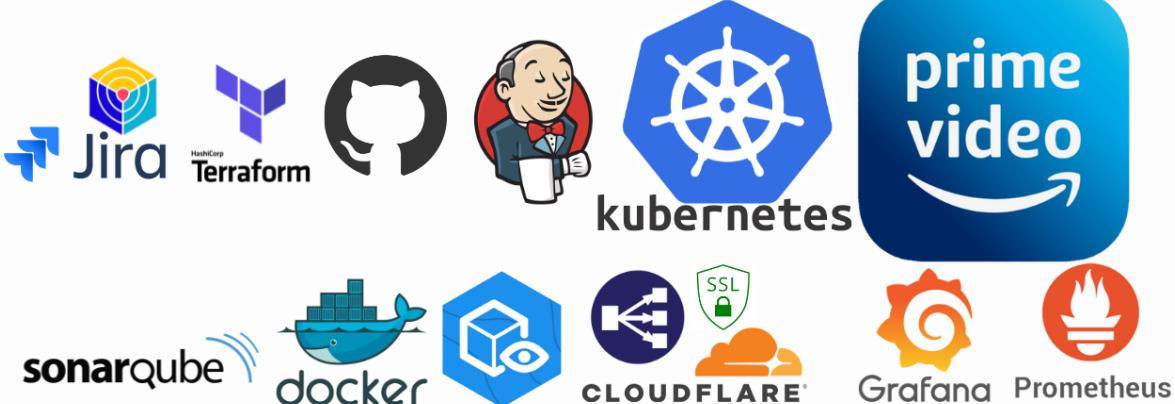


Amazon-prime-video App — End-to-End Secure & Scalable Deployment with DevSecOps & Jenkins !

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

DEVSECOPS PROJECT

Production Deployment of Amazon Prime Video App on Kubernetes Eks | Monitoring



The slide features a central title "Production Deployment of Amazon Prime Video App on Kubernetes Eks | Monitoring" in large red and black font. Below the title is a grid of logos for various tools: Jira, Terraform, GitHub, Jenkins (represented by a cartoon character), Kubernetes, Prime Video, SonarQube, Docker, Cloudflare, Grafana, and Prometheus. The background of the slide has a faint watermark of the word "CloudAseem".

Follow me for projects Links :

YouTube: <https://youtu.be/1FNhSda0sV4> - @clouddevopswithaseem

Project GitHub Repo: [Aseemakram19/starbucks-kubernetes](https://github.com/Aseemakram19/starbucks-kubernetes)

LinkedIn: [Mohammed Aseem Akram](https://www.linkedin.com/in/mohammed-aseem-akram/)

Github Account: [Aseemakram19](https://github.com/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

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)

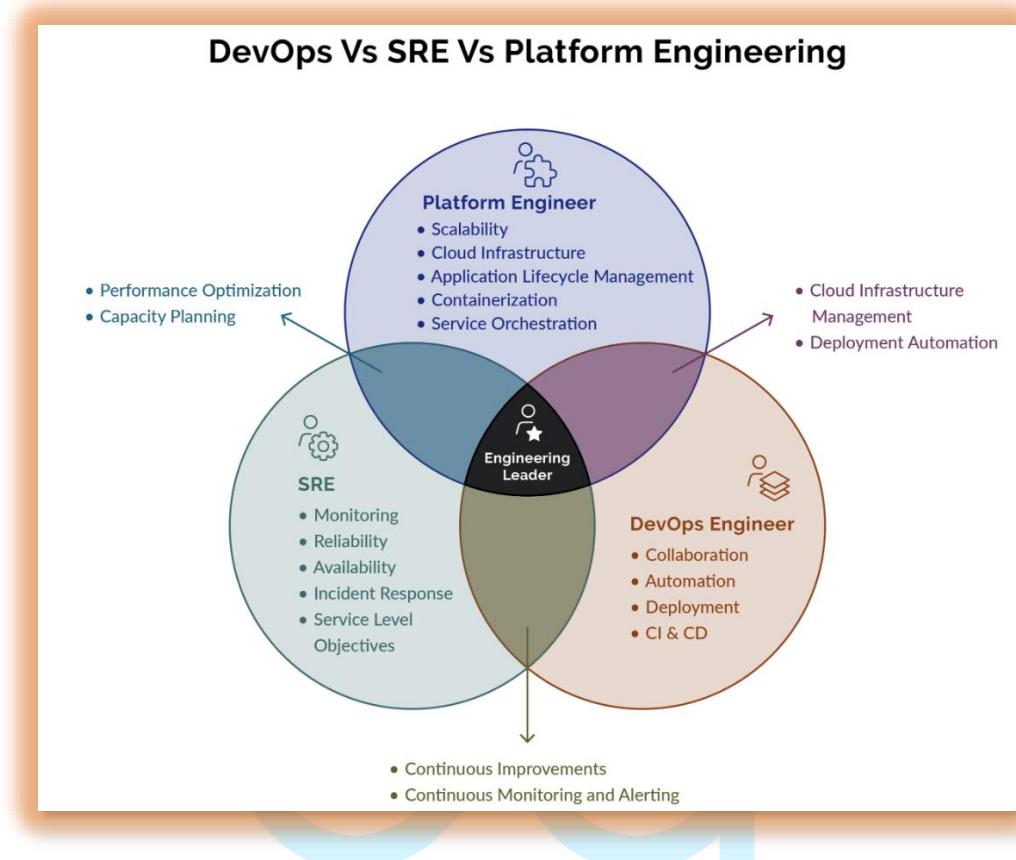
-  Ensures system reliability, availability, and performance
-  Implements monitoring, alerting, and incident response workflows
-  Automates failover, backup, and recovery strategies
-  Defines SLOs/SLIs and tracks error budgets for services



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

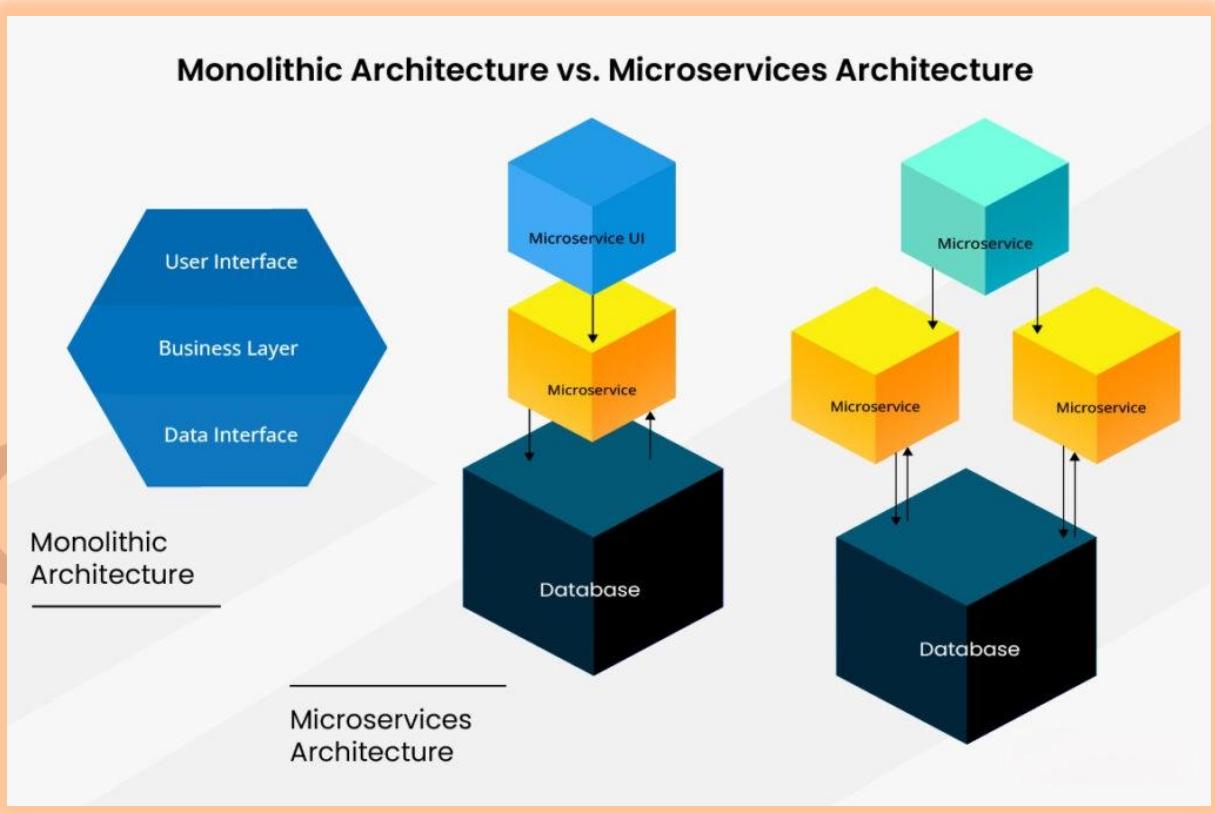
-  **Faster Development (initially)** — Simple to build and get started
-  **Easier Testing** — No inter-service communication to mock
-  **Simplified Deployment** — One deployment package to manage

! 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

E 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



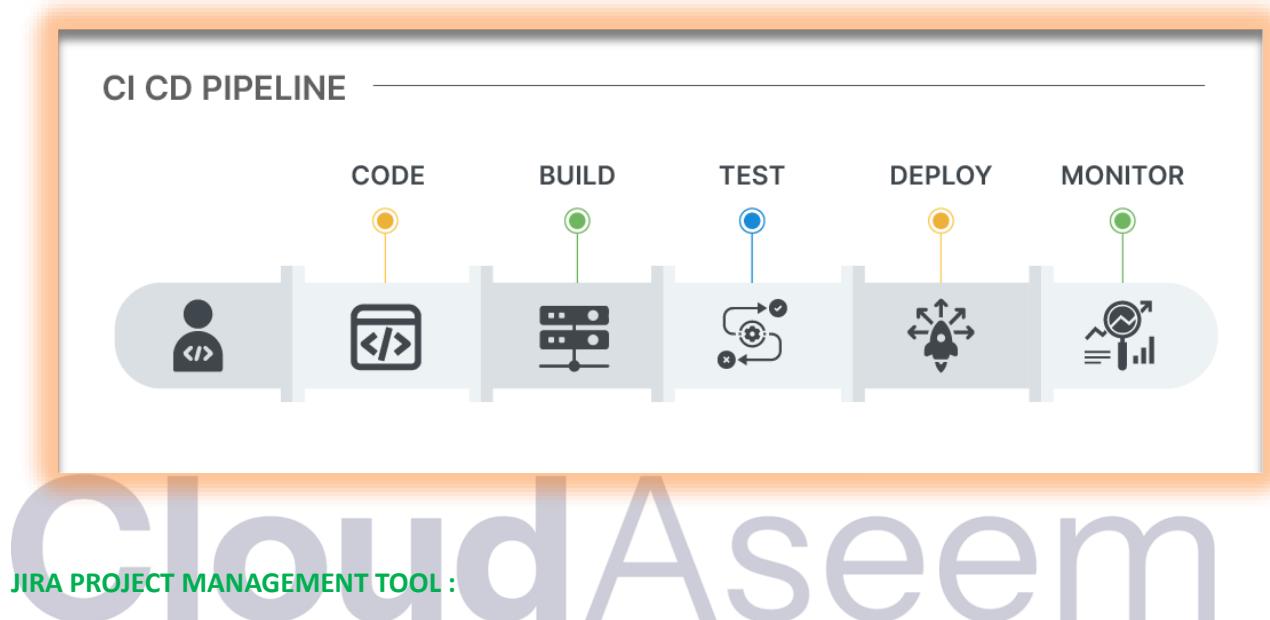
CI/CD pipeline diagram

 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 :



CloudAseem

JIRA PROJECT MANAGEMENT TOOL :

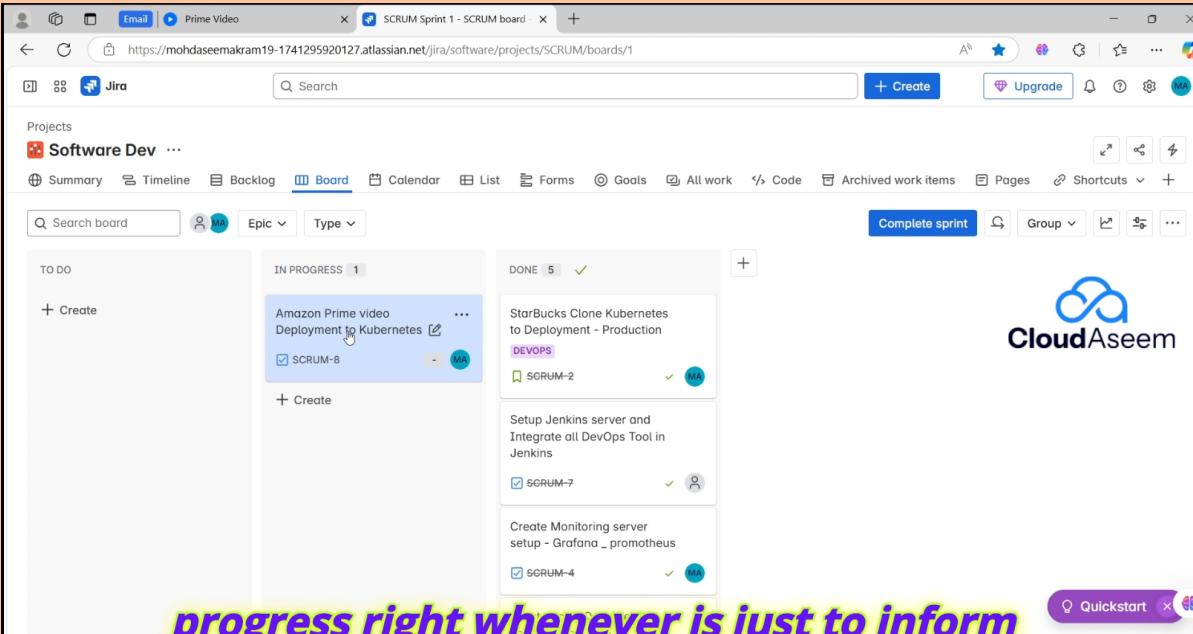
💡 What is Jira?

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



The screenshot shows a Jira Scrum board titled "SCRUM Sprint 1 - SCRUM board". The board has three columns: TO DO, IN PROGRESS, and DONE. The TO DO column has one task: "Amazon Prime video Deployment to Kubernetes". The IN PROGRESS column has two tasks: "StarBucks Clone Kubernetes to Deployment - Production" and "Setup Jenkins server and Integrate all DevOps Tool in Jenkins". The DONE column has two tasks: "Create Monitoring server setup - Grafana _ prometheus" and "SCRUM-4". The right side of the screen shows a sidebar with the CloudAseem logo and a banner that says "progress right whenever is just to inform".

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...

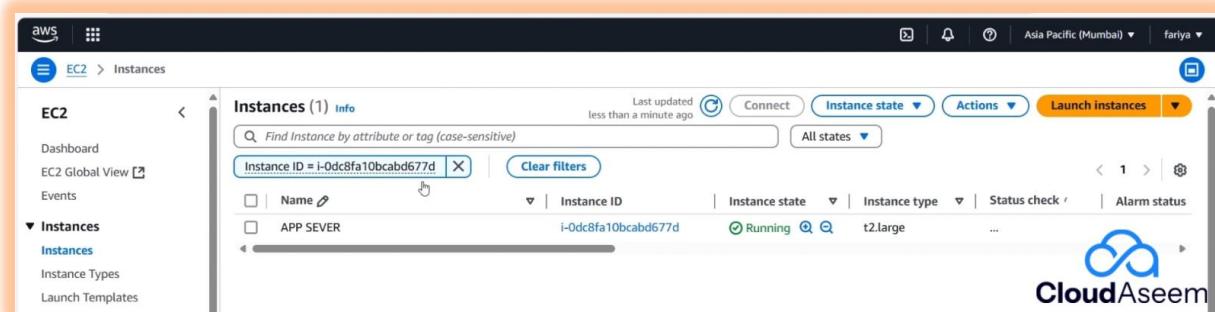
We'll begin with:

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).



