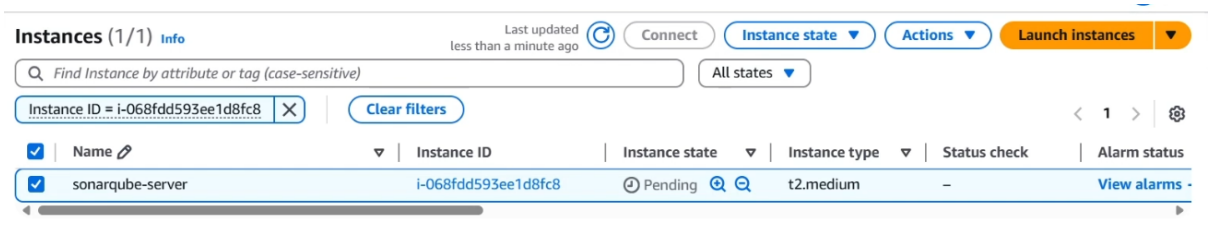

NodeJS
Name
node
Install automatically
Install from nodejs.org
Version
NodeJS 23.11.0
For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail
Force 32bit architecture
Global npm packages to install
Save
Apply

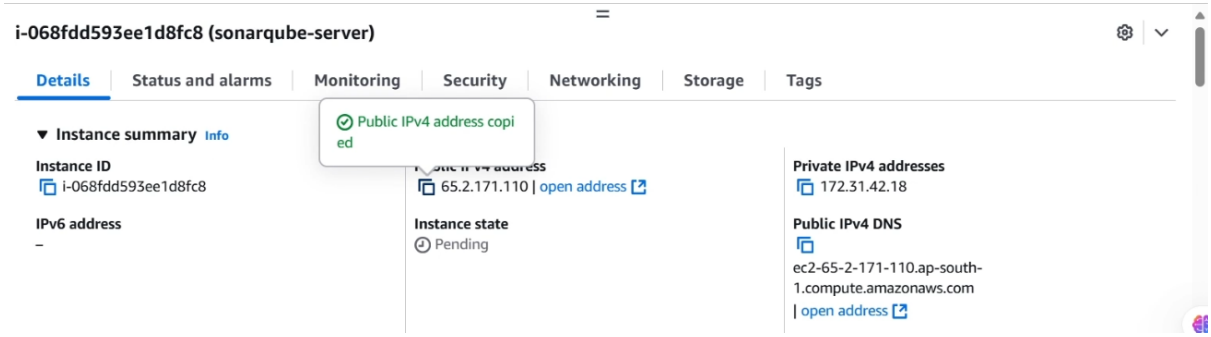
Add Docker credential in Jenkins credentials Vault as username as password:



SonarQube server

Sonarqube Server : Setup a Ec2





Access the server and execute this script

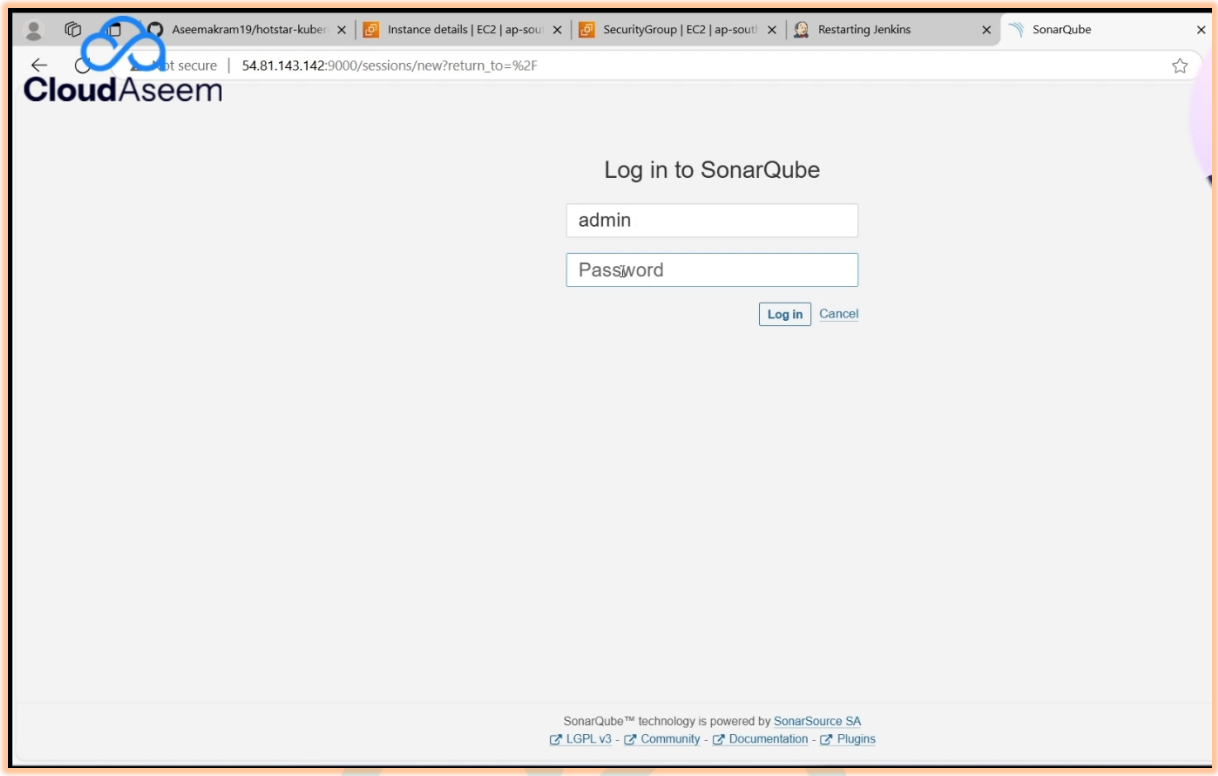
```
#!/bin/bash
# Script to install Docker on an EC2 instance and configure permissions

# Update the package list
sudo apt-get update -y

# Install Docker
sudo apt-get install docker.io -y

# Add the 'ubuntu' and 'jenkins' users to the 'docker' group to allow running Docker without sudo
sudo usermod -aG docker ubuntu
sudo usermod -aG docker jenkins

# Apply the new group settings immediately
newgrp docker

# Set correct permissions for the Docker socket to allow 'docker' group members to access it
sudo chmod 660 /var/run/docker.sock
sudo chown root:docker /var/run/docker.sock

# Restart Docker service to apply changes
sudo systemctl restart docker

# Verify installation
docker -version
# Run SonarQube container in detached mode with port mapping
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```


we create a sonarqube container

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

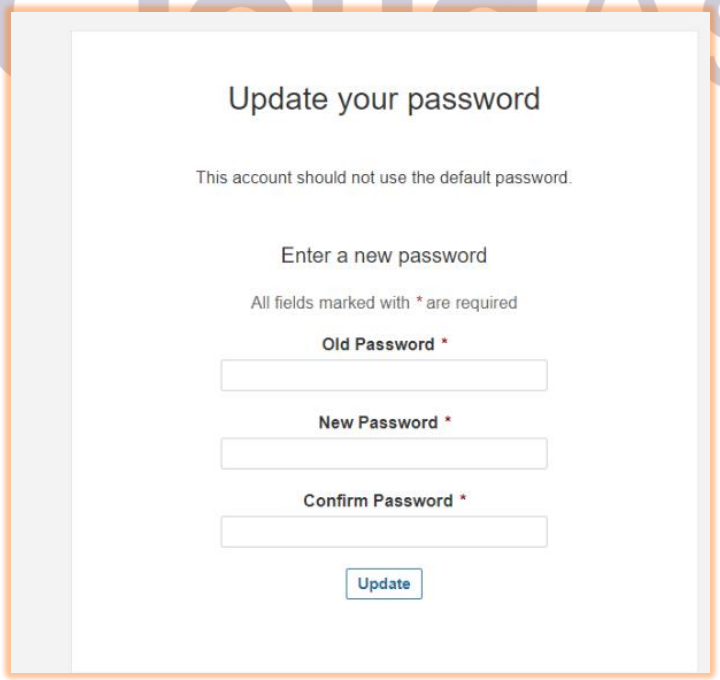
```
00:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
9cb31e2e37ea: Pull complete
13876c96bdc5: Pull complete
25fdfc9faee8: Pull complete
b682cc54ed35: Pull complete
4615ed3a3407: Pull complete
c2e1e3bdd7bc: Pull complete
b2dce0ce5cad: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:95b1826b68e606678882148e28c01c70dc0efe464a0d0c568570b17eedf1a9f9
Status: Downloaded newer image for sonarqube:lts-community
73467ceddea5dbb6cf2970ca1299b617974ec4276a33d9f95f796b73155589b6
ubuntu@ip-10-0-1-103:~/hotstar-kubernetes/scripts$
```

Now our Sonarqube is up and running

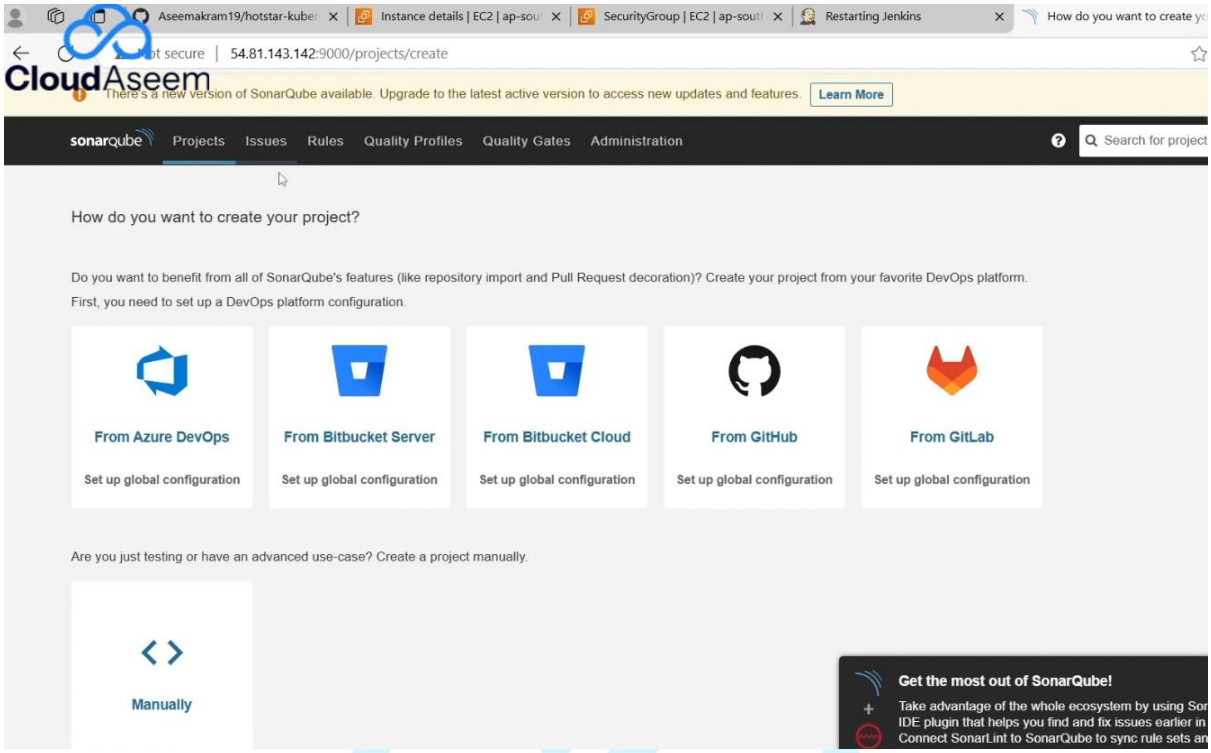


Enter username and password, click on login and change password

username admin
password admin

