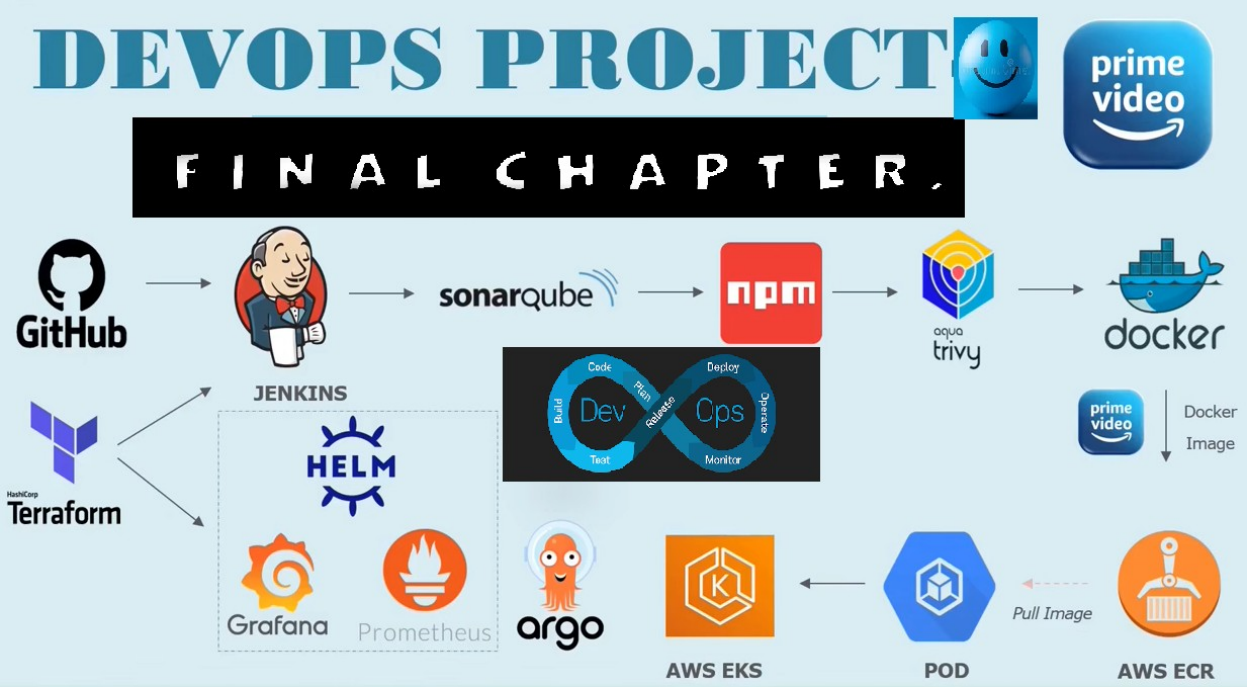
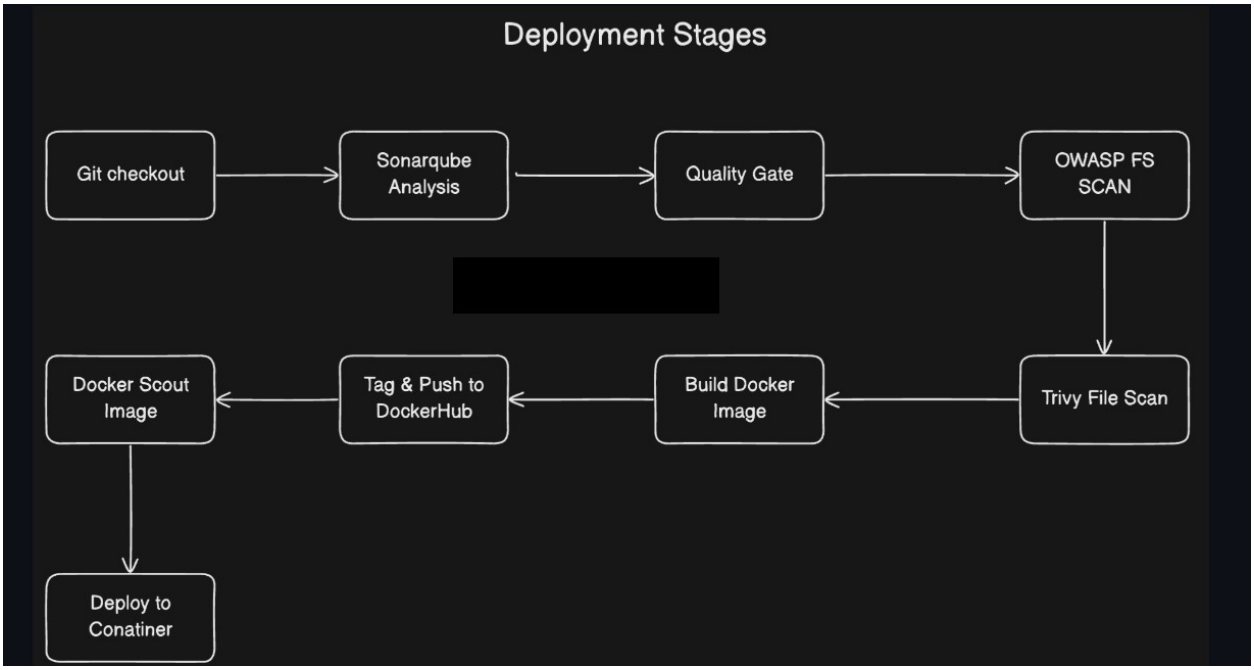
AWS DevOps- CI/CD Automation Project- Final Chapter





This is a fully automated DevOps CI/CD project. In this project, we automate a complete DevOps pipeline using enkins, SonarQube, Docker, ECR (Amazon Elastic Container Registry), and AWS services. Set up a Jenkins pipeline to build and deploy an application into AWS, with quality checks using SonarQube and security scans using Trivy. The pipeline will also push Docker images to AWS ECR and handle Docker image cleanup. I will walk you through step-by-step, from Jenkins setup to deployment. I have added --how 2 create sub-domain from the load balancer endpoint using Route53.

Overview:

This project focuses on implementing a Continuous Integration (CI) and Continuous Deployment (CD) pipeline using Jenkins. The goal is to deploy a clone of the Amazon Prime Video application to Amazon Elastic Container Registry (ECR) while incorporating security vulnerability scanning using Trivy. The pipeline also integrates SonarQube for static code analysis and quality gate checks.

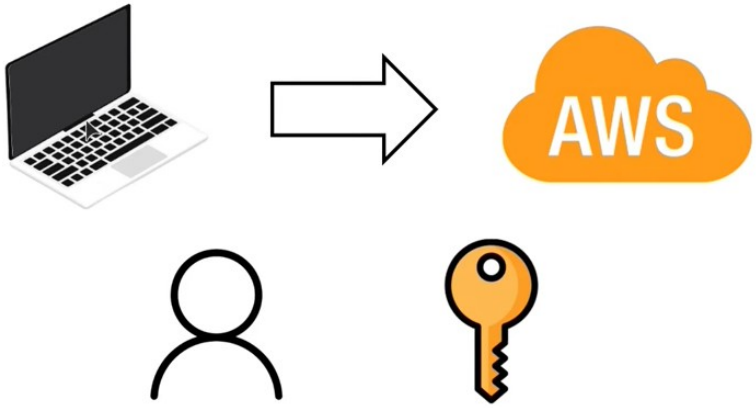
This project uses: Jenkins for Continuous Integration (CI) SonarQube for Code Quality Analysis Docker for Containerization AWS ECR for storing Docker images Trivy for vulnerability scanning Terraform for infrastructure as code (IAC)

Introduction

- **Pre-requisites**
- **Configuration**
- **Creating Jenkins server using Terraform**
- **Access Jenkins Server**
- **Configure SonarQube**
- **Configure Jenkins**
- **Build Pipeline Overview**
- **Writing Build pipeline code from scratch**
- **Running Build pipeline**
- **Creating EKS using Terraform**
- **Deployment pipeline overview**
- **Running Deployment pipeline**
- **Access Prometheus, Grafana & ArgoCD**
- **Deploy Application using ArgoCD**
- **Access Amazon prime Application**
- **Creating sub-domain using Route53**
- **Access Grafana & create dashboard**
- **Running Cleanup pipeline**
- **Cleanup Jenkins Server & AWS EKS using Terraform**

PRE-REQUISITES

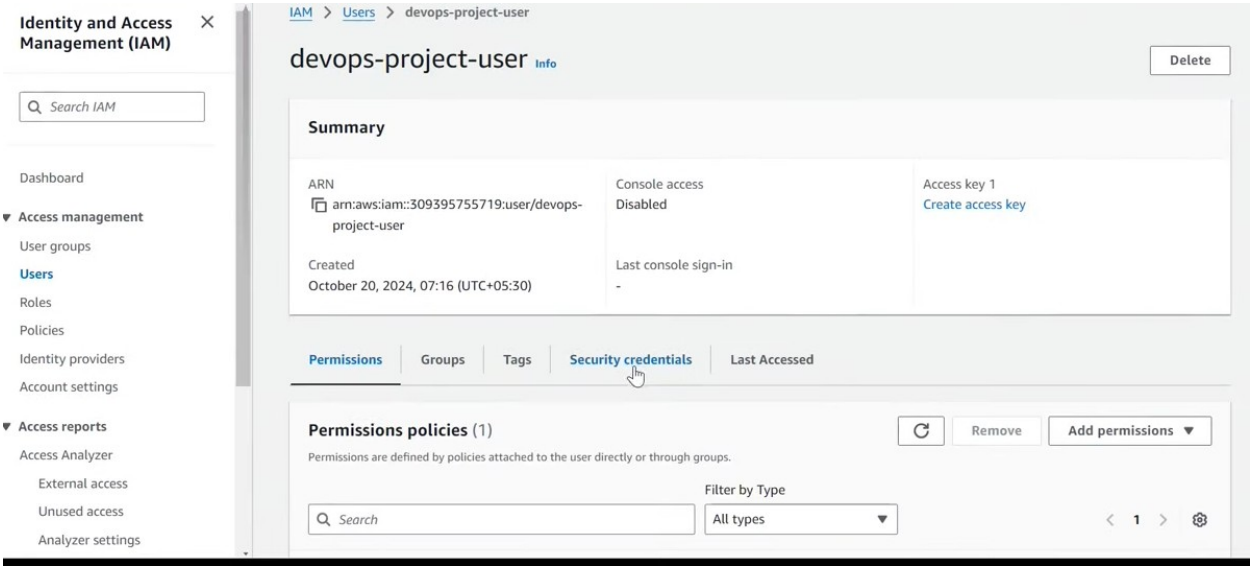
CONFIGURATION



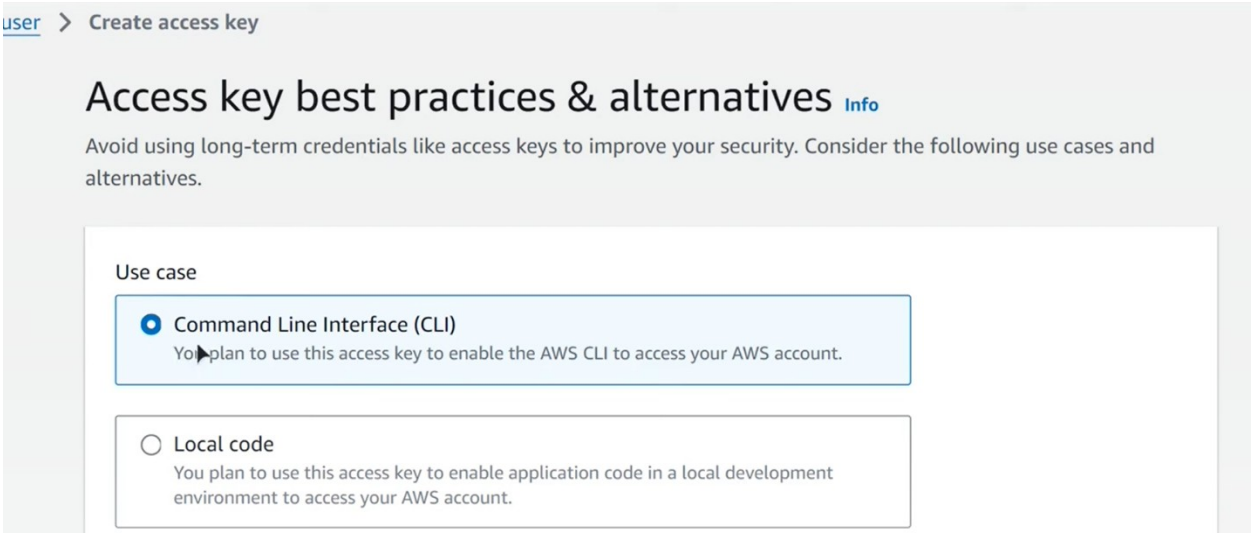
Tools & Technologies:

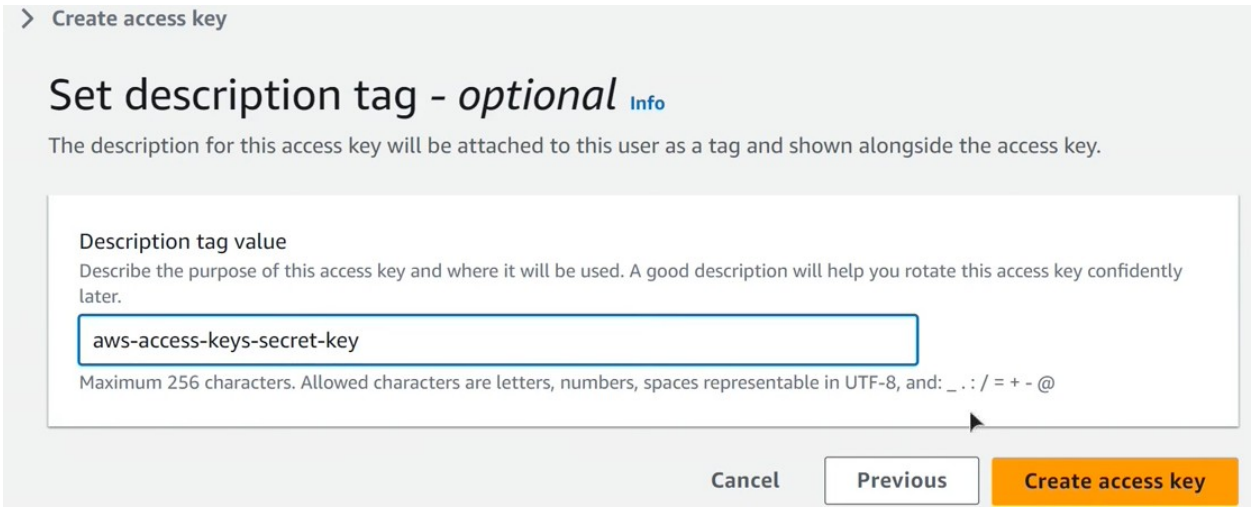
- **Jenkins**: An automation server used to manage the CI/CD pipeline.
- **SonarQube**: A platform for continuous inspection of code quality and security vulnerabilities.
- **Trivy**: A security vulnerability scanner specifically for Docker images.
- **Docker**: A containerization platform used to package the application into lightweight containers.
- **AWS ECR (Elastic Container Registry)**: A fully managed Docker container registry for storing Docker images.
- **AWS CLI**: Command-line tool to interact with AWS services.

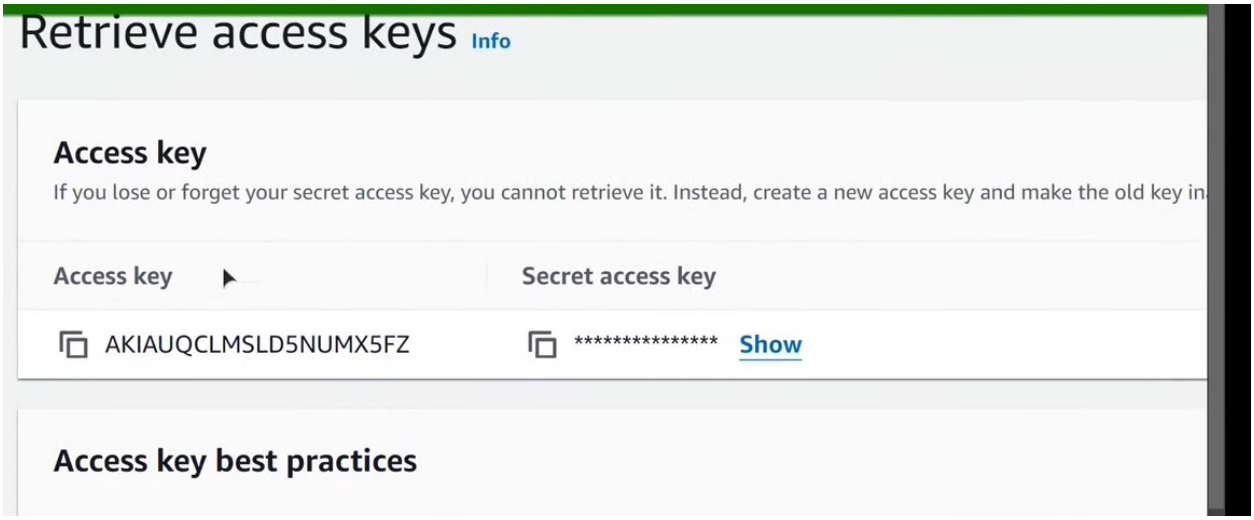
 # Go to AWS IAM .. create user – give Admin Access Permision



 # now Generate Access Key


user > Create access key
Access key best practices & alternatives Info
Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.
Use case
Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.
Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.


Create access key
Set description tag - optional Info
The description for this access key will be attached to this user as a tag and shown alongside the access key.
Description tag value
Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.
aws-access-keys-secret-key
Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . : / = + - @
Cancel
Previous
Create access key


Retrieve access keys 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 in
Access key
Secret access key
AKIAUQCLMSLD5NUMX5FZ
****
Show
Access key best practices

```
# Access key:
AKIAUQCLMSLD5NUMX5FZ

# Secret key:
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
```

 # Jenkin Plugins

```
# JENKINS PLUGINS
SonarQube scanner
NodeJS
Pipeline: Stage View
Docker, Docker Commons, Docker pipeline, Docker API, docker-build-step
Eclipse Temurin Installer
Prometheus metrics
```

- **Pipeline Stages:**

1. **Git Checkout**:

   This stage pulls the source code from a GitHub repository, which contains the Amazon Prime Video clone project.
2. **SonarQube Analysis**:

   Performs static code analysis using SonarQube to check for code smells, bugs, and security vulnerabilities. It uses the pre-configured SonarQube scanner.
3. **Quality Gate**:

   Ensures that the code meets the defined quality standards using SonarQube's Quality Gate. If the quality gate fails, the pipeline stops.
4. **Install npm Dependencies**:

   This stage installs the required npm packages for the application using the command npm install.
5. **Trivy Security Scan**:

   Scans the application's directory for known vulnerabilities using Trivy. The scan results are stored in a file named trivy.txt.

6. **Build Docker Image**:

   This stage builds the Docker image for the application using the Dockerfile found in the repository. The image is tagged based on the user-defined ECR repository name.

7. **Create ECR Repository (if it doesn't exist)**:

   The pipeline checks whether the specified ECR repository exists in the AWS account. If not, it creates the repository automatically. This step ensures that the Docker image has a valid destination for storage.

8. **Tag & Push Docker Image to AWS ECR**:

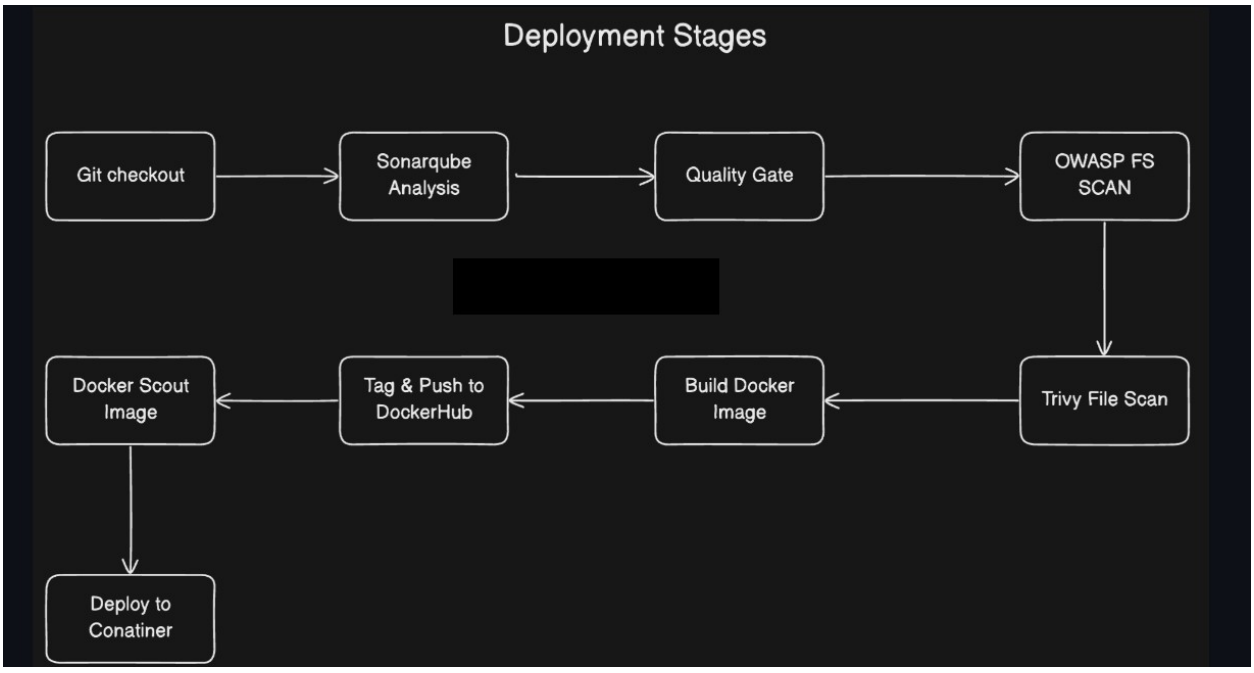
   The Docker image is tagged with both a unique build number and the latest tag. Afterward, the image is pushed to the specified ECR repository in the user's AWS account.