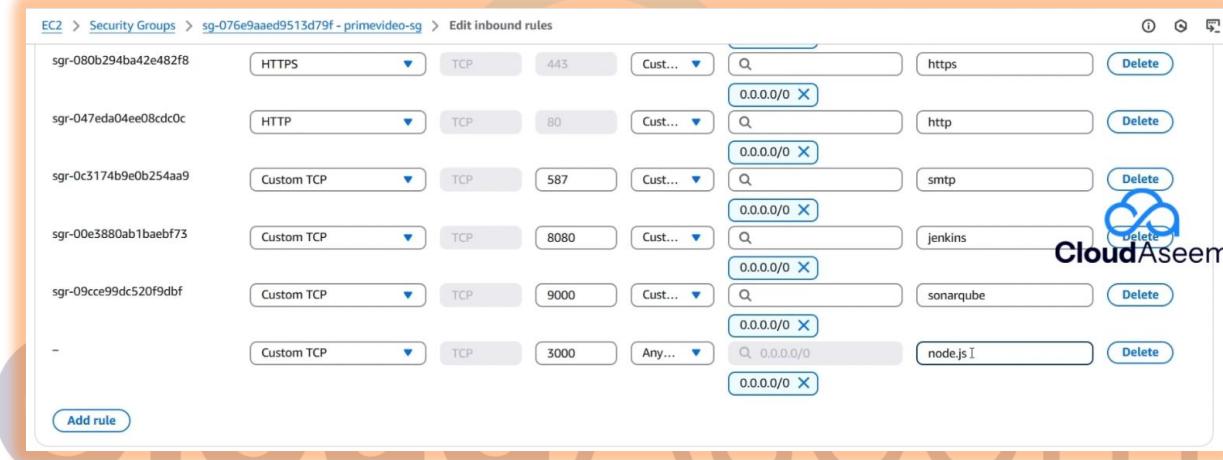
The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with 'EC2' selected under 'Instances'. The main area displays a table titled 'Instances (1)'. The table has columns for 'Name', 'Instance ID', 'Instance state', 'Instance type', and 'Status check'. One row is shown, labeled 'APP SERVER' with Instance ID 'i-0dc8fa10bcabd677d', status 'Running', instance type 't2.large', and a green checkmark in the status check column. At the top right of the main area, there are buttons for 'Actions', 'Launch instances', and a dropdown menu. Below the table, there are filters for 'Find Instance by attribute or tag (case-sensitive)' and 'Clear filters'. The bottom right corner of the screenshot features the 'CloudAseem' logo.

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.



The screenshot shows the AWS EC2 Security Groups inbound rules configuration page. The URL is [EC2 > Security Groups > sg-076e9aaed9513d79f - primevideo-sg > Edit inbound rules](#). The page displays several existing rules and a form for adding a new one.

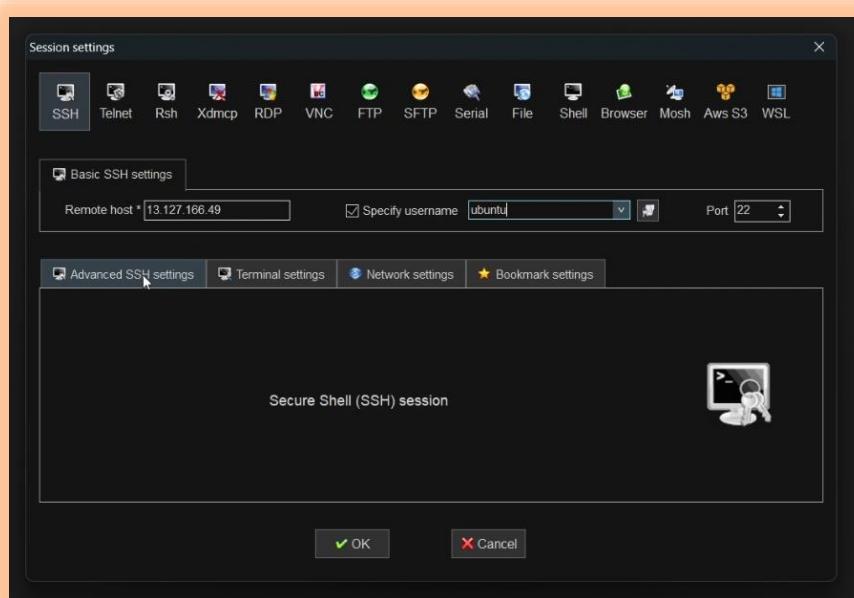
Rule ID	Protocol	Port Range	Action	Source	Custom Target
sgr-080b294ba42e482f8	HTTPS	443	Allow	0.0.0.0/0	
sgr-047eda04ee08cdc0c	HTTP	80	Allow	0.0.0.0/0	
sgr-0c3174b9e0b254aa9	Custom TCP	587	Allow	0.0.0.0/0	
sgr-00e3880ab1baebf73	Custom TCP	8080	Allow	jenkins	
sgr-09cce99dc520f9dbf	Custom TCP	9000	Allow	sonarqube	
-	Custom TCP	3000	Allow	node.js	

Add rule

New rule details:

- Protocol:** TCP
- Port range:** Any... (3000)
- Source:** 0.0.0.0/0
- Custom target:** node.js

- **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
```

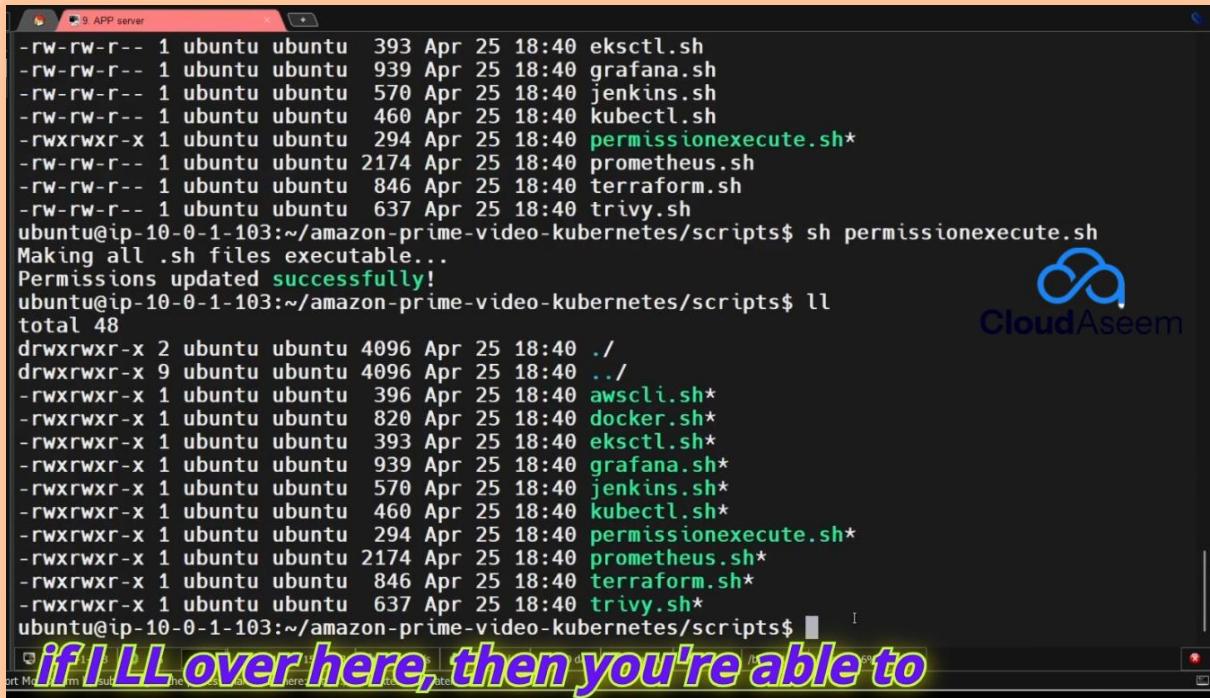
Navigate to the project directory:

```
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



```

ubuntu@ip-10-0-1-103:~/amazon-prime-video-kubernetes/scripts$ sh permissionexecute.sh
Making all .sh files executable...
Permissions updated successfully!
ubuntu@ip-10-0-1-103:~/amazon-prime-video-kubernetes/scripts$ ll
total 48
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr 25 18:40 .
drwxrwxr-x 9 ubuntu ubuntu 4096 Apr 25 18:40 ../
-rwxrwxr-x 1 ubuntu ubuntu 396 Apr 25 18:40 awscli.sh*
-rwxrwxr-x 1 ubuntu ubuntu 820 Apr 25 18:40 docker.sh*
-rwxrwxr-x 1 ubuntu ubuntu 393 Apr 25 18:40 eksctl.sh*
-rwxrwxr-x 1 ubuntu ubuntu 939 Apr 25 18:40 grafana.sh*
-rwxrwxr-x 1 ubuntu ubuntu 570 Apr 25 18:40 jenkins.sh*
-rwxrwxr-x 1 ubuntu ubuntu 460 Apr 25 18:40 kubectl.sh*
-rwxrwxr-x 1 ubuntu ubuntu 294 Apr 25 18:40 permissionexecute.sh*
-rw-rw-r-- 1 ubuntu ubuntu 2174 Apr 25 18:40 prometheus.sh
-rw-rw-r-- 1 ubuntu ubuntu 846 Apr 25 18:40 terraform.sh
-rw-rw-r-- 1 ubuntu ubuntu 637 Apr 25 18:40 trivy.sh
ubuntu@ip-10-0-1-103:~/amazon-prime-video-kubernetes/scripts$ ll

```

if I'll over here, then you're able to

Jenkins Installed



```

No VM guests are running outdated hypervisor (qemu) binaries on this host.
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/system
d-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-10-0-1-103:~/amazon-prime-video-kubernetes/scripts$ jenkins --version
2.492.3

```



```

● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-04-25 18:45:06 UTC; 1min 14s ago
     Main PID: 489639 (java)
        Tasks: 49 (limit: 18927)
       Memory: 1.0G (peak: 1.1G)
          CPU: 24.565s
        CGroup: /system.slice/jenkins.service
                └─489639 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war

Apr 25 18:45:03 ip-10-0-1-103 jenkins[489639]: 9183cafee27048cead1d22c77acd3b6
Apr 25 18:45:03 ip-10-0-1-103 jenkins[489639]: This may also be found at: /var/lib/jenkins/>
Apr 25 18:45:03 ip-10-0-1-103 jenkins[489639]: ****>
Apr 25 18:45:03 ip-10-0-1-103 jenkins[489639]: ****>
Apr 25 18:45:03 ip-10-0-1-103 jenkins[489639]: ****>
Apr 25 18:45:06 ip-10-0-1-103 jenkins[489639]: 2025-04-25 18:45:06.790+0000 [id=33] >
Apr 25 18:45:06 ip-10-0-1-103 jenkins[489639]: 2025-04-25 18:45:06.809+0000 [id=24] >
Apr 25 18:45:06 ip-10-0-1-103 systemd[1]: Started jenkins.service - Jenkins Continuous Inte>
Apr 25 18:45:06 ip-10-0-1-103 jenkins[489639]: 2025-04-25 18:45:06.924+0000 [id=53] >
Apr 25 18:45:06 ip-10-0-1-103 jenkins[489639]: 2025-04-25 18:45:06.926+0000 [id=53] >
lines 1-20

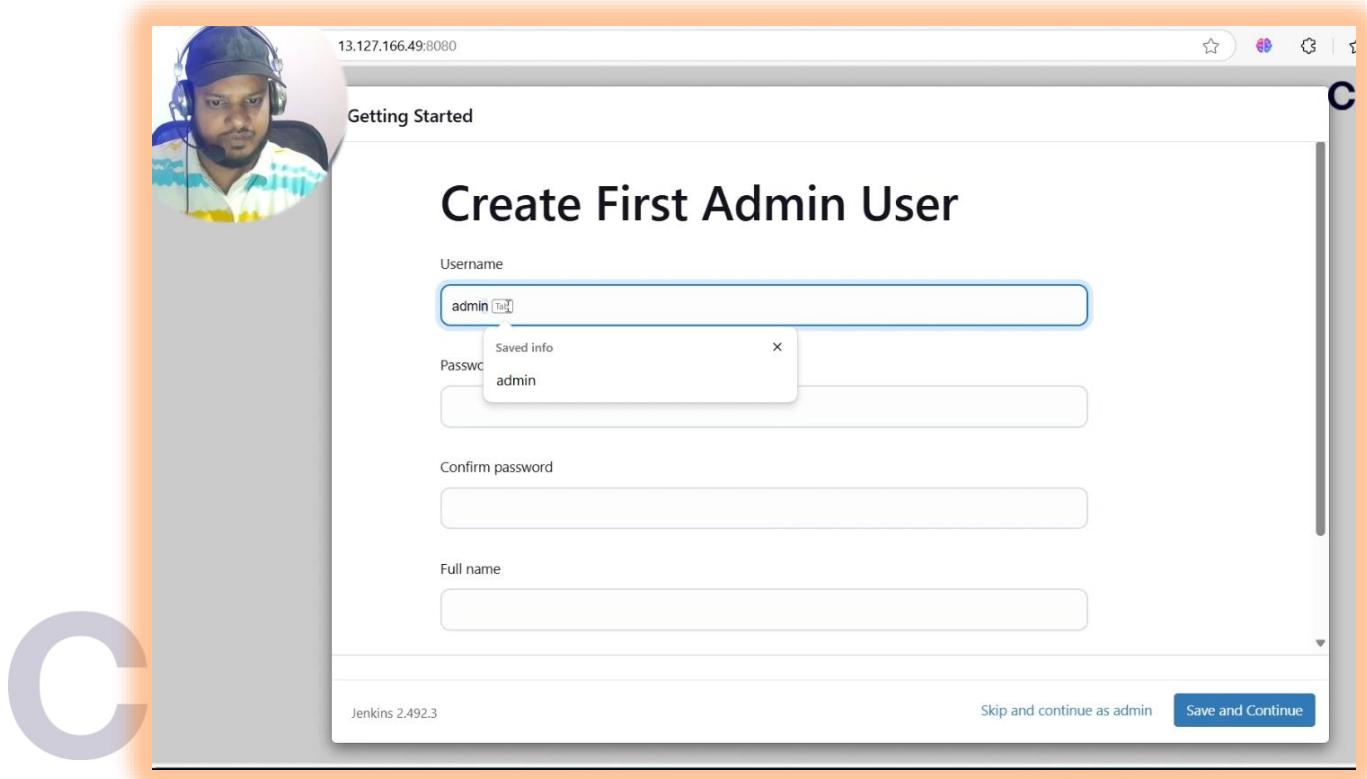
```

- Access Jenkins in your browser:

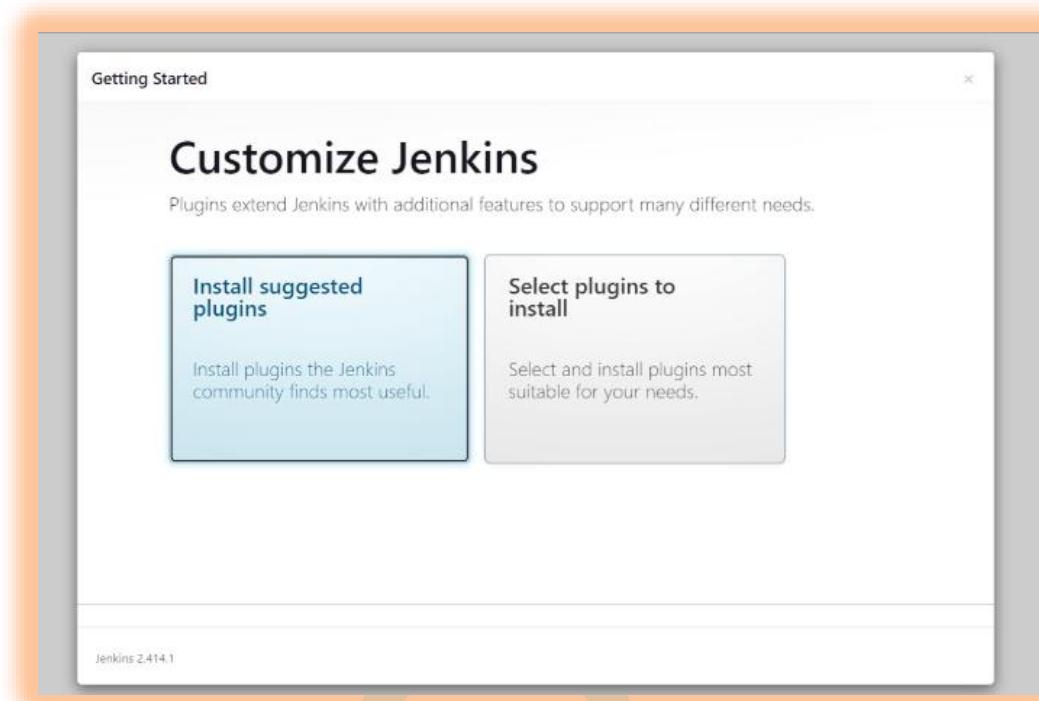
```
http://<PUBLIC_IP>:8080
```

- 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

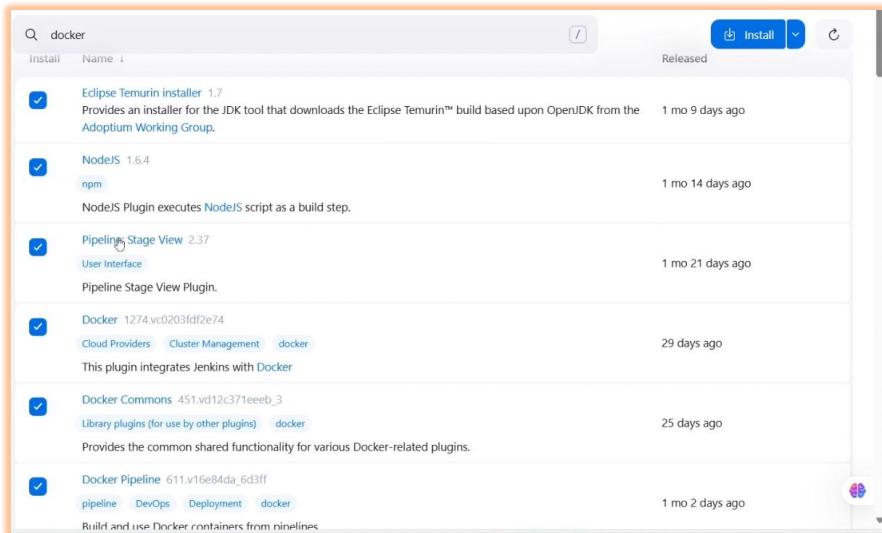
- Create a user click on save and continue.
- Jenkins Getting Started Screen.
- Follow the setup wizard and install recommended plugins.