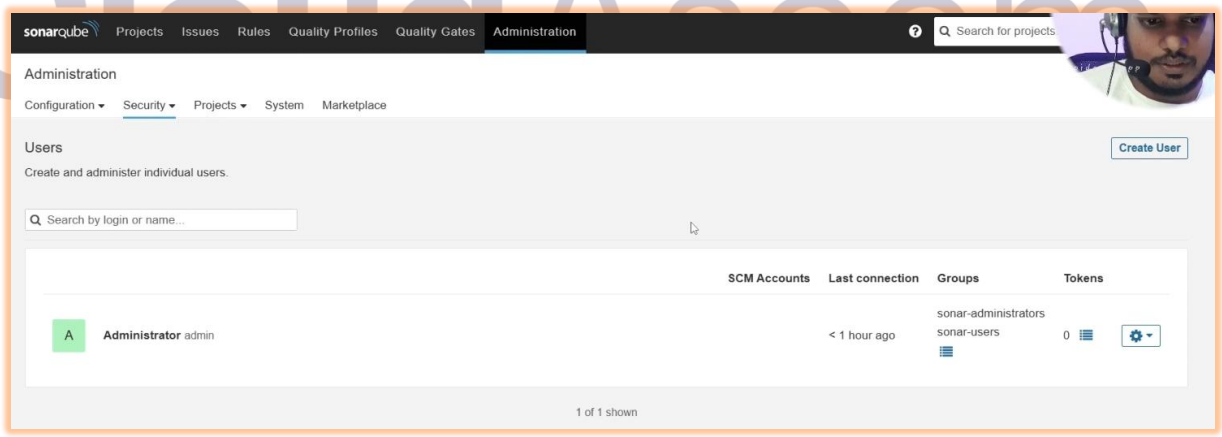


Update New password, This is Sonar Dashboard.



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

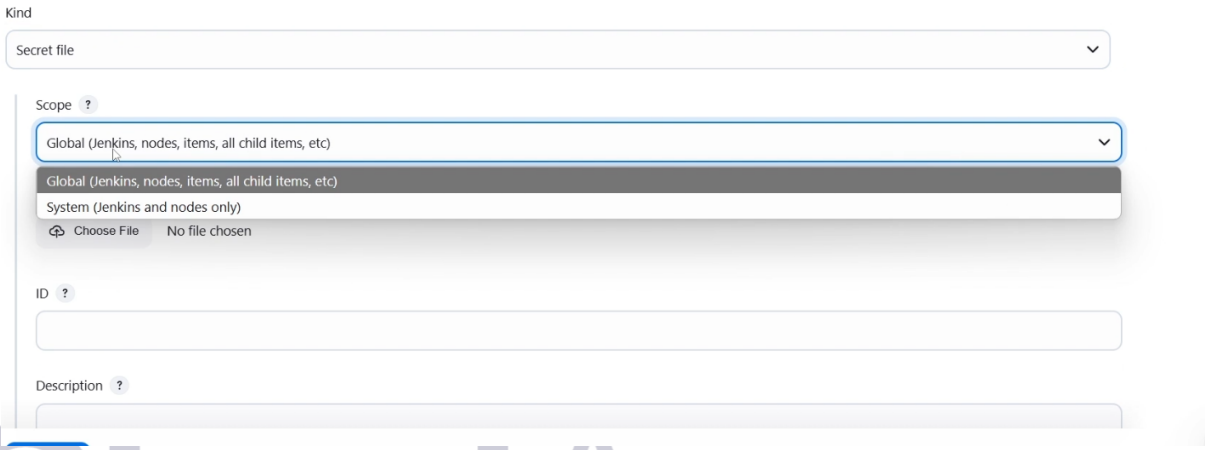
Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token



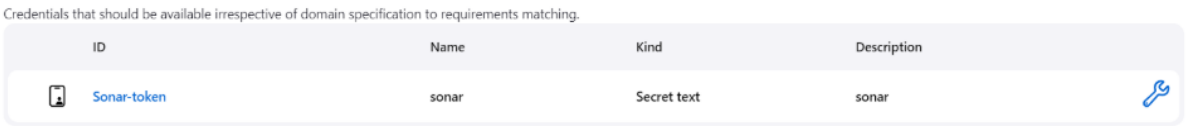
Create a token with a name and generate

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

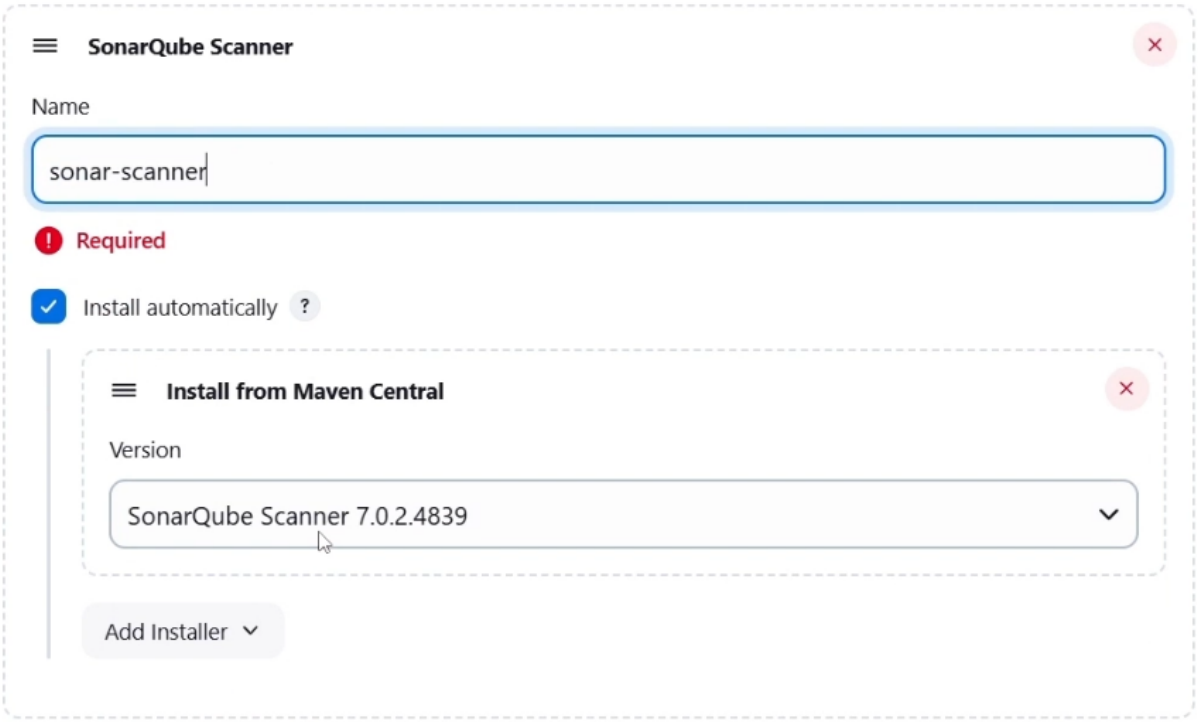
Add Credentials → Add Secret Text. It should look like this



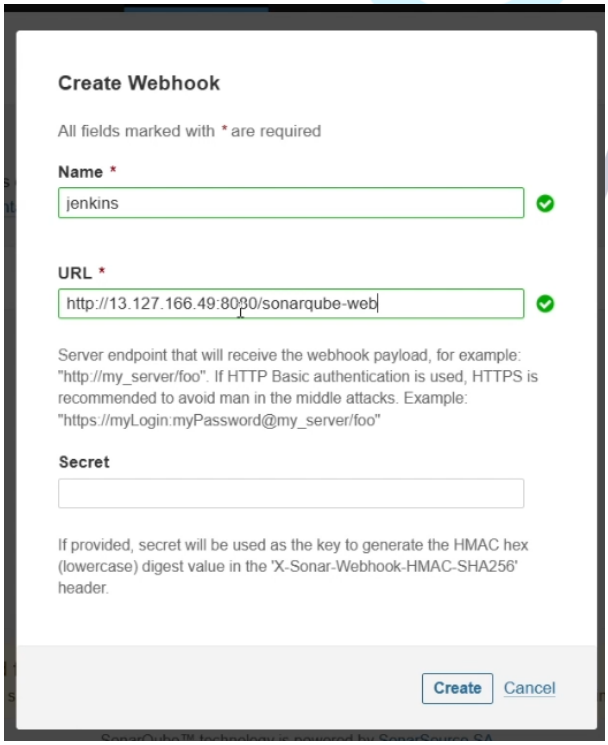
You will this page once you click on create



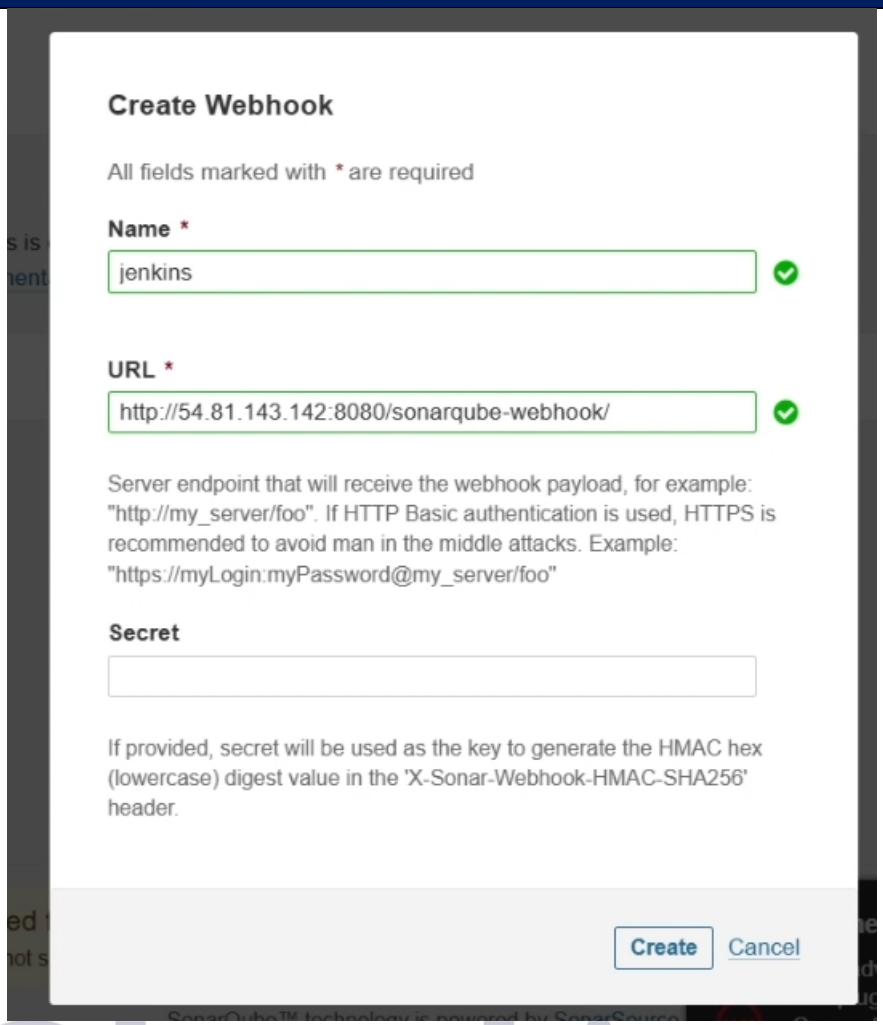
The Configure System option is used in Jenkins to configure different server
Global Tool Configuration is used to configure different tools that we install using Plugins
We will install a sonar scanner in the tools.
Manage Jenkins –> Tools –> SonarQube Scanner



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



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

- 

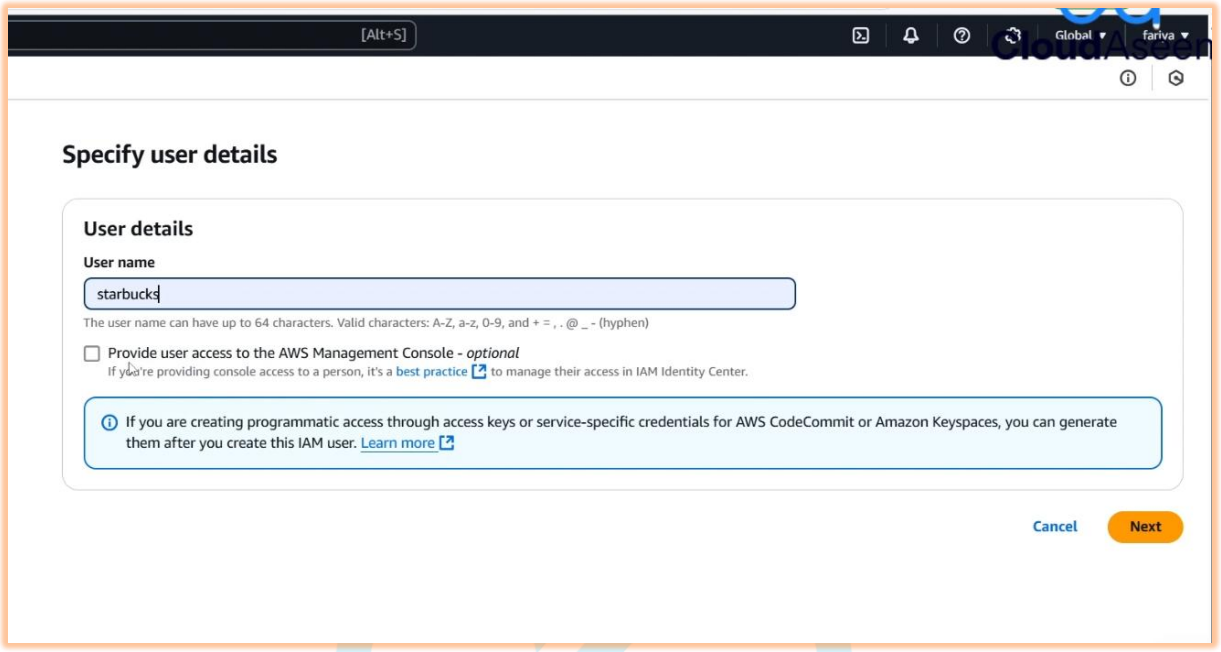
To set up the **AWS Access Key** and **Secret Access Key** for configuring AWS EKS from an EC2 instance, follow these steps:

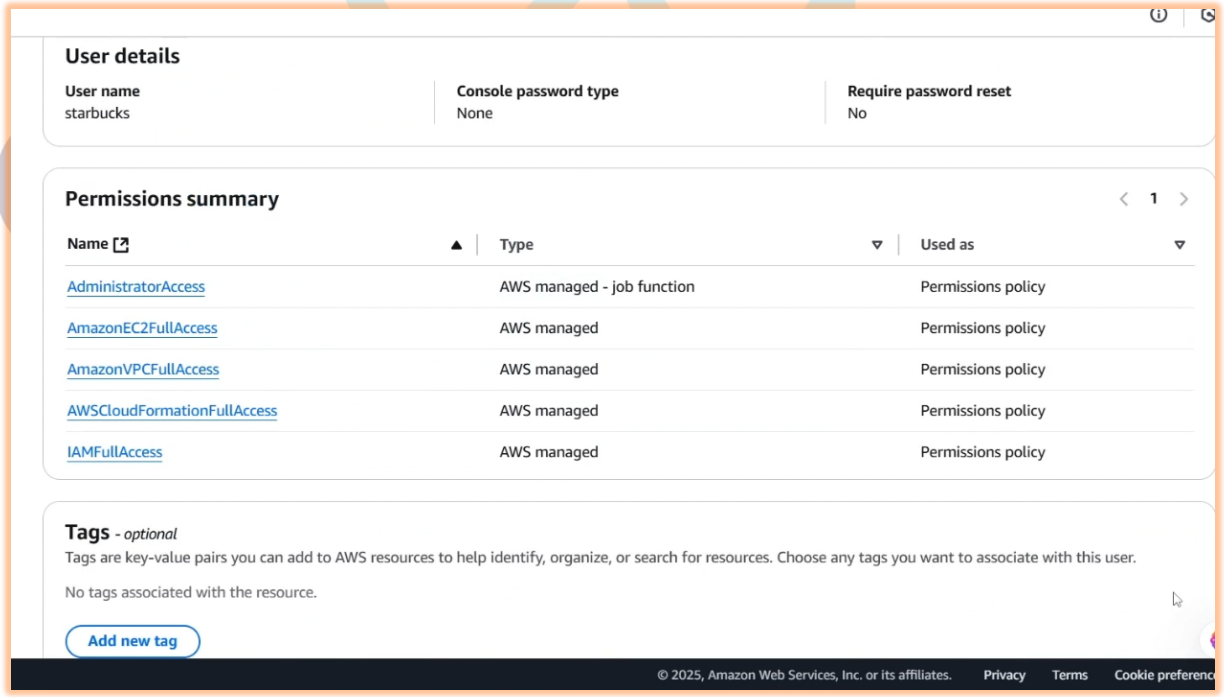
1. Create IAM User with EKS Permissions

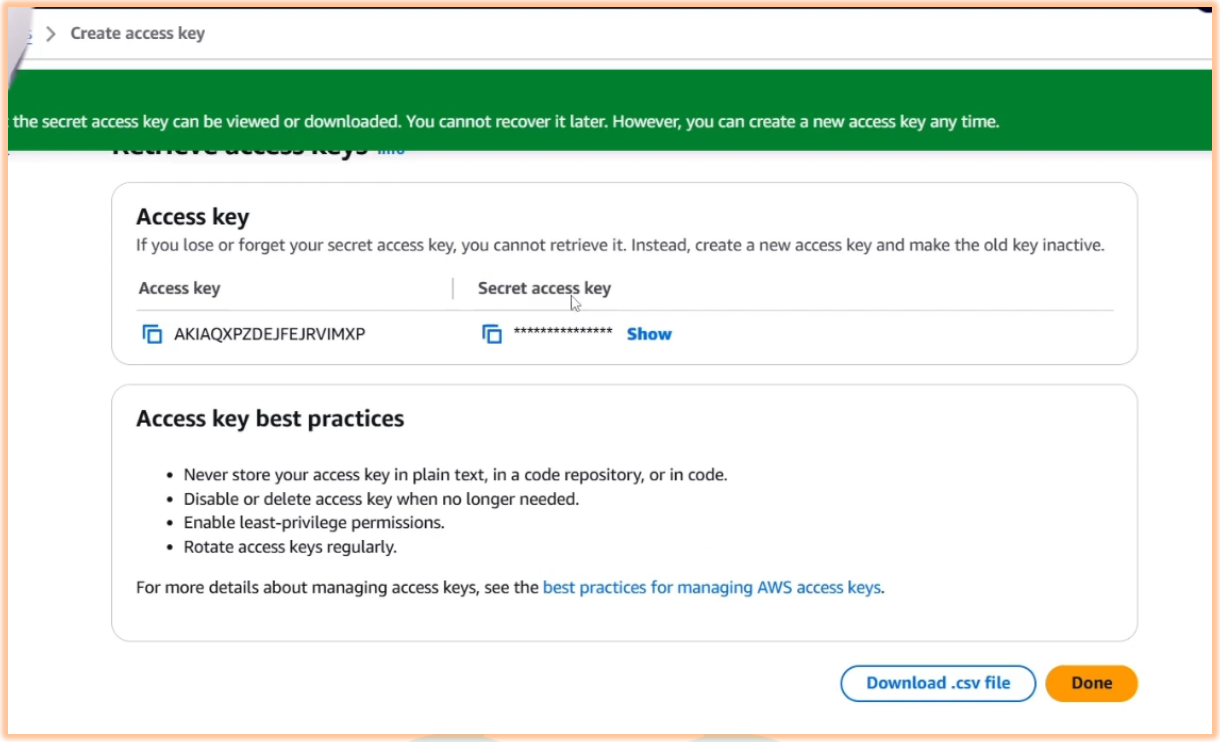
1. **Log in to the AWS Management Console**.
2. **Go to IAM (Identity and Access Management)**.
3. **Create a new IAM user**:
   - Click on **Users** from the left sidebar and then click **Add user**.
   - Provide a username (e.g., `eks-setup-user`).
   - Choose **Programmatic access** to allow the user 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.

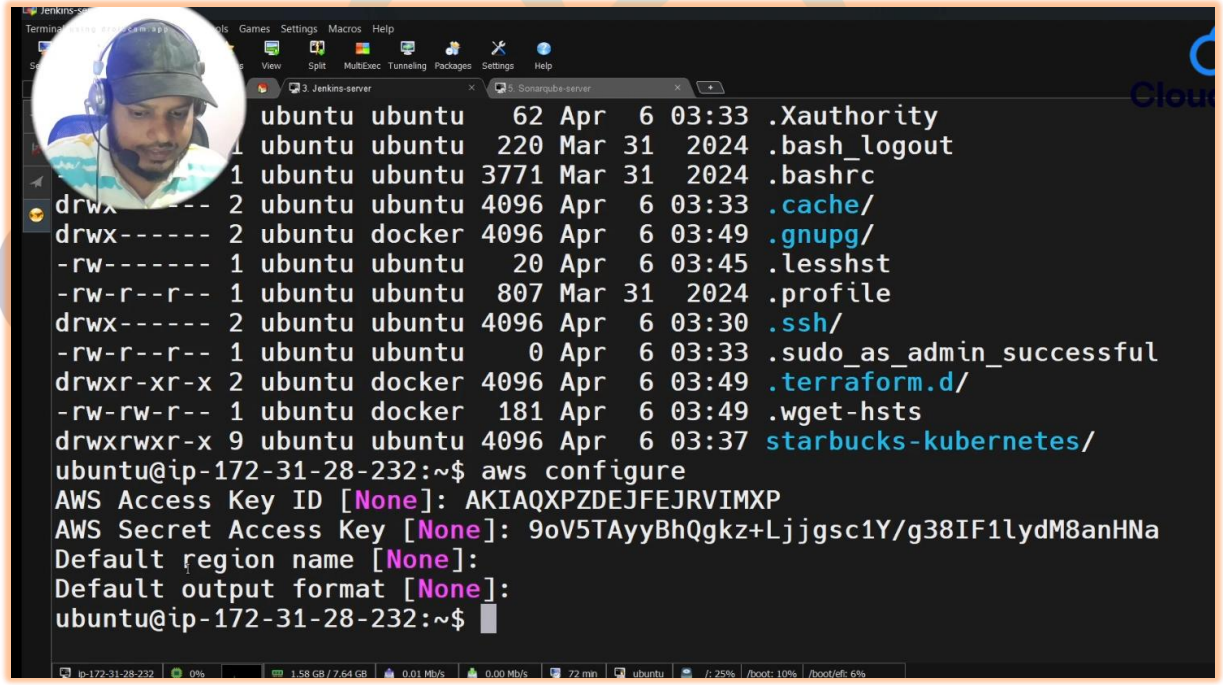
Create IAM user for AWS credentials to auth for EKS cluster setup







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
```