9. **Delete Docker Images from Jenkins Server**

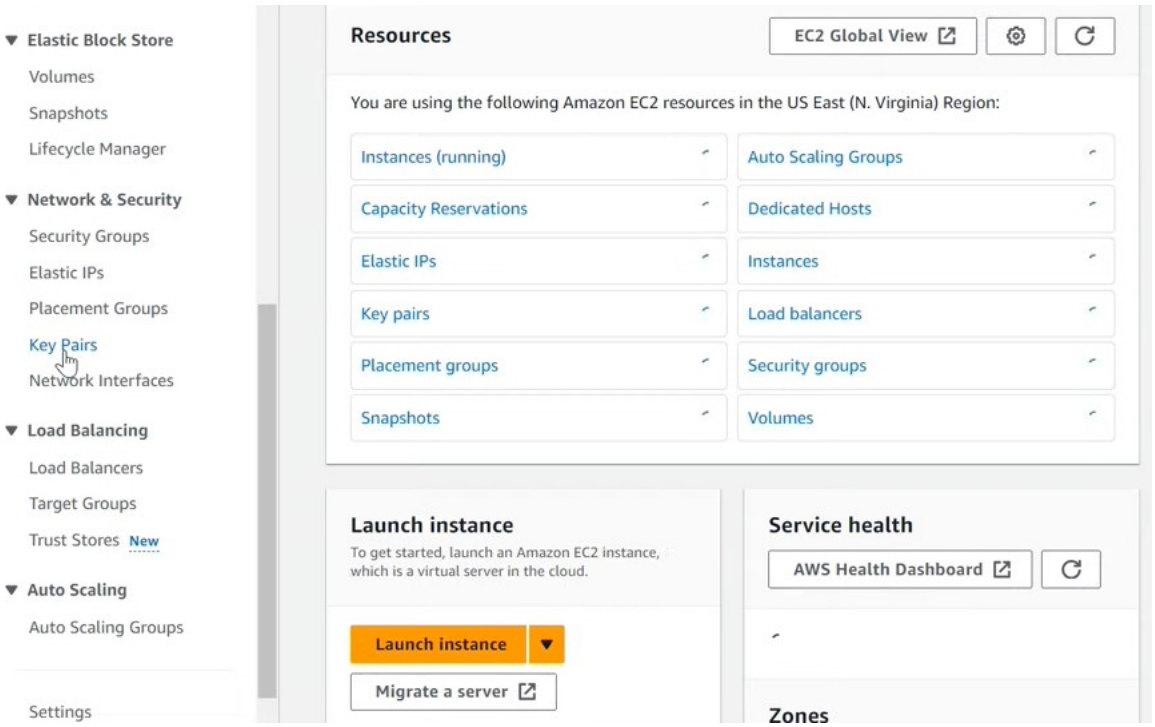
   Once the images are pushed to ECR, the unused (dangling) docker images will be deleted to avoid storage issues.

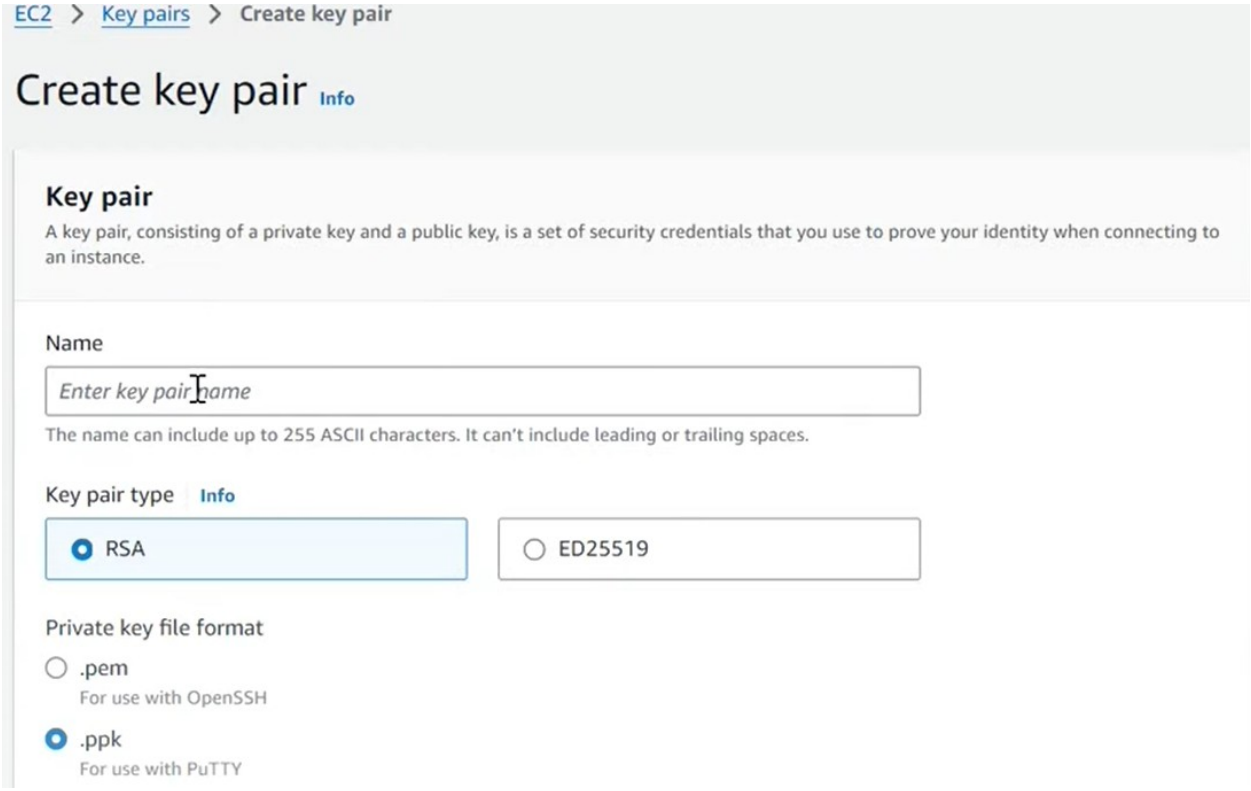
10. **Create ArgoCD, Grafana & Prometheus**

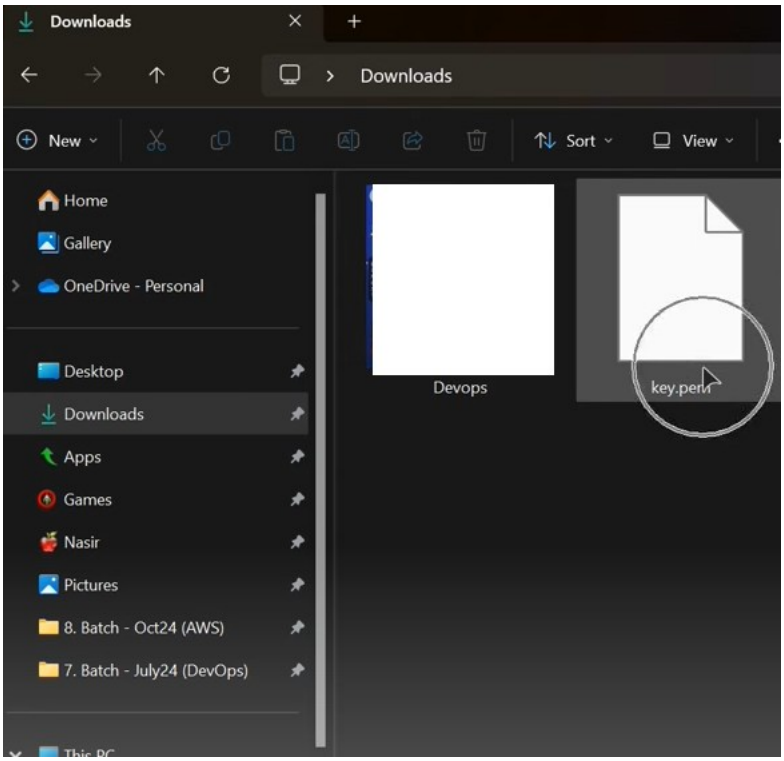
    The pipeline will create ArgoCD, Grafana & Prometheus. ArgoCD will be used to deploy Docker image from ECR. Grafana & Prometheus will be used for monitoring the application.



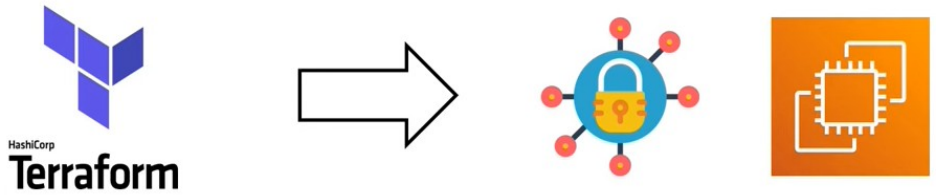
#Key pair for SSH Generation


Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores New
Auto Scaling
Auto Scaling Groups
Settings
Resources
EC2 Global View
You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:
Instances (running)
Auto Scaling Groups
Capacity Reservations
Dedicated Hosts
Elastic IPs
Instances
Key pairs
Load balancers
Placement groups
Security groups
Snapshots
Volumes
Launch instance
To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.
Launch instance
Migrate a server
Service health
AWS Health Dashboard
Zones


EC2 > Key pairs > Create key pair
Create key pair Info
Key pair
A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.
Name
Enter key pair name
The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.
Key pair type Info
RSA
ED25519
Private key file format
.pem
For use with OpenSSH
.ppk
For use with PuTTY



CREATE INFRA USING TERRAFORM



INFRASTRUCTURE CONFIGURATION:

- **EC2 Instance:**
  - **Operating System**: Ubuntu 24.04

- **Security Group:**
  - **SSH**: Port 22
  - **HTTP**: Port 80
  - **HTTPS**: Port 443
  - **NodeJS**: Port 3000
  - **Jenkins**: Port 8080
  - **SonarQube**: Port 9000

- **Jenkins Plugins:**

1. **SonarQube Scanner**:
   Used to integrate Jenkins with SonarQube to check code quality and perform quality gate checks.
2. **NodeJS**:
   Manages and allows the use of different versions of Node.js within Jenkins.
3. **Pipeline: Stage View**:
   Provides a visual interface to monitor the stages of the Jenkins pipeline.
4. **Docker**:
   Allows Jenkins to manage Docker containers directly.
5. **Docker Commons**:
   Facilitates the sharing of common Docker configurations across Jenkins jobs.
6. **Docker Pipeline**:
   Supports building and running Docker containers within Jenkins pipelines.
7. **Docker API**:
   Provides an API to interact with Docker within Jenkins.
8. **docker-build-step**:
   Adds Docker-related build steps, like building images or running containers in freestyle jobs.
9. **SSH Agent**:
   Manages SSH credentials for remote server access within Jenkins pipelines.
10. **Eclipse Temurin Installer**:
    Used to install OpenJDK within Jenkins.
11. **Prometheus Metrics**:
    Enables Jenkins to expose metrics for monitoring and alerting.

SONARQUBE CONFIGURATION:

- **SonarQube Webhook:**
  The SonarQube webhook is used to send analysis results back to Jenkins.

- **Configuration**:
    - Name: sonarqube-webhook
    - URL: http://<jenkins-url>:8080/sonarqube-webhook/
    - Secret: None

- **SonarQube Token:**
    - Generated token to authenticate Jenkins with SonarQube for analysis and quality gate checks.

- **Credentials:**

    - **SonarQube Token:** Used to authenticate and interact Jenkins with SonarQube.

    - **AWS credentials (Access Key and Secret Key):** These credentials are used to authenticate and interact Jenkins with AWS, to push the Docker Image to AWS ECR.

- **Jenkins SonarQube System Configuration:**
    - Helps to store SonarQube server details globally in Jenkins instead of specifying them in each pipeline script.

    - **Configuration**:
        - Name: sonar-server
        - URL: http://<sonar-server-url>:9000

JENKINS GLOBAL TOOL CONFIGURATION:

- **JDK Installation:**
  - **Name**: jdk17
  - **Install From**: adoptium.net
  - **Version**: jdk17.0.8.1+1
- **SonarQube Scanner Installation:**
  - **Name**: sonar-scanner
  - **Install From**: Maven Central
  - **Version**: 6.2.0.4584
- **NodeJS Installation:**
  - **Name**: node16
  - **Install From**: nodejs.org
  - **Version**: 16.20.0
- **Docker Installation:**
  - **Name**: docker
  - **Install From**: docker.com
  - **Version**: Latest

- **Flow of Execution:**

1. **Source Code Fetching**: The pipeline begins by pulling the code from GitHub.
2. **Static Analysis**: SonarQube performs code quality and security checks on the pulled code.
3. **Build & Testing**: The application's dependencies are installed, and the application is packaged into a Docker image.
4. **Security Scanning**: Trivy scans the Docker image for vulnerabilities and produces a report.
5. **Deployment**: The final Docker image is pushed to AWS ECR, making it ready for deployment to AWS ECS or EKS. The application is deployed using ArgoCD. The pods are monitored using Grafana & Prometheus.

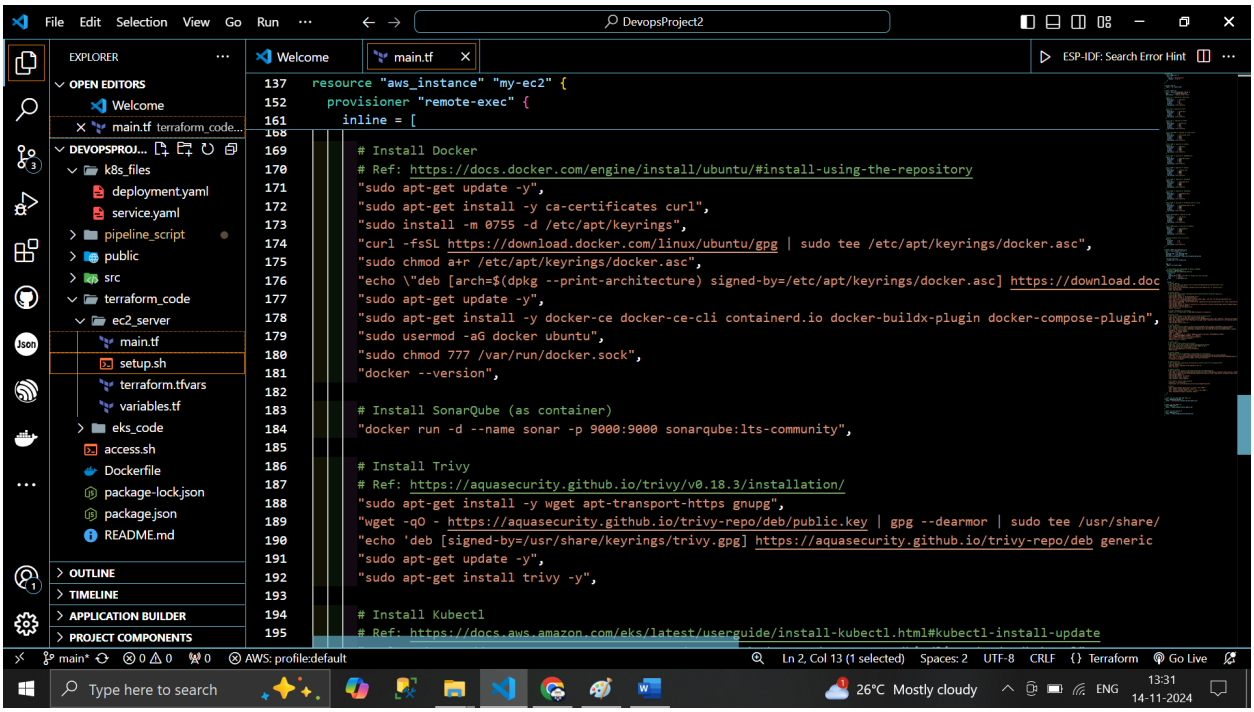
Conclusion:

This project provides a comprehensive CI/CD pipeline from source code management to deployment with integrated code quality checks and security scans. Using AWS ECR ensures that the Dockerized application is securely stored and readily available for deployment in a scalable AWS environment. The application is deployed using ArgoCD. The pods are monitored using Grafana & Prometheus.

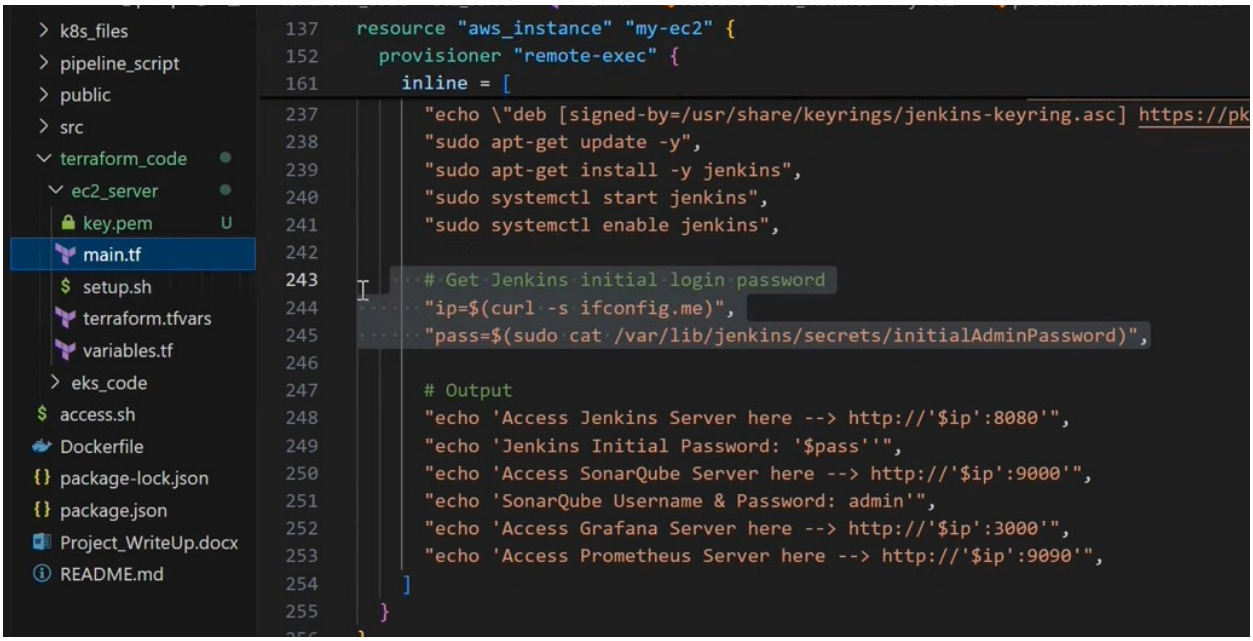
 # Practicals

 # Create Terraform main.tf that initiates all necessary microservices

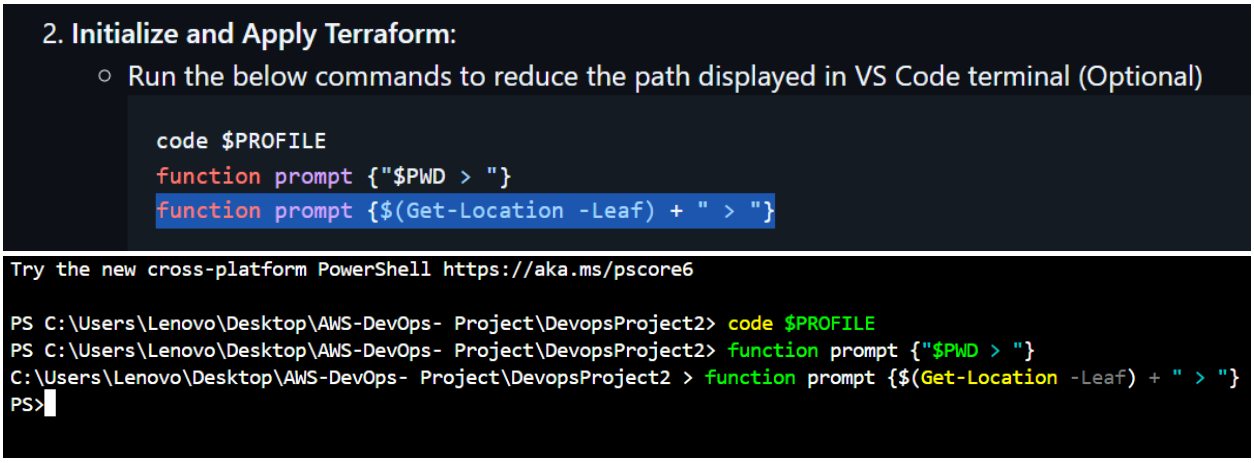
Docker /ec2/argocd/sonarQube/grafana/Prometheus/Trivy/k8s/java/helm/java17



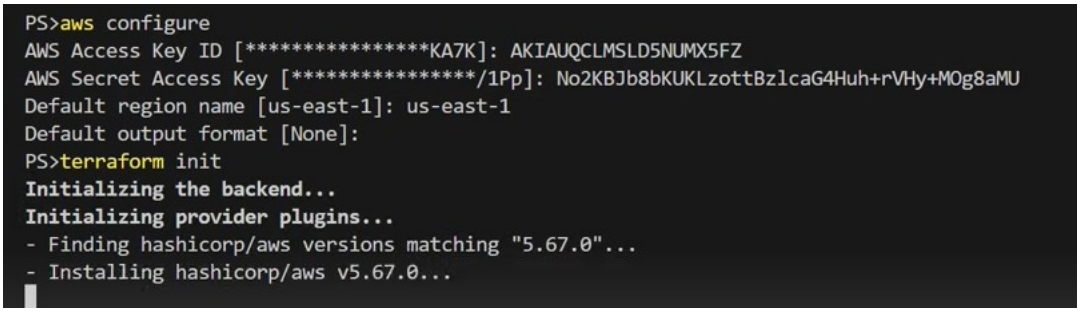
#note key.pem file must be placed in same folder of terraform main.tf