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

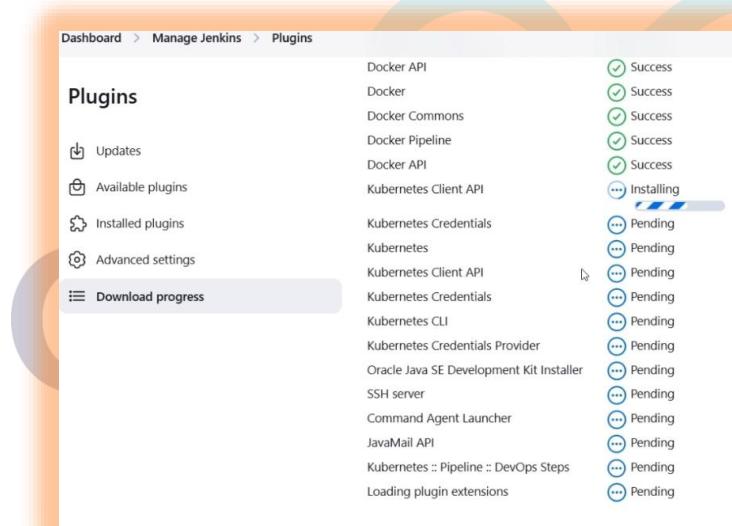
Goto Manage Jenkins → Plugins → Available Plugins →

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

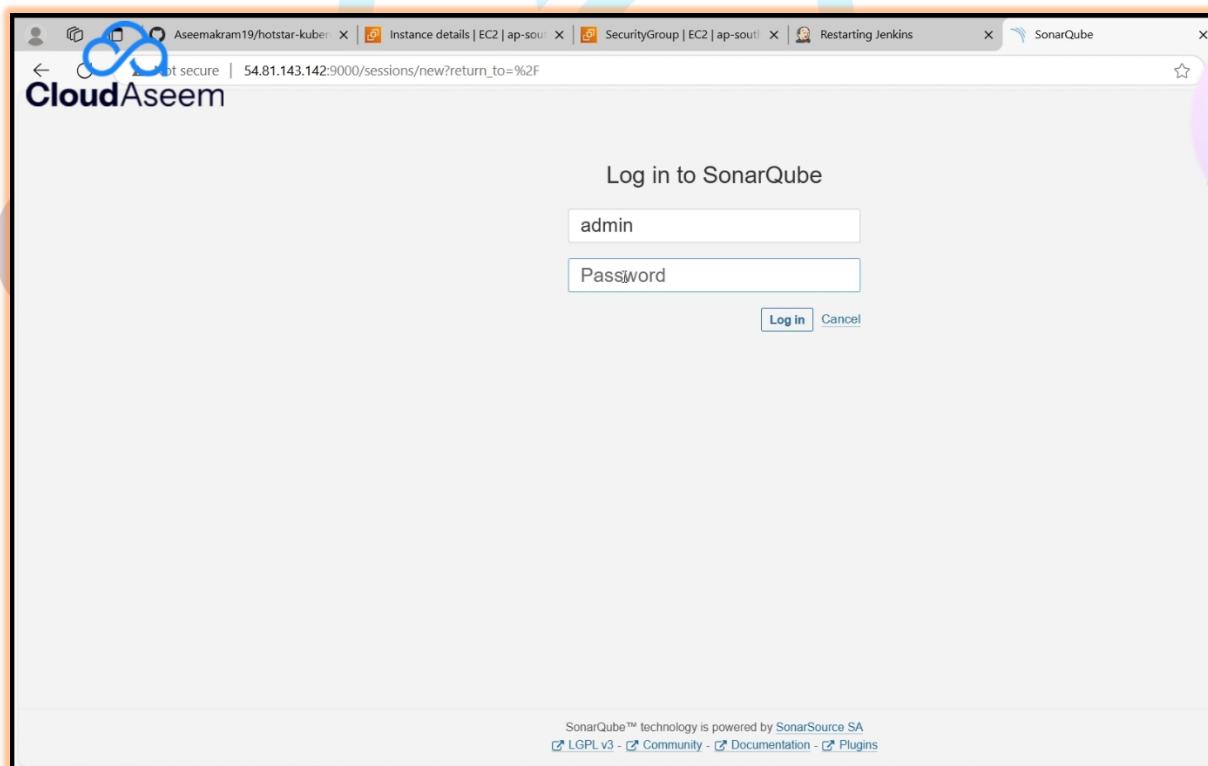
```
# Verify installation
docker -version
# Run SonarQube container in detached mode with port mapping
```

```
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: sha256:3abad2e9d705543f5a24831db2fcdf9f24aaec864f372a97b2b0dc7a2f46
Status: Downloaded newer image for sonarqube:lts-community
4fa2d45a5edadcd98d594428bc9d0e4cbaf6f3c361f16618f7145818910dca236
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 your password

This account should not use the default password.

Enter a new password

All fields marked with * are required

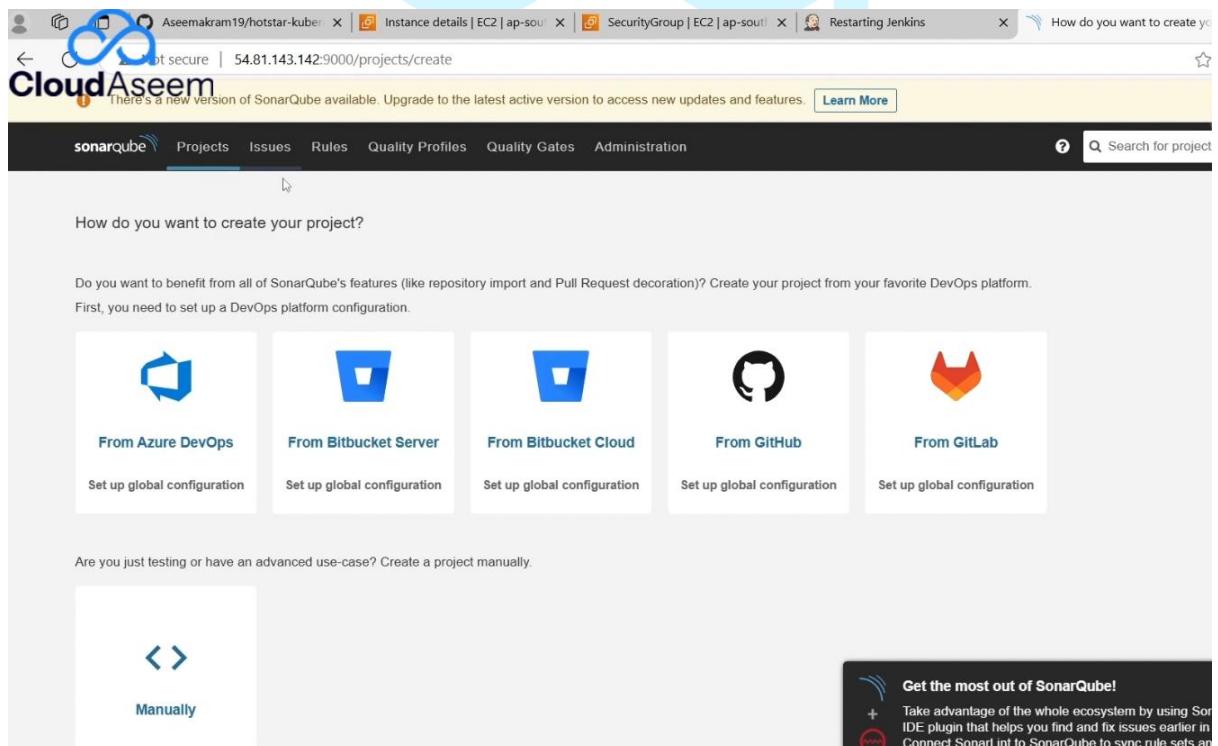
Old Password *

New Password *

Confirm Password *

Update

Update New password, This is Sonar Dashboard.



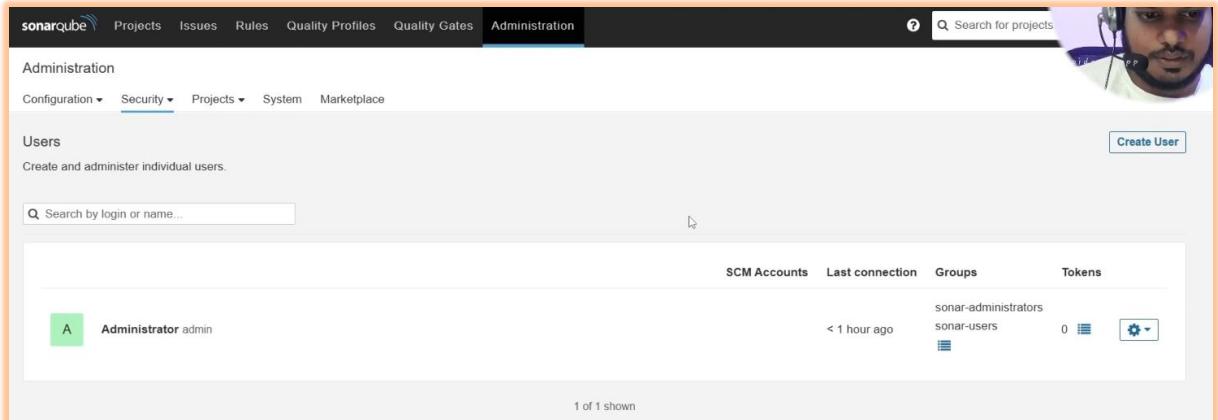
A screenshot of a web browser showing the SonarQube interface. The address bar indicates the URL is 54.81.143.142:9000/projects/create. The page title is "CloudAseem". A banner at the top says "There's a new version of SonarQube available. Upgrade to the latest active version to access new updates and features." with a "Learn More" button. The main navigation menu includes "sonarqube", "Projects", "Issues", "Rules", "Quality Profiles", "Quality Gates", "Administration", and a search bar. Below the menu, a section titled "How do you want to create your project?" lists several options:

- From Azure DevOps**: Set up global configuration
- From Bitbucket Server**: Set up global configuration
- From Bitbucket Cloud**: Set up global configuration
- From GitHub**: Set up global configuration
- From GitLab**: Set up global configuration

At the bottom left, there is a link "Are you just testing or have an advanced use-case? Create a project manually." with a "Manually" button. On the right side, there is a sidebar with the heading "Get the most out of SonarQube!" and a list of items including "Take advantage of the whole ecosystem by using SonarQube IDE plugin that helps you find and fix issues earlier in your development cycle.", "Connect SonarLint to SonarQube to sync rule sets and analysis results.", and "Integrate SonarQube with your CI/CD pipeline to automatically detect code quality issues and failures during builds."

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

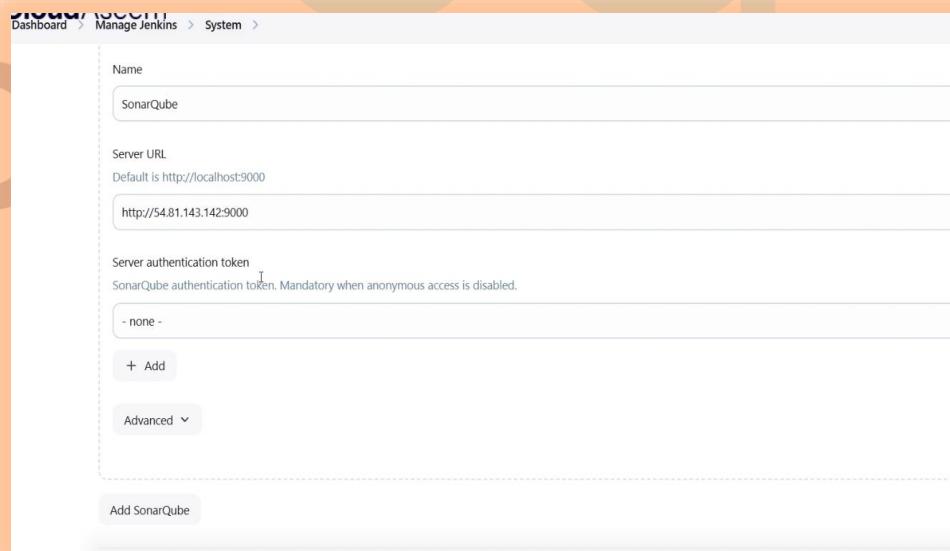
Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token



The screenshot shows the SonarQube Administration interface. In the top navigation bar, 'Administration' is selected. Under 'Security', 'Users' is selected. On the right, there is a profile picture of a person and a 'Create User' button. The main area shows a table with columns: 'SCM Accounts', 'Last connection', 'Groups', and 'Tokens'. The 'Tokens' column for the 'Administrator admin' row shows '0' tokens.

Create a token with a name and generate

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



The screenshot shows the Jenkins 'Manage Jenkins' → 'System' configuration page. The 'Add SonarQube' form is displayed. It has fields for 'Name' (set to 'SonarQube'), 'Server URL' (set to 'Default is http://localhost:9000' and 'http://54.81.143.142:9000'), and 'Server authentication token' (set to '- none -'). There is a '+ Add' button and an 'Advanced' dropdown.

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

Kind

Secret file

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Global (Jenkins, nodes, items, all child items, etc)

System (Jenkins and nodes only)

Choose File No file chosen

ID ?

Description ?

You will see this page once you click on create

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 Sonar-token	sonar	Secret text	sonar 

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

≡ SonarQube Scanner

Name

sonar-scanner 

Required

Install automatically 

≡ Install from Maven Central

Version

SonarQube Scanner 7.0.2.4839 

Add Installer 

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

Create Webhook

All fields marked with * are required

Name *
jenkins

URL *
`http://13.127.166.49:8080/sonarqube-webhook`

Server endpoint that will receive the webhook payload, for example:
`"http://my_server/foo"`. If HTTP Basic authentication is used, HTTPS is recommended to avoid man in the middle attacks. Example:
`"https://myLogin:myPassword@my_server/foo"`

Secret

If provided, secret will be used as the key to generate the HMAC hex (lowercase) digest value in the 'X-Sonar-Webhook-HMAC-SHA256' header.

Create **Cancel**

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

Create Webhook

All fields marked with * are required

Name *
jenkins

URL *
`http://54.81.143.142:8080/sonarqube-webhook`

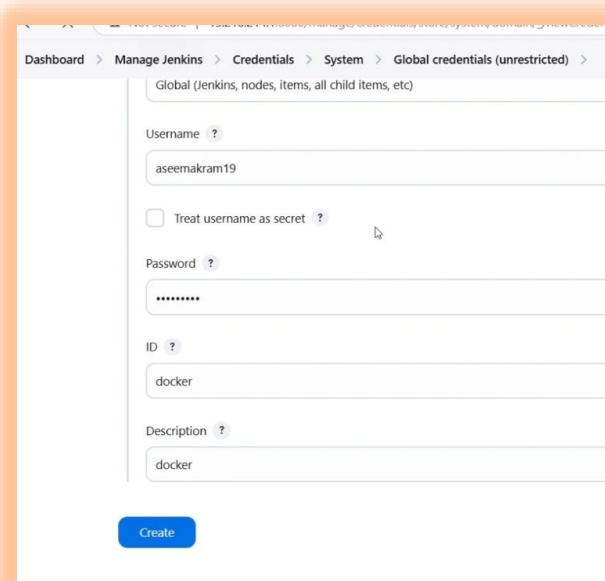
Server endpoint that will receive the webhook payload, for example:
`"http://my_server/foo"`. If HTTP Basic authentication is used, HTTPS is recommended to avoid man in the middle attacks. Example:
`"https://myLogin:myPassword@my_server/foo"`

Secret

If provided, secret will be used as the key to generate the HMAC hex (lowercase) digest value in the 'X-Sonar-Webhook-HMAC-SHA256' header.

Create **Cancel**

Docker Hub credentials in Jenkins secret

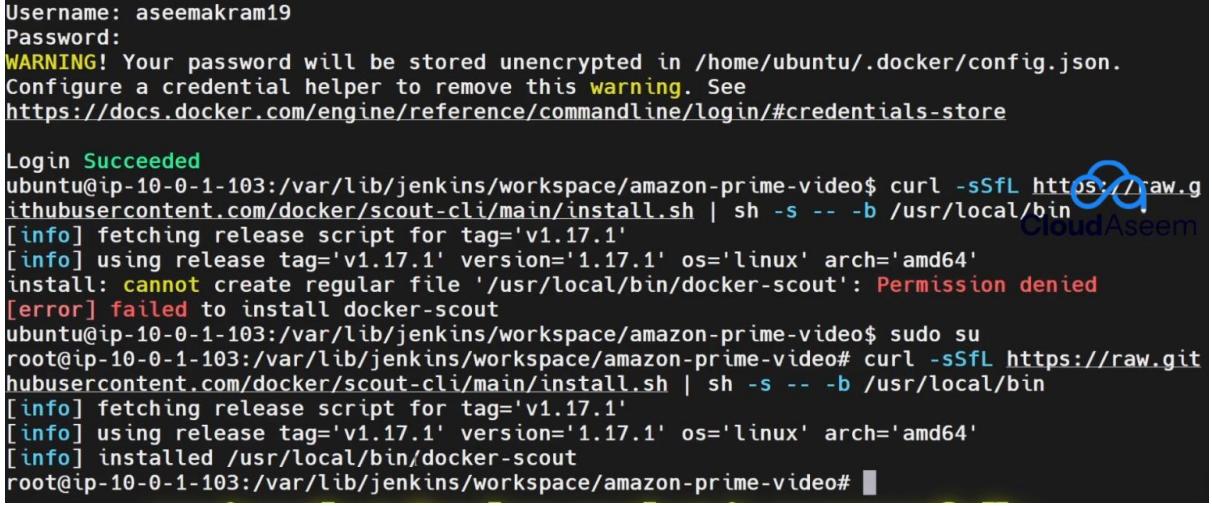


Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Global (Jenkins, nodes, items, all child items, etc)
Username ? aseemakram19
<input type="checkbox"/> Treat username as secret ?
Password ? *****
ID ? docker
Description ? docker
Create

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.githubusercontent.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.githubusercontent.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#

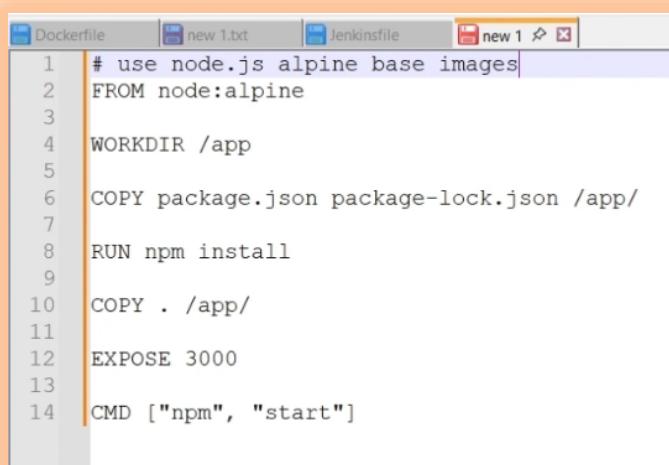
```