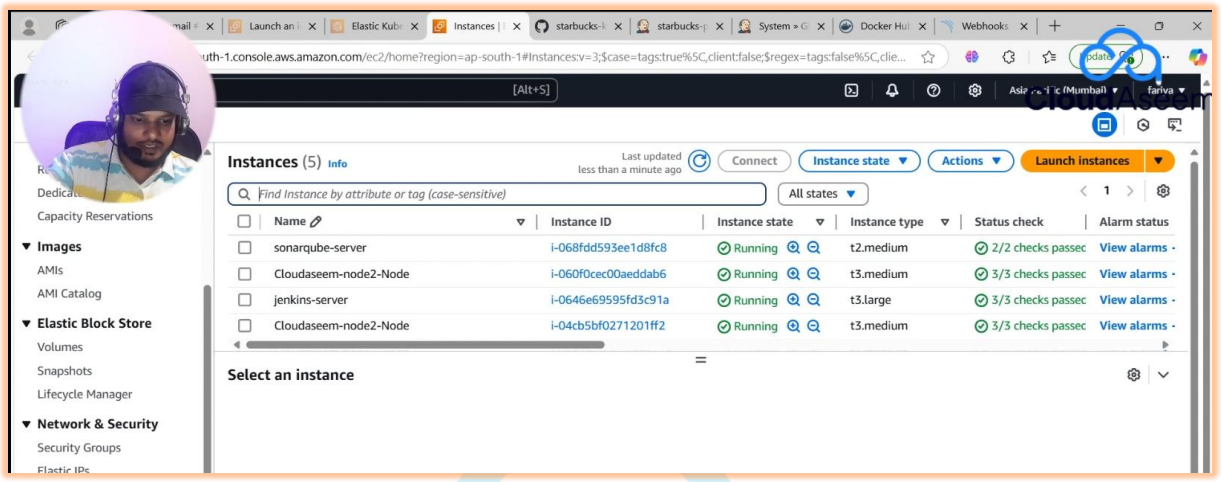
The above command is crucial when setting up an EKS cluster because it enables IAM roles for service accounts (IRSA)
Amazon EKS uses OpenID Connect (OIDC) to authenticate Kubernetes service accounts with IAM roles.
Associating the IAM OIDC provider allows Kubernetes workloads (Pods) running in the cluster 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.

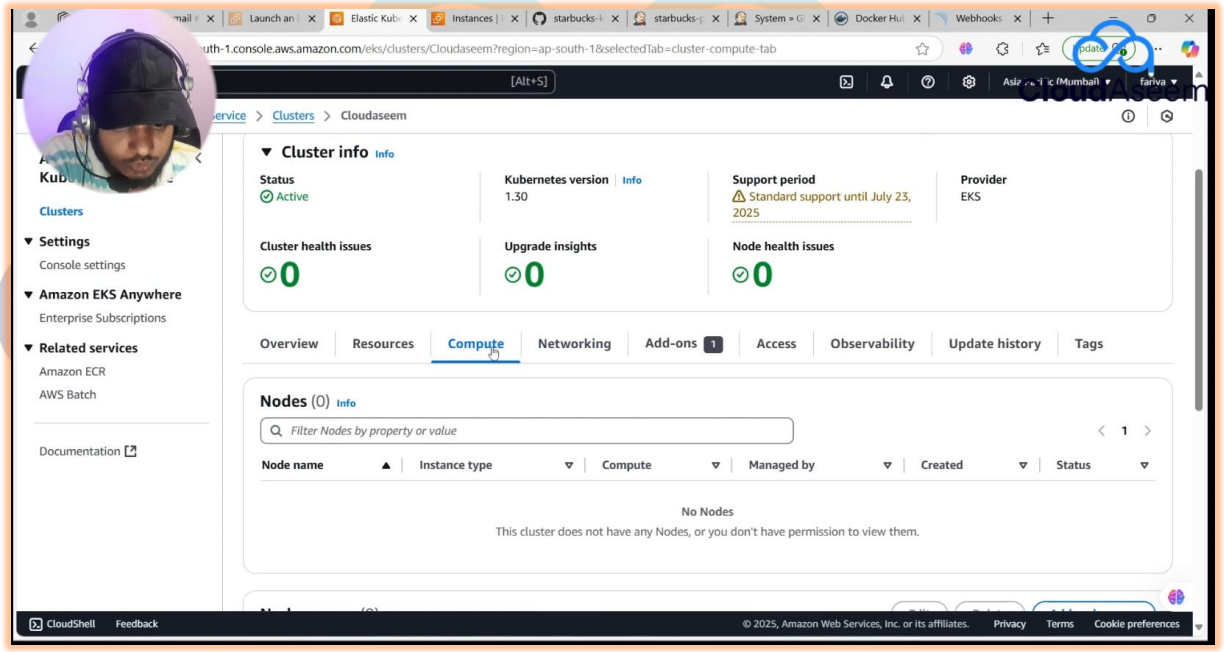
(c)
Before executing the below command, in the 'ssh-public-key' keep the '<PEM FILE NAME>' (dont give .pem. Just give the pem file name) which was used to create Jenkins Server

```
eksctl create nodegroup --cluster=Cloudaseem \
                --region=ap-south-1 \
                --name=node2 \
                --node-type=t3.medium \
                --nodes=3 \
                --nodes-min=2 \
                --nodes-max=4 \
                --node-volume-size=20 \
                --ssh-access \
                --ssh-public-key=starbucks1 \
                --managed \
                --asg-access \
                --external-dns-access \
                --full-ecr-access \
                --appmesh-access \
```

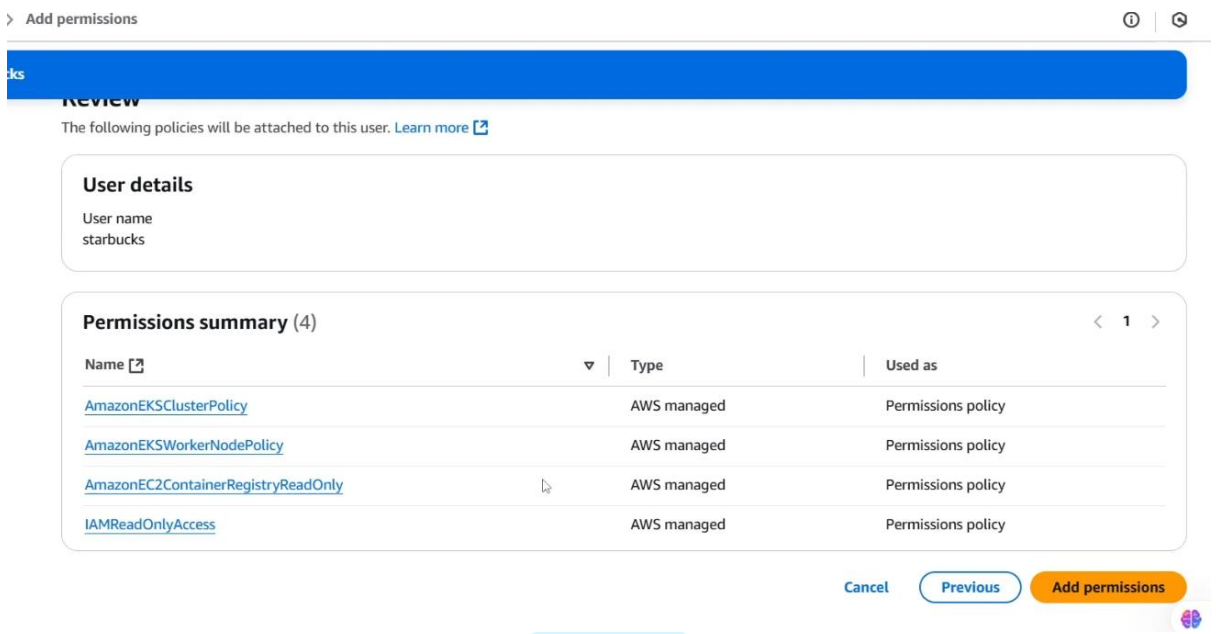
--alb-ingress-access

It will take 5-10 minutes





Add more policies to Starbucks user

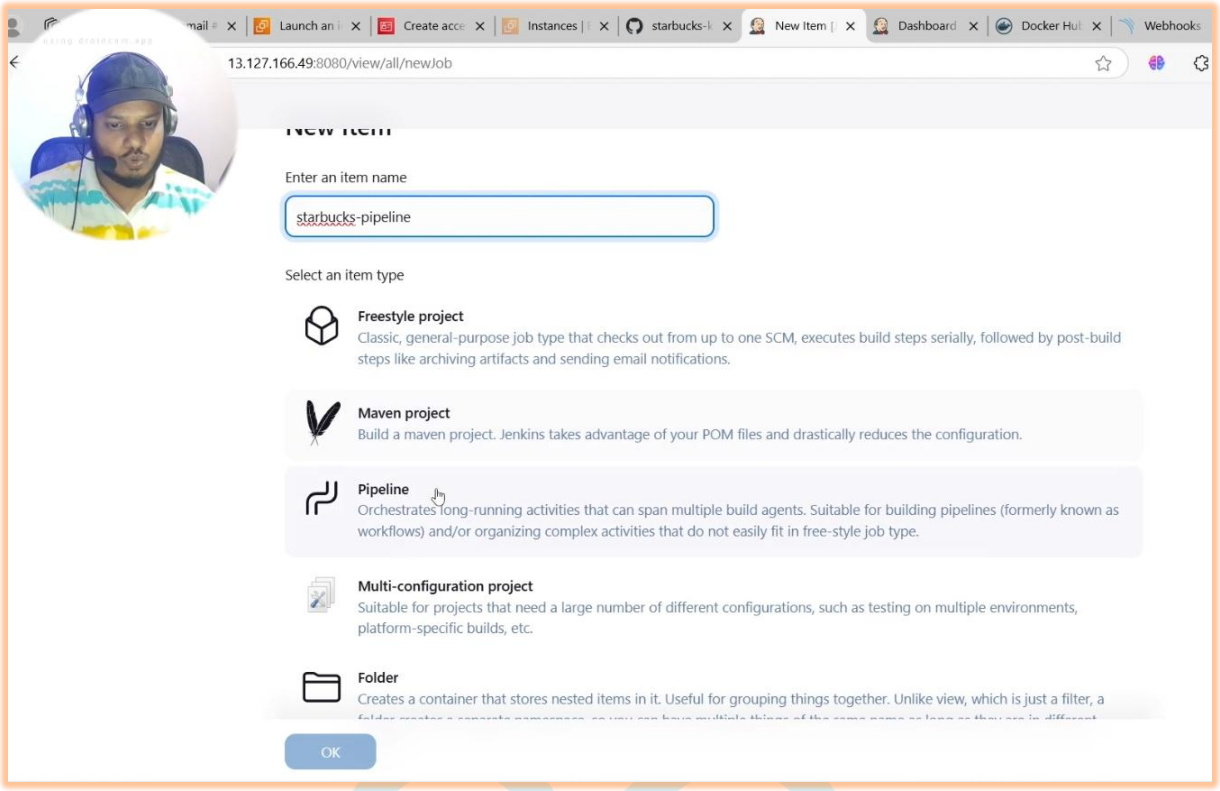


Stage – 2

Lets setup our first Jenkins pipeline

Create Job for

Let’s add a pipeline , to test the Github Clone stage of Private Registry



🔐 **1. Add GitHub Credentials to Jenkins**

1. **Go to Jenkins Dashboard > Manage Jenkins > Credentials.**
2. **Add a new credential:**
   - **Kind: Username with password**
   - **Username: Your GitHub username**
   - **Password: Your GitHub Personal Access Token (PAT)**
   - **ID: github-creds (you'll use this in the pipeline)**
   - **Description: GitHub Access Token**