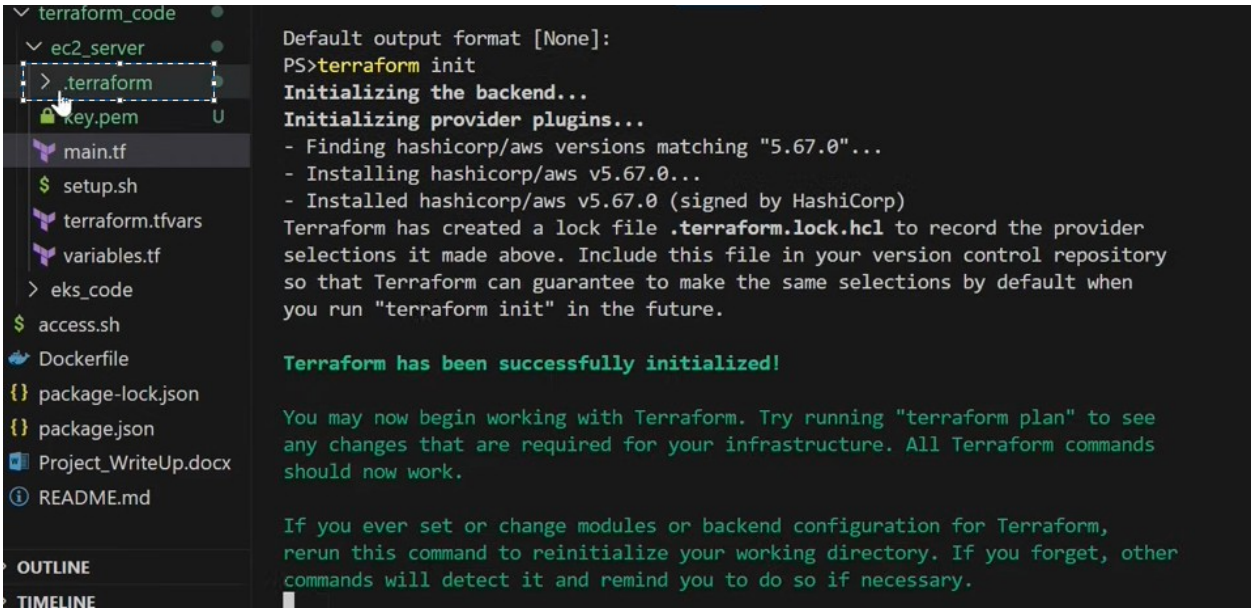
 # Terminal

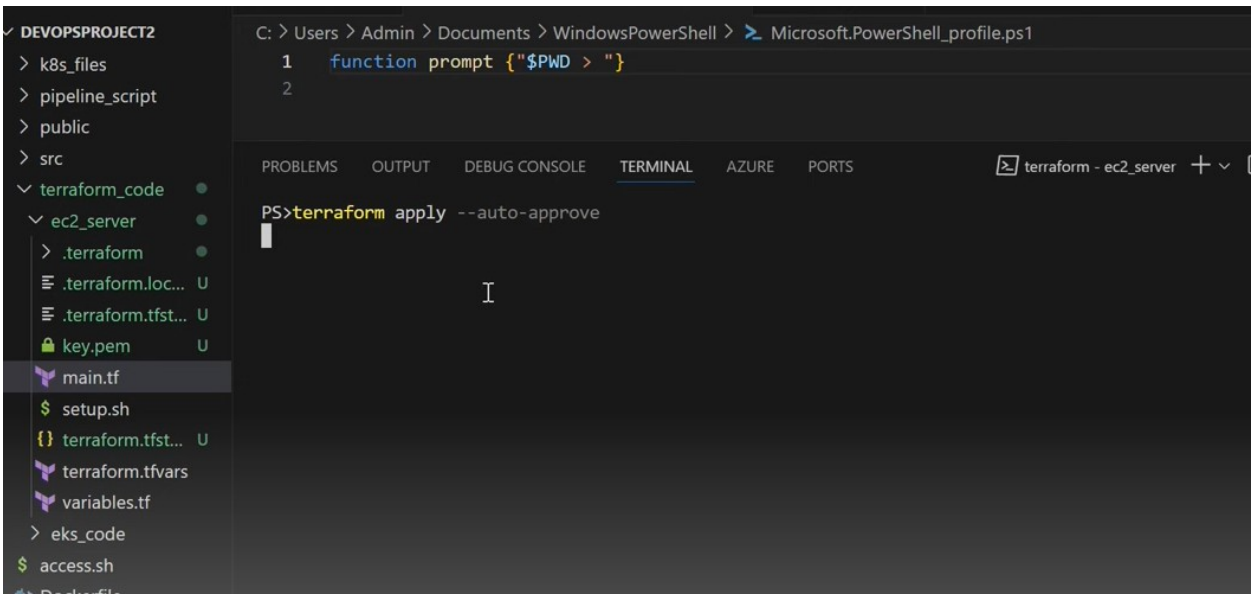


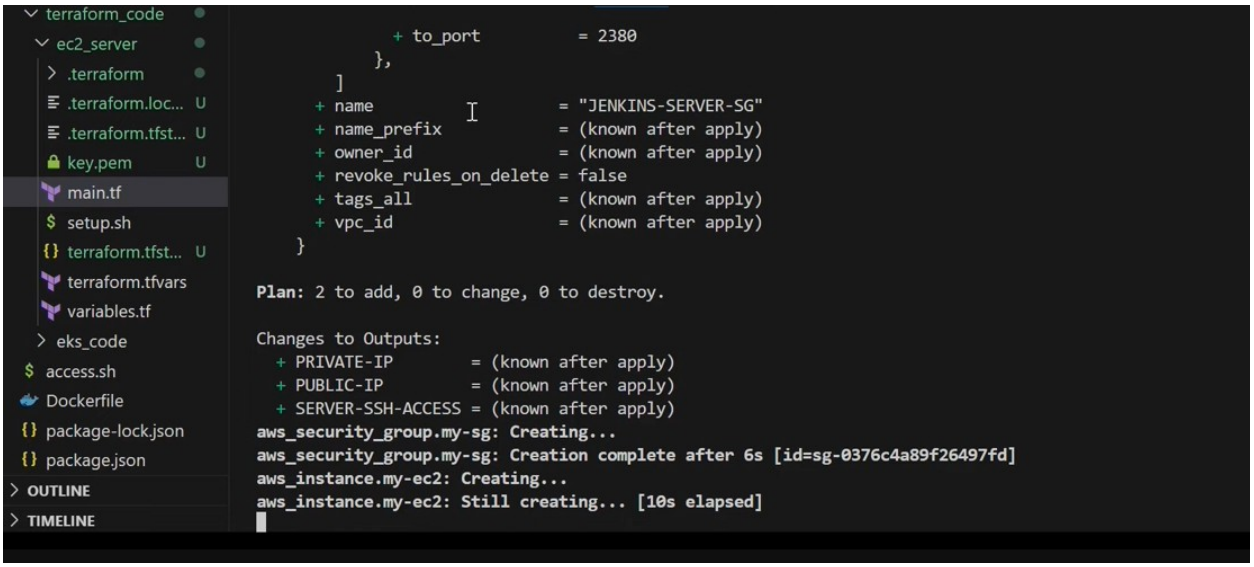
 # AWS Configure .. Connect to AWS Cloud + initiate TF

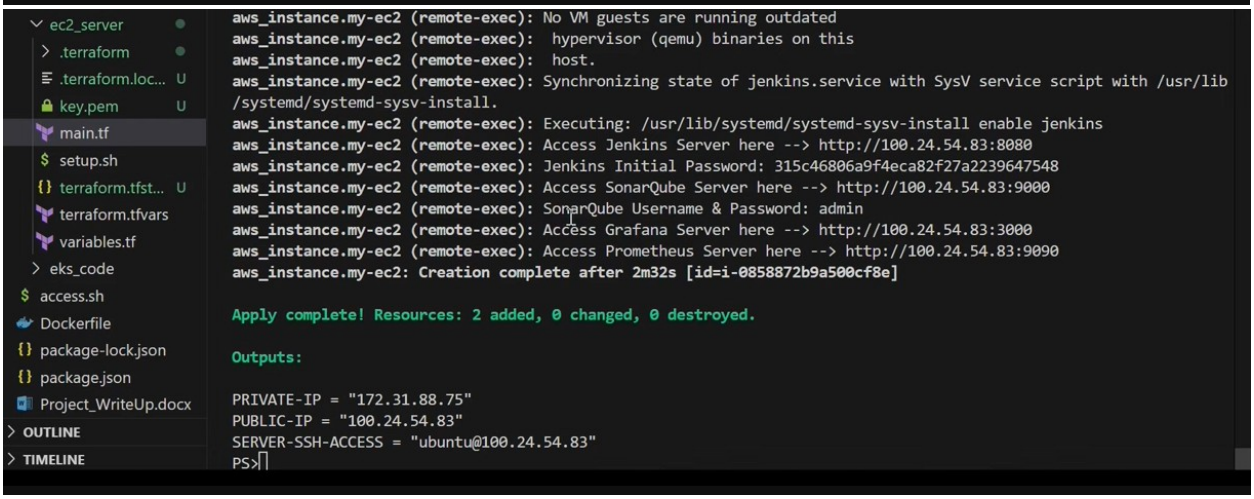


#.terraform Folder is created

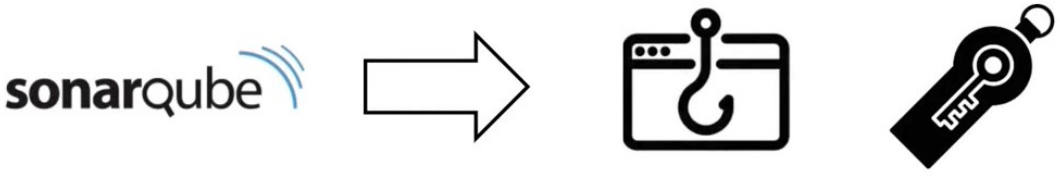

terraform_code
ec2_server
.terraform
key.pem
main.tf
setup.sh
terraform.tfvars
variables.tf
eks_code
access.sh
Dockerfile
package-lock.json
package.json
Project_WriteUp.docx
README.md
OUTLINE
TIMELINE
Default output format [None]:
PS>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.67.0"...
- Installing hashicorp/aws v5.67.0...
- Installed hashicorp/aws v5.67.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.
If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.


DEVOPSPROJECT2
k8s_files
pipeline_script
public
src
terraform_code
ec2_server
.terraform
.terraform.loc...
.terraform.tfst...
key.pem
main.tf
setup.sh
terraform.tfst...
terraform.tfvars
variables.tf
eks_code
access.sh
C: > Users > Admin > Documents > WindowsPowerShell > Microsoft.PowerShell_profile.ps1
1 function prompt {"$PWD > "}
2
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
AZURE
PORTS
terraform - ec2_server
PS>terraform apply --auto-approve





CONFIGURE SONARQUBE

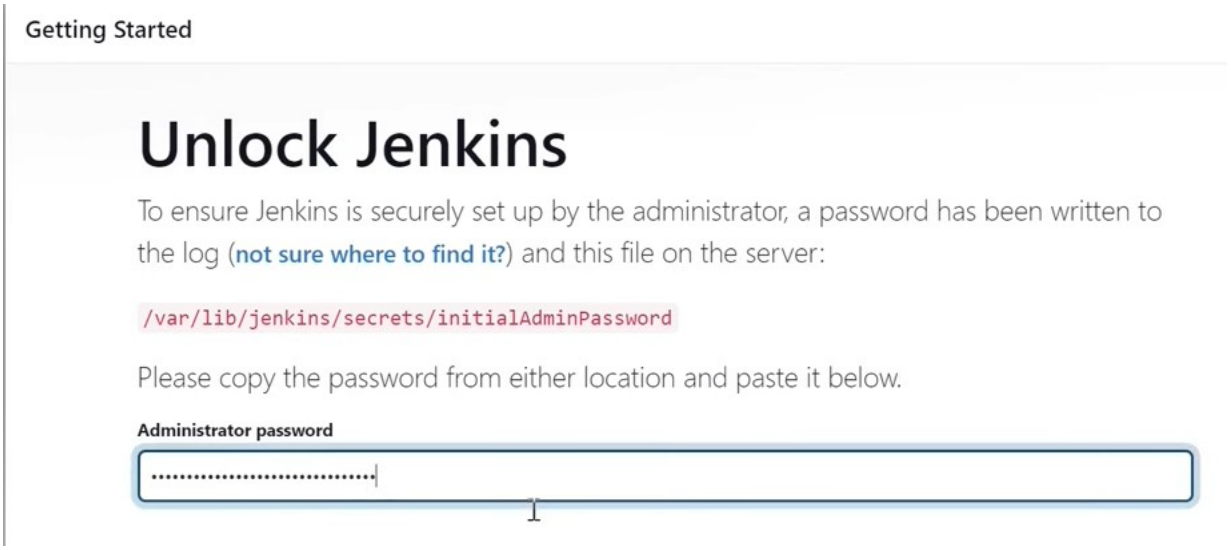


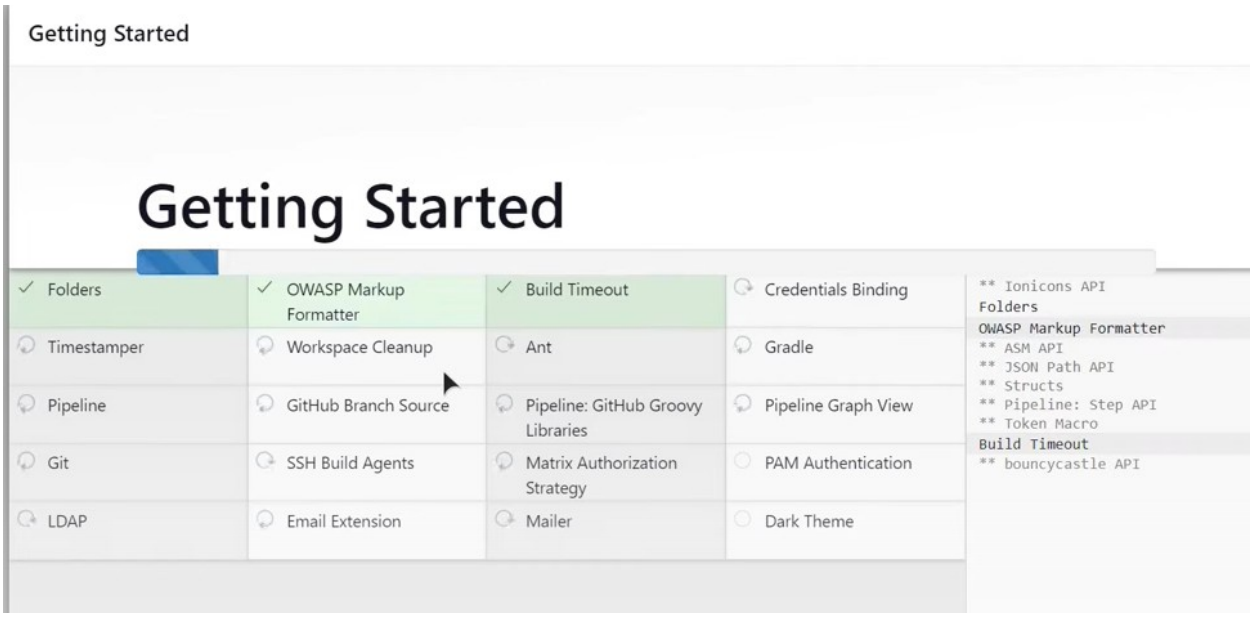
 # cp Jenkins URL

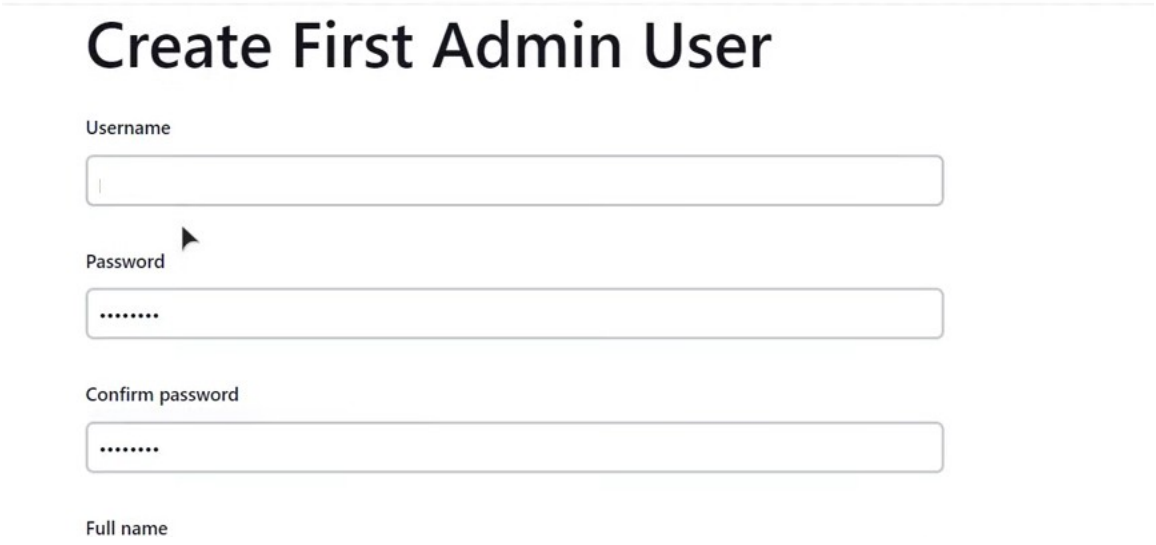
```
aws_instance.my-ec2 (remote-exec): Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
aws_instance.my-ec2 (remote-exec): Access Jenkins Server here --> http://100.24.54.83:8080
aws_instance.my-ec2 (remote-exec): Jenkins Initial Password: 315c46806a9f4eca82f27a2239647548
aws_instance.my-ec2 (remote-exec): Access SonarQube Server here --> http://100.24.54.83:9000
aws_instance.my-ec2 (remote-exec): SonarQube Username & Password: admin
aws_instance.my-ec2 (remote-exec): Access Grafana Server here --> http://100.24.54.83:3000
aws_instance.my-ec2 (remote-exec): Access Prometheus Server here --> http://100.24.54.83:9090
aws_instance.my-ec2: Creation complete after 2m32s [id=i-0858872b9a500cf8e]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

PRIVATE-IP = "172.31.88.75"
PUBLIC-IP = "100.24.54.83"
SERVER-SSH-ACCESS = "ubuntu@100.24.54.83"
```

 # password

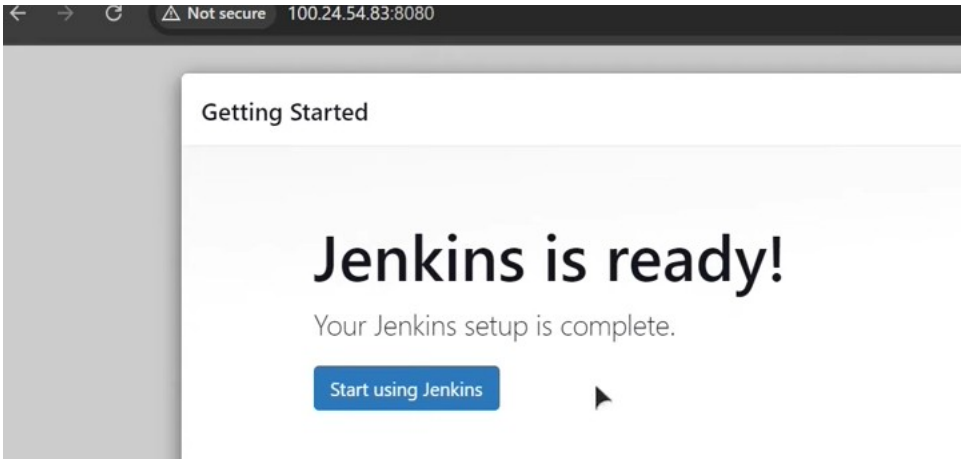
```
aws_instance.my-ec2 (remote-exec): Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
aws_instance.my-ec2 (remote-exec): Access Jenkins Server here --> http://100.24.54.83:8080
aws_instance.my-ec2 (remote-exec): Jenkins Initial Password: 315c46806a9f4eca82f27a2239647548
aws_instance.my-ec2 (remote-exec): Access SonarQube Server here --> http://100.24.54.83:9000
aws_instance.my-ec2 (remote-exec): SonarQube Username & Password: admin
aws_instance.my-ec2 (remote-exec): Access Grafana Server here --> http://100.24.54.83:3000
aws_instance.my-ec2 (remote-exec): Access Prometheus Server here --> http://100.24.54.83:9090
aws_instance.my-ec2: Creation complete after 2m32s [id=i-0858872b9a500cf8e]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

PRIVATE-IP = "172.31.88.75"
PUBLIC-IP = "100.24.54.83"
SERVER-SSH-ACCESS = "ubuntu@100.24.54.83"
PS>
```



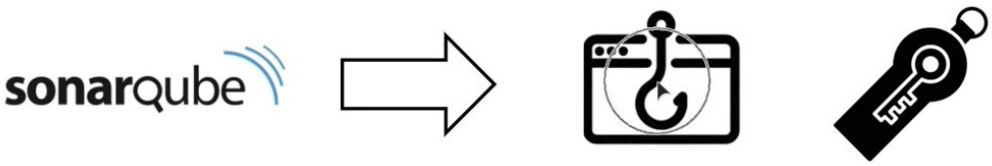

Getting Started
Getting Started
Folders
OWASP Markup Formatter
Build Timeout
Credentials Binding
Timestamper
Workspace Cleanup
Ant
Gradle
Pipeline
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
Pipeline Graph View
Git
SSH Build Agents
Matrix Authorization Strategy
PAM Authentication
LDAP
Email Extension
Mailer
Dark Theme
** Ionicons API
Folders
OWASP Markup Formatter
** ASM API
** JSON Path API
** Structs
** Pipeline: Step API
** Token Macro
Build Timeout
** bouncycastle API


Create First Admin User
Username
Password
Confirm password
Full name


Getting Started
Instance Configuration
Jenkins URL:
http://100.24.54.83:8080/
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.
The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.