Dockerfile

Write Dockerfile from scratch

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.



```

1 # use node.js alpine base images
2 FROM node:alpine
3
4 WORKDIR /app
5
6 COPY package.json package-lock.json /app/
7
8 RUN npm install
9
10 COPY . /app/
11
12 EXPOSE 3000
13
14 CMD ["npm", "start"]

```

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"]**

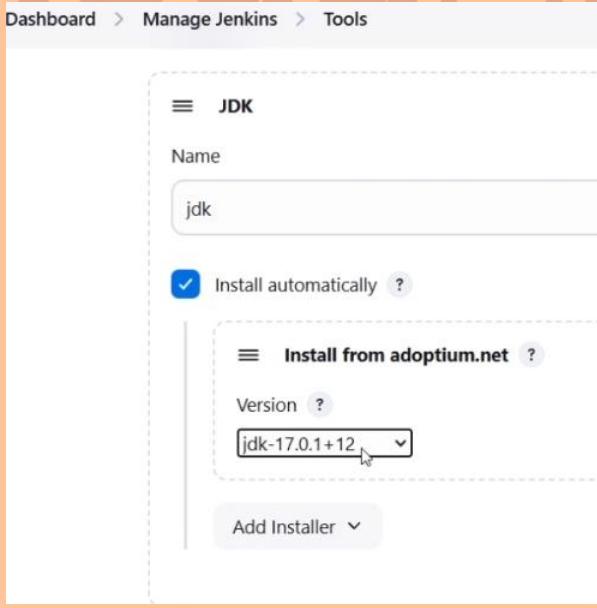
Specifies the **default command** that runs when the container starts.

In this case, it runs npm start, which should start your Node.js app (as defined in package.json).

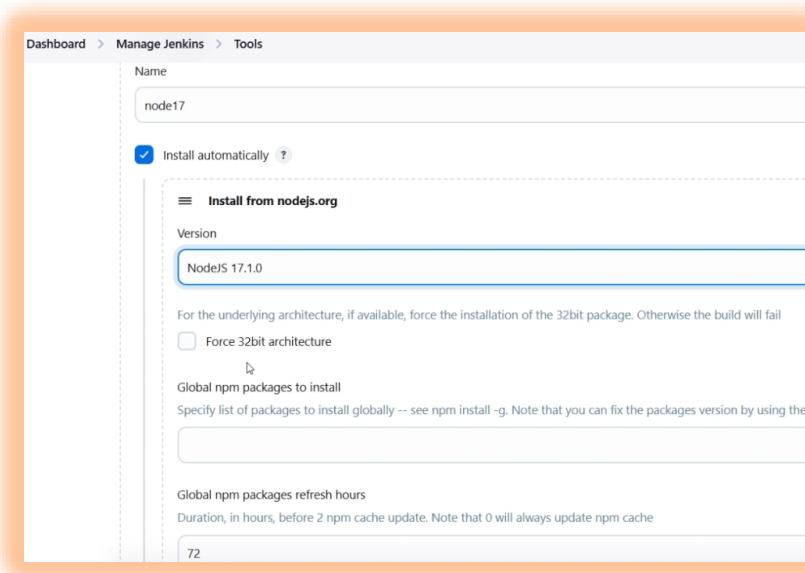
Configure Tools in jenkins

1. JDK
2. Nodejs

CloudAseem



The screenshot shows the Jenkins interface for managing tools. The URL is `Dashboard > Manage Jenkins > Tools`. Under the 'JDK' section, a new entry named 'jdk' is being configured. The 'Install automatically' checkbox is checked. Under the 'Install from adoptium.net' section, the 'Version' dropdown is set to 'jdk-17.0.1+12'. There is also a 'Add Installer' button.



Dashboard > Manage Jenkins > Tools

Name: node17

Install automatically ?

Install from nodejs.org

Version: NodeJS 17.1.0

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
 Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the

Global npm packages refresh hours
 Duration, in hours, before 2 npm cache update. Note that 0 will always update npm cache
 72

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

Update credentials

Scope [?](#)

Global (Jenkins, nodes, items, all child items, etc)

Username [?](#)

mohdaseemakram19@gmail.com

Treat username as secret [?](#)

Password [?](#)

 Concealed

[Change Password](#)

ID [?](#)

smtp-gmail

Description [?](#)

smtp-gmail

[Save](#)

7.

← App passwords

App passwords help you sign into your Google Account on older apps and services that don't support modern security standards.

App passwords are less secure than using up-to-date apps and services that use modern security standards. Before you create an app password, you should check to see if your app needs this in order to sign in.

[Learn more](#)

You don't have any app passwords.

To create a new app specific password, type a name for it below...

App name
starbucks

[Create](#)

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**.

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:**

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

E-mail Notification

SMTP server

smtp.gmail.com

Default user e-mail suffix 

@gmail.com

 Advanced 

Use SMTP Authentication 

User Name

mohdaseemakram19@gmail.com

Password

 Concealed

Use SSL 

Use TLS

SMTP Port 

465

Reply-To Address

mohdaseemakram19@gmail.com

Charset

UTF-8

Test configuration by sending test e-mail



2. Extended Email Notification

SMTP Server: smtp.gmail.com

SMTP Port: 465 (for tls)

Use SSL: Checked

Credentials: Select from the

Extended E-mail Notification

SMTP server

smtp.gmail.com

SMTP Port

587

Advanced ▾  Edited

Default user e-mail suffix 

@gmail.com

Advanced ▾  Edited

Default Content Type 

Plain Text (text/plain)

List ID 

Add 'Precedence: bulk' E-mail Header 

Default Recipients 

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

```

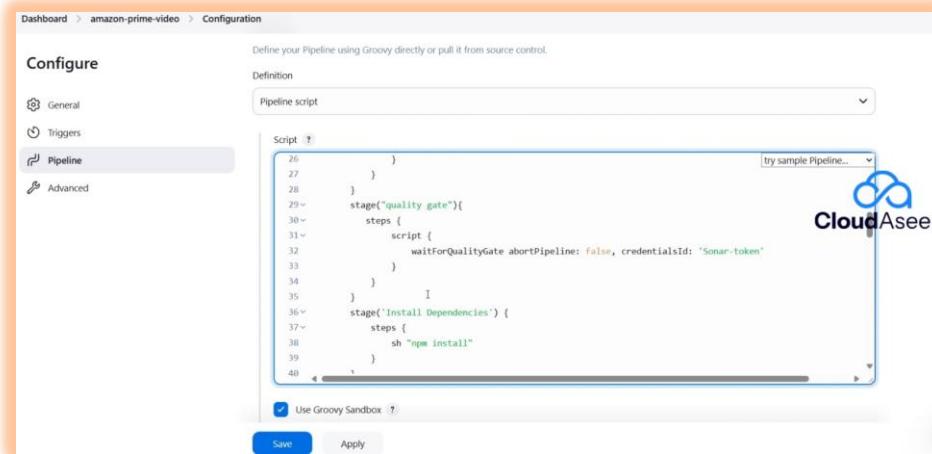
```

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

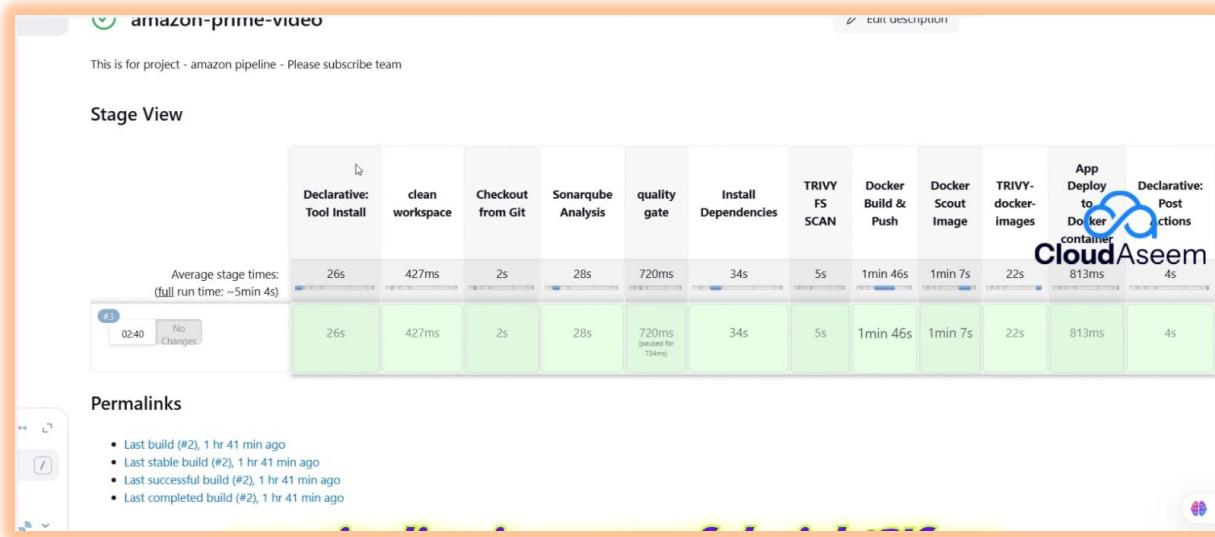
emailext (
subject: "Pipeline ${buildStatus}: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
body: """
<p>This is a Jenkins amazon-prime-video CICD pipeline status.</p>
<p>Project: ${env.JOB_NAME}</p>
<p>Build Number: ${env.BUILD_NUMBER}</p>
<p>Build Status: ${buildStatus}</p>
<p>Started by: ${buildUser}</p>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



This is for project - amazon pipeline - Please subscribe team

Stage View

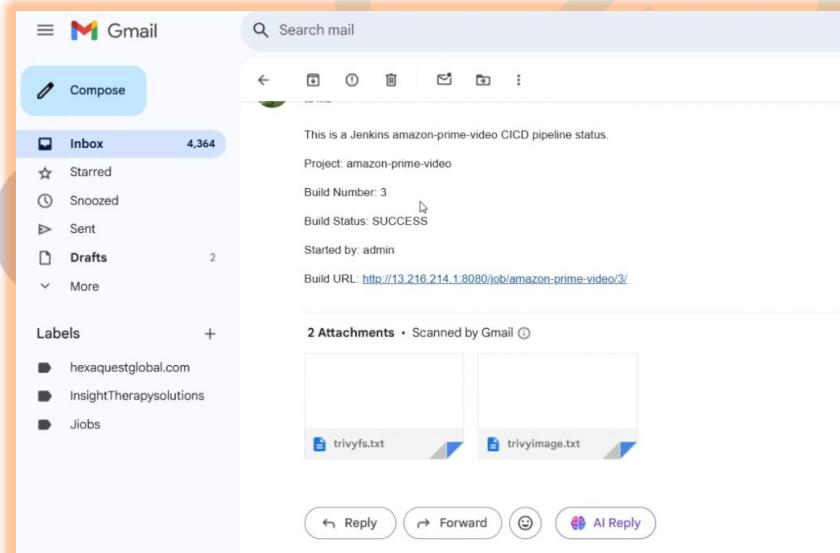
Declarative: Tool Install	clean workspace	Checkout from Git	Sonarqube Analysis	quality gate	Install Dependencies	TRIVY FS SCAN	Docker Build & Push	Docker Scout Image	TRIVY-docker-images	App Deploy to Docker container	Declarative: Post Actions
Average stage times: (full run time: ~5min 4s)	26s	427ms	2s	28s	720ms	34s	5s	1min 46s	1min 7s	22s	813ms
02:40 No Changes	26s	427ms	2s	28s	720ms (paused for 734ms)	34s	5s	1min 46s	1min 7s	22s	813ms

Permalinks

- Last build (#2), 1 hr 41 min ago
- Last stable build (#2), 1 hr 41 min ago
- Last successful build (#2), 1 hr 41 min ago
- Last completed build (#2), 1 hr 41 min ago

Result :

- Emails received



This is a Jenkins amazon-prime-video CI/CD pipeline status.