Definition

Pipeline script

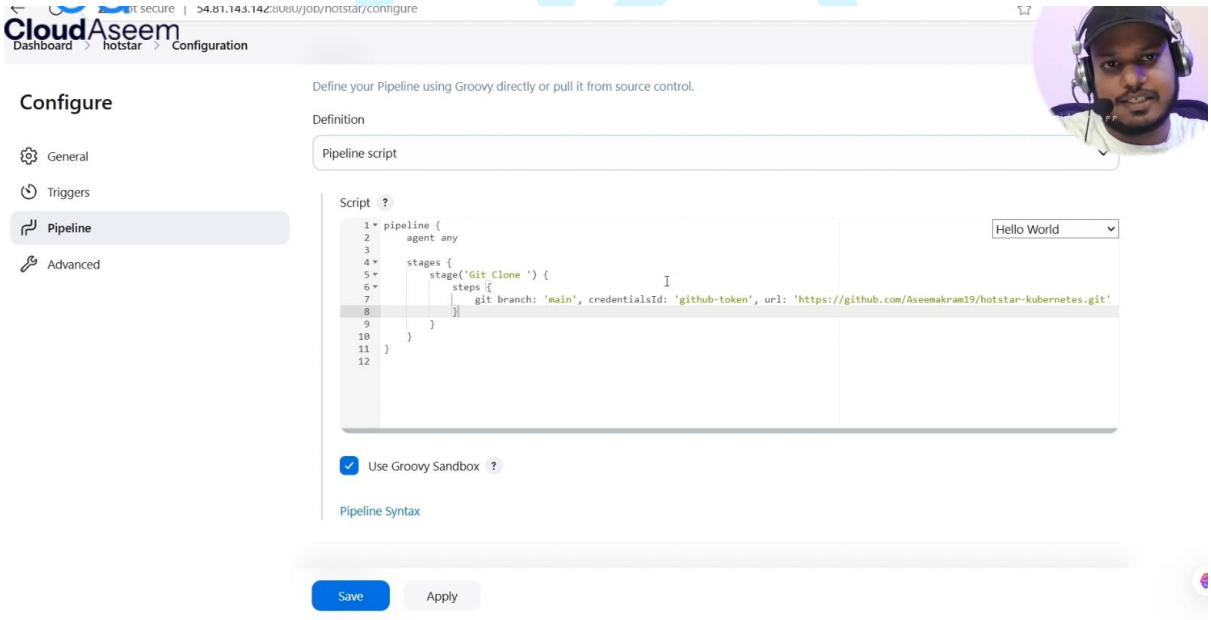
Script ?

```
pipeline {
    agent any

    stages {
        stage('Git-clone') {
            steps {
                git branch: 'main', credentialsId: 'github-token', url: 'https://github.com/Aseemakram19/starbu
            }
        }
    }
}
```

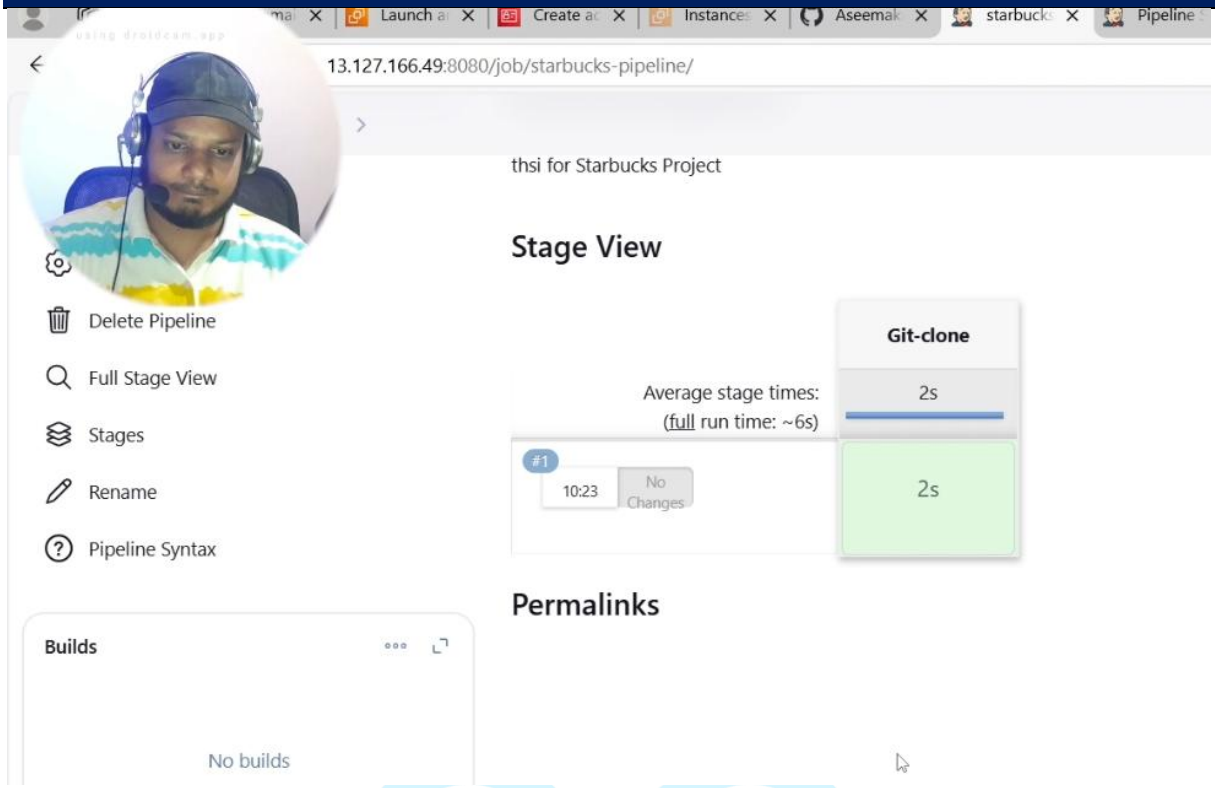
Hello World

Use Groovy Sandbox ?

Pipeline Syntax



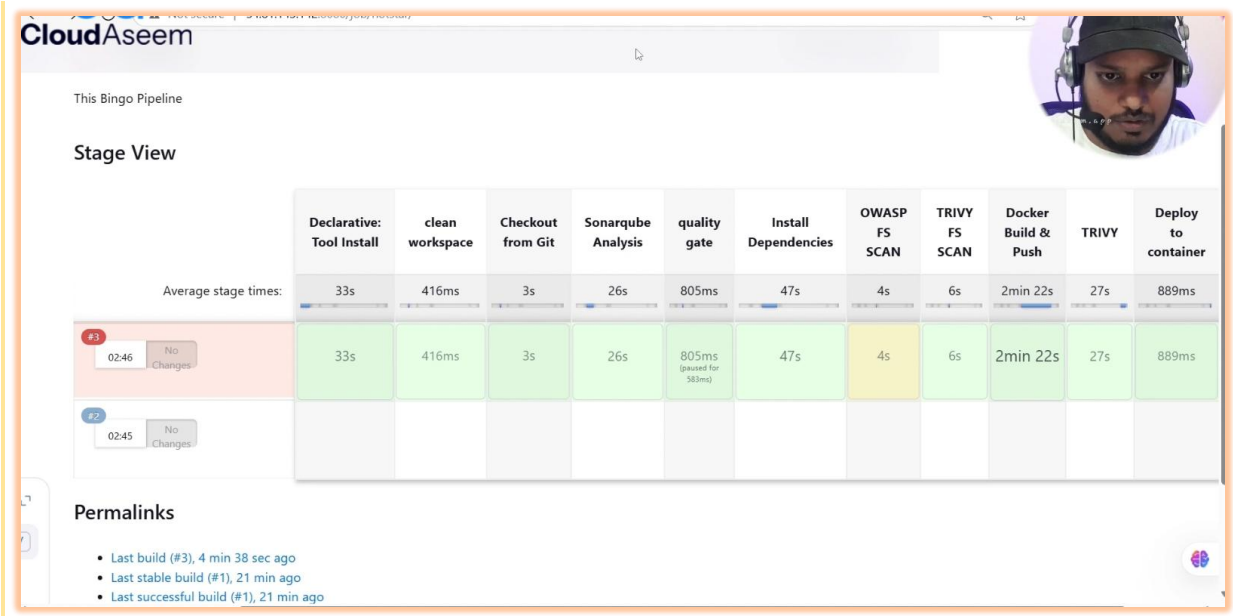
Apply and Save and click on Build

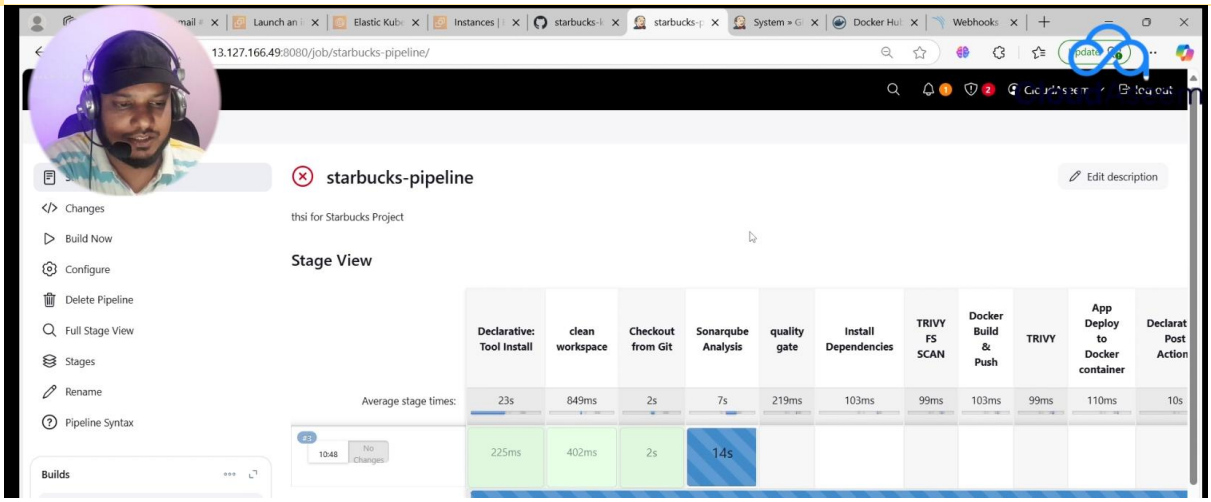


Let’s add a pipeline of Project

```
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', credentialsId: 'github-token', url: 'https://github.com/Aseemakram19/starbucks-kubernetes.git'
      }
    }
    stage("Sonarqube Analysis "){
      steps{
        withSonarQubeEnv('SonarQube') {
          sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=starbucks \
          -Dsonar.projectKey=starbucks '''
        }
      }
    }
    stage("quality gate"){
      steps {
        script {
          waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
        }
```

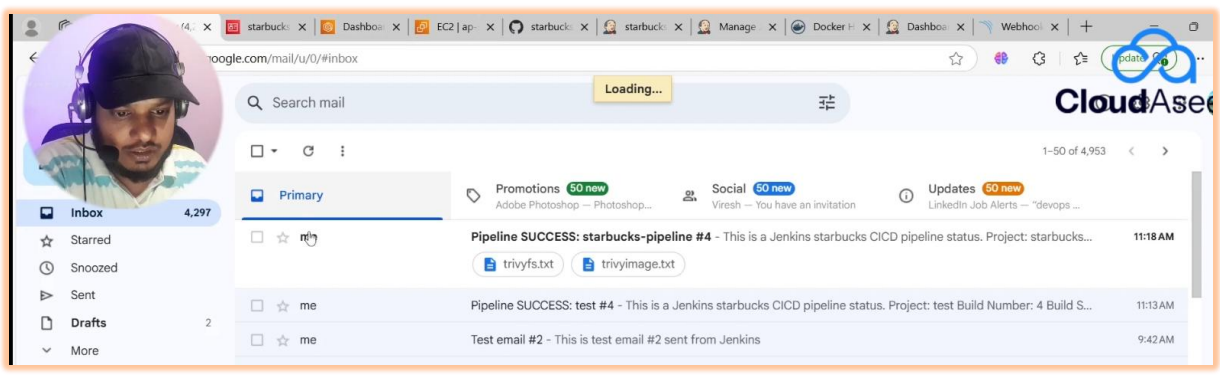
```
        }
    }
    stage('Install Dependencies') {
        steps {
            sh "npm install"
        }
    }
    stage('TRIVY FS SCAN') {
        steps {
            sh "trivy fs . > trivyfs.txt"
        }
    }
    stage("Docker Build & Push"){
        steps{
            script{
              withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
                sh "docker build -t starbucks ."
                sh "docker tag starbucks aseemakram19/starbucks:latest "
                sh "docker push aseemakram19/starbucks:latest "
              }
            }
        }
    }
    stage("TRIVY"){
        steps{
            sh "trivy image aseemakram19/starbucks:latest > trivyimage.txt"
        }
    }
    stage('App Deploy to Docker container'){
        steps{
            sh 'docker run -d --name starbucks -p 3000:3000 aseemakram19/starbucks: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 starbucks 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'
        )
       }
    }

  }

}
```



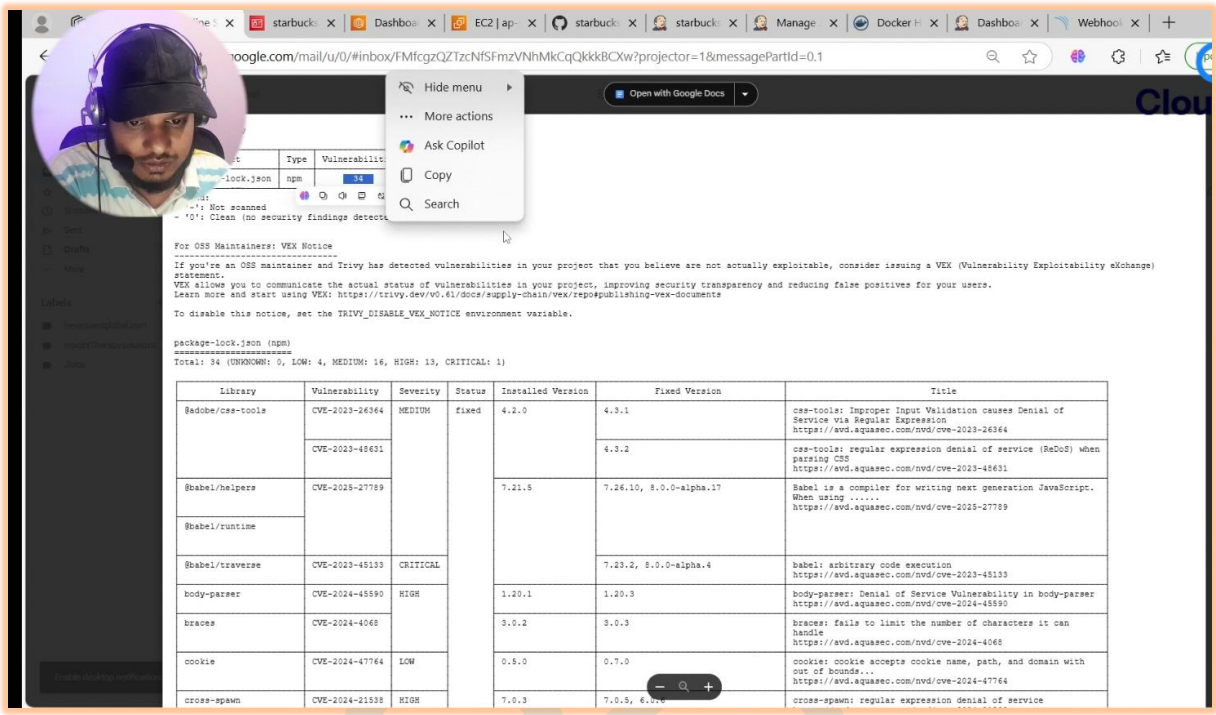


Result :