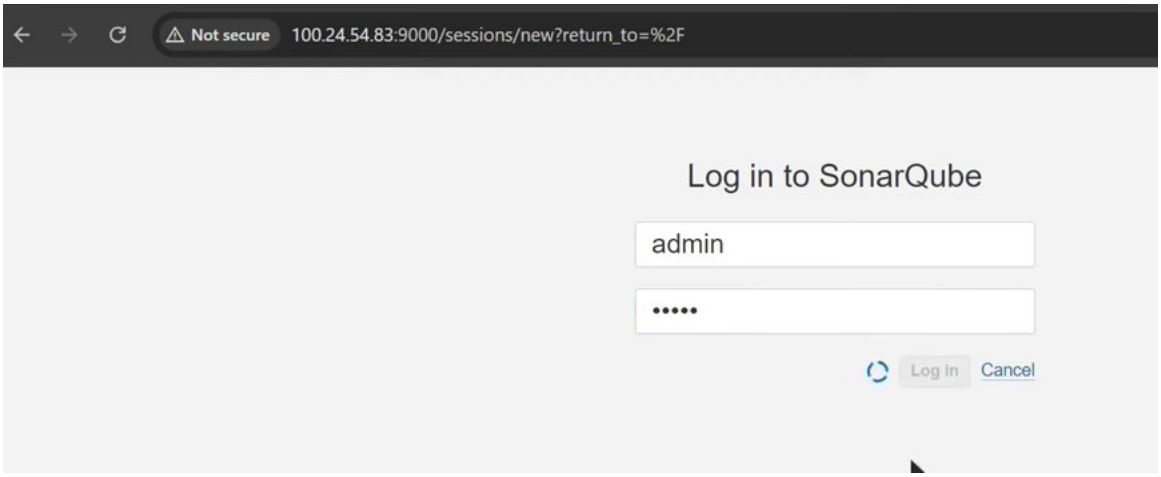
CONFIGURE SONARQUBE



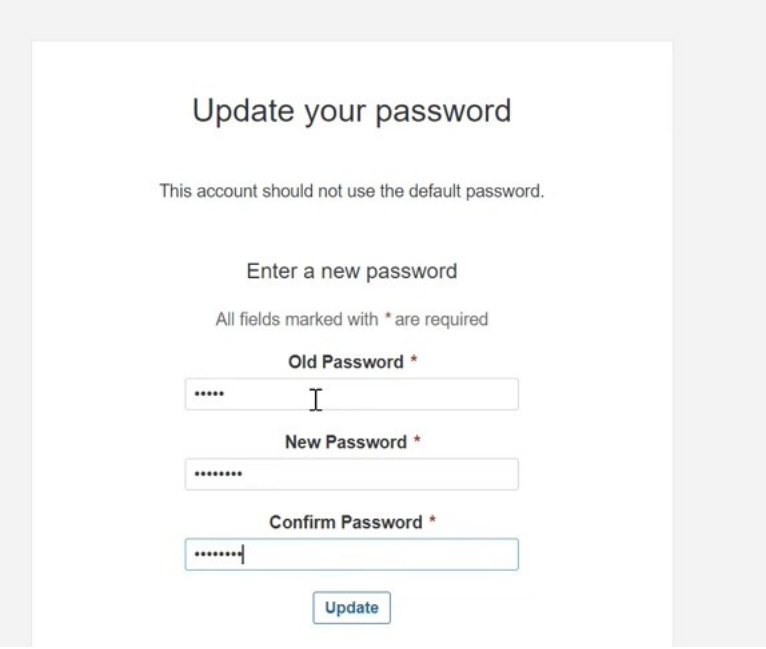
 # setting uo Webhook - generating the Token

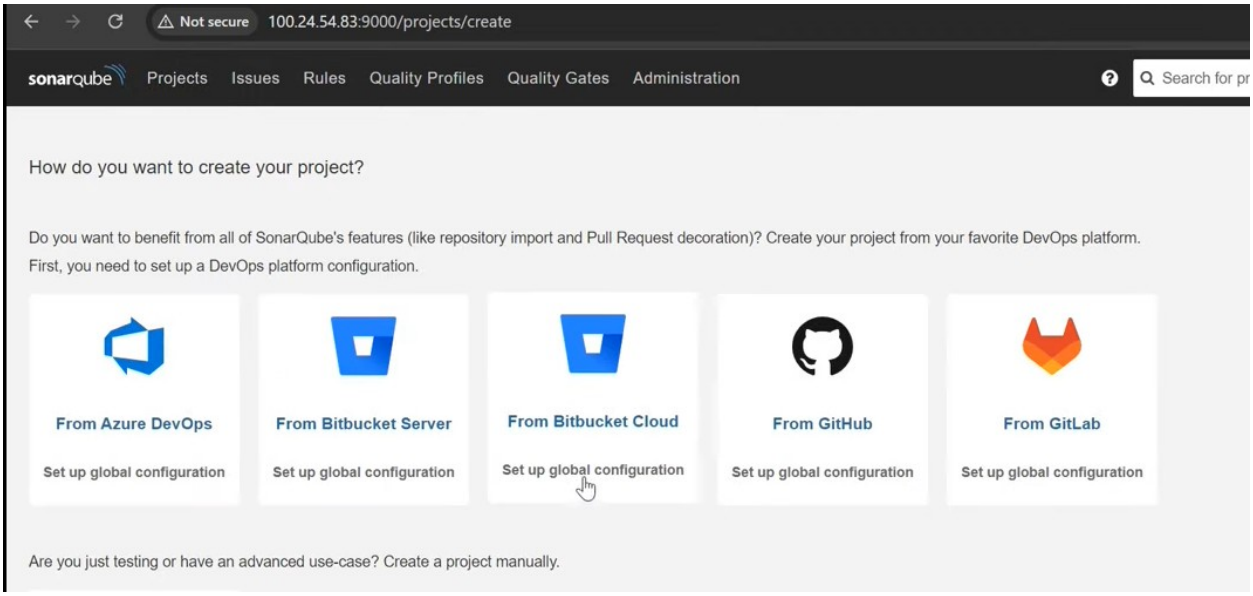
 # SonarCube URL

```
aws_instance.my-ec2 (remote-exec): Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
aws_instance.my-ec2 (remote-exec): Access Jenkins Server here --> http://100.24.54.83:8080
aws_instance.my-ec2 (remote-exec): Jenkins Initial Password: 315c46806a9f4eca82f27a2239647548
aws_instance.my-ec2 (remote-exec): Access SonarQube Server here --> http://100.24.54.83:9000
aws_instance.my-ec2 (remote-exec): SonarQube Username & Password: admin
aws_instance.my-ec2 (remote-exec): Access Grafana Server here --> http://100.24.54.83:3000
aws_instance.my-ec2 (remote-exec): Access Prometheus Server here --> http://100.24.54.83:9090
aws_instance.my-ec2: Creation complete after 2m32s [id=i-0858872b9a500cf8e]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

PRIVATE-IP = "172.31.88.75"
PUBLIC-IP = "100.24.54.83"
SERVER-SSH-ACCESS = "ubuntu@100.24.54.83"
PS>
```



 # change the pswd :

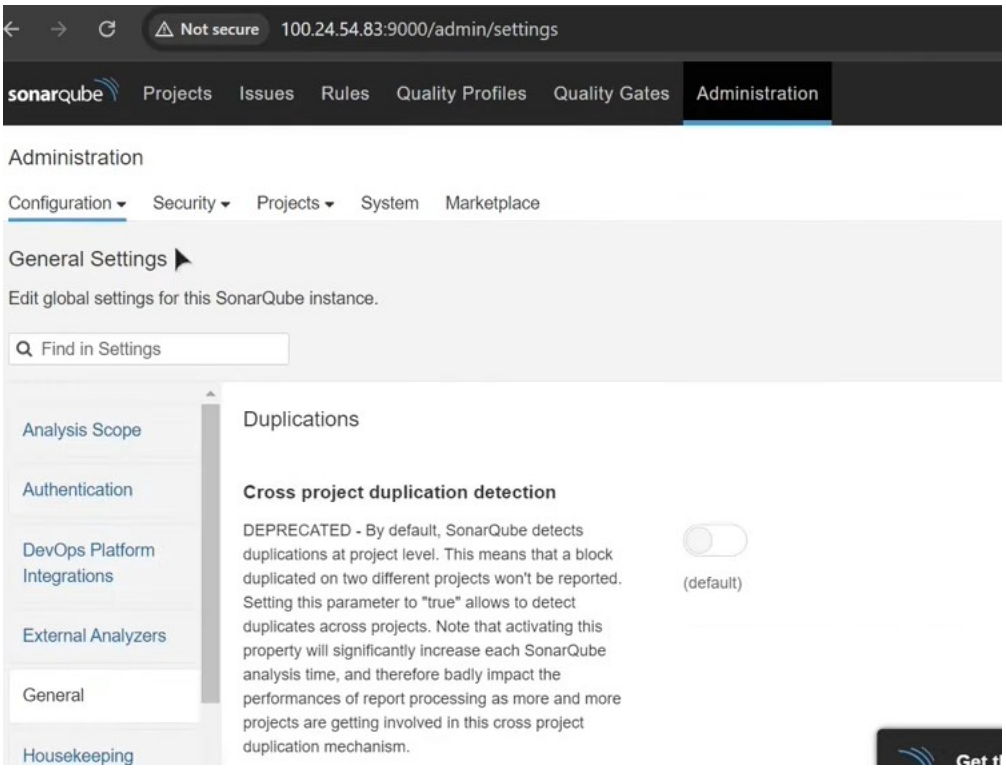




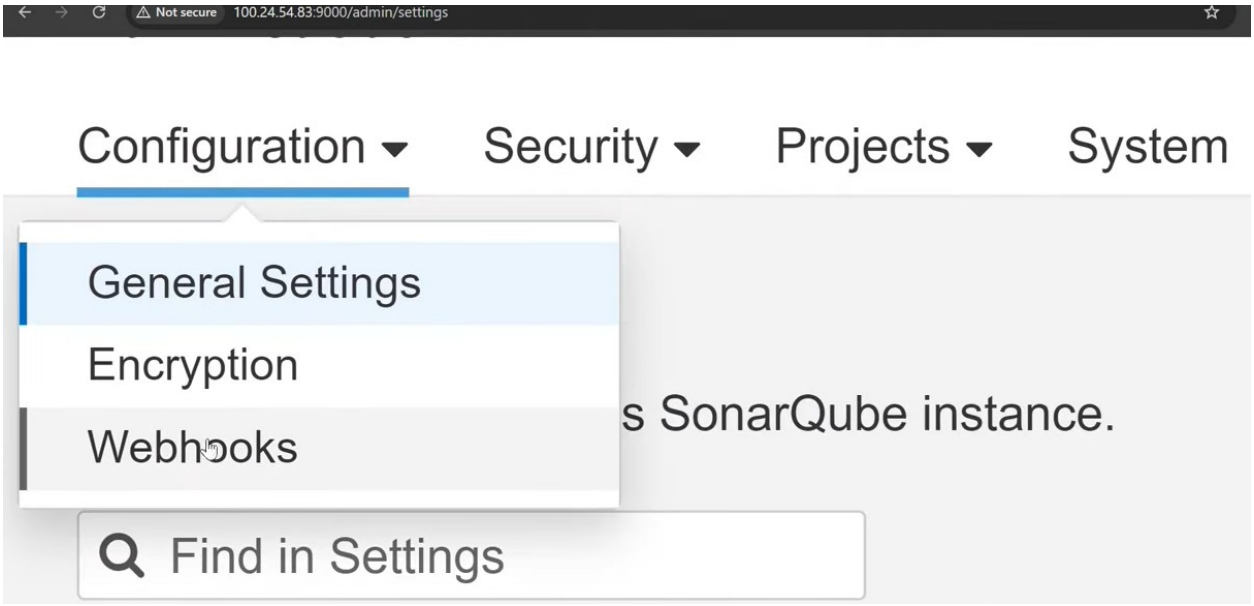
WEBHOOK FOR SONARQUBE

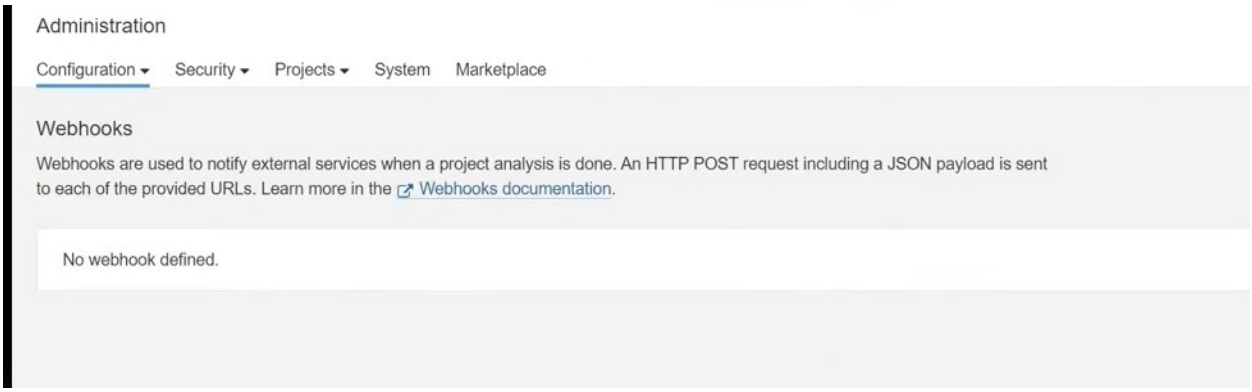
Send Notifications to Jenkins – WebHook is used

 # go to Administration

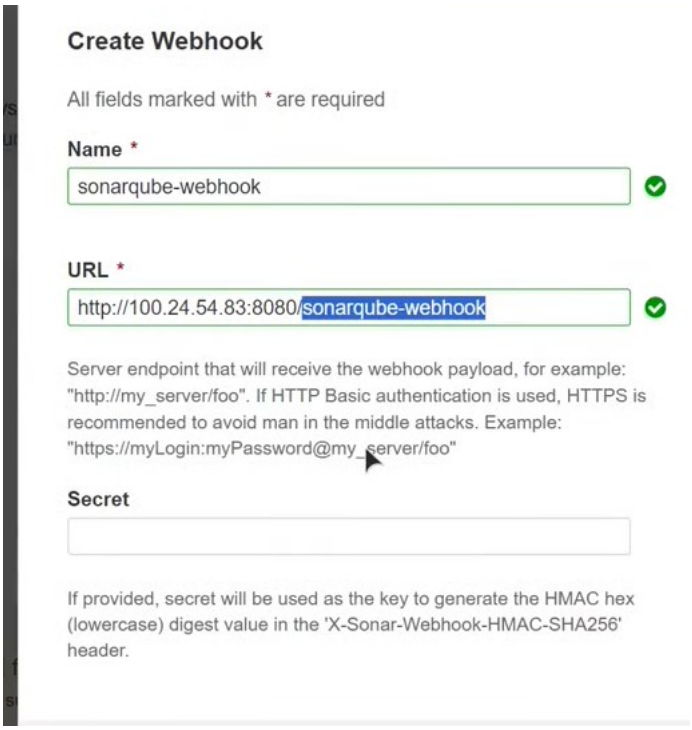

Not secure 100.24.54.83:9000/admin/settings
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Administration
Configuration
Security
Projects
System
Marketplace
General Settings
Edit global settings for this SonarQube instance.
Find in Settings
Analysis Scope
Authentication
DevOps Platform Integrations
External Analyzers
General
Housekeeping
Duplications
Cross project duplication detection
DEPRECATED - By default, SonarQube detects duplications at project level. This means that a block duplicated on two different projects won't be reported. Setting this parameter to "true" allows to detect duplicates across projects. Note that activating this property will significantly increase each SonarQube analysis time, and therefore badly impact the performances of report processing as more and more projects are getting involved in this cross project duplication mechanism.
(default)

Click on

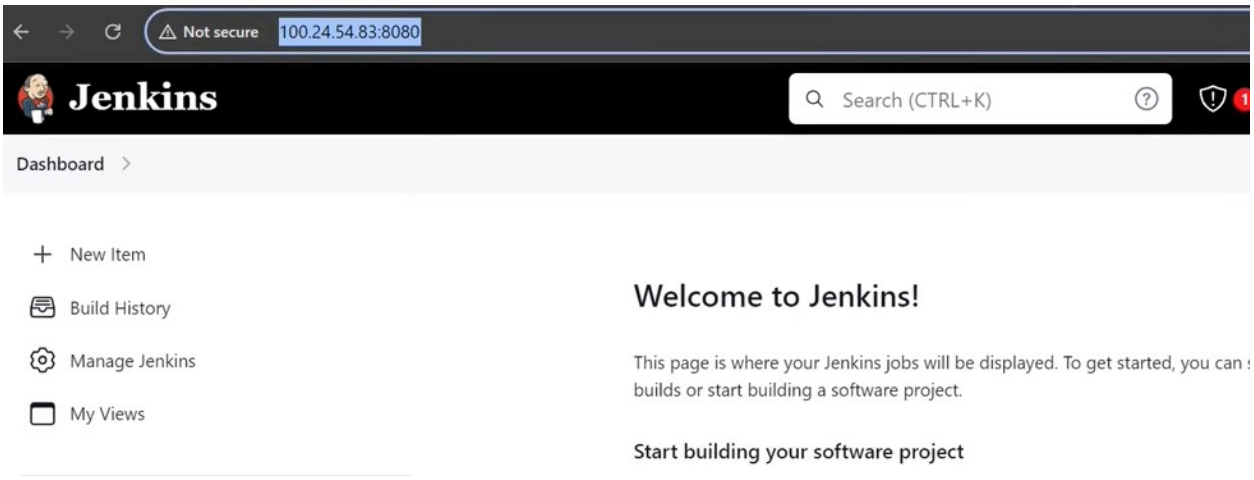

Not secure 100.24.54.83:9000/admin/settings
Configuration
Security
Projects
System
General Settings
Encryption
Webhooks
s SonarQube instance.
Find in Settings



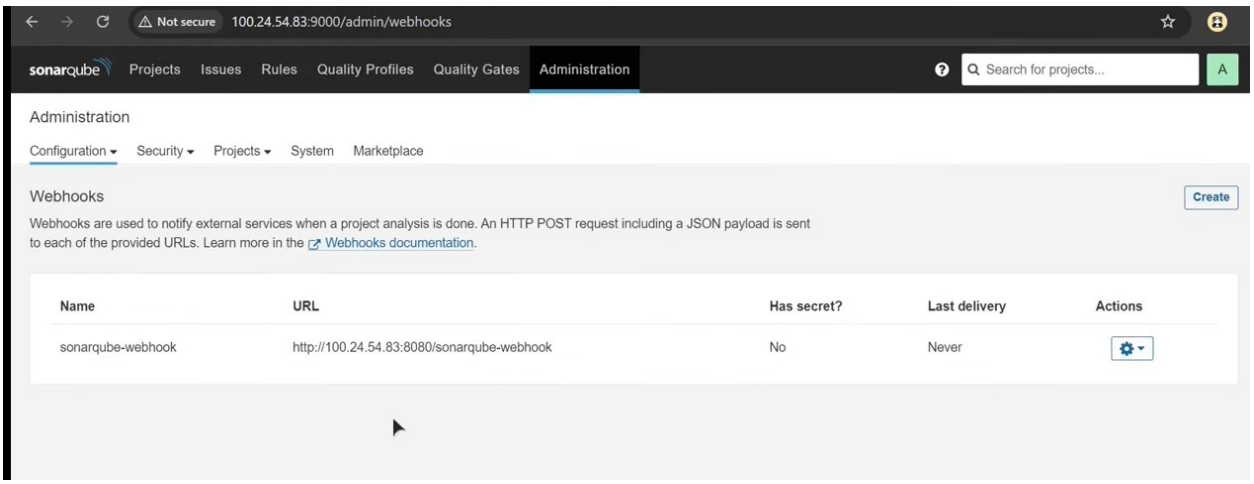
 # Create



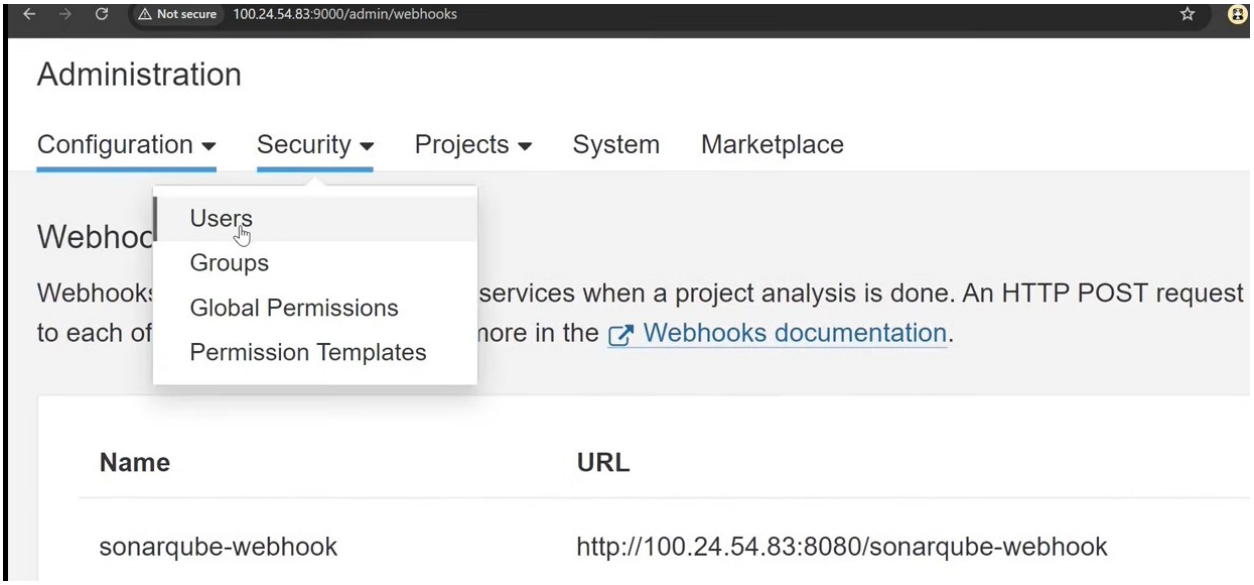
 # add url of Jenkins ..