Project: amazon-prime-video

Build Number: 3

Build Status: SUCCESS

Started by: admin

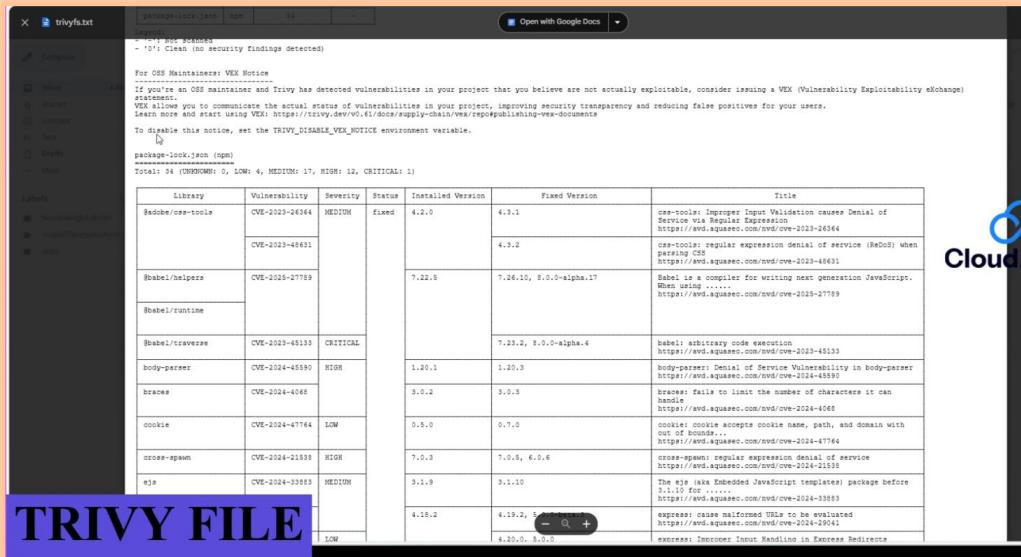
Build URL: <http://13.216.214.1:8080/job/amazon-prime-video/3/>

2 Attachments • Scanned by Gmail

trivyfs.txt trivyimage.txt

Reply Forward AI Reply

Report attached in emails

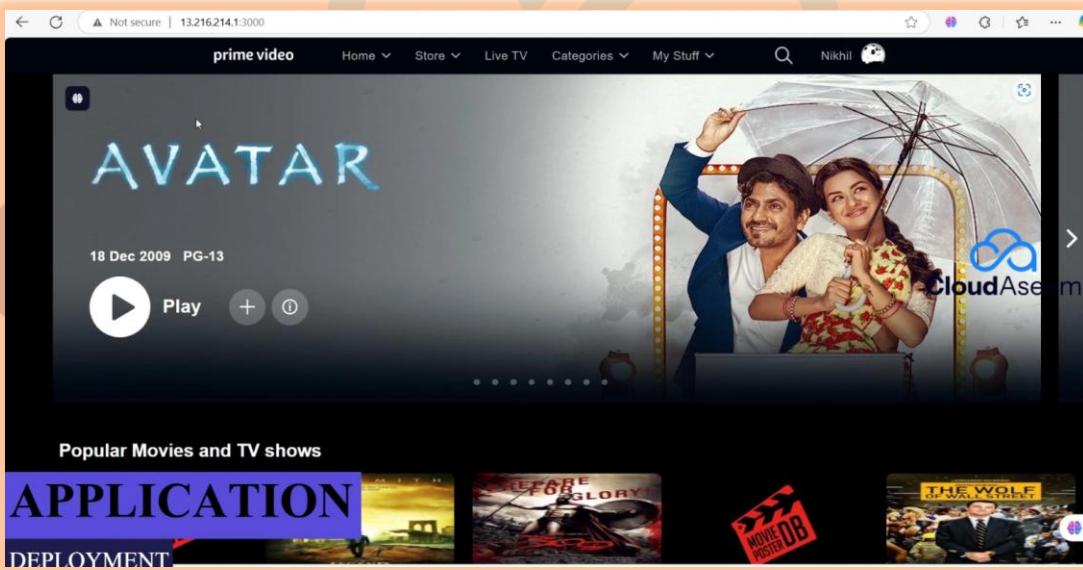


TRIVY FILE

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
@adobe/cse-tools	CVE-2023-24644	MEDIUM	fixed	4.2.0	4.3.1	cse-tools: Improper Input Validation causes Denial of Service
	CVE-2023-48631				4.3.2	cse-tools: regular expression denial of service (ReDoS) when parsing CSS
@babel/helpers	CVE-2023-27789			7.22.0	7.26.10, 8.0.0-alpha.17	Babel is a compiler for writing next generation JavaScript. When using
					7.23.2, 8.0.0-alpha.4	babel: arbitrary code execution
body-parser	CVE-2024-45133	CRITICAL		1.20.1	1.20.3	body-parser: Denial of Service Vulnerability in body-parser
	CVE-2024-45590	HIGH		3.0.2	3.0.3	braces: fails to limit the number of characters it can handle
braces	CVE-2024-40488			0.5.0	0.7.0	cookie: cookie accepts cookie name, path, and domain with
				7.0.0	7.0.5, 8.0.6	cross-spawn: regular expression denial of service
cookie	CVE-2024-47764	LOW		3.1.9	3.1.10	The ejs (aka Embedded JavaScript template) package before 3.1.17 fix for
				4.19.2	4.19.2, 5	ejson: cause malformed URL to be evaluated
cross-spawn	CVE-2024-21538	HIGH				express: cause malformed URL to be evaluated
					4.20.0, 3.0.0	express: Improper Input Handling in Express Redirects
ejs	CVE-2024-33855	MEDIUM				
		LOW				

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

Our Application is live with this output



2. Verify Docker Images in Dockerhub



A screenshot of the Docker Hub interface. On the left is a sidebar for the user "aseemakram19" with sections for Repositories, Settings, Billing, Usage, Pulls, and Storage. The main area shows a table of repositories under the heading "Repositories". One row for "aseemakram19/amazon-prime-video" is highlighted with a red arrow pointing to it. The table includes columns for Name, Last Pushed, Contains, Visibility, and Status. At the bottom, there are banners for "DOCKER HUB" and "APPLICATION IMAGE".

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.

A screenshot of the SonarQube web interface. The top navigation bar shows "sonarqube" and "Projects". The main content area displays a summary for the project "amazon-prime-video". It shows a green "Passed" status with a "Last analysis: 14 minutes ago" timestamp. Below this, there's a dashboard with metrics: Bugs (1), Vulnerabilities (6), Hotspots Reviewed (E), Code Smells (A), Coverage (0.0%), and Duplications (5.2%). A note indicates "13k" lines scanned across "CSS, Java" files. On the left, there are filters for Quality Gate (Passed, Failed), Reliability (A rating, B rating, C rating, D rating, E rating), and a search bar.



CloudAseem

CloudAseem
Mohammed Aseem Akram

amazon-prime-video main

Last analysis of this Branch had 1 warning April 26, 2025 at 2:41 AM Version not provided

Overview Issues Security Hotspots Measures Code Activity Project Settings Project Information

1 Vulnerabilities

3 Security Hotspots 0.0% Reviewed Security Review E

3h 57min Debt 87 Code Smells Maintainability A

0.0% 5.2% 55

vulnerability security Duplications on 13k Lines Duplicated Blocks

Docker Scout review

```
+ docker-scout quickview aseemakram19/amazon-prime-video:latest
...Storing image for indexing
✓ Image stored for indexing
...Indexing
✓ Indexed 1426 packages

i Base image was auto-detected. To get more accurate results, build images with max-mode provenance attestations.
Review https://docs.docker.com/build/attestations/slsa-provenance/ for more information.
```

Target	aseemakram19/amazon-prime-video:latest	1C	12H	17M	4L
digest	aa26c4c1c82e				
Base image	node:23-alpine	0C	0H	0M	0L
Updated base image	node:slim	0C	0H	0M	25L
					+25

What's next:

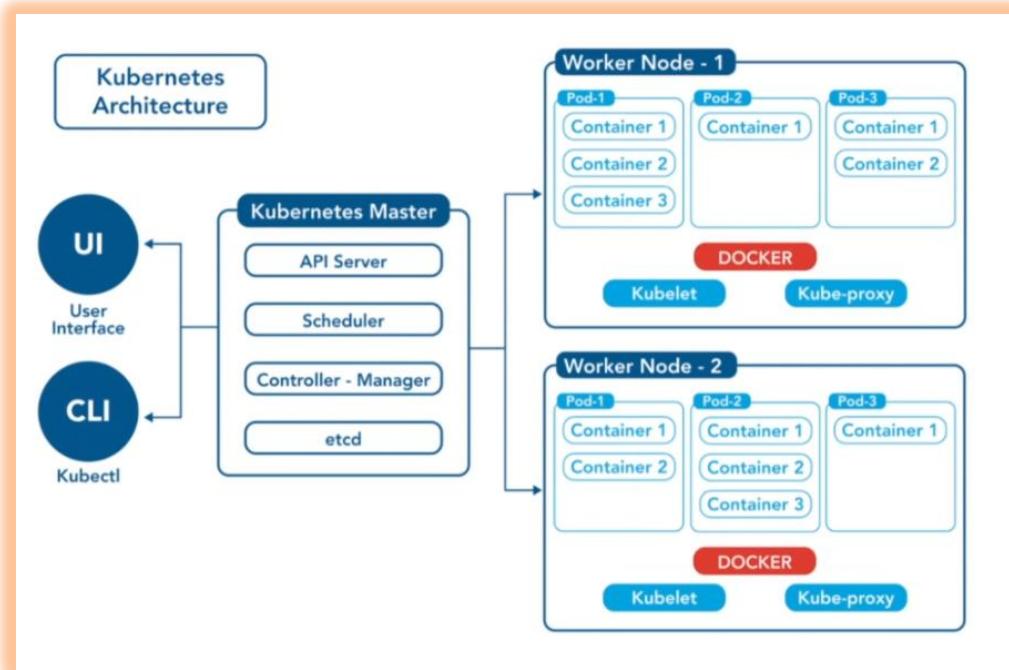
- View vulnerabilities → docker scout cves aseemakram19/amazon-prime-video:latest
- View base image update recommendations → docker scout recommendations aseemakram19/amazon-prime-video:latest
- Include policy results in your quickview by supplying an organization → docker scout quickview aseemakram19/amazon-prime-video:latest --org <organization>

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:

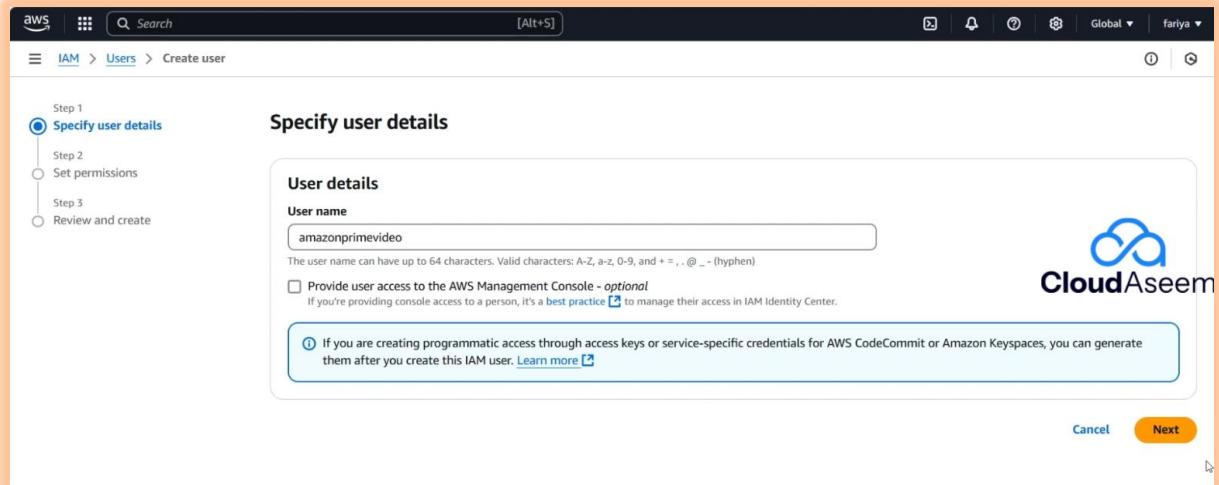
Managed Node Groups (recommended)

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



The screenshot shows the 'Specify user details' step of the AWS IAM 'Create user' wizard. The user name is set to 'amazonprimevideo'. The 'Provide user access to the AWS Management Console - optional' checkbox is unchecked. A note at the bottom states: 'If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user.' Buttons for 'Cancel' and 'Next Step' are visible.

Permissions summary		
Name	Type	Used as
AdministratorAccess	AWS managed - job function	Permissions policy
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonEKSClusterPolicy	AWS managed	Permissions policy
AmazonEKSServicePolicy	AWS managed	Permissions policy
AmazonVPCFullAccess	AWS managed	Permissions policy
AWSCloudFormationFullAccess	AWS managed	Permissions policy
IAMFullAccess	AWS managed	Permissions policy

> Create access key

the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

[Create access key](#) [info](#)

Access key
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
 AKIAQXPZDEJFEJRVMXP	 ***** Show

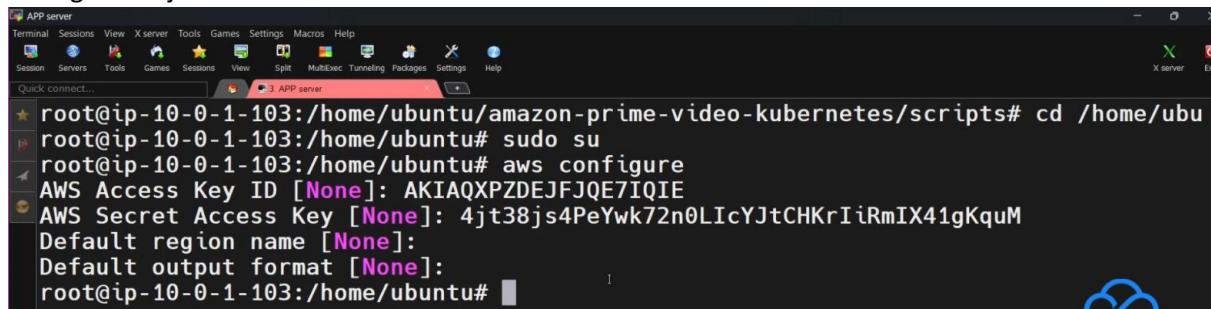
Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .csv file](#) [Done](#)

Configure on jenkins server



```

APP server
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
root@ip-10-0-1-103:/home/ubuntu/amazon-prime-video-kubernetes/scripts# cd /home/ubuntu
root@ip-10-0-1-103:/home/ubuntu# sudo su
root@ip-10-0-1-103:/home/ubuntu# aws configure
AWS Access Key ID [None]: AKIAQXPZDEJFJQE7IQIE
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:
root@ip-10-0-1-103:/home/ubuntu#

```

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.

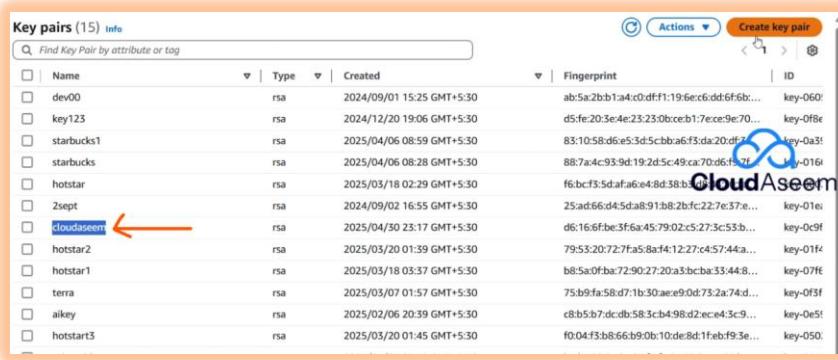
Associating the IAM OIDC provider allows Kubernetes workloads (Pods) running in the cluster to assume IAM roles securely.