- Emails received



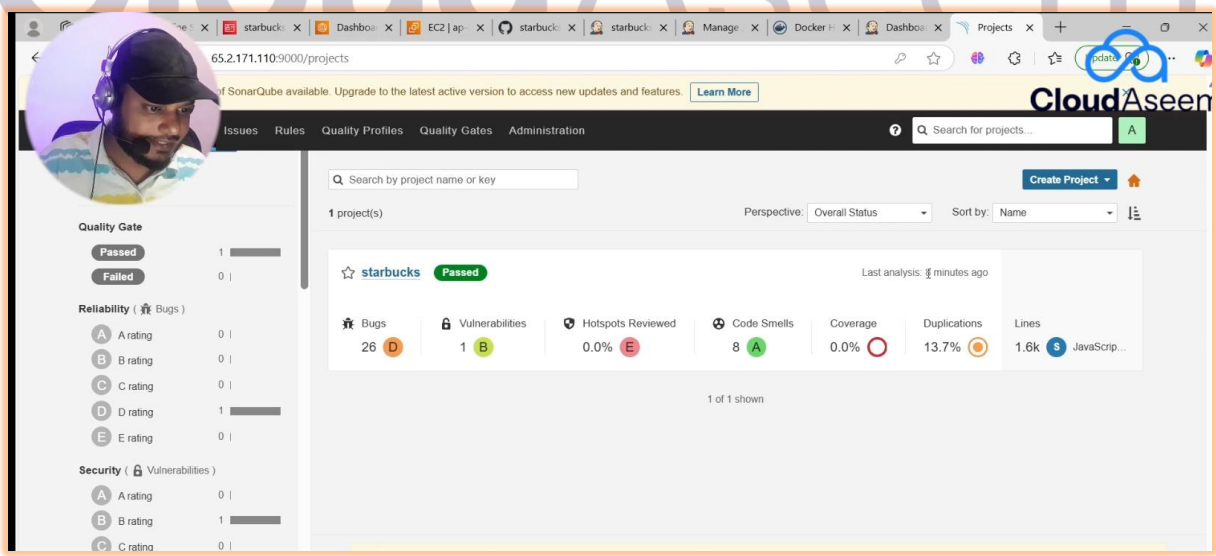
Report attached in emails

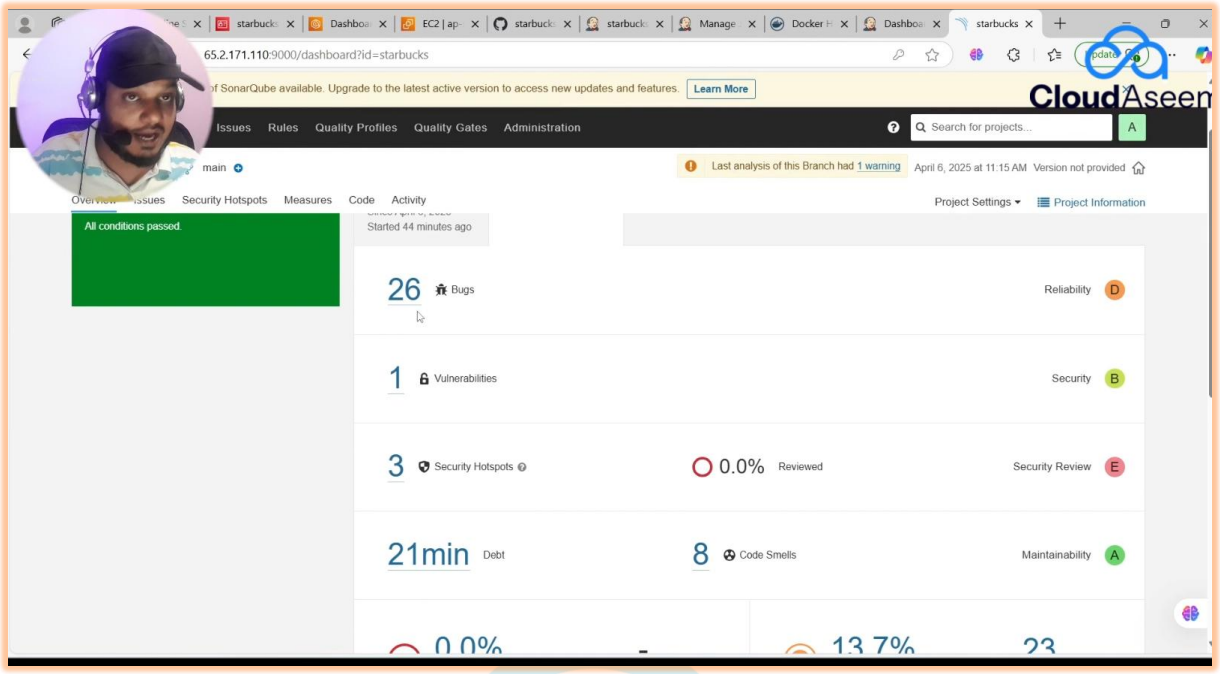


Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
@adobe/css-tools	CVE-2023-26364	MEDIUM	fixed	4.2.0	4.3.1	css-tools: Improper Input Validation causes Denial of Service via Regular Expression https://avd.aquasec.com/nvd/cve-2023-26364
	CVE-2023-48631				4.3.2	css-tools: regular expression denial of service (ReDoS) when parsing CSS https://avd.aquasec.com/nvd/cve-2023-48631
@babel/helpers	CVE-2025-27789			7.21.5	7.26.10, 8.0.0-alpha.17	Babel is a compiler for writing next generation JavaScript. When using https://avd.aquasec.com/nvd/cve-2025-27789
@babel/runtime						
@babel/traverse	CVE-2023-45133	CRITICAL			7.23.2, 8.0.0-alpha.4	babel: arbitrary code execution https://avd.aquasec.com/nvd/cve-2023-45133
body-parser	CVE-2024-45590	HIGH		1.20.1	1.20.3	body-parser: Denial of Service Vulnerability in body-parser https://avd.aquasec.com/nvd/cve-2024-45590
braces	CVE-2024-4068			3.0.2	3.0.3	braces: fails to limit the number of characters it can handle https://avd.aquasec.com/nvd/cve-2024-4068
cookie	CVE-2024-47764	LOW		0.5.0	0.7.0	cookie: cookie accepts cookie name, path, and domain with out of bounds... https://avd.aquasec.com/nvd/cve-2024-47764
cross-spawn	CVE-2024-21538	HIGH		7.0.3	7.0.5, 6.0.6	cross-spawn: regular expression denial of service

3. SonarQube Project scanned :

You can see the report has been generated and the status shows as passed. You can see that there are 943 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.

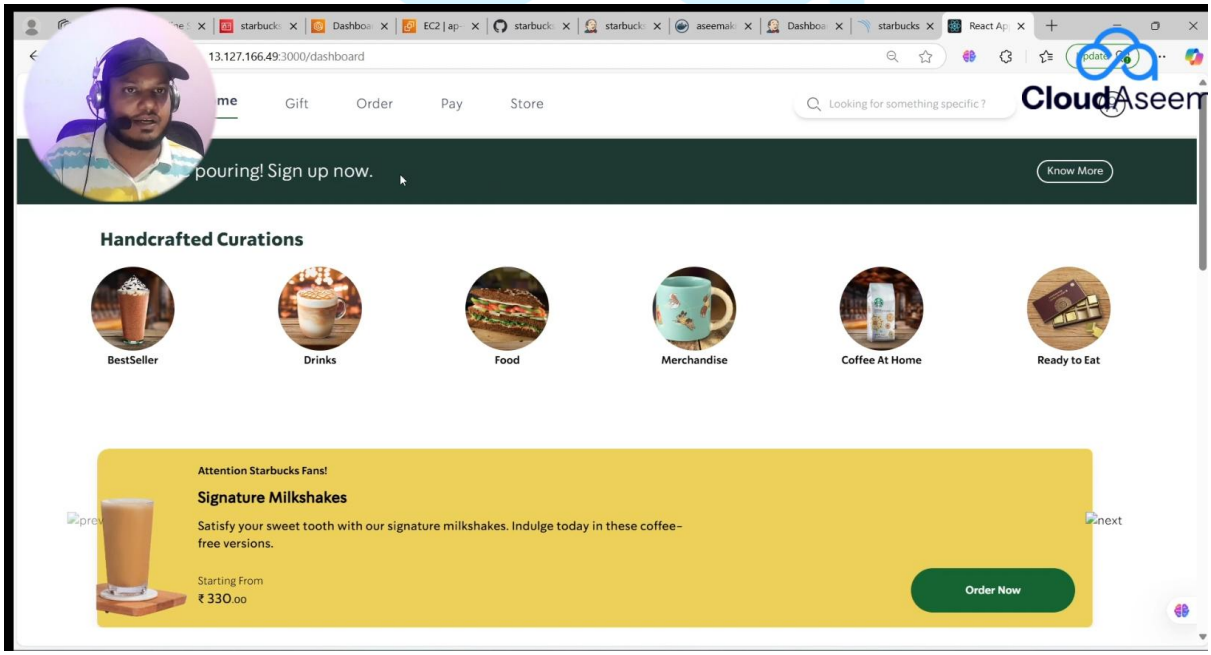




4. Application as Test Deploy as Docker container

<public-ip:3000>

Our Applciation is live with this output



Email alert with Post build

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

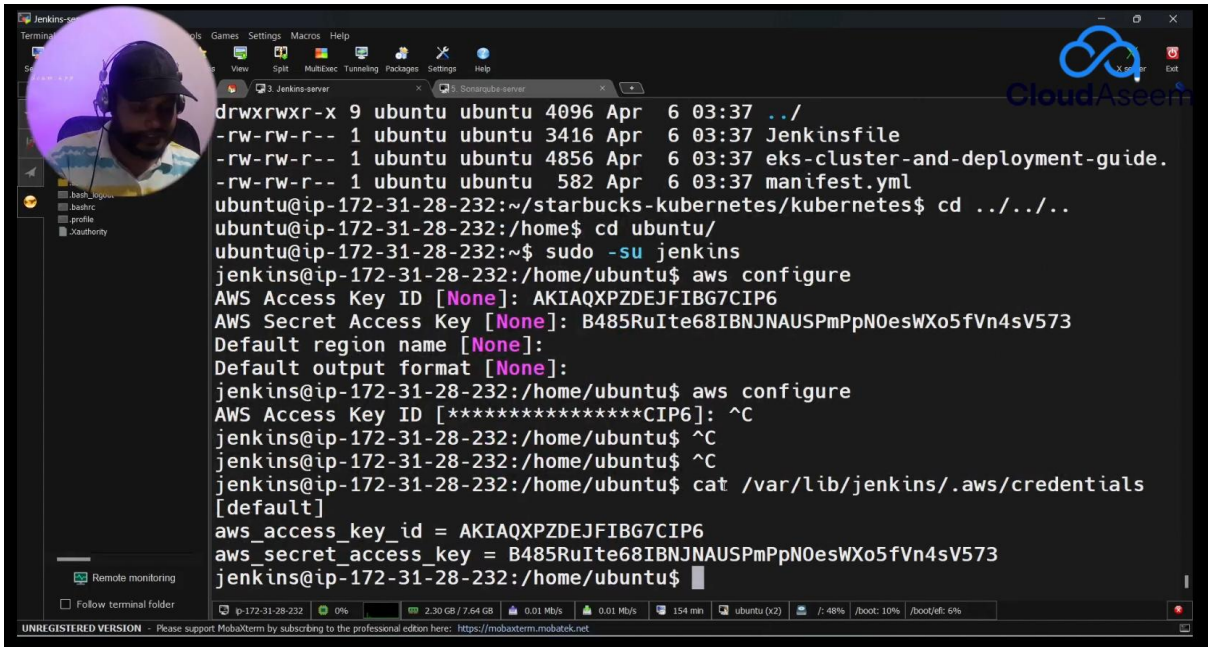
- Writes a kubeconfig file for `kubectl` in `/var/lib/jenkins/.kube/config`

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

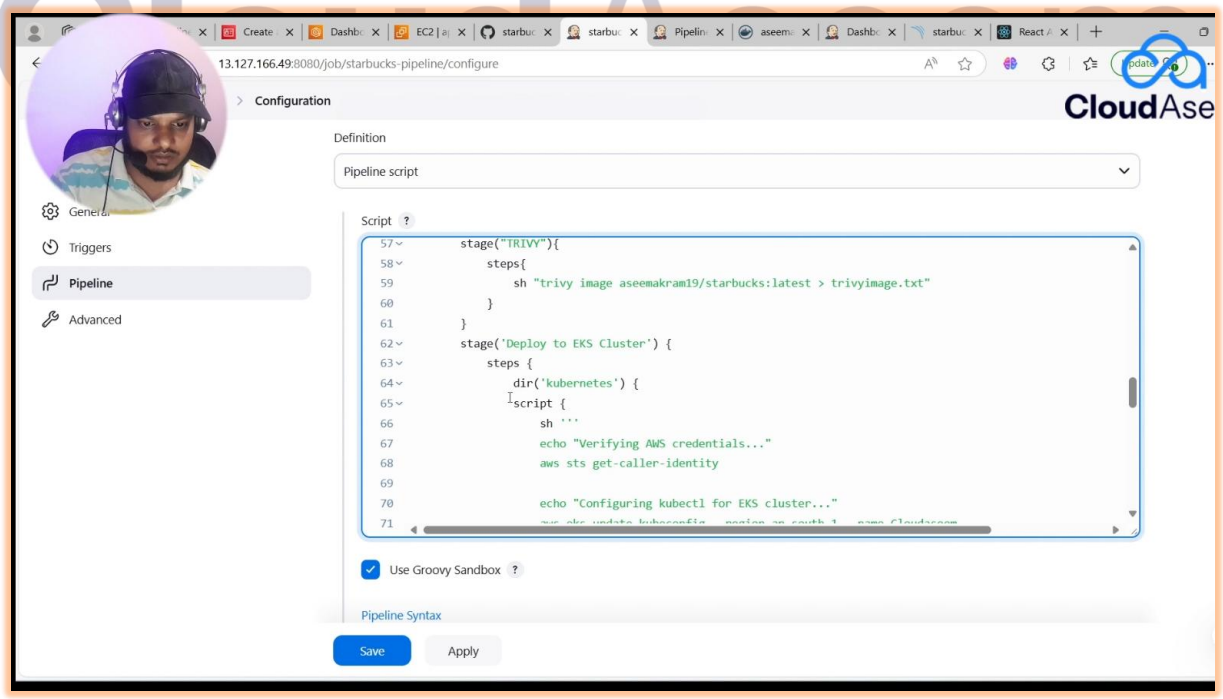
```
stage('Deploy to EKS Cluster') {
          steps {
             dir('kubernetes') {
             script {
                sh '''
                echo "Verifying AWS credentials..."
                aws sts get-caller-identity

                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
                '''
             }
          }
       }

    }
```