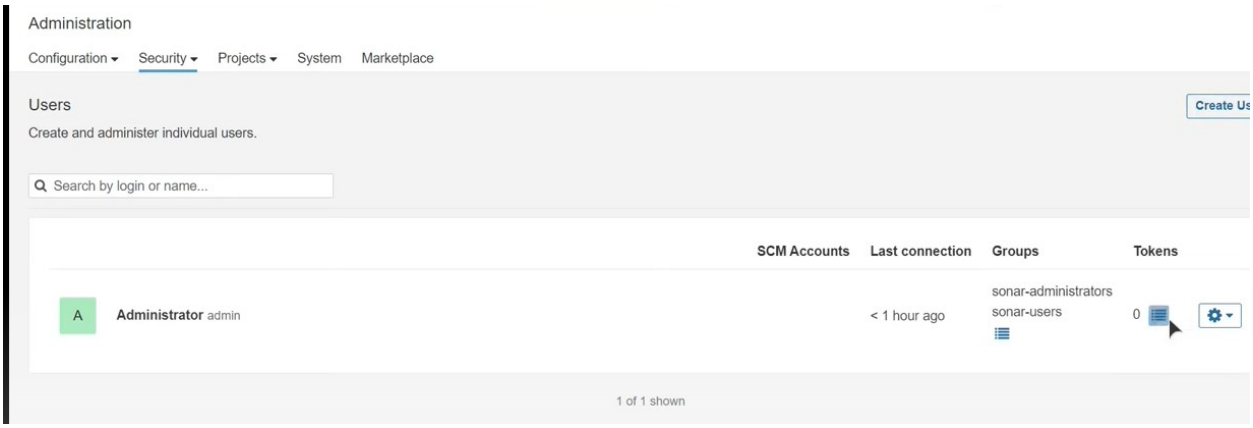


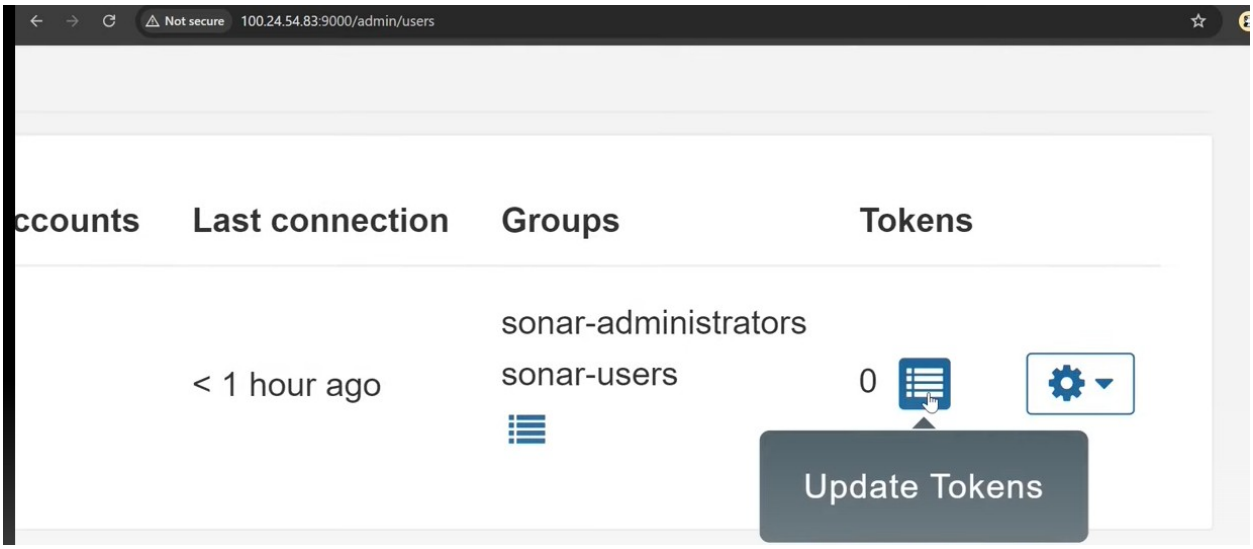
 # Webhook Created



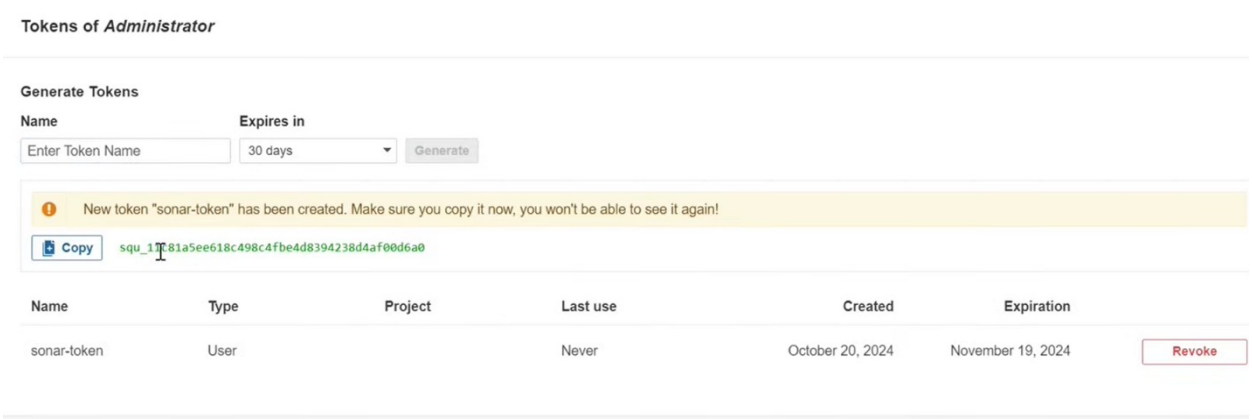
 # Token



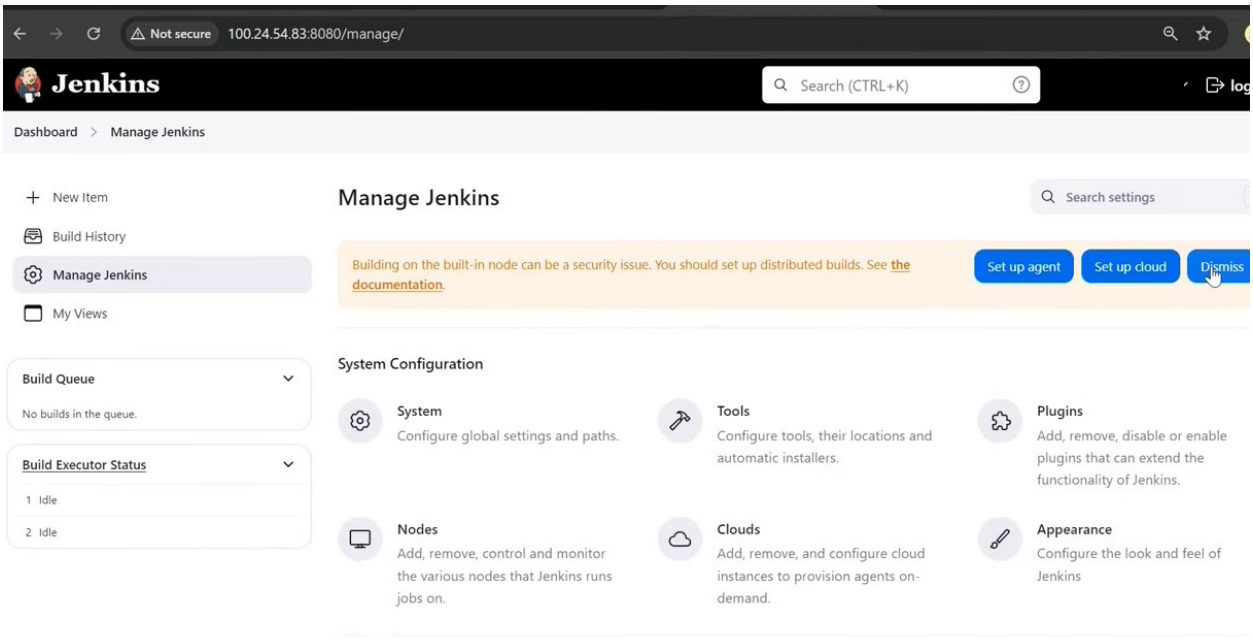




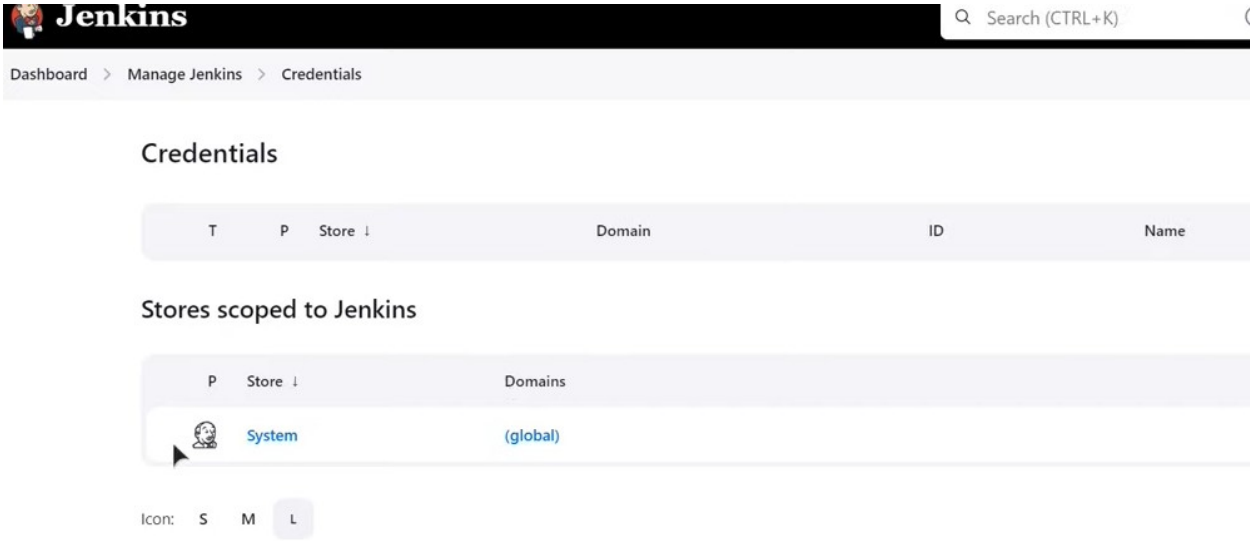
 # Token Generated - that will be saved in Jenkins - Credential Section (two way Commuication Sonar <-> Jenkins)

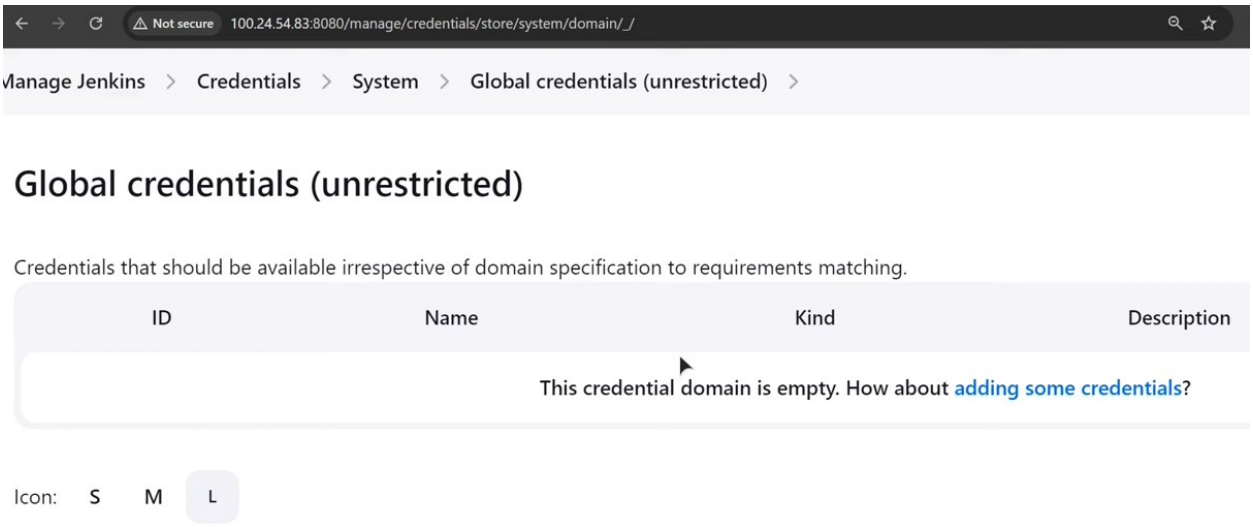


 # cp – sonar token .. Go to Jenkins -manage jenkins – credentials

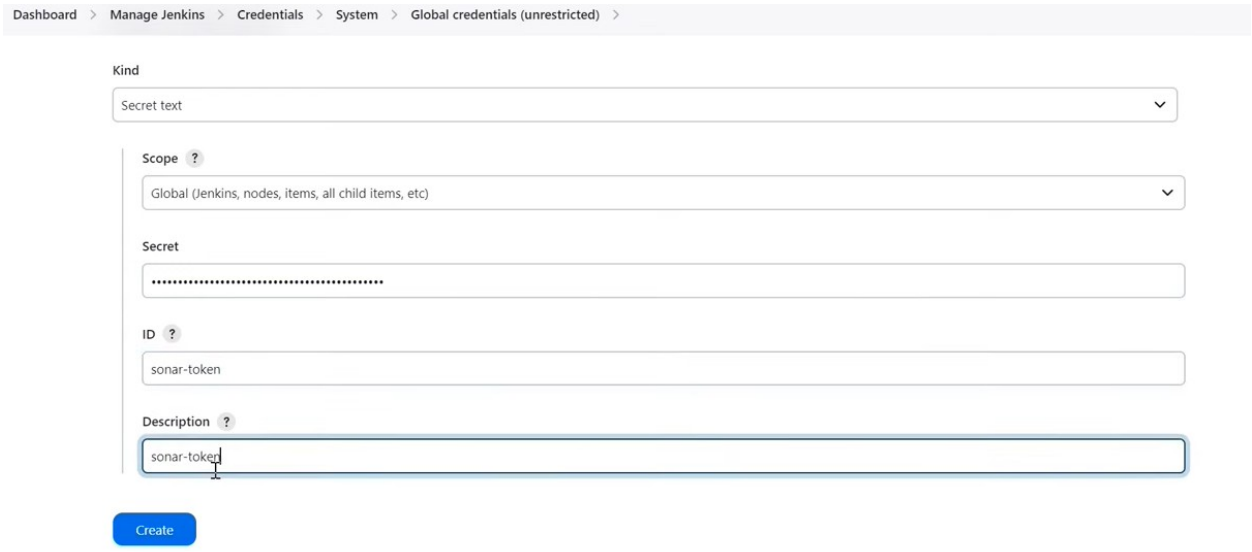


 # Credentials



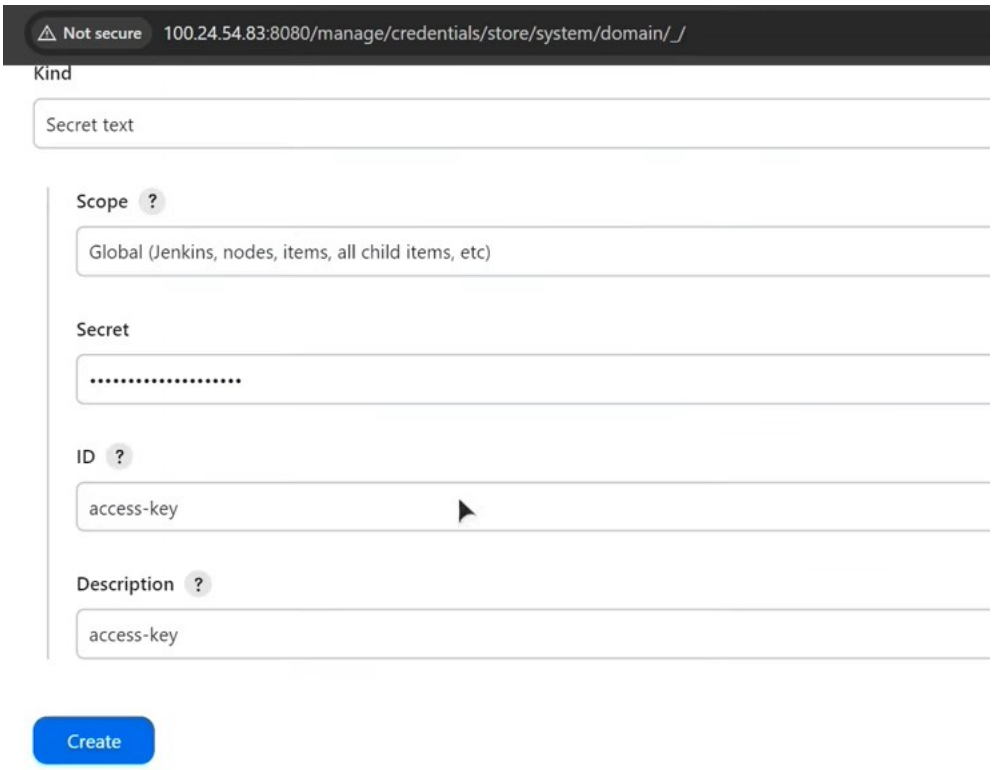


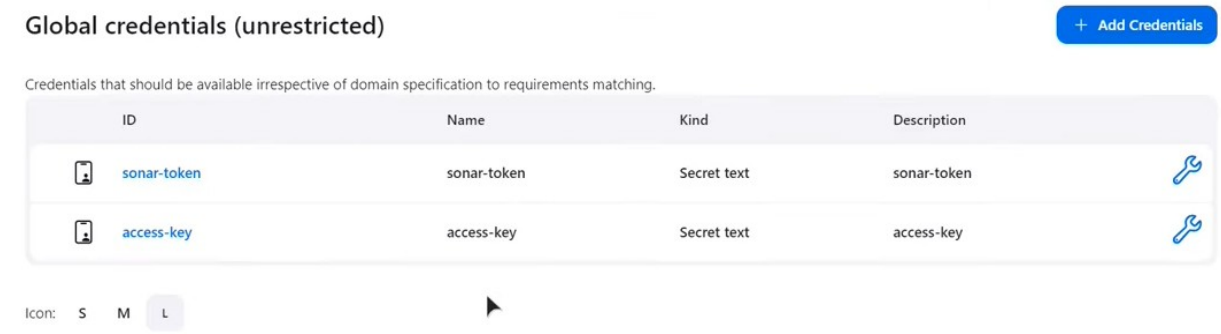
 # Add Credentials

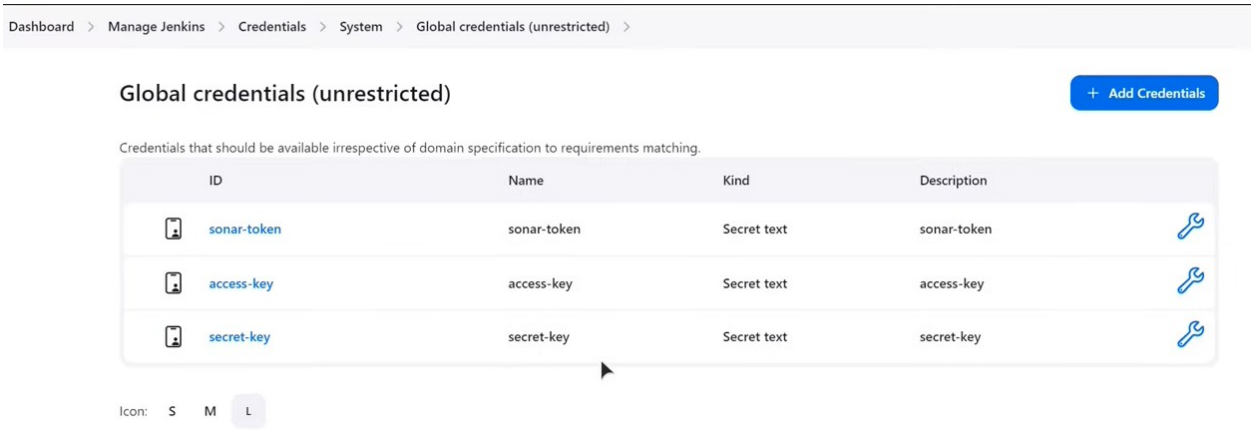


 # I need to Add both AWS Access key + secrete Key to run AWS ECR + EKS Cmds from Jenkins

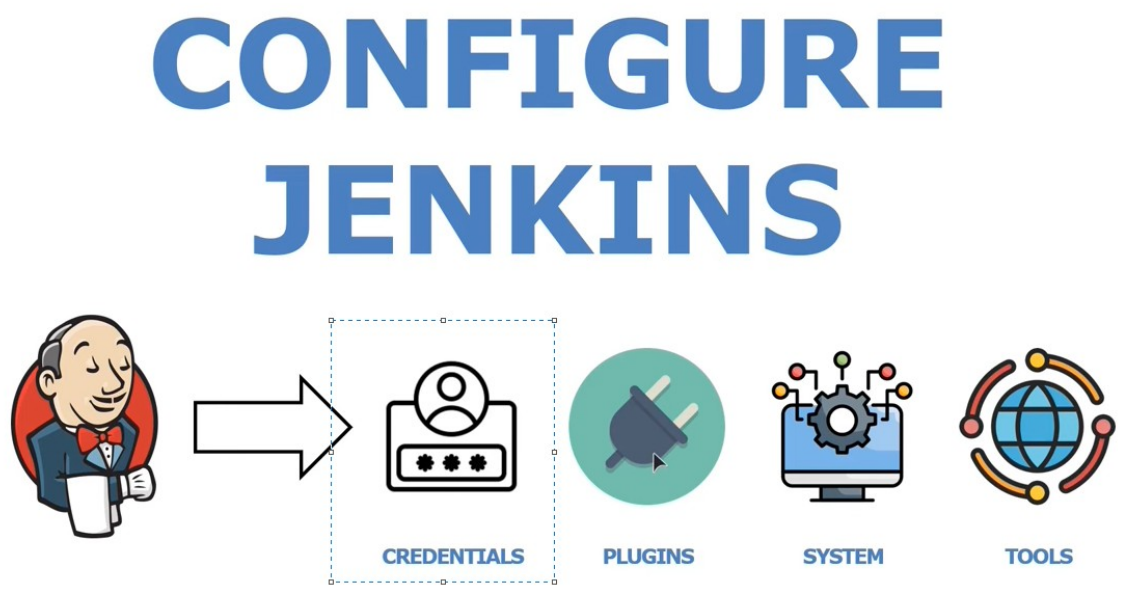
```
# Access key:
AKIAUQCLMSLD5NUMX5FZ

# Secret key:
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
```


Not secure
100.24.54.83:8080/manage/credentials/store/system/domain/_/
Kind
Secret text
Scope ?
Global (Jenkins, nodes, items, all child items, etc)
Secret
ID ?
access-key
Description ?
access-key
Create

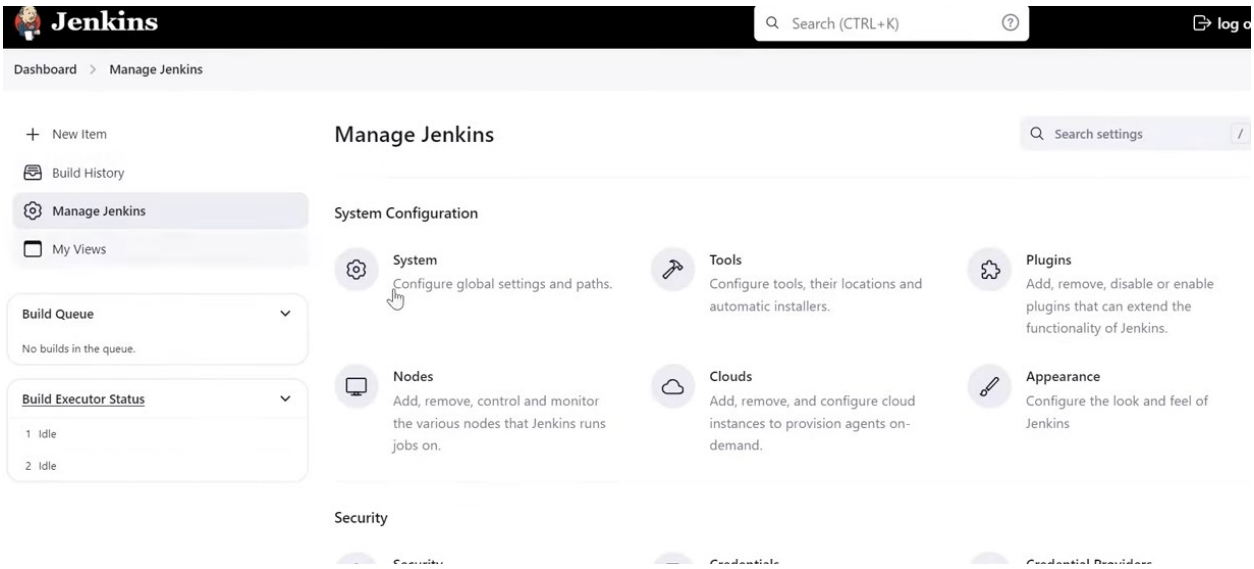

Global credentials (unrestricted)
+ Add Credentials
Credentials that should be available irrespective of domain specification to requirements matching.
ID
Name
Kind
Description
sonar-token
sonar-token
Secret text
sonar-token
access-key
access-key
Secret text
access-key
Icon: S M L


Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >
Global credentials (unrestricted)
+ Add Credentials
Credentials that should be available irrespective of domain specification to requirements matching.
ID
Name
Kind
Description
sonar-token
sonar-token
Secret text
sonar-token
access-key
access-key
Secret text
access-key
secret-key
secret-key
Secret text
secret-key
Icon: S M L

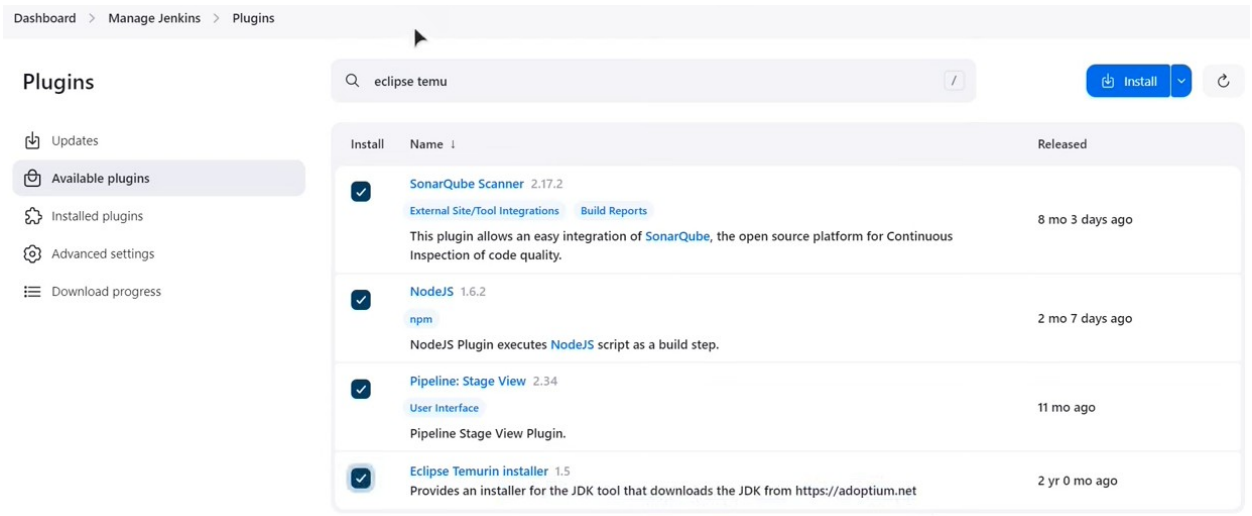
 # Configure Jenkins



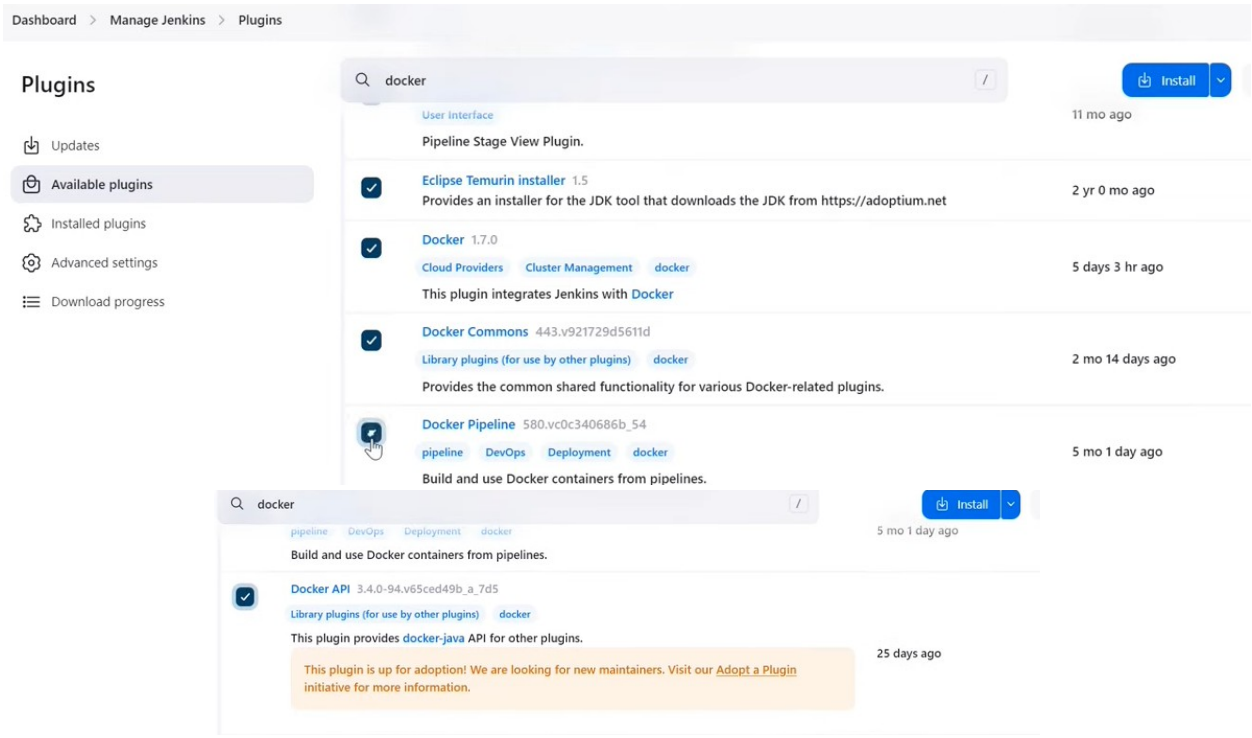
 # Credentials part is done .. heading to other 3 .. sections



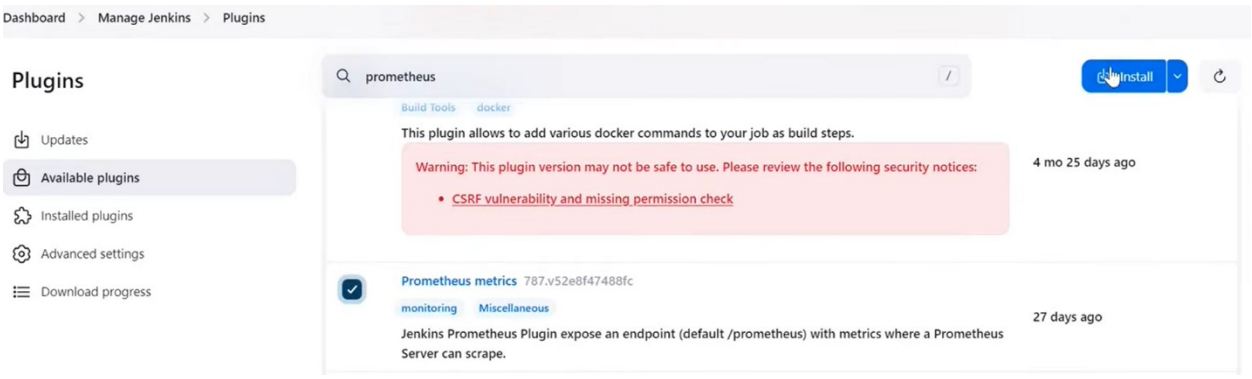
 # Click on Plugins

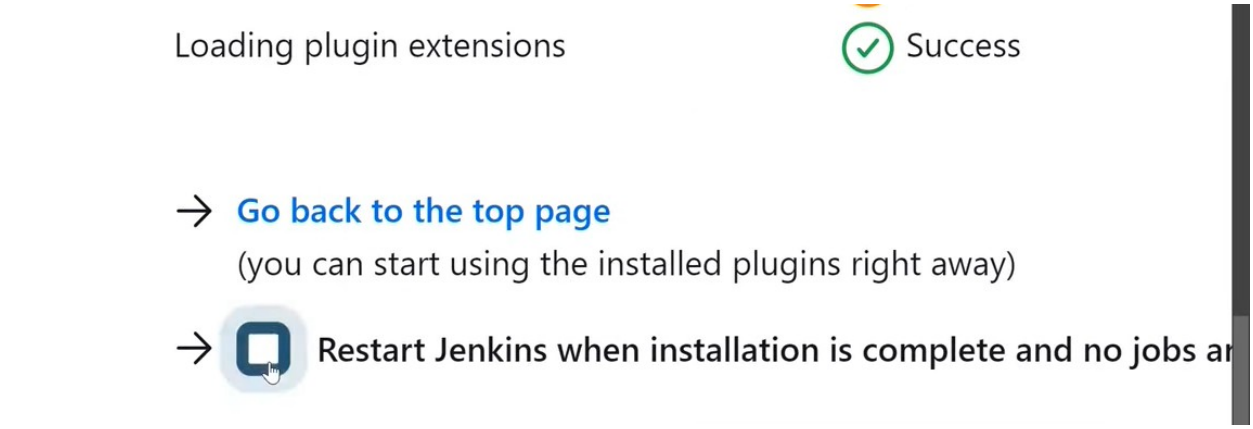


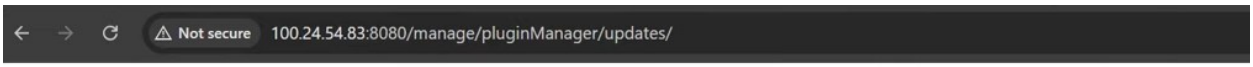
 # last Docker

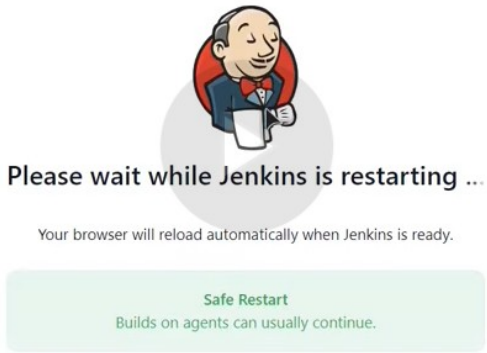


#Prometheusp-metric



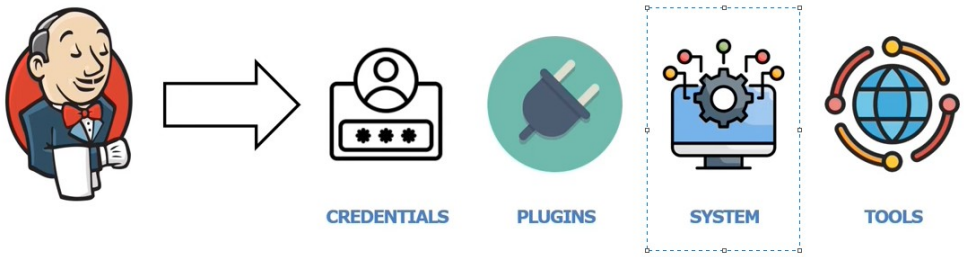






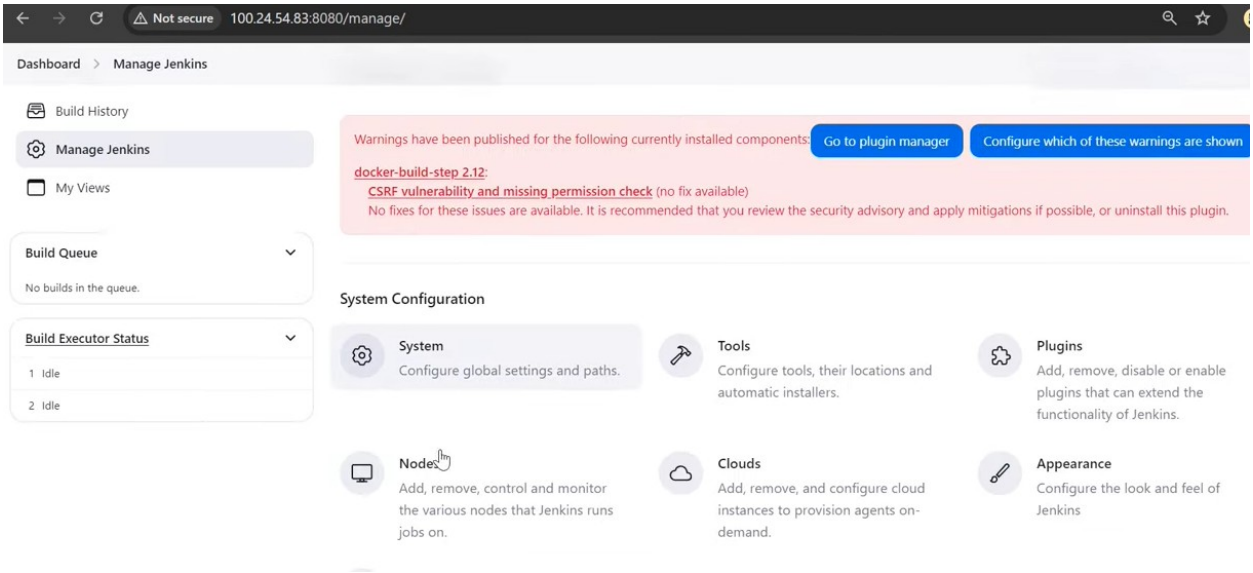
 # 3rd Part- System

CONFIGURE JENKINS

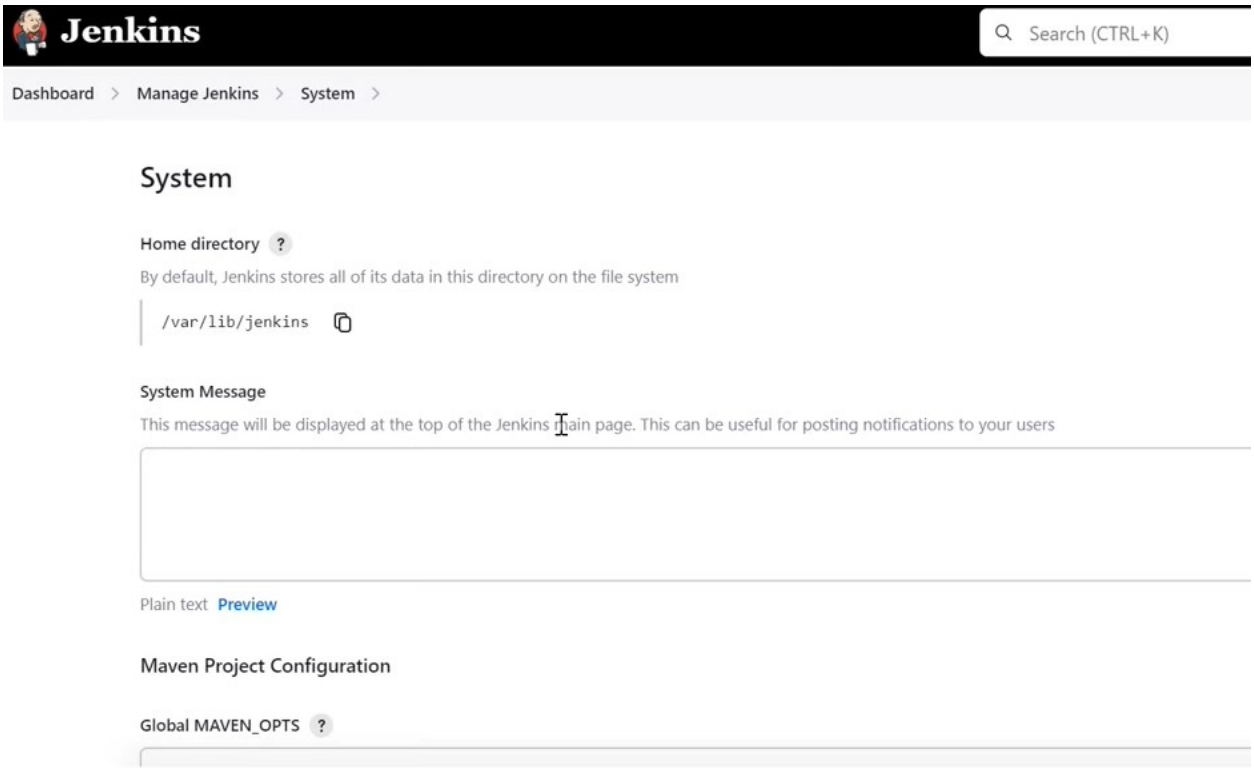


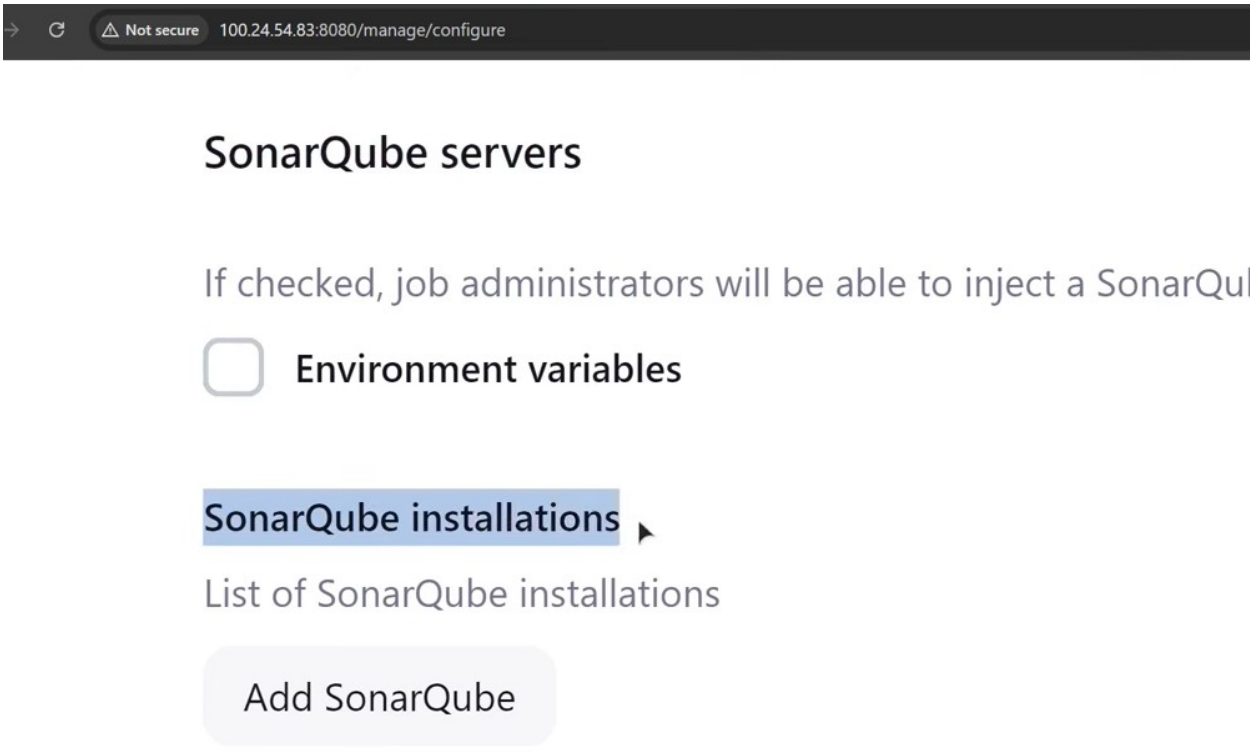
 # in System .. setting up SonarQube in jenkins Server..

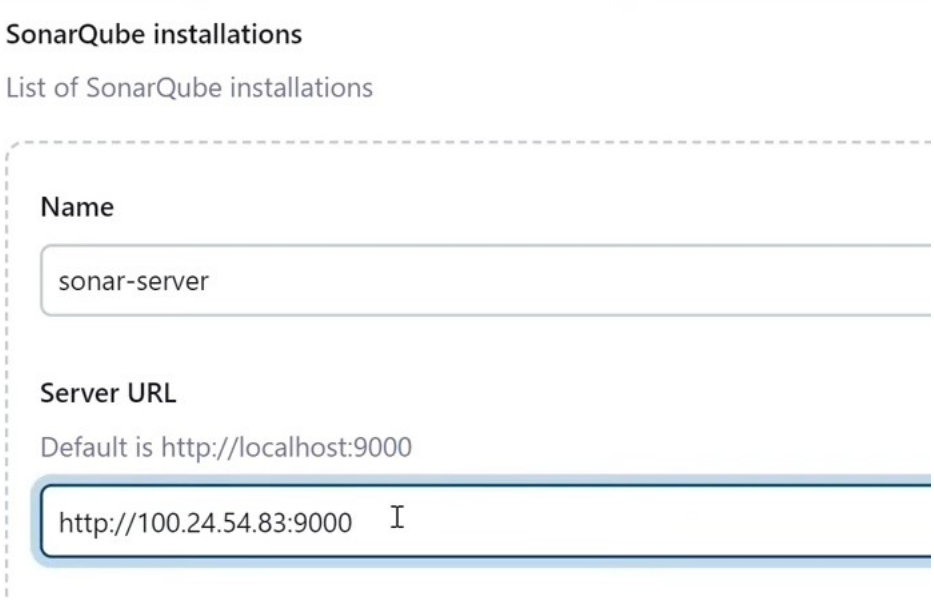
 # Manage Jenkins - Click System – to setup sonarQube server



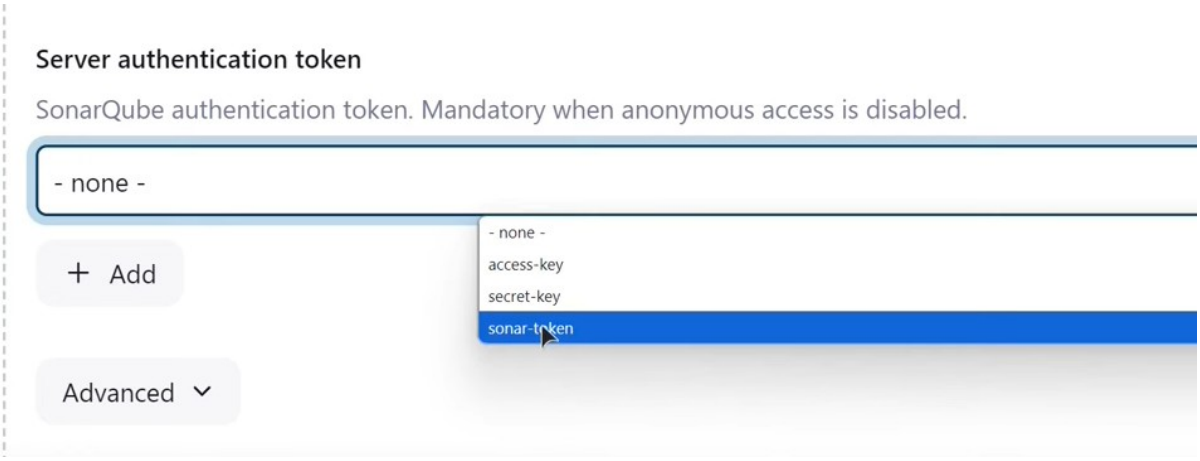
 # integrate Jekins with SonarQube


Jenkins
Search (CTRL+K)
Dashboard > Manage Jenkins > System >
System
Home directory
By default, Jenkins stores all of its data in this directory on the file system
/var/lib/jenkins
System Message
This message will be displayed at the top of the Jenkins main page. This can be useful for posting notifications to your users
Plain text Preview
Maven Project Configuration
Global MAVEN_OPTS