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

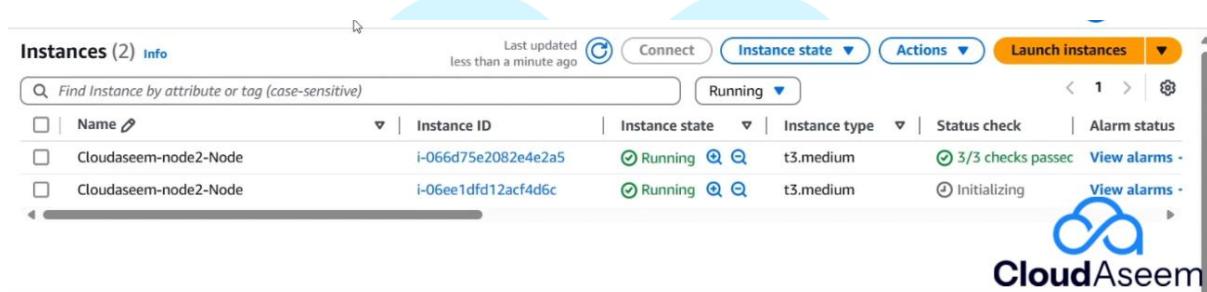


Key pairs (15) Info			
	Name	Type	Created
<input type="checkbox"/>	dev00	rsa	2024/09/01 15:25 GMT+5:30
<input type="checkbox"/>	key123	rsa	2024/12/20 19:06 GMT+5:30
<input type="checkbox"/>	starbucks1	rsa	2025/04/08 08:59 GMT+5:30
<input type="checkbox"/>	starbucks	rsa	2025/04/06 08:28 GMT+5:30
<input type="checkbox"/>	hoststar	rsa	2025/03/18 02:29 GMT+5:30
<input type="checkbox"/>	2sept	rsa	2024/09/02 16:55 GMT+5:30
<input type="checkbox"/>	cloudaseem	rsa	2025/04/30 23:17 GMT+5:30
<input type="checkbox"/>	hoststar2	rsa	2025/03/20 01:39 GMT+5:30
<input type="checkbox"/>	hoststar1	rsa	2025/03/18 03:37 GMT+5:30
<input type="checkbox"/>	terra	rsa	2025/03/07 01:57 GMT+5:30
<input type="checkbox"/>	aikey	rsa	2025/02/06 20:39 GMT+5:30
<input type="checkbox"/>	hoststart3	rsa	2025/03/20 01:45 GMT+5:30

	Fingerprint	ID
<input type="checkbox"/>	ab:5a:2bb1:a4:c0:df:f1:19:6:c6:dd:6f:6b...	key-060:
<input type="checkbox"/>	d5:fe:20:3e:4e:23:23:0:bc:b1:7:e:ce:9e:70...	key-0f8e
<input type="checkbox"/>	83:10:58:d6:e5:3d:5:cb:b6:f3:da:20:df:7...	key-0a3:
<input type="checkbox"/>	88:7:a4:c9:3:9d:19:2d:5:c4:9:ca:70:6:f:7...	key-016i
<input type="checkbox"/>	f6:b:c:f5:5:df:a6:e4:8:d:3:b:4:5:1:1:1:1...	key-01ei
<input type="checkbox"/>	25:ad:66:d4:5:da:8:9:1:b:2:b:fc:22:7:e:37:e...	key-0c9f
<input type="checkbox"/>	d6:16:6:f:fb:5:f6:6:45:7:9:0:2:c:27:3:c:5:b...	key-01fz
<input type="checkbox"/>	79:53:20:7:2:f:a5:8:a4:1:2:27:c:4:5:7:4:4:a...	key-07ff
<input type="checkbox"/>	b8:5:a:0:fb:a:7:2:9:0:2:7:20:a:3:b:c:b:3:3:4:8...	key-0f3f
<input type="checkbox"/>	75:b:9:fa:5:8:d:7:1:b:30:a:e:9:0:d:7:3:2:a:74:d...	key-0e5f
<input type="checkbox"/>	c:8:b:5:b:7:dc:db:5:8:3:c:b:4:9:8:d:2:c:e:4:5:c:9...	key-050:

```
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



Name	Instance ID	Instance state	Instance type	Status check
Cloudaseem-node2-Node	i-066d75e2082e4e2a5	Running	t3.medium	3/3 checks passed
Cloudaseem-node2-Node	i-06ee1dfd12acf4d6c	Running	t3.medium	Initializing

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

```

[2025-04-30 17:52:42 [i] checking security group configuration for all nodegroups
2025-04-30 17:52:42 [i] all nodegroups have up-to-date cloudformation templates
root@ip-10-0-1-103:/home/ubuntu# aws configure
AWS Access Key ID [*****IQIE]: ^C
root@ip-10-0-1-103:/home/ubuntu# root^C
root@ip-10-0-1-103:/home/ubuntu# sudo -su jenkins
jenkins@ip-10-0-1-103:/home/ubuntu$ aws configure
AWS Access Key ID [None]: AKIAQXPZDEJFJQE7IQIE
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:
jenkins@ip-10-0-1-103:/home/ubuntu$ whoami
jenkins
jenkins@ip-10-0-1-103:/home/ubuntu$ aws sts get-caller-identity
{
    "UserId": "AIDAQXPZDEJFMUQGNL00",
    "Account": "050451391050",
    "Arn": "arn:aws:iam::050451391050:user/amazonprimevideo"
}
jenkins@ip-10-0-1-103:/home/ubuntu$ 

```

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'  
  )  
}  
}  
  
}  
  
}
```

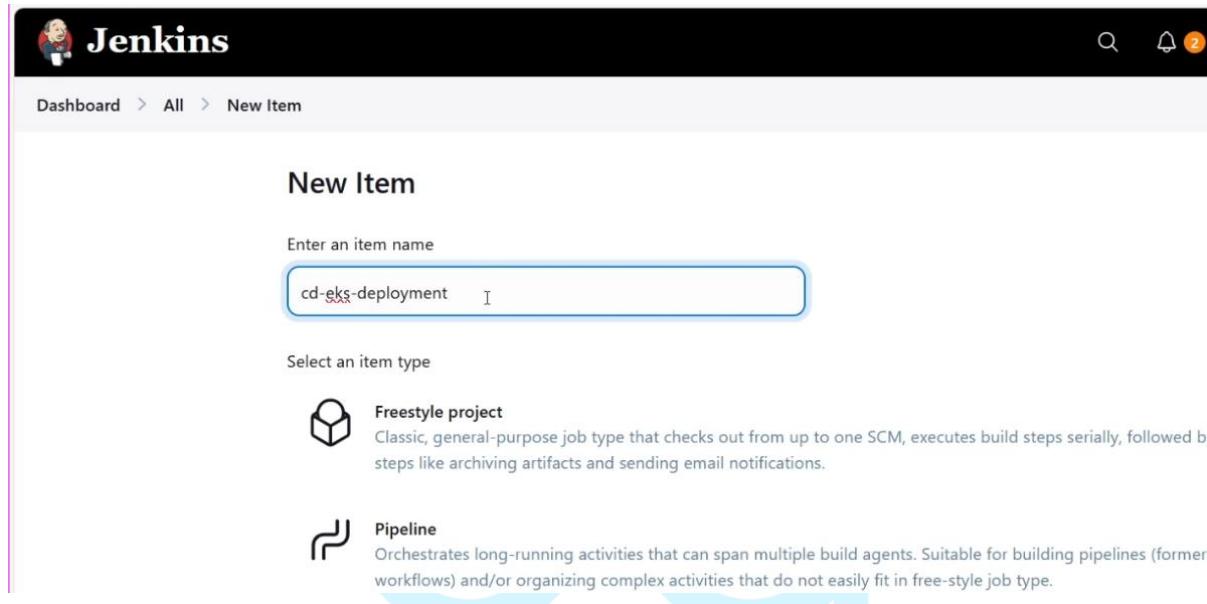
File exist ,

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
```



Dashboard > All > New Item

New Item

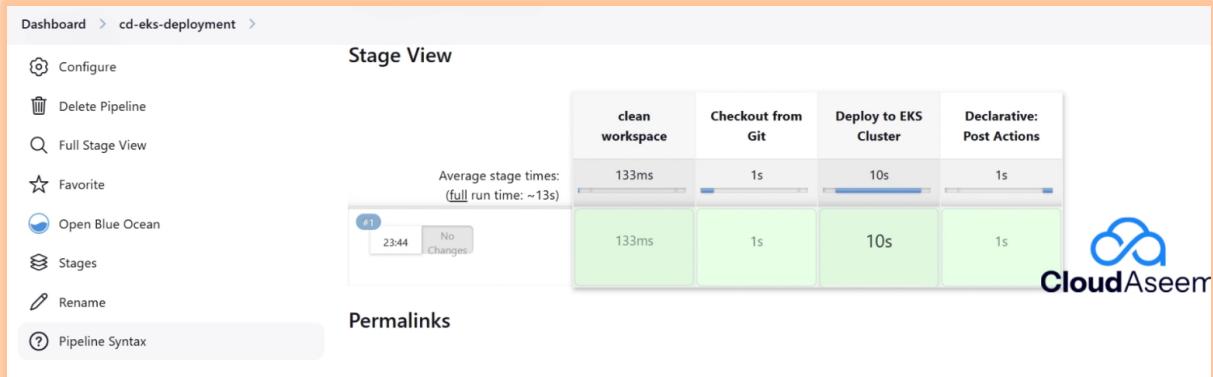
Enter an item name

Select an item type

-  **Freestyle project**
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by steps like archiving artifacts and sending email notifications.
-  **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly workflows) and/or organizing complex activities that do not easily fit in free-style job type.

CloudAseem

and execute the pipeline now



Dashboard > cd-eks-deployment >

Configure

- Delete Pipeline
- Full Stage View
- Favorite
- Open Blue Ocean
- Stages
- Rename
- Pipeline Syntax

Stage View

Average stage times: (full run time: ~13s)

clean workspace	Checkout from Git	Deploy to EKS Cluster	Declarative: Post Actions
133ms	1s	10s	1s
133ms	1s	10s	1s

#1 23:44 No Changes

Permalinks

CloudAseem

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   AGE
deployment.apps/amazon-prime-video-deployment  2/2     2           2           2m
                                             34s

NAME                                DESIRED  CURRENT   READ
/ 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                PORT(S)        TYPE        CLUSTER-IP   EXTERNAL-IP
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
                                             34s

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:

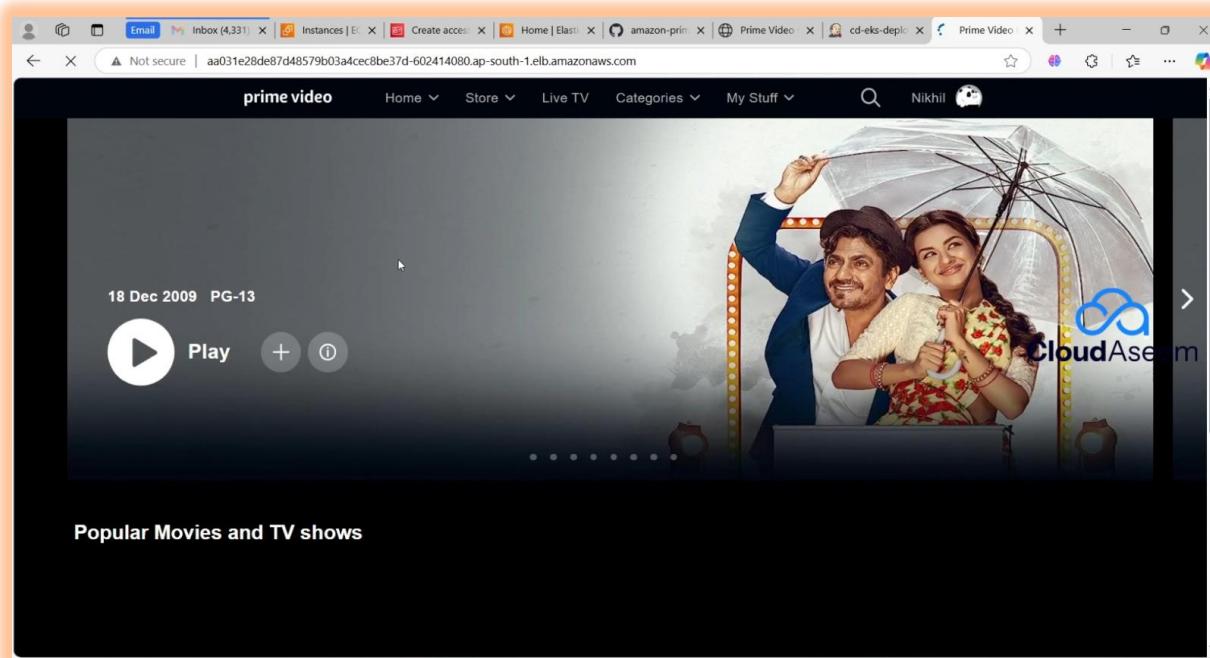
Kubernetes will automatically recreate the pod to maintain the desired state.

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

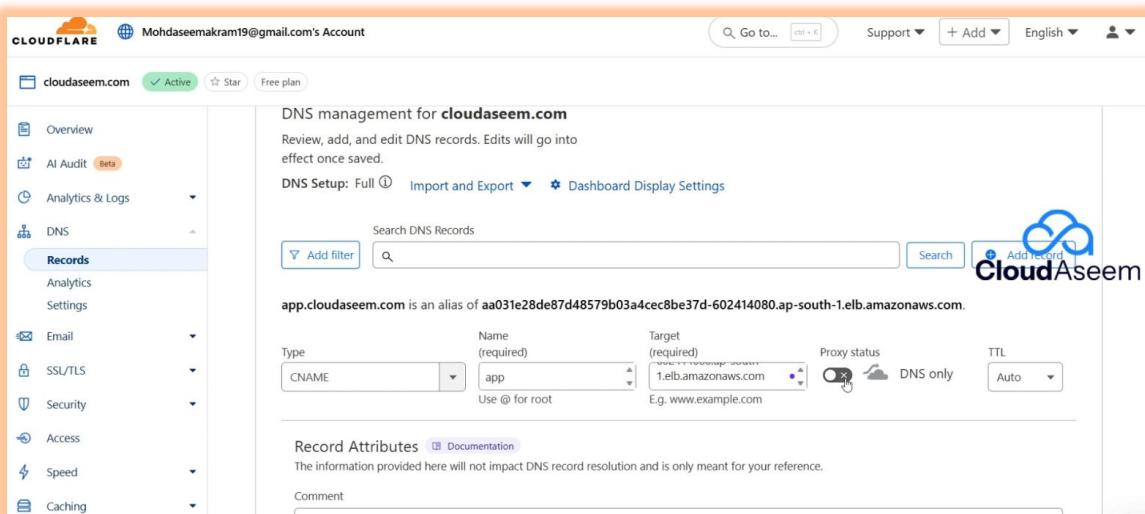


Add Domain to Our application

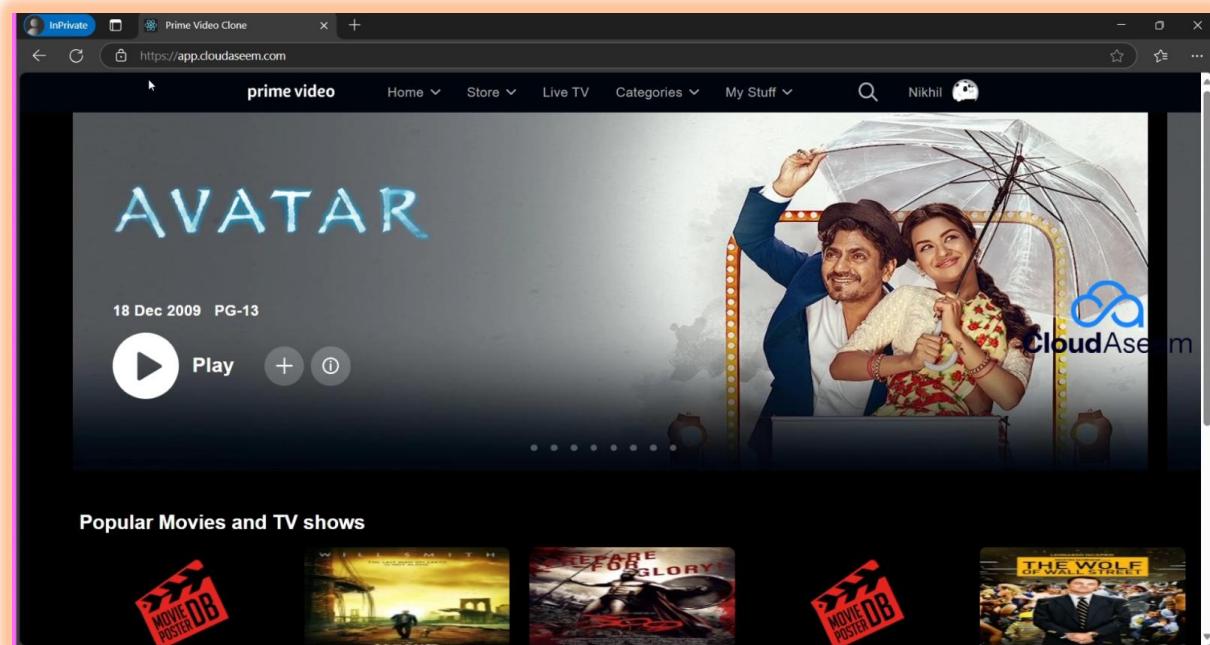
Add Loadbalance in Cloudflare to apply domain and ssl to access by client with cname entry

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

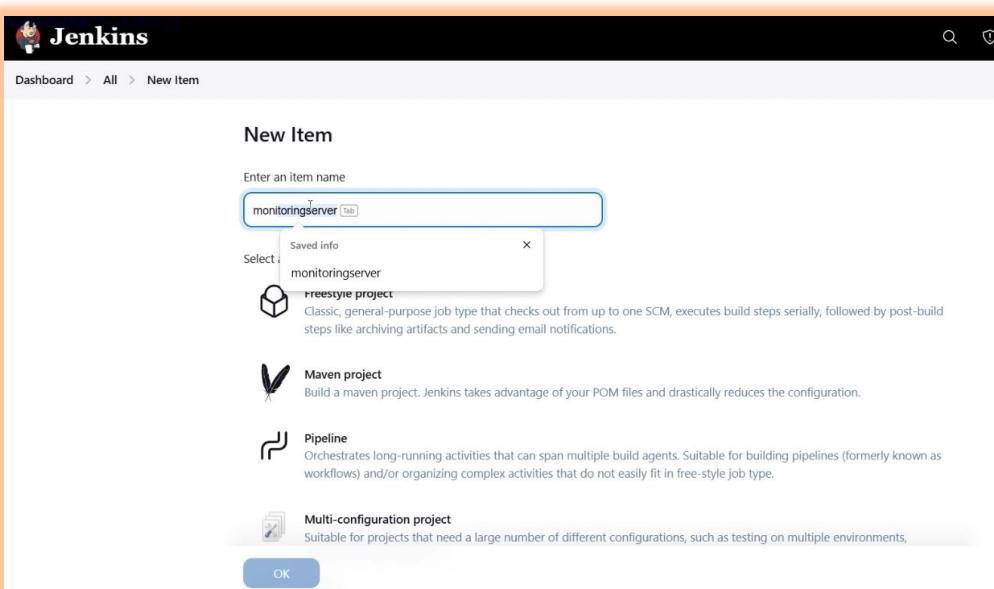


Part – 03

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

Global credentials (unrestricted)

[+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description	Actions
Sonar-token	Sonar-token	Secret text	Sonar-token	
github-token	Aseemakram19/******** (github-token)	Username with password	github-token	
docker	aseemakram19/******** (docker)	Username with password	docker	
smtp-gmail	mohdaseemakram19@gmail.com/******** (smtp-gmail)	Username with password	smtp-gmail	
<u>AWS_ACCESS_KEY_ID</u>	AWS_ACCESS_KEY_ID ✓	Secret text ✓	AWS_ACCESS_KEY_ID ✓	
<u>AWS_SECRET_ACCESS_KEY</u>	AWS_SECRET_ACCESS_KEY ✓	Secret text ✓	AWS_SECRET_ACCESS_KEY ✓	

Icon: S M L

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

This project is parameterized ?

 Choice Parameter ?

Name ?

Choices ?

Description ?

Plain text [Preview](#)

Save
Apply

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

- Status
- Changes
- Build with Parameters
- Configure
- Delete Pipeline
- Full Stage View
- Rename

Pipeline Terraform-Eks

This build requires parameters:

action

 Build Cancel