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

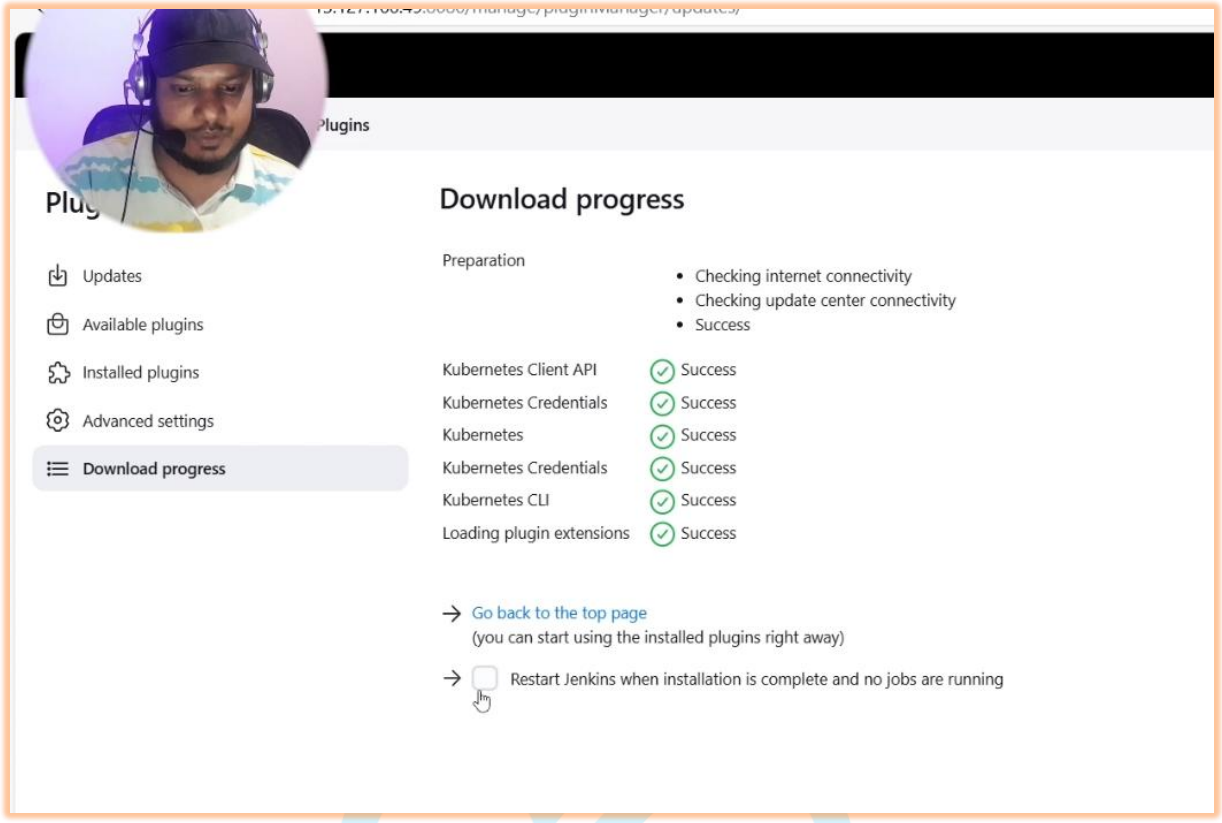
manifest.yml file

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: starbucks-deployment
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
```

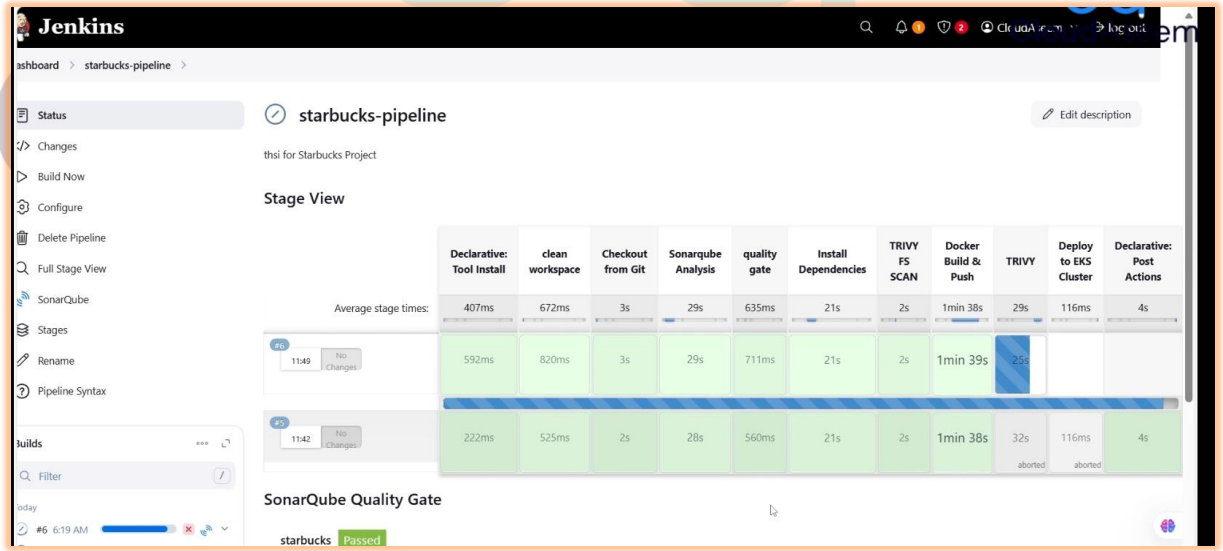
```
  selector:
    matchLabels:
      app: starbucks
  template:
    metadata:
      labels:
        app: starbucks
    spec:
      containers:
      - name: starbucks-container
        image: aseemakram19/starbucks
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: starbucks-service
spec:
  type: LoadBalancer
  selector:
    app: starbucks
  ports:
  - port: 80
    targetPort: 3000
```

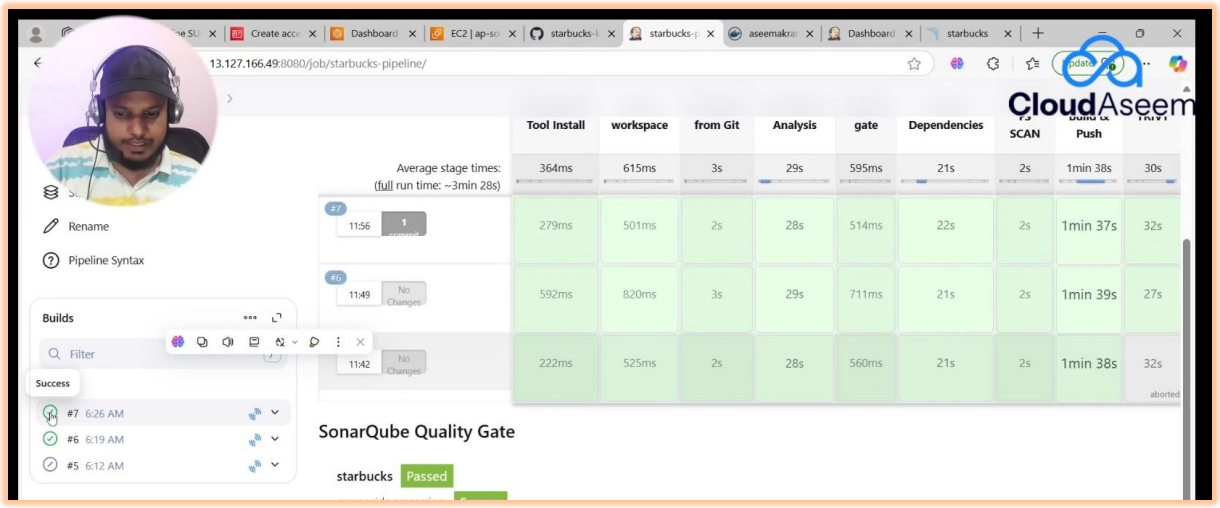


Add this additional Plguins jenkins

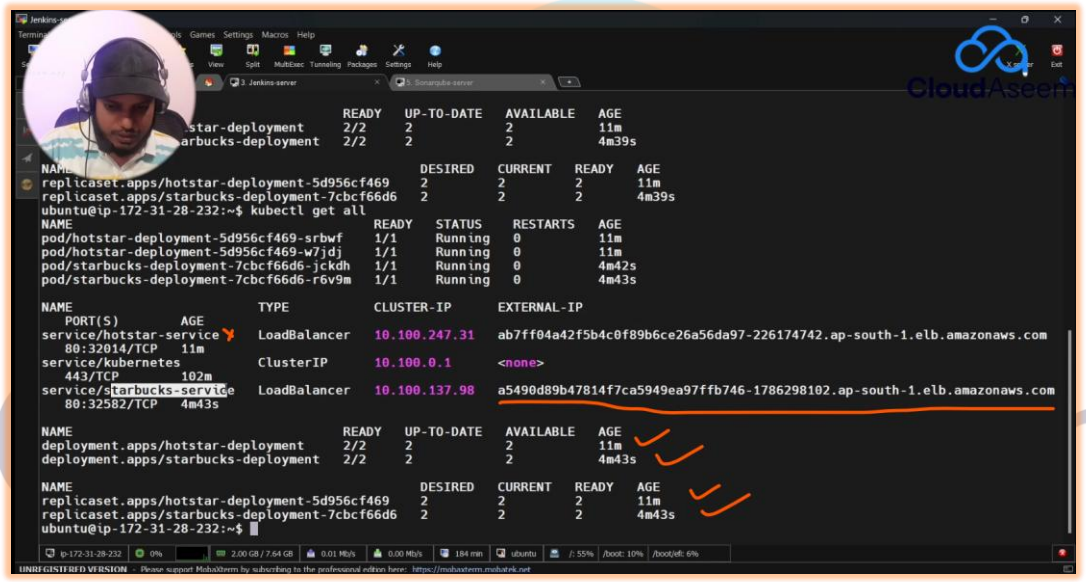


and execute the pipeline now

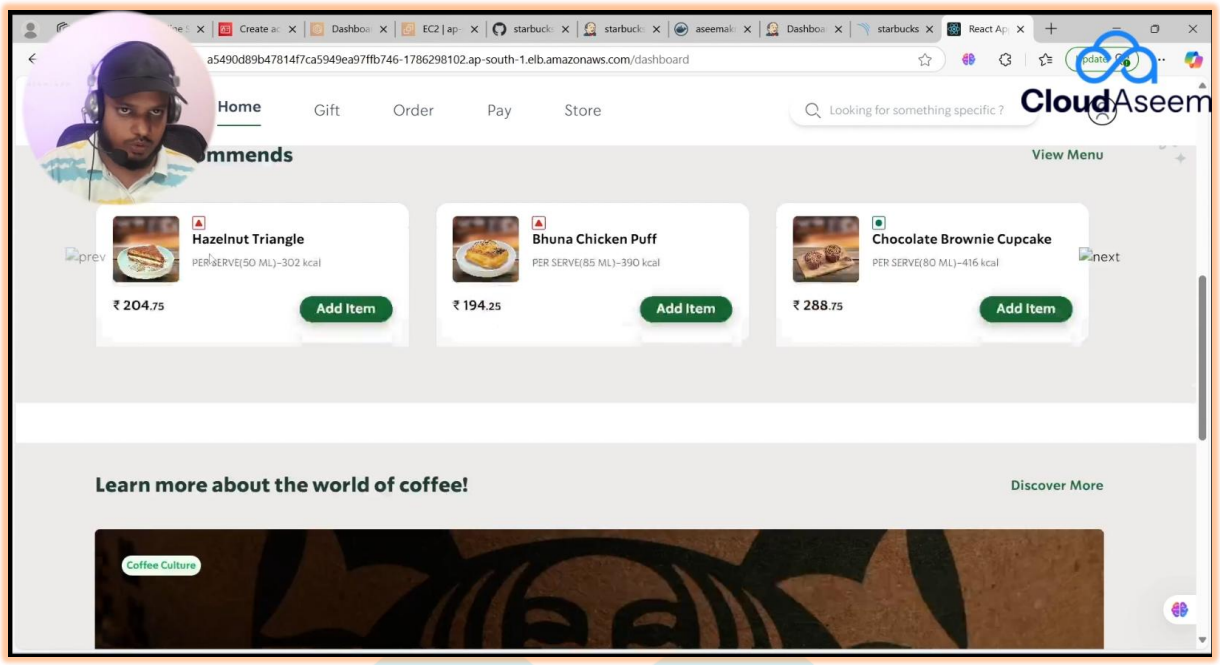




kubectl get all



Access app with Loadbalancer

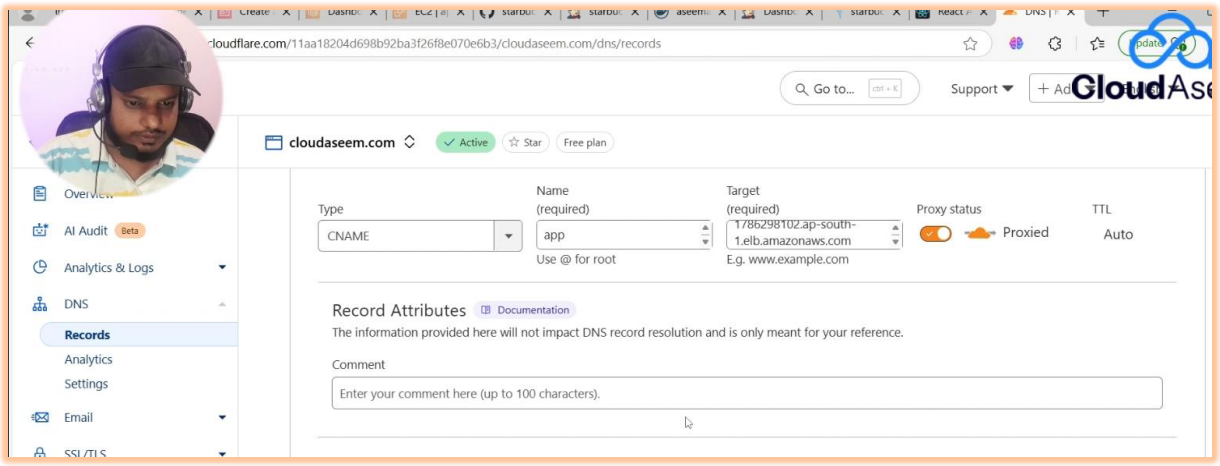


Add Doamin to Our application

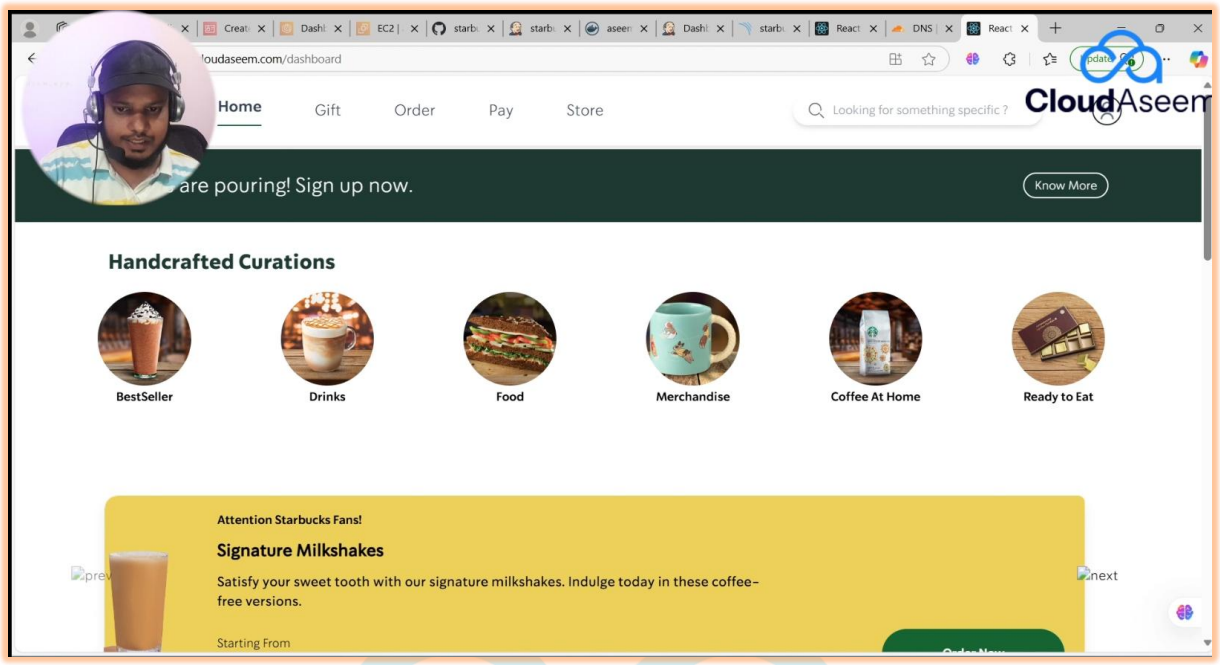
Add Loadbalance in Cloudflare to apply domain and ssl to access by client with cname entry

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

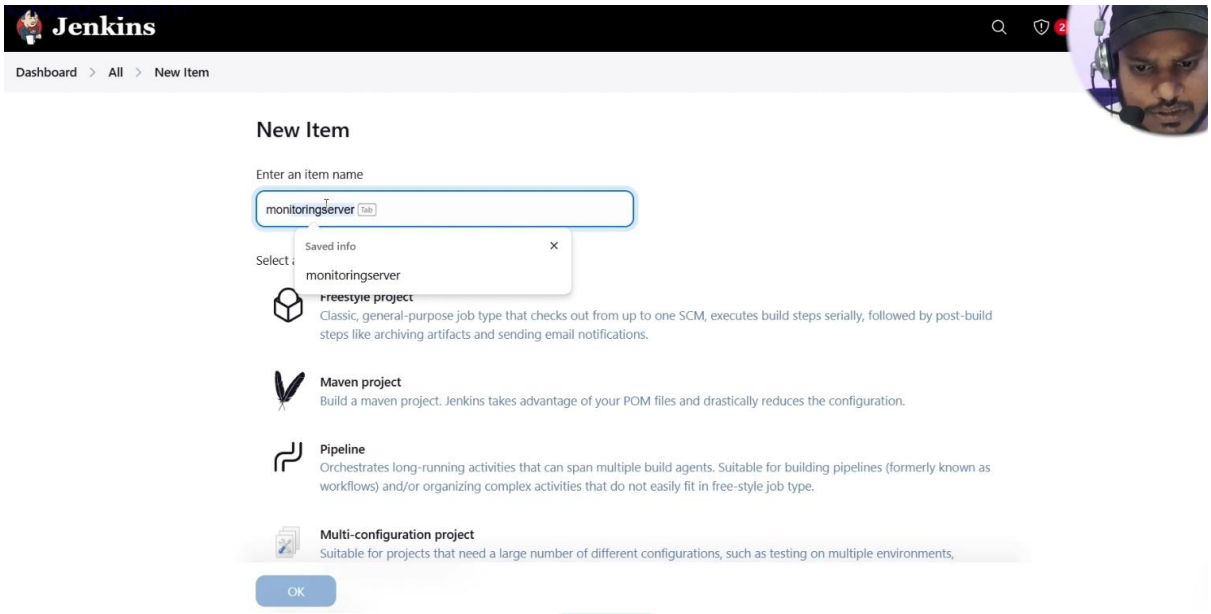


Part – 03

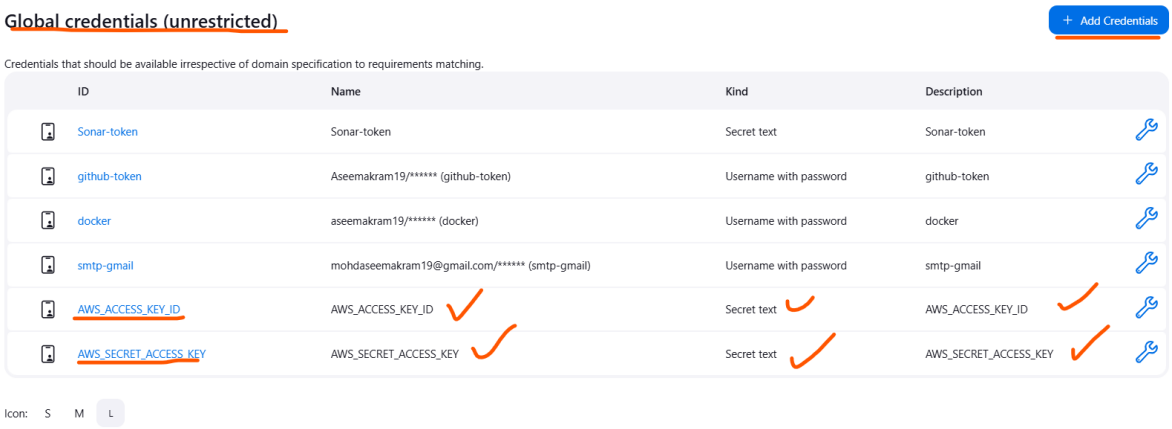
monitoring server with JENKINS + TERRAFORM = MONITORING SERVER SETUP

Create a new pipeline as below :

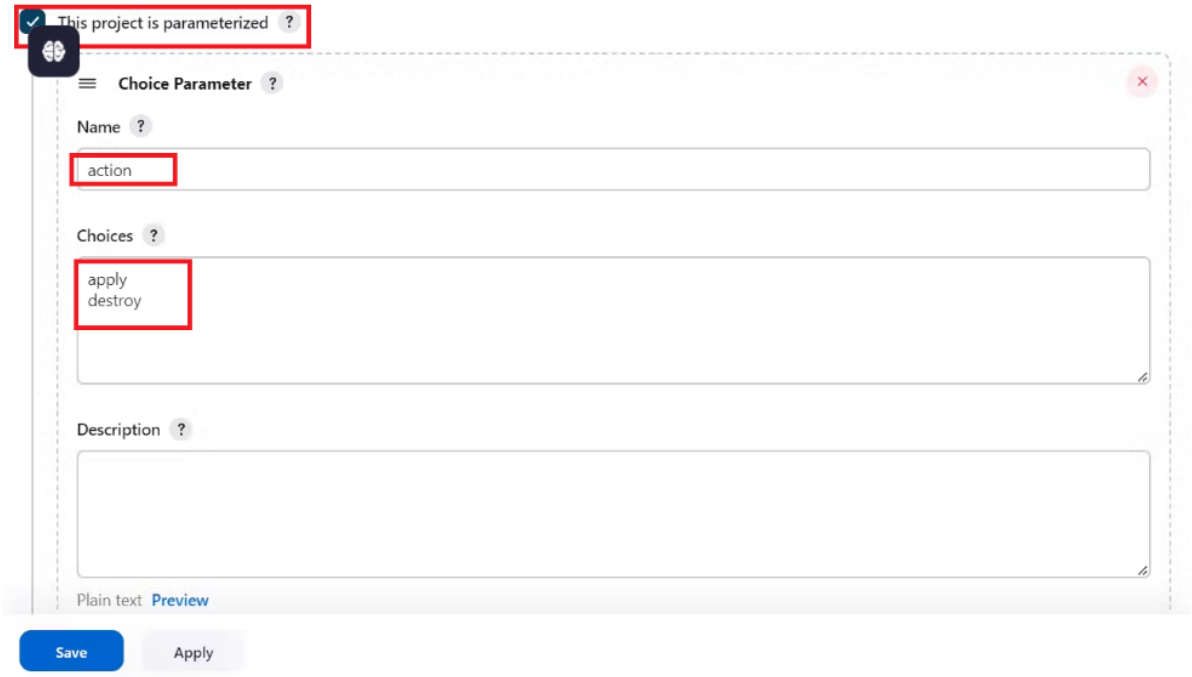
Monitoring Server setup with Jenkins + Terraform



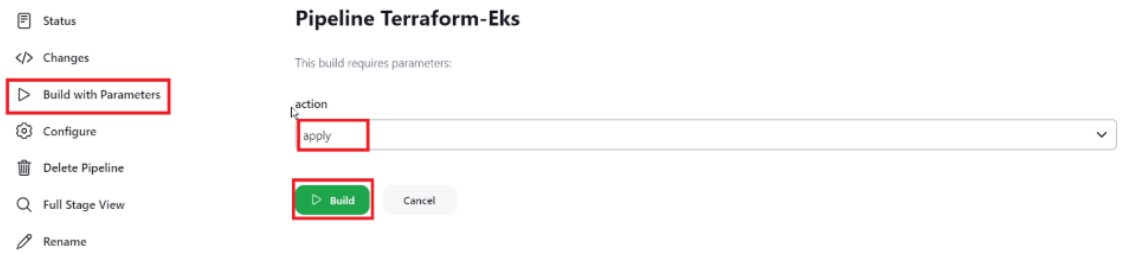
add Secret in AWS credentials



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

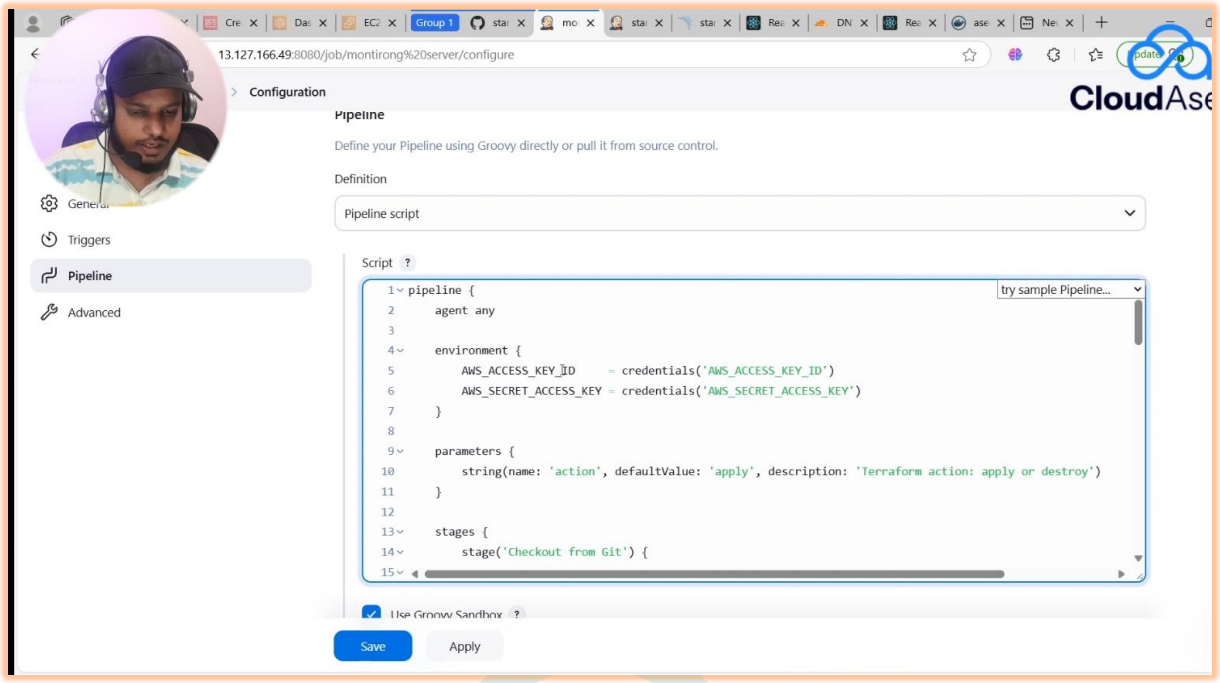


Let's apply and save and Build with parameters and select action as apply

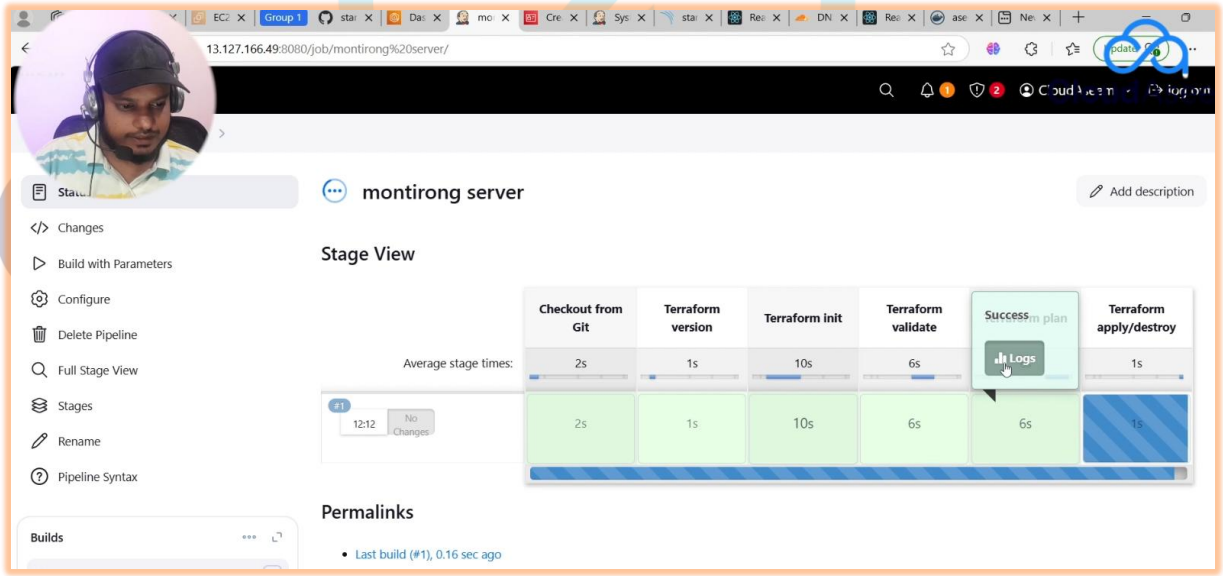