Not secure
100.24.54.83:8080/manage/configure
SonarQube servers
If checked, job administrators will be able to inject a SonarQu
Environment variables
SonarQube installations
List of SonarQube installations
Add SonarQube



 # choose Sonar Token

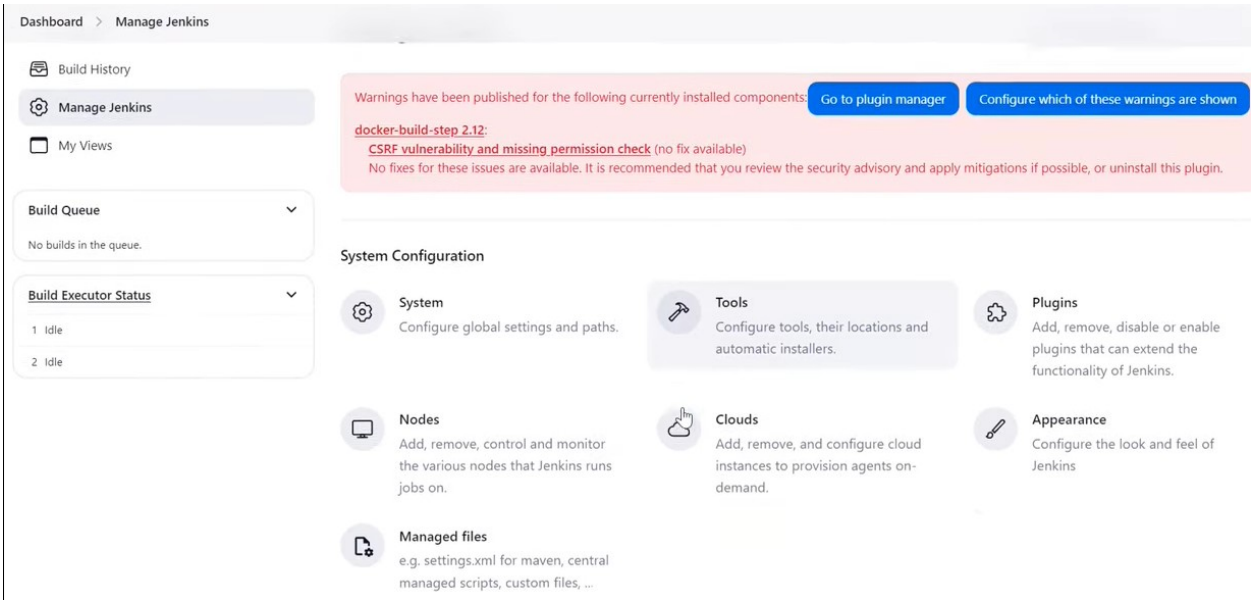


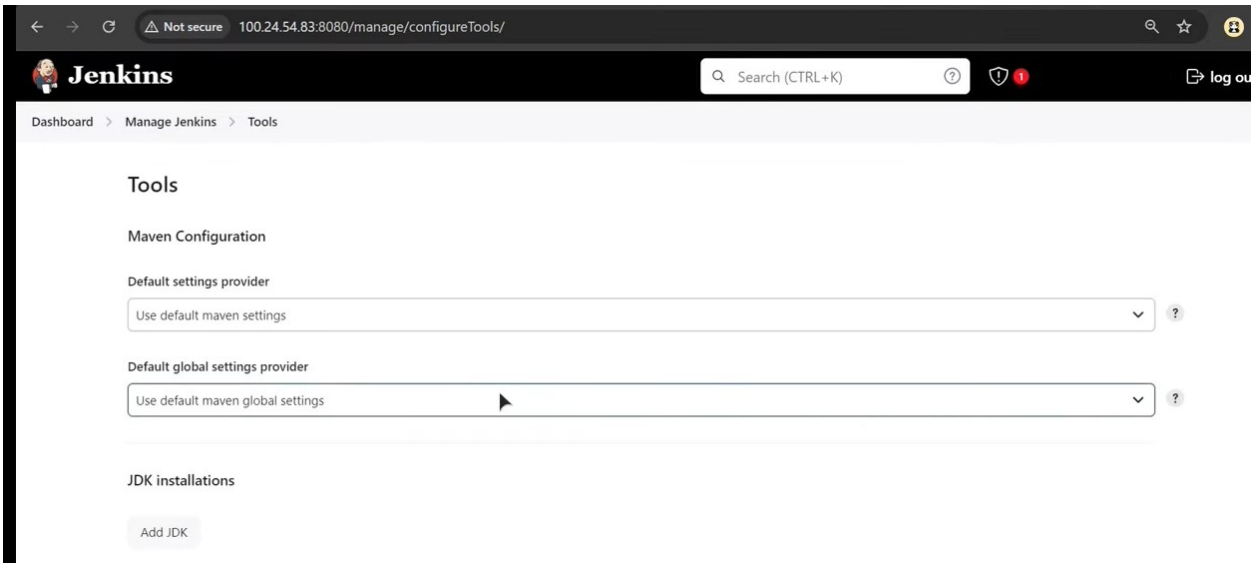
 # Save

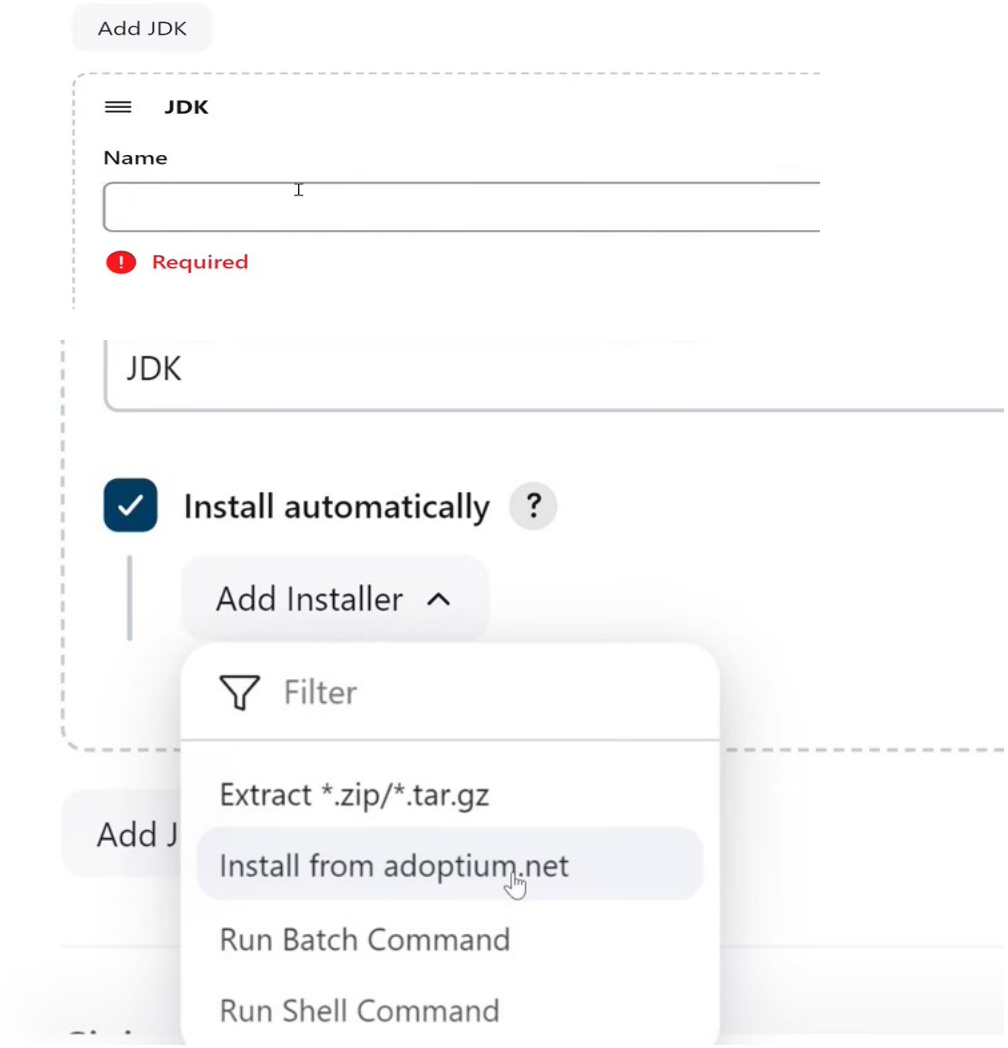
 # 4th Part Configure the tools

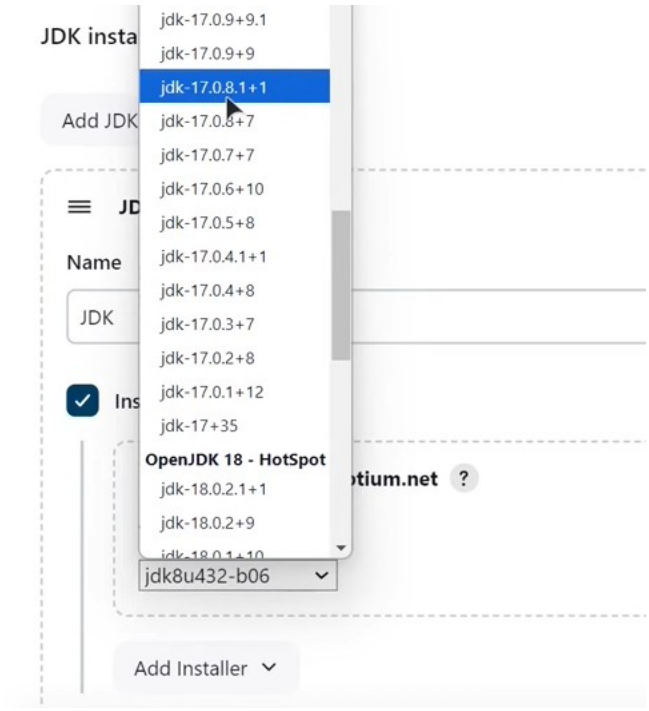
JDK --> 17.0.8
SONAR-SCANNER --> latest
NodeJS --> 16.20.0
Docker --> latest

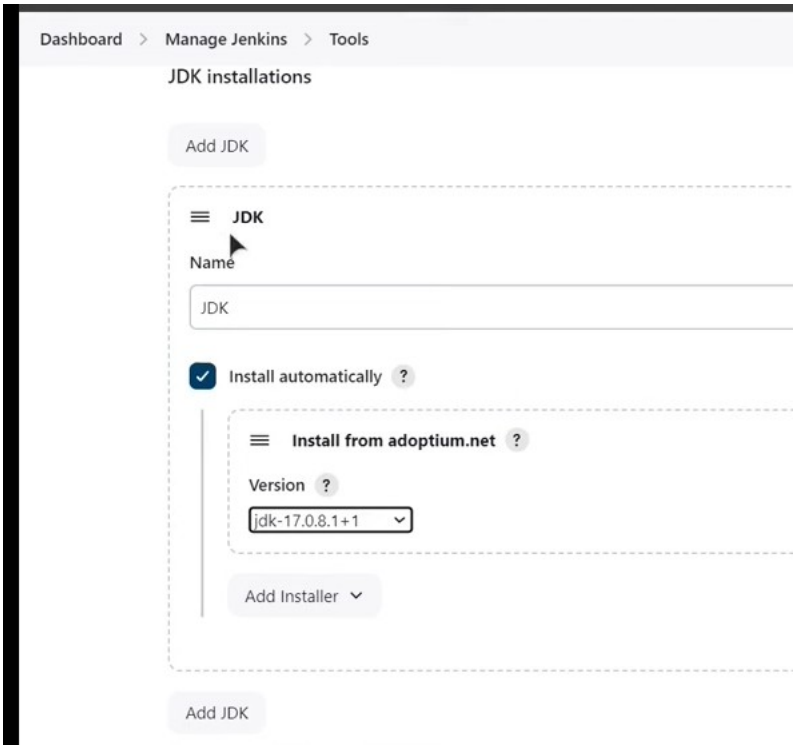
 # Click on tools in manage Jenkins

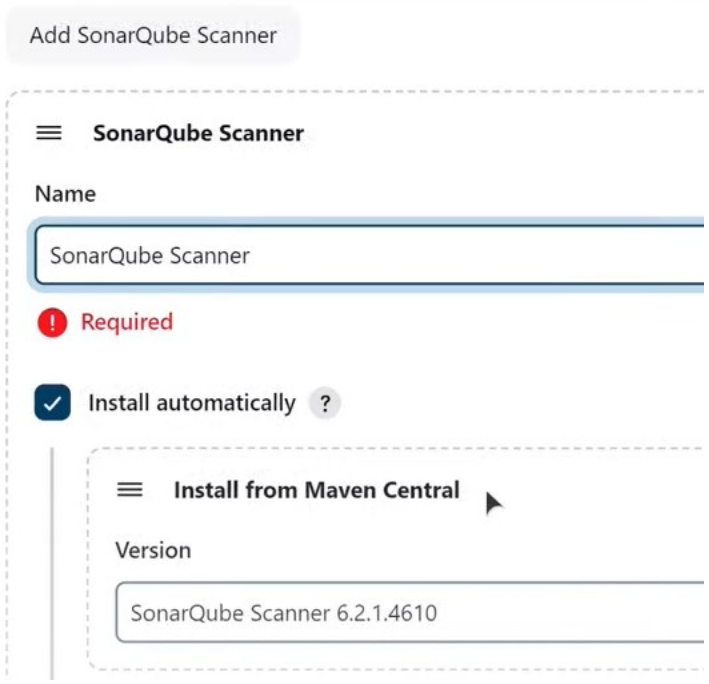


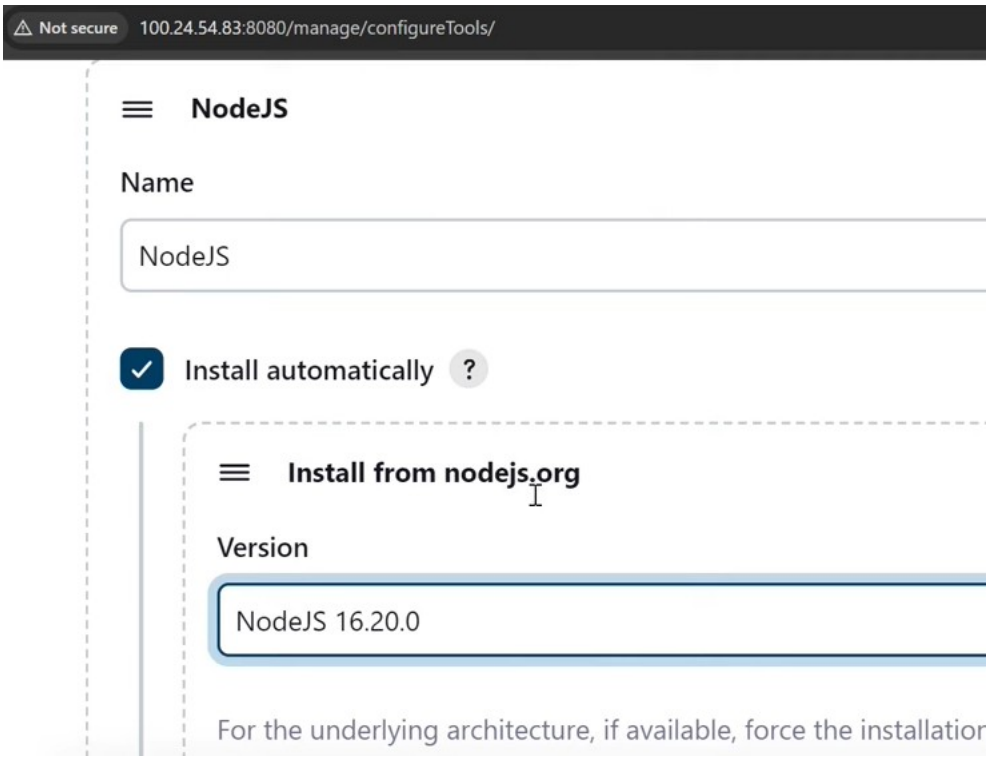


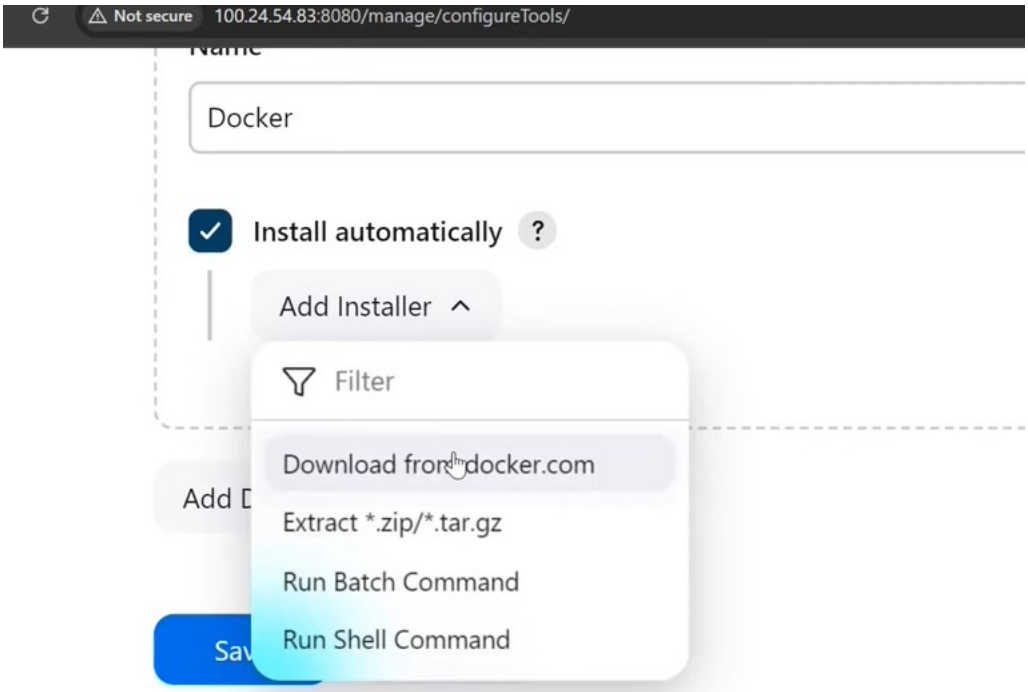

Add JDK
JDK
Name
Required
JDK
Install automatically
Add Installer
Filter
Extract *.zip/*.tar.gz
Install from adoptium.net
Run Batch Command
Run Shell Command


JDK insta
Add JDK
Name
JDK
jdk-17.0.9+9.1
jdk-17.0.9+9
jdk-17.0.8.1+1
jdk-17.0.8+7
jdk-17.0.7+7
jdk-17.0.6+10
jdk-17.0.5+8
jdk-17.0.4.1+1
jdk-17.0.4+8
jdk-17.0.3+7
jdk-17.0.2+8
jdk-17.0.1+12
jdk-17+35
OpenJDK 18 - HotSpot
jdk-18.0.2.1+1
jdk-18.0.2+9
jdk8u432-b06
Add Installer

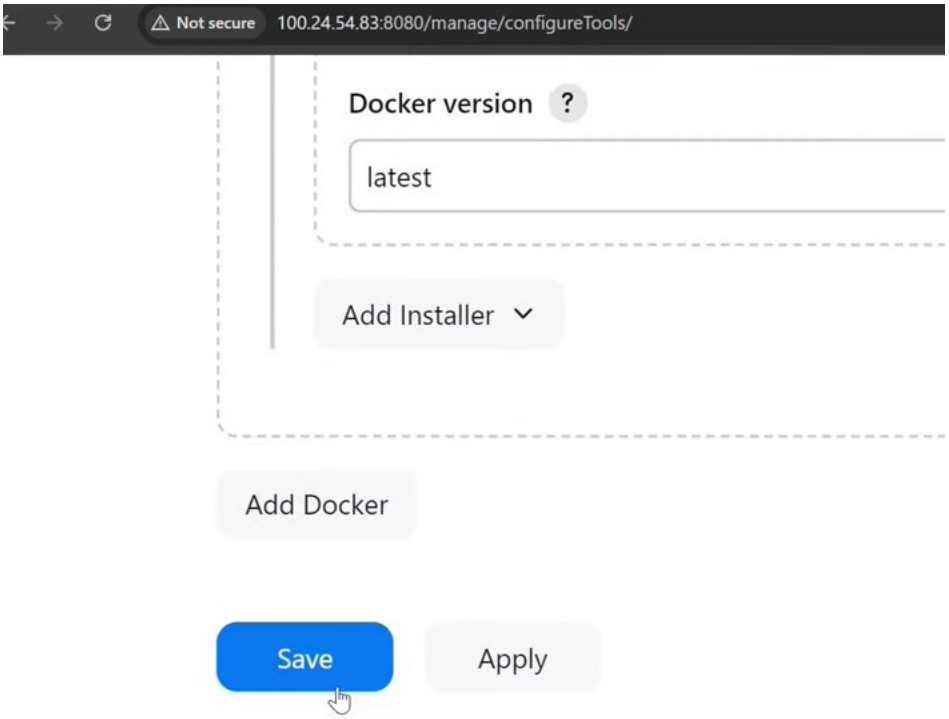

Dashboard
Manage Jenkins
Tools
JDK installations
Add JDK
JDK
Name
JDK
Install automatically
Install from adoptium.net
Version
jdk-17.0.8.1+1
Add Installer
Add JDK


Add SonarQube Scanner
SonarQube Scanner
Name
SonarQube Scanner
Required
Install automatically
Install from Maven Central
Version
SonarQube Scanner 6.2.1.4610


Not secure
100.24.54.83:8080/manage/configureTools/
NodeJS
Name
NodeJS
Install automatically
Install from nodejs.org
Version
NodeJS 16.20.0
For the underlying architecture, if available, force the installatior