```
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"
""

}

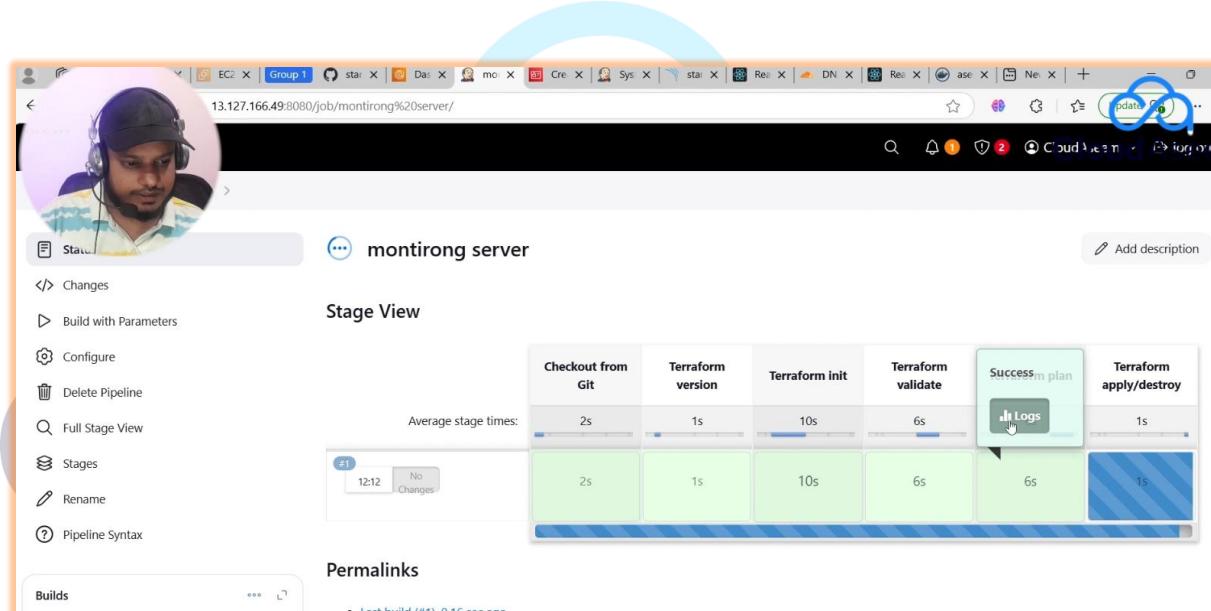
}

}

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

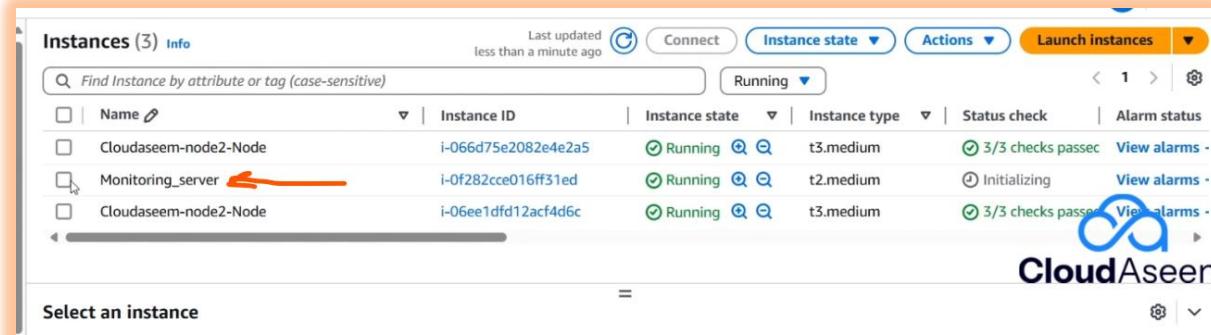
```

Note: Store aws secret and secret access key in jenkins credentials below



The screenshot shows a Jenkins pipeline interface. On the left, there's a sidebar with options like 'Changes', 'Build with Parameters', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Stages', 'Rename', and 'Pipeline Syntax'. The main area is titled 'Stage View' and shows a timeline of stages: 'Checkout from Git' (2s), 'Terraform version' (1s), 'Terraform init' (10s), 'Terraform validate' (6s), 'Success plan' (6s), and 'Terraform apply/destroy' (1s). The 'Success plan' stage is highlighted with a green box and a blue arrow points to its 'Logs' button. Below the timeline, it says 'Average stage times:' and shows a bar chart with the same values: 2s, 1s, 10s, 6s, 6s, and 1s. At the bottom, there's a 'Permalinks' section with a link to 'Last build (#1), 0.16 sec ago'.

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



The screenshot shows the AWS CloudWatch Instances page. It displays three instances: 'Cloudaseem-node2-Node', 'Monitoring_server' (highlighted with a red arrow), and 'Cloudaseem-node2-Node'. The 'Monitoring_server' instance has an Instance ID of i-0f282cce016ff31ed and is in a 'Running' state. The other two instances have Instance IDs i-066d75e2082e4e2a5 and i-06ee1dfd12acf4d6c respectively, and are also in a 'Running' state. The page includes filters for 'Name', 'Instance ID', 'Instance state', 'Instance type', 'Status check', and 'Alarm status'. There are buttons for 'Connect', 'Actions', and 'Launch instances'. At the bottom, there's a 'Select an instance' dropdown and some additional buttons.

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 execute 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
```

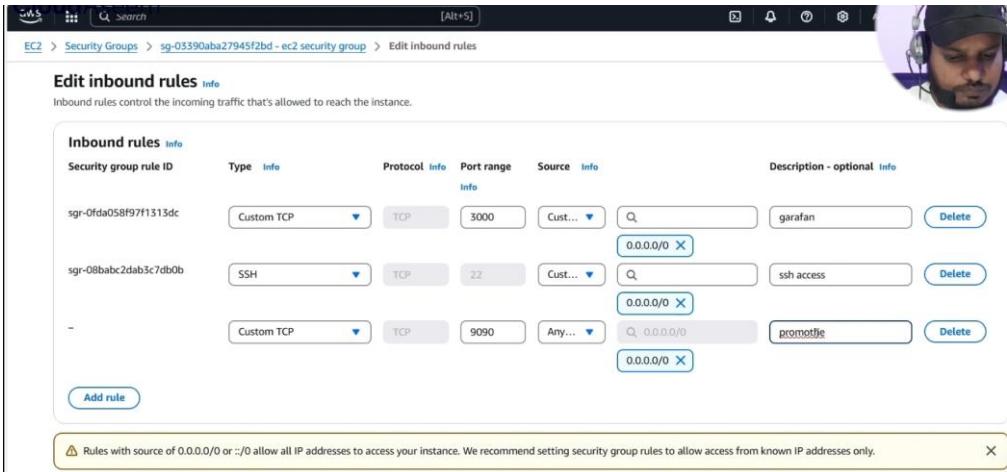
```
sudo apt-get install grafana -y
```

```
# Start and enable Grafana service
```

```
sudo systemctl start grafana-server
```

```
sudo systemctl enable grafana-server
```

Open ports in SG group



The screenshot shows the AWS Management Console interface for managing security groups. The user is editing the inbound rules for the security group 'sg-05390aba27945f2bd'. There are three existing rules:

- Custom TCP rule (port 3000) from 'garafan' (IP 0.0.0.0/0).
- SSH rule (port 22) from 'ssh access' (IP 0.0.0.0/0).
- Custom TCP rule (port 9090) from 'promotjje' (IP 0.0.0.0/0).

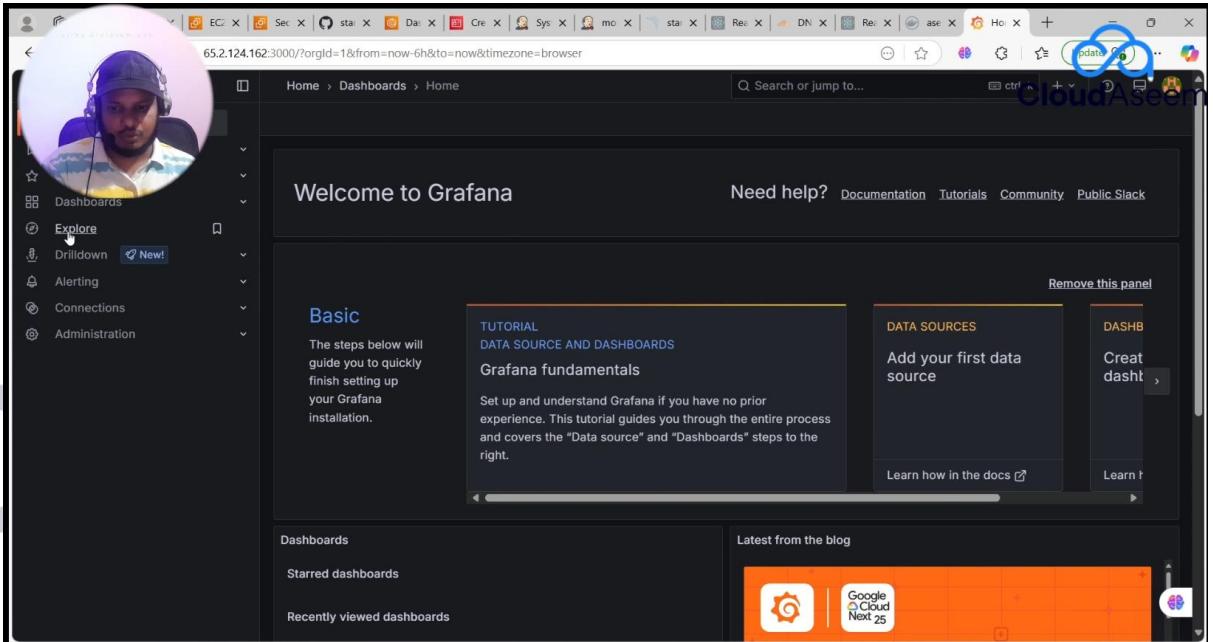
A new rule is being added with the following details:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-08babc2dab3c7db0b	SSH	TCP	22	Cust... (0.0.0.0/0)	ssh access
-	Custom TCP	TCP	9090	Any... (0.0.0.0/0)	promotjje

A warning message at the bottom states: "⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only." A small video thumbnail of the speaker is visible in the top right corner.

After installation, you can access Grafana at:

<http://your-server-ip:3000> (default user: admin, password: admin)



The screenshot shows the Grafana home dashboard. The URL in the browser is 65.2.124.162:3000/?orgId=1&from=now-6h&to=now&timezone=browser. The dashboard features a central 'Welcome to Grafana' section with a video thumbnail of the speaker. To the left is a sidebar with navigation links: Dashboards, Explore (highlighted), Drilldown, Alerting, Connections, and Administration. The main content area includes sections for 'Basic' setup steps, a 'TUTORIAL' for 'DATA SOURCE AND DASHBOARDS', 'Grafana fundamentals', and a 'DATA SOURCES' panel prompting to 'Add your first data source'. At the bottom, there are sections for 'Dashboards' (Starred dashboards, Recently viewed dashboards) and 'Latest from the blog'.

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${PROMETHEUS_VERSION}/prometheus-${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 \
EOF
```

```
--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

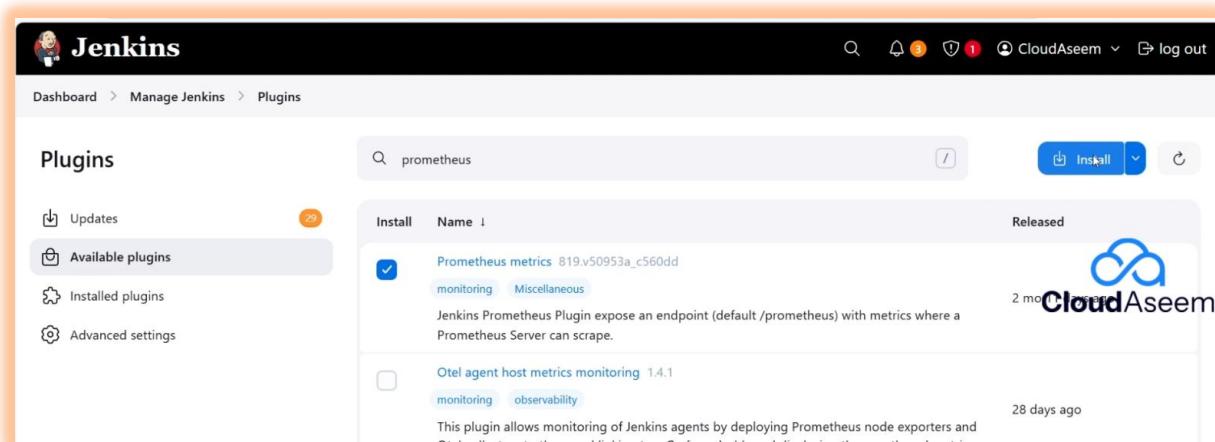
```
t=0.70.0.0-Linux-amd64-ctrl.gz
--2025-03-20 01:29:24-- https://github.com/prometheus/blackbox_exporter/releases/download/v0.26.0/blackbox_exporter-0.26.0.linux-amd64.tar.gz
Resolving github.com (github.com)... 20.207.73.82
Connecting to github.com (github.com)|20.207.73.82|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/41964498/37e20444-c65c-45e1-9674-99ed1137b43
b?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=releaseassetproduction%2F20250320%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20
250320T012924Z&X-Amz-Signature=2d54029d913c128ab7b481bfaba38b65fbef005b05aa20b986af70d939579869&X-Amz-SignedHeade
rs=host&response-content-disposition=attachment%3B%20filename%3Dblackbox_exporter-0.26.0.linux-amd64.tar.gz&response-content-type=a
pplication%2Foctet-stream [following]
--2025-03-20 01:29:24-- https://objects.githubusercontent.com/github-production-release-asset-2e65be/41964498/37e20444-c65c-45e1-9
674-99ed1137b43b?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=releaseassetproduction%2F20250320%2Fus-east-1%2Fs3%2Faws4_
request&X-Amz-Date=20250320T012924Z&X-Amz-Signature=300&X-Amz-Signature=2d54029d913c128ab7b481bfaba38b65fbef005b05aa20b986af70d939579869&X-
Amz-SignedHeaders=host&response-content-disposition=attachment%3B%20filename%3Dblackbox_exporter-0.26.0.linux-amd64.tar.gz&response-
content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12370868 (12M) [application/octet-stream]
Saving to: 'blackbox_exporter-0.26.0.linux-amd64.tar.gz'

blackbox_exporter-0.26.0.linux-amd64.tar.gz 100%[=====] 11.80M 40.0MB/s in 0.3s
2025-03-20 01:29:26 (40.0 MB/s) - 'blackbox_exporter-0.26.0.linux-amd64.tar.gz' saved [12370868/12370868]

root@ip-172-31-38-246:/home/ubuntu#
```

Note:

install Prometheus plugin in Jenkins server



The screenshot shows the Jenkins Plugins management interface. A search bar at the top right contains the text 'prometheus'. Below it, a list of available plugins is shown, with the 'Prometheus metrics' plugin selected for installation. The plugin details page is visible, showing its name, version, and a brief description.