```
pipeline {
  agent any

  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/starbucks-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.'
    }
  }
}
```

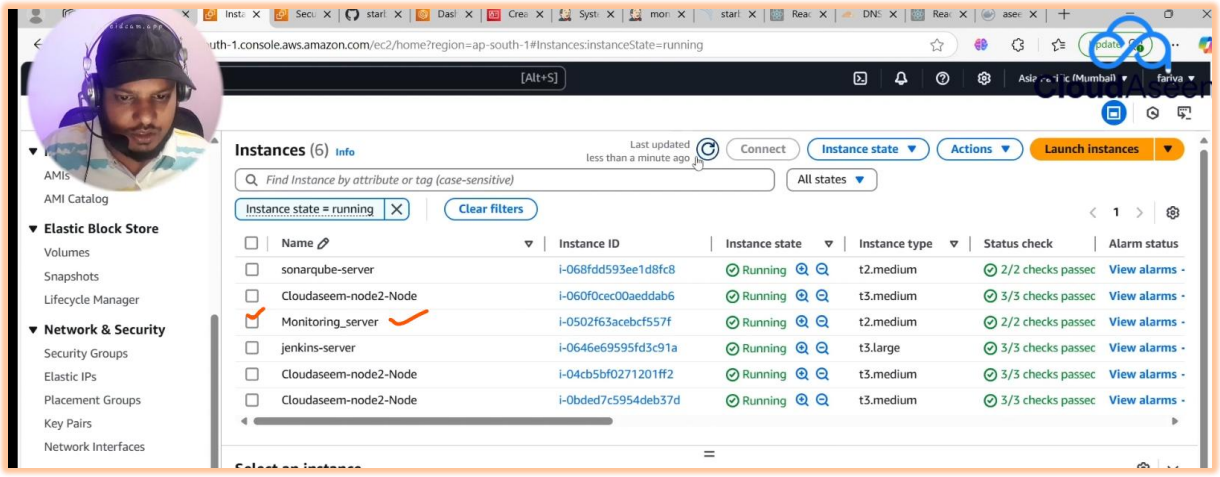


Note: Store aws secret and secret access key in jenkins crednetials s below



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.

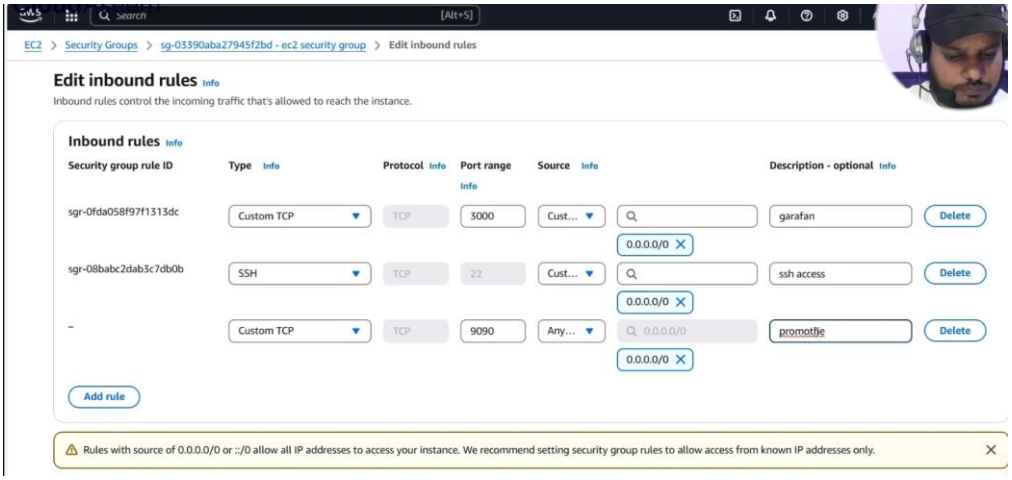
1. **Grafana installation**

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

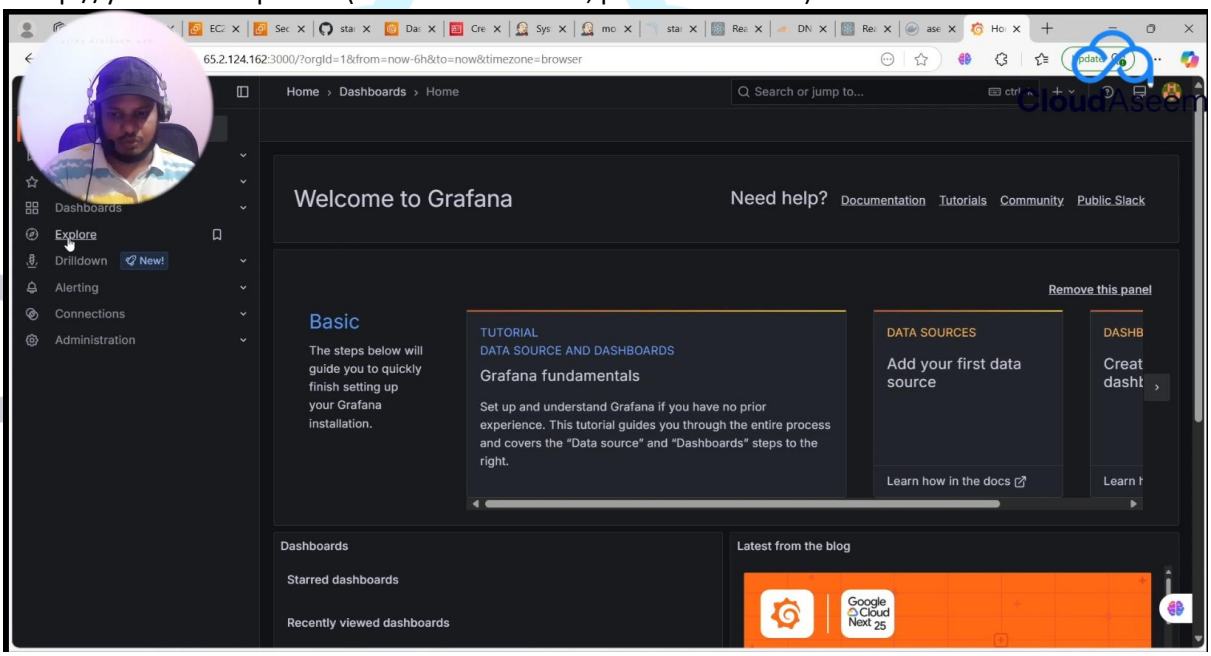
```
#!/bin/bash
# Script to install Grafana on a Linux instance

# 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)



2. **Install Prometheus**

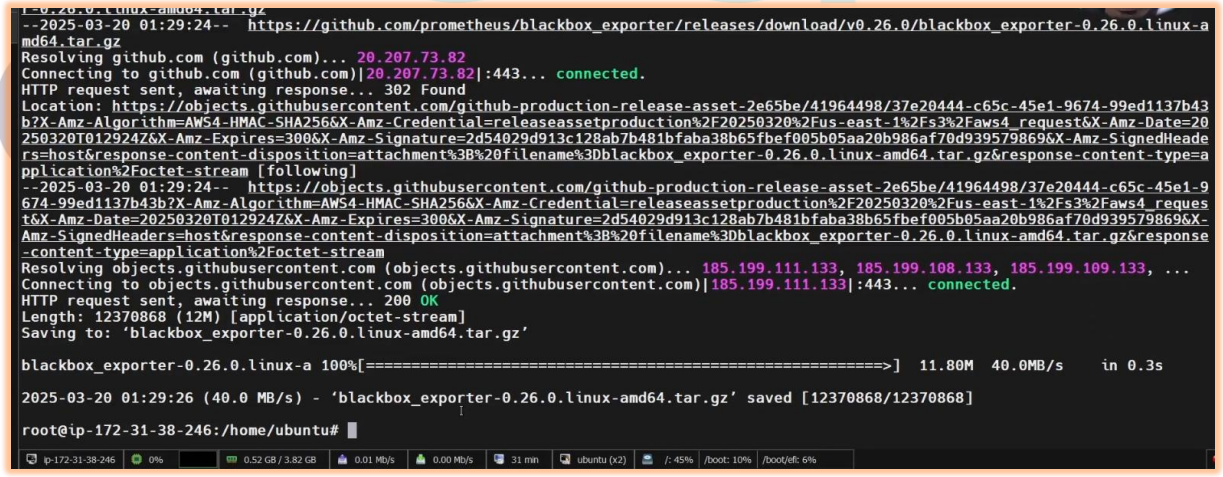
#!/bin/bash

```
#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



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: IP:9115  # The blackbox exporter's real hostname.


- 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

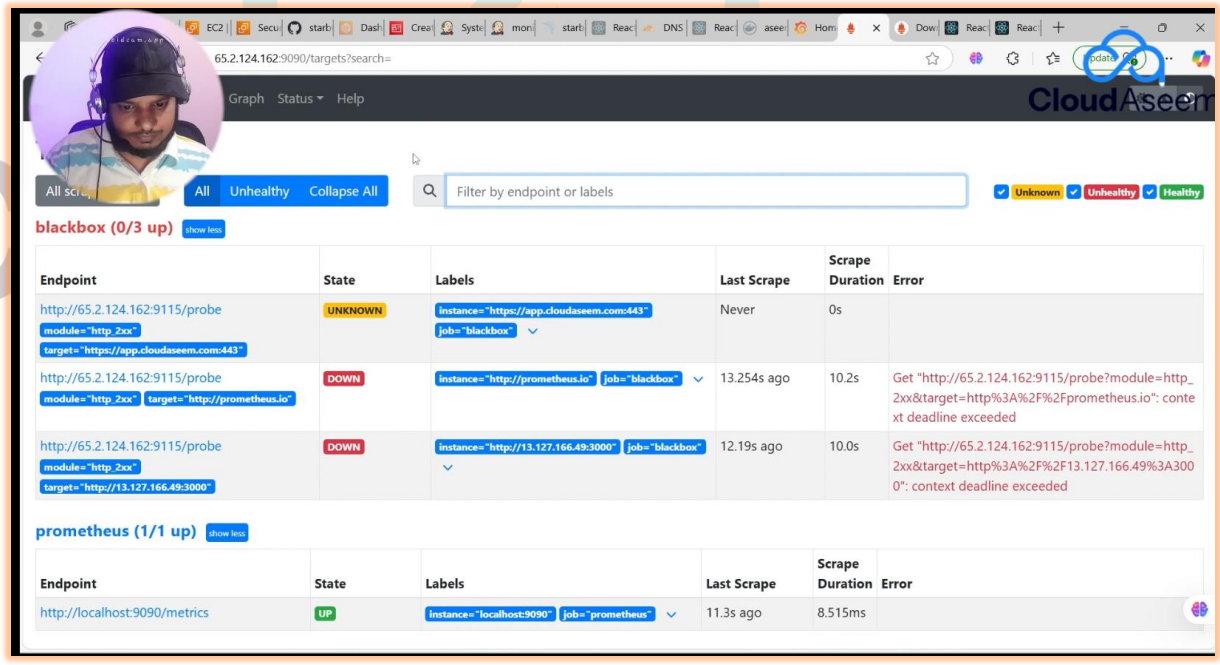
pgrep Prometheus and kill PID

```
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 &

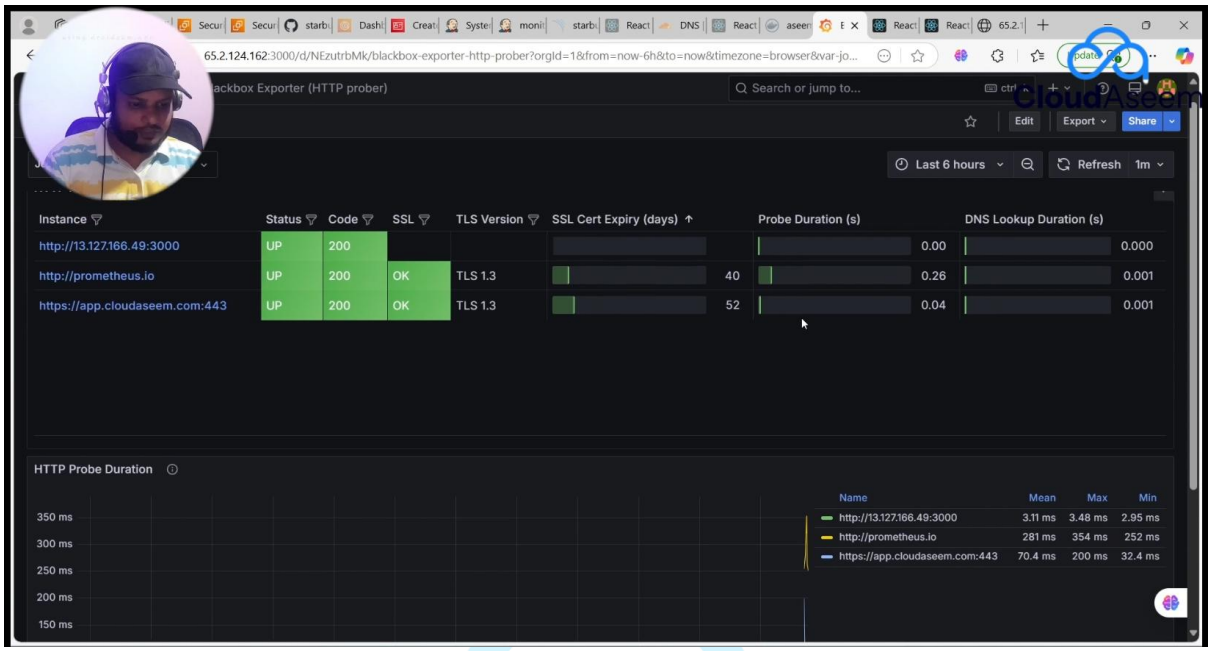
Step 3: Connect Prometheus to Grafana

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.**



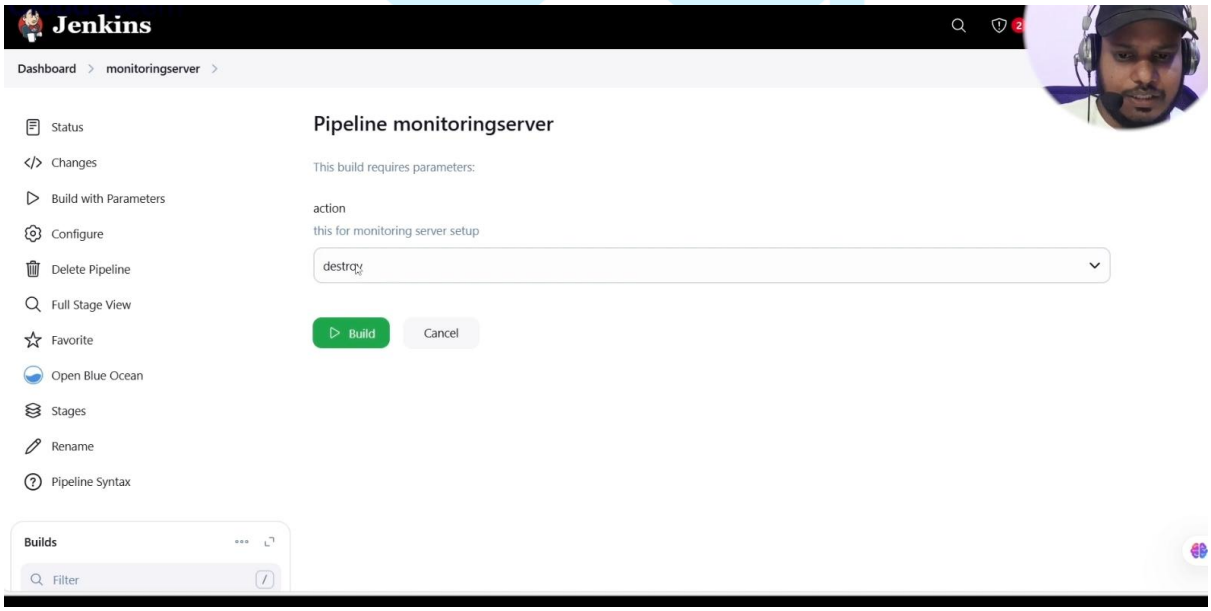
Add
Go to **Dashboards** → **Import**.
Enter **Dashboard ID: 13659** (Prometheus Blackbox Exporter).
Click **Load**.
Select **Prometheus** as the **Data Source**.

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