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=WARNING 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#

```

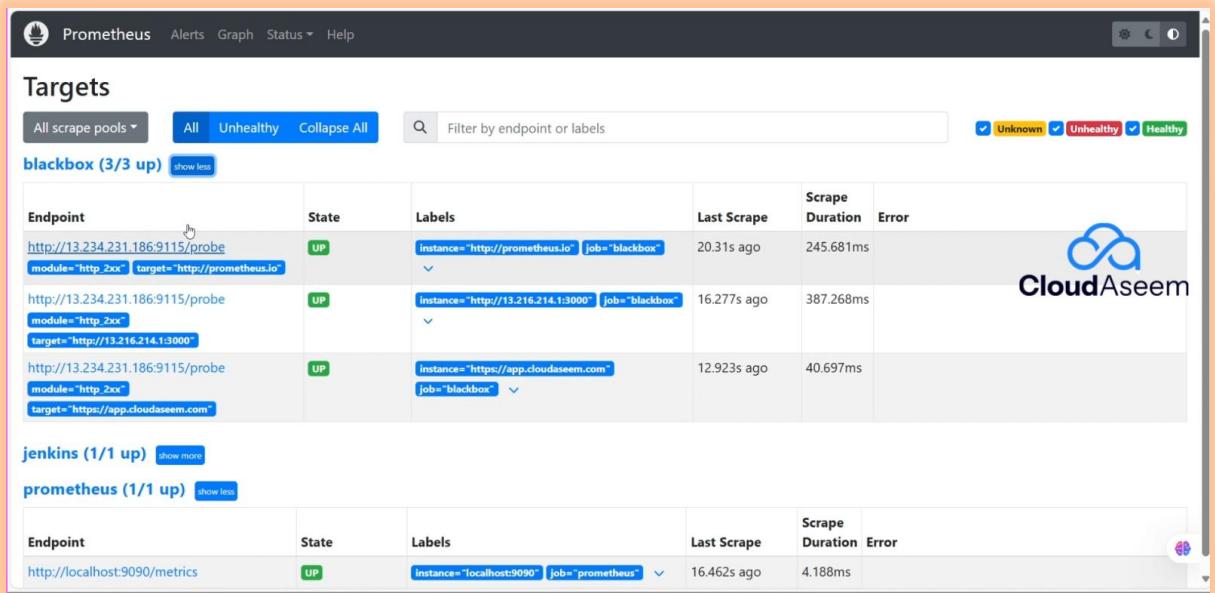
restart

./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.



The screenshot shows the Prometheus Targets page with three groups of targets:

- blackbox (3/3 up)**:
 - http://13.234.231.186:9115/probe (UP, module="http_2xx", target="http://prometheus.io")
 - http://13.234.231.186:9115/probe (UP, module="http_2xx", target="http://13.216.214.1:3000")
 - http://13.234.231.186:9115/probe (UP, module="http_2xx", target="https://app.cloudaseem.com")
- jenkins (1/1 up)**:
 - http://localhost:9090/metrics (UP, instance="localhost:9090", job="prometheus")
- prometheus (1/1 up)**:
 - http://localhost:9090/metrics (UP, instance="localhost:9090", job="prometheus")

Add Jenkins Dashboard

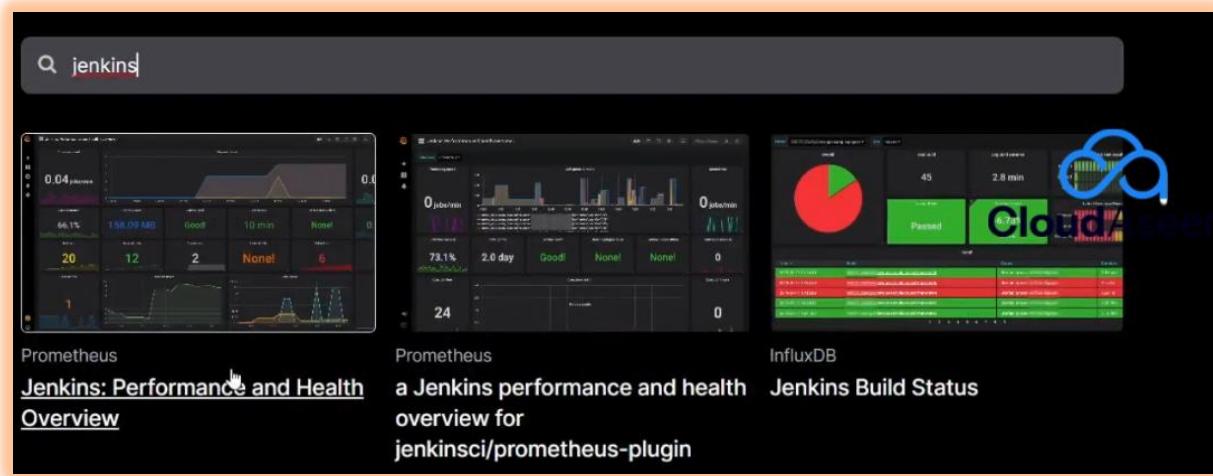
Go to Dashboards → Import.

Enter Dashboard ID: 9964

Click Load.

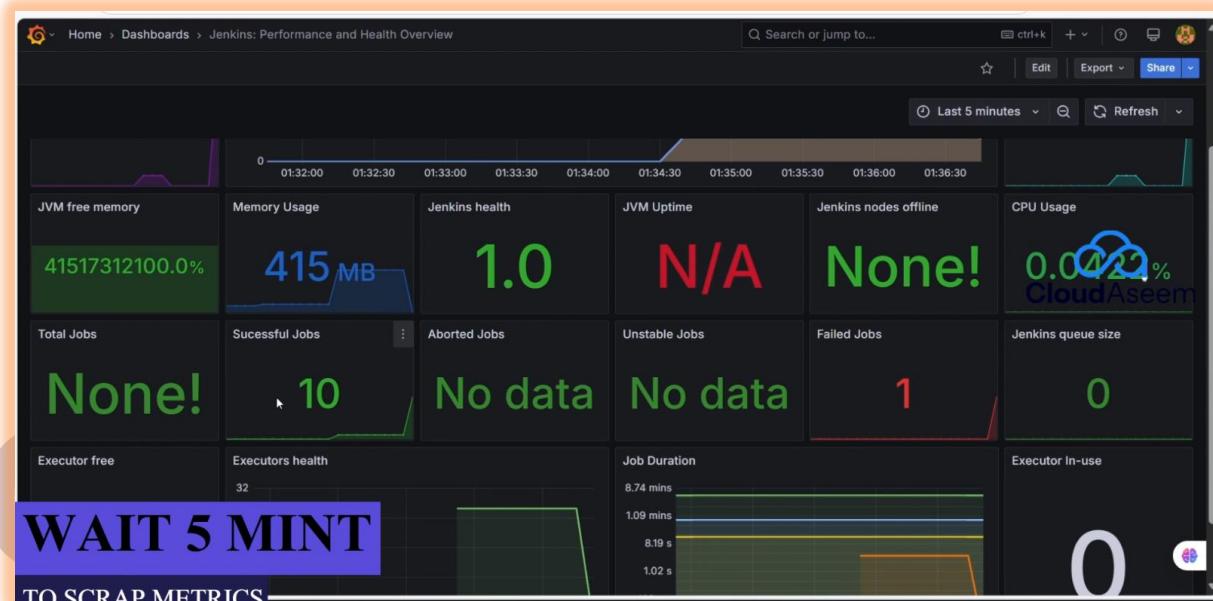
Select Prometheus as the Data Source.

CloudAseem



The screenshot shows three separate Grafana dashboards related to Jenkins monitoring:

- Prometheus Jenkins: Performance and Health Overview**: This dashboard includes a graph of JVM free memory usage over time, showing values like 415 MB. It also displays metrics for total jobs (None!), successful jobs (10), and failed jobs (1).
- Prometheus Jenkins performance and health**: This dashboard provides a Jenkins health overview with a status of "Good". It shows metrics such as 73.1% Jenkins health, 2.0 day Jenkins uptime, and 0 Jenkins nodes offline.
- InfluxDB Jenkins Build Status**: This dashboard displays Jenkins build status with a pie chart showing 45 builds. It includes a table of build results with categories like "Passed" and "Failed".



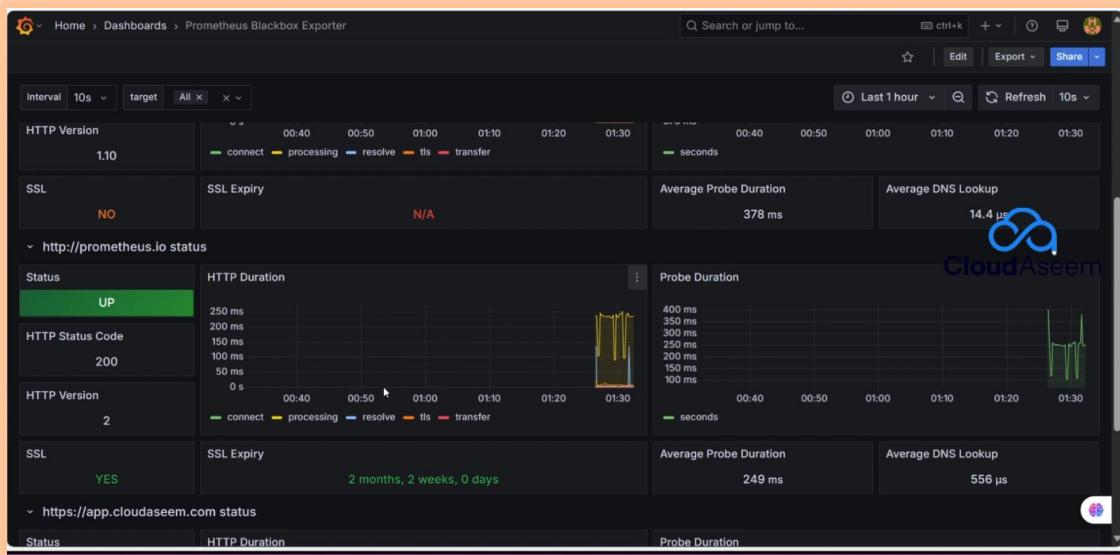
Add

Go to **Dashboards → Import**.

Enter **Dashboard ID: 13659** (Prometheus Blackbox Exporter).

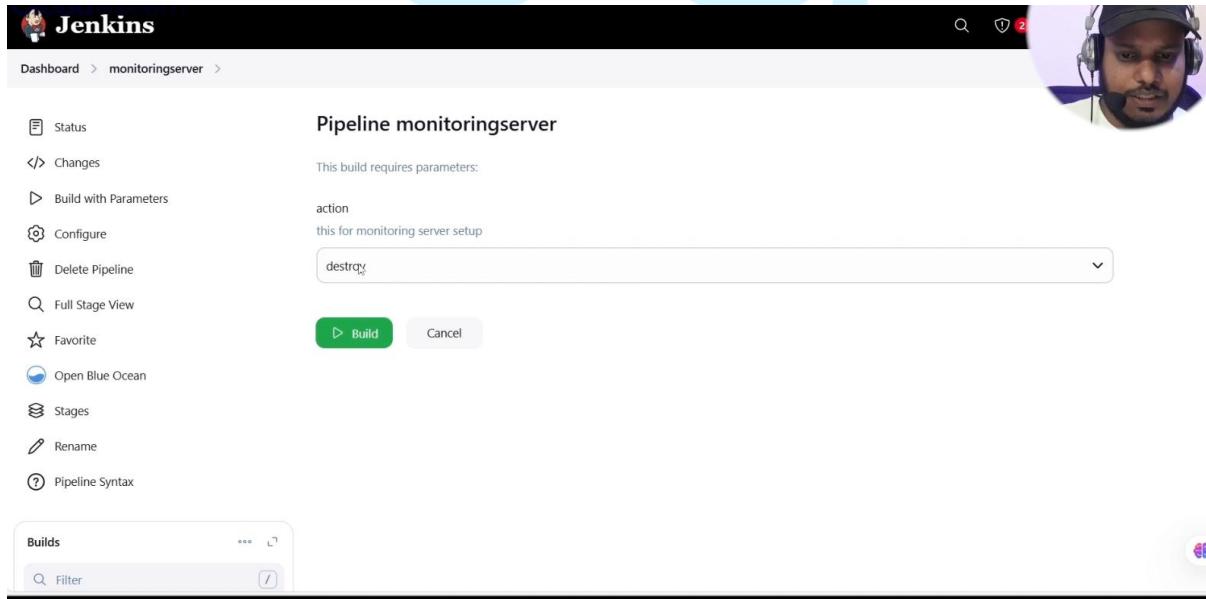
Click **Load**.

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



Step - 4 :

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



The screenshot shows a Jenkins pipeline configuration page. The pipeline is named "Pipeline monitoringserver". The configuration includes:

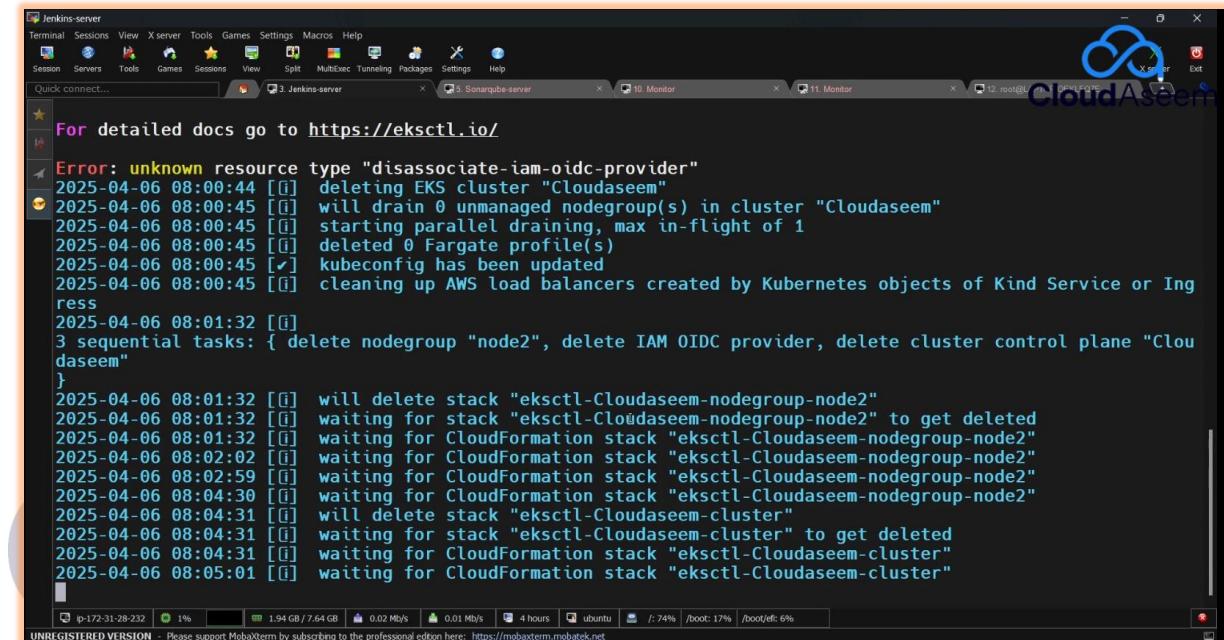
- Changes:** This build requires parameters.
- Configure:** Action: "this for monitoring server setup".
- Parameters:** A dropdown menu showing "destroy" as the selected value.
- Build Buttons:** "Build" and "Cancel".

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-cluster4"
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 of 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 Ingress
2025-04-06 08:01:32 [i]
3 sequential tasks: { delete nodegroup "node2", delete IAM OIDC provider, delete cluster control plane "Cloudaseem"
}
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) - <https://www.youtube.com/@clouddevopswithaseem>
- 👉 LinkedIn: [Mohammed Aseem Akram](https://www.linkedin.com/in/mohammed-aseem-akram) - www.linkedin.com/in/mohammed-aseem-akram
- 👉 GitHub: [AseemAkram19](https://github.com/AseemAkram19)



CloudAseem

[Mohammed Aseem Akram](#)

🌟 Stay tuned for more DevOps insights! 🚀

#Cloud #DevOps #Technology #Kubernetes



CloudAseem