 # save



 # Build Pipeline..

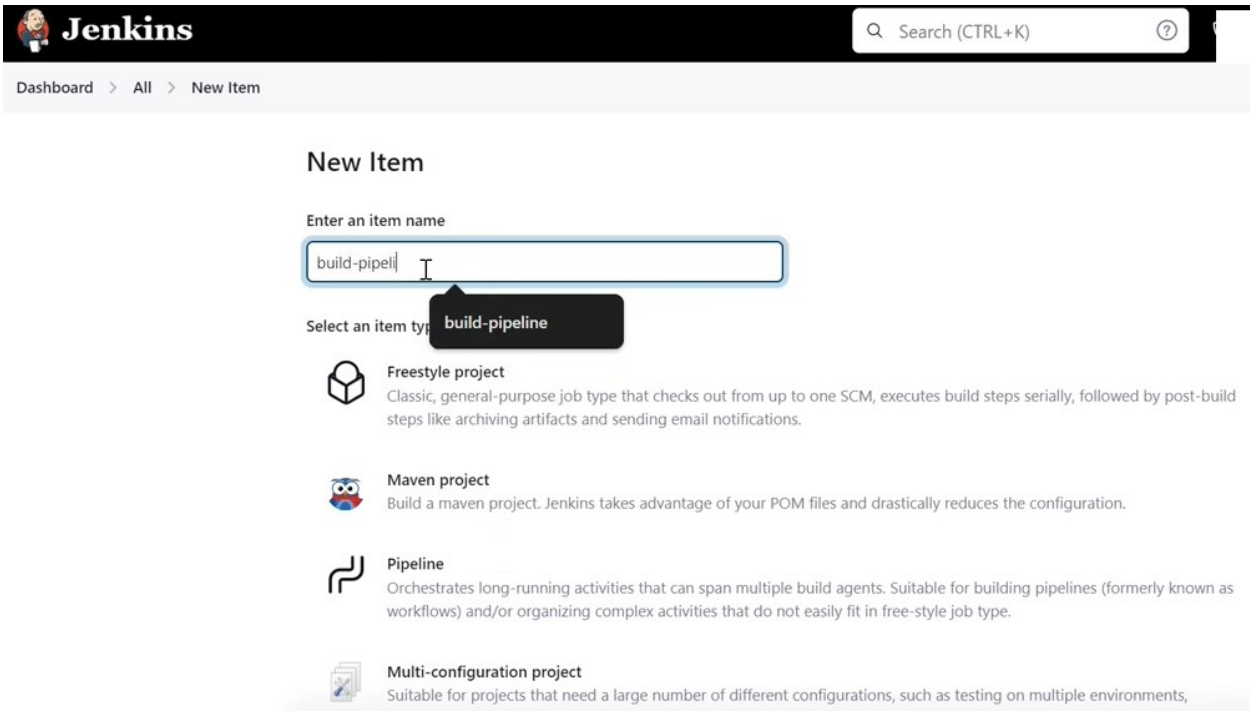
BUILD PIPELINE OVERVIEW

Pipeline Overview

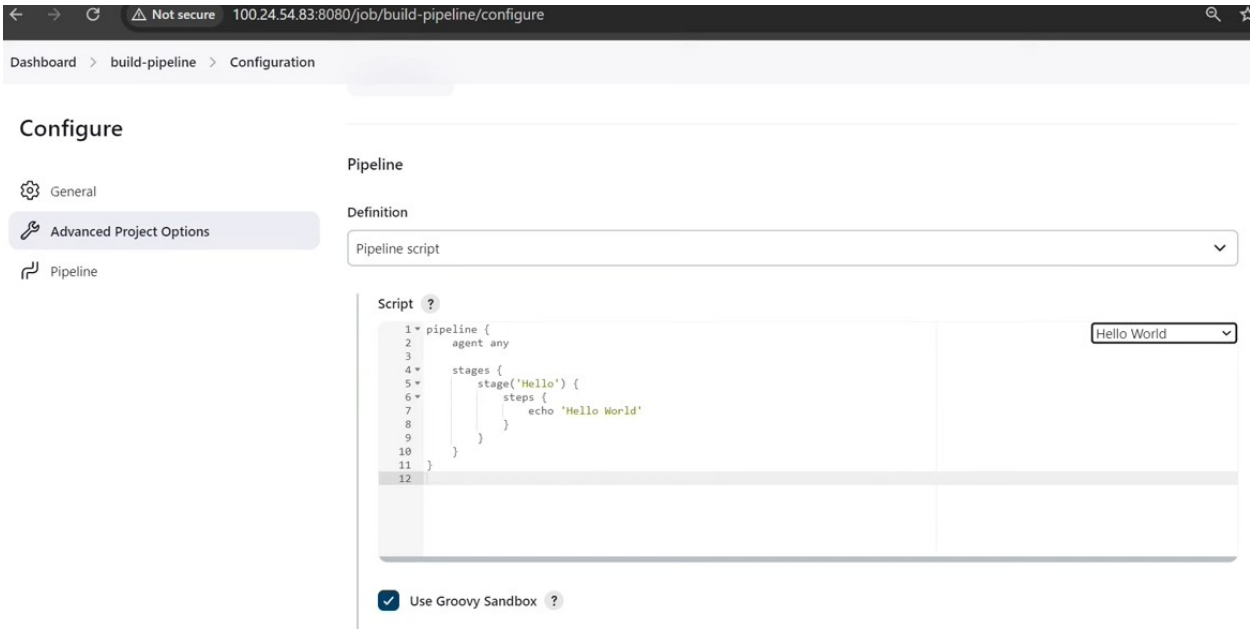
Pipeline Stages

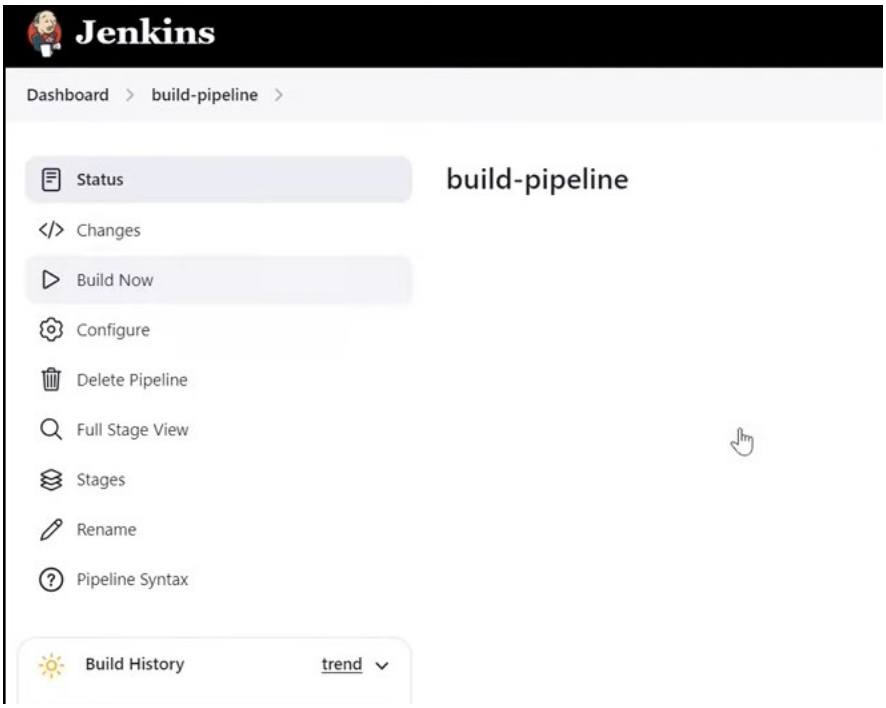
1. **Git Checkout**: Clones the source code from GitHub.
2. **SonarQube Analysis**: Performs static code analysis.
3. **Quality Gate**: Ensures code quality standards.
4. **Install NPM Dependencies**: Installs NodeJS packages.
5. **Trivy Security Scan**: Scans the project for vulnerabilities.
6. **Docker Build**: Builds a Docker image for the project.
7. **Push to AWS ECR**: Tags and pushes the Docker image to ECR.
8. **Image Cleanup**: Deletes images from the Jenkins server to save space.

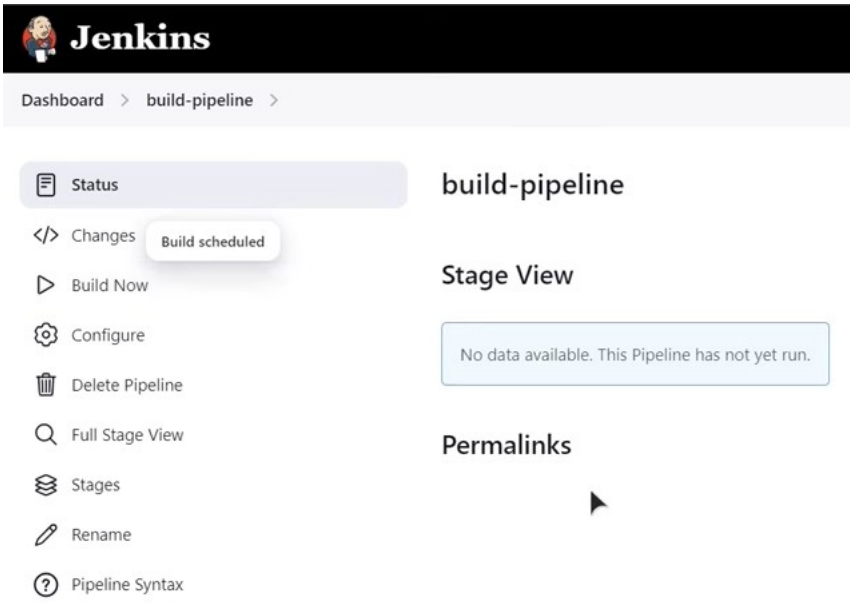
 # Create – pipeline – Test it sample one

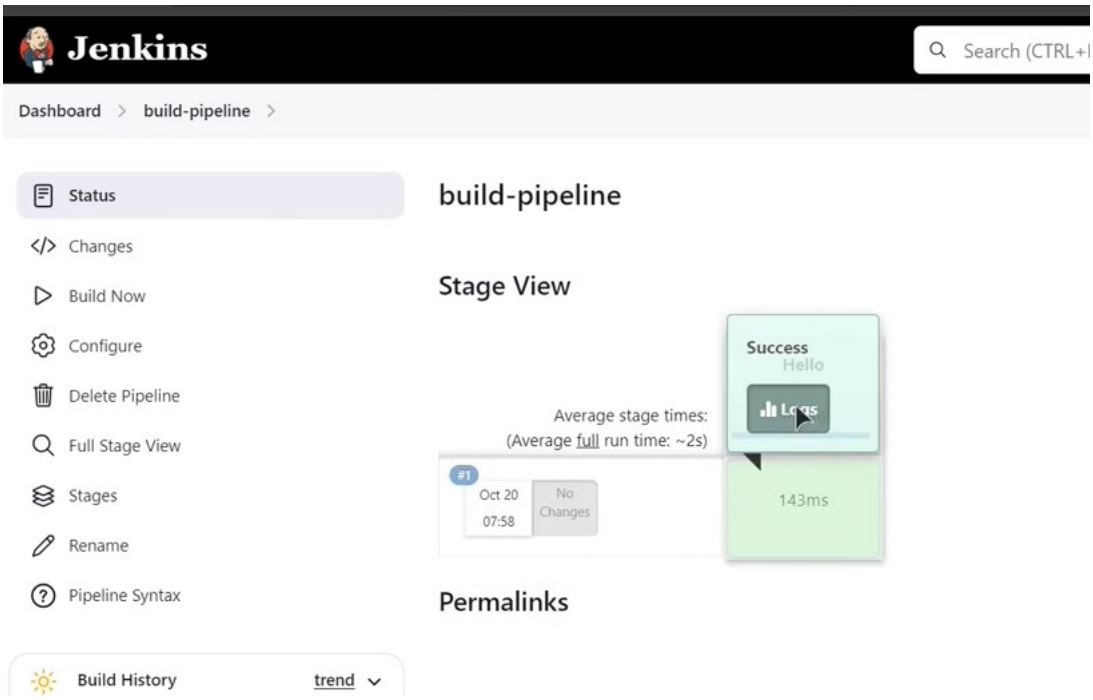


 # Generate the script- sample

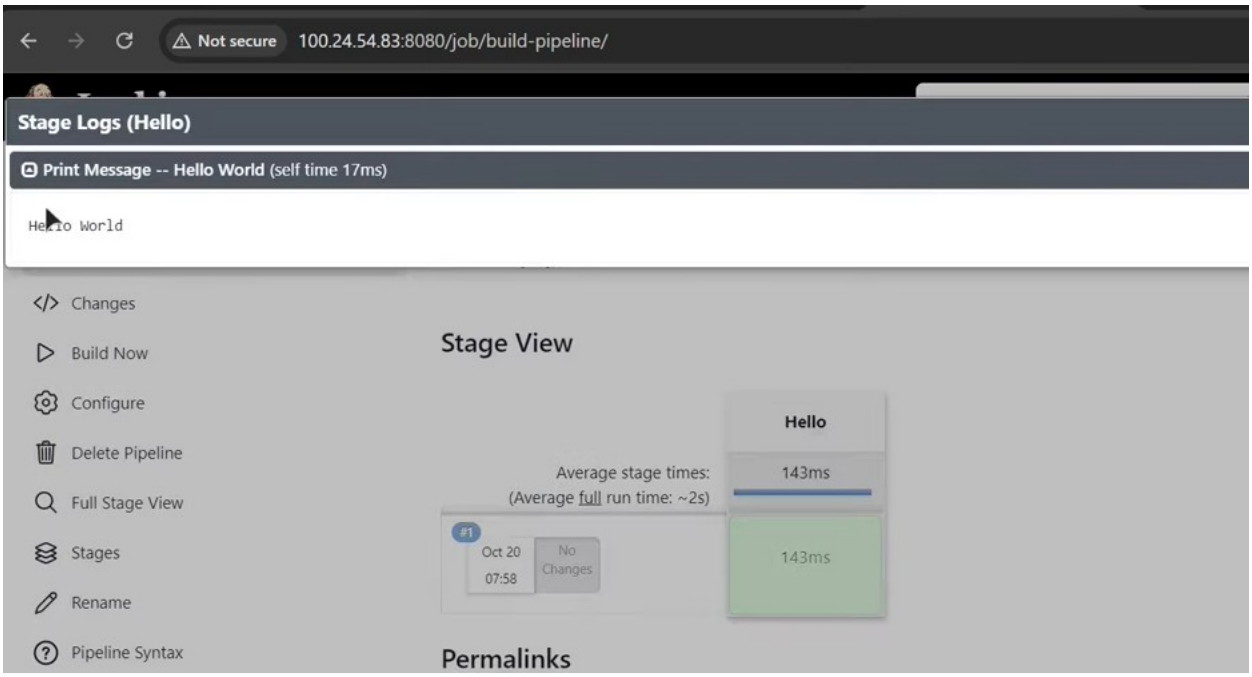



Jenkins
Dashboard > build-pipeline >
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
Build History
trend
build-pipeline

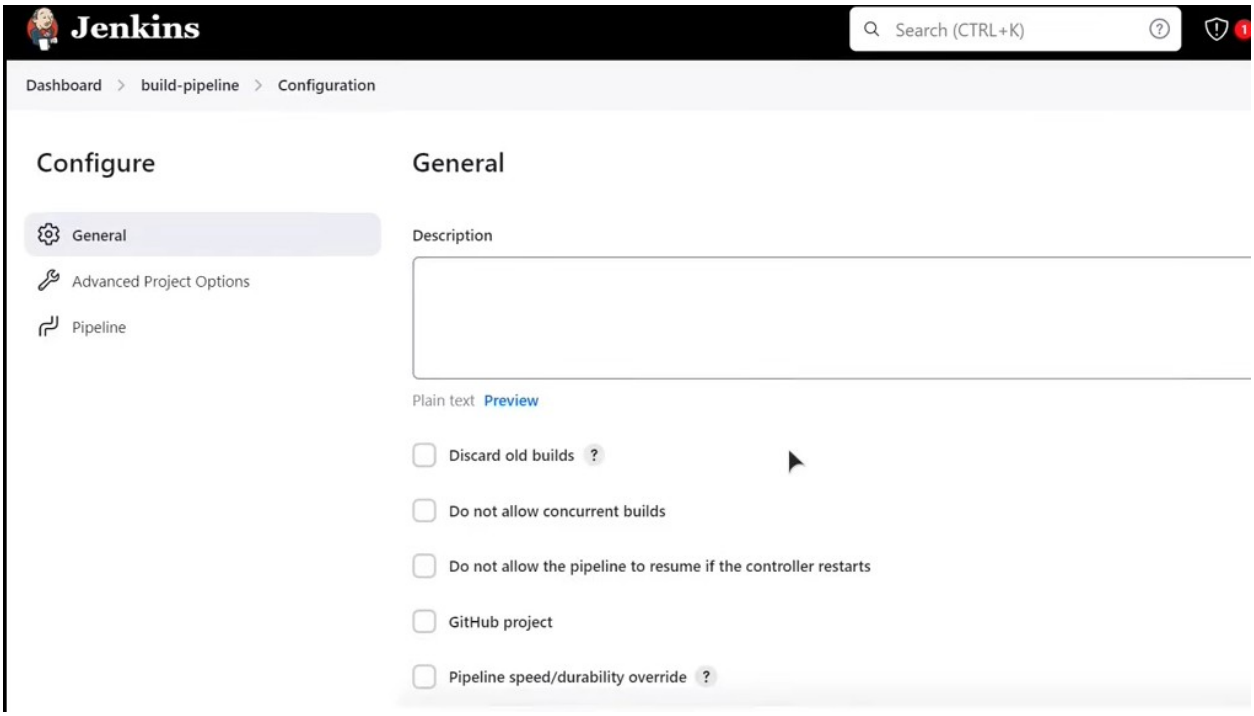

Jenkins
Dashboard > build-pipeline >
Status
Changes
Build scheduled
Build Now
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
build-pipeline
Stage View
No data available. This Pipeline has not yet run.
Permalinks



 # Logs – o/p

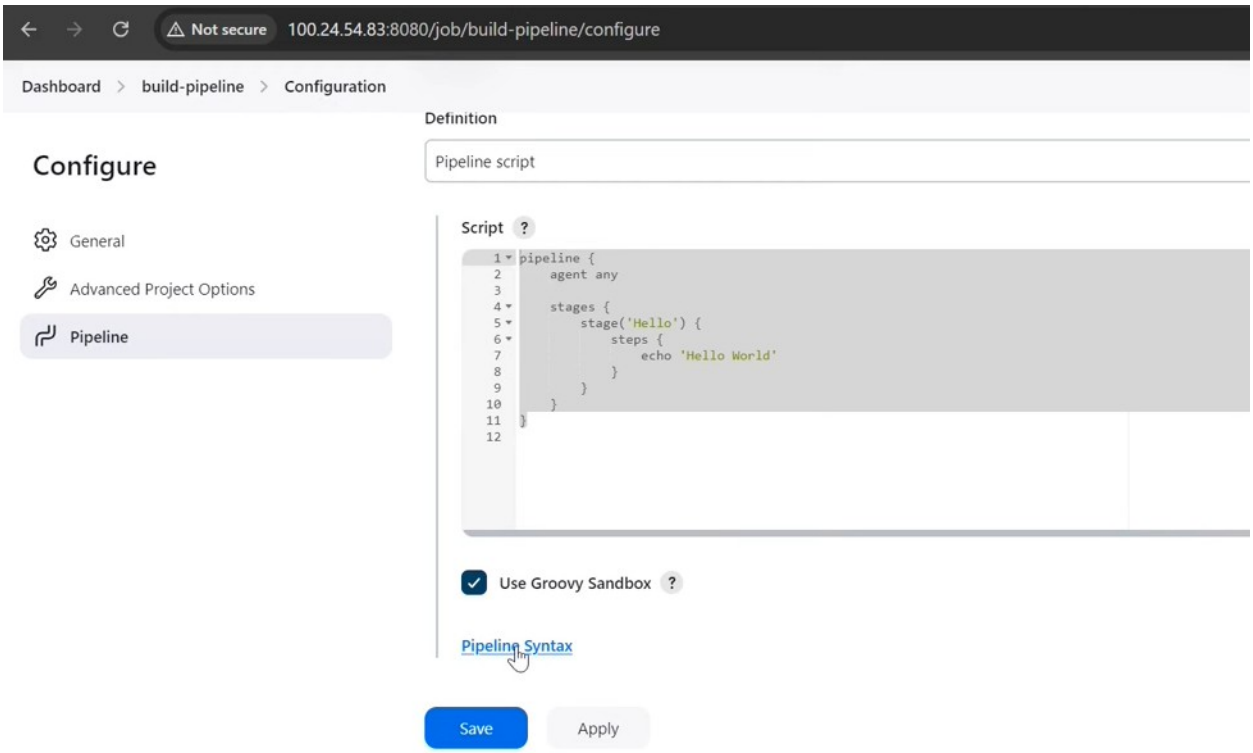


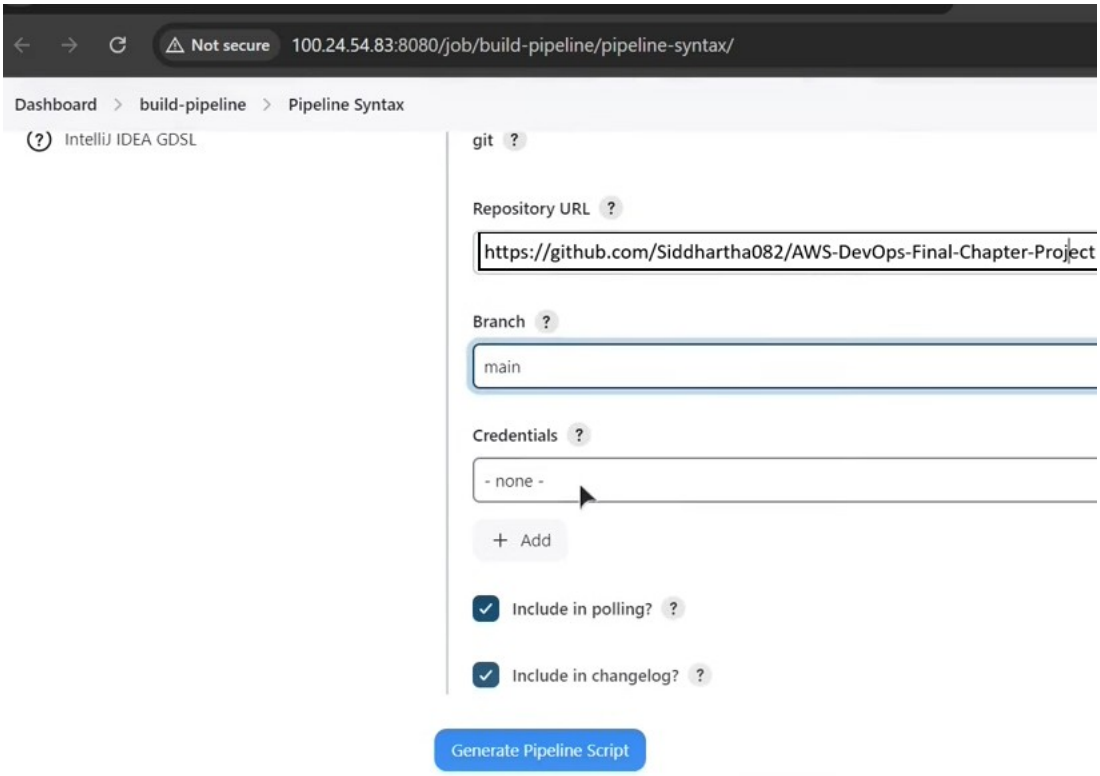
 # Go back to Configure



 # 1st stage .. Git checkout

 # Click on Pipeline Syntax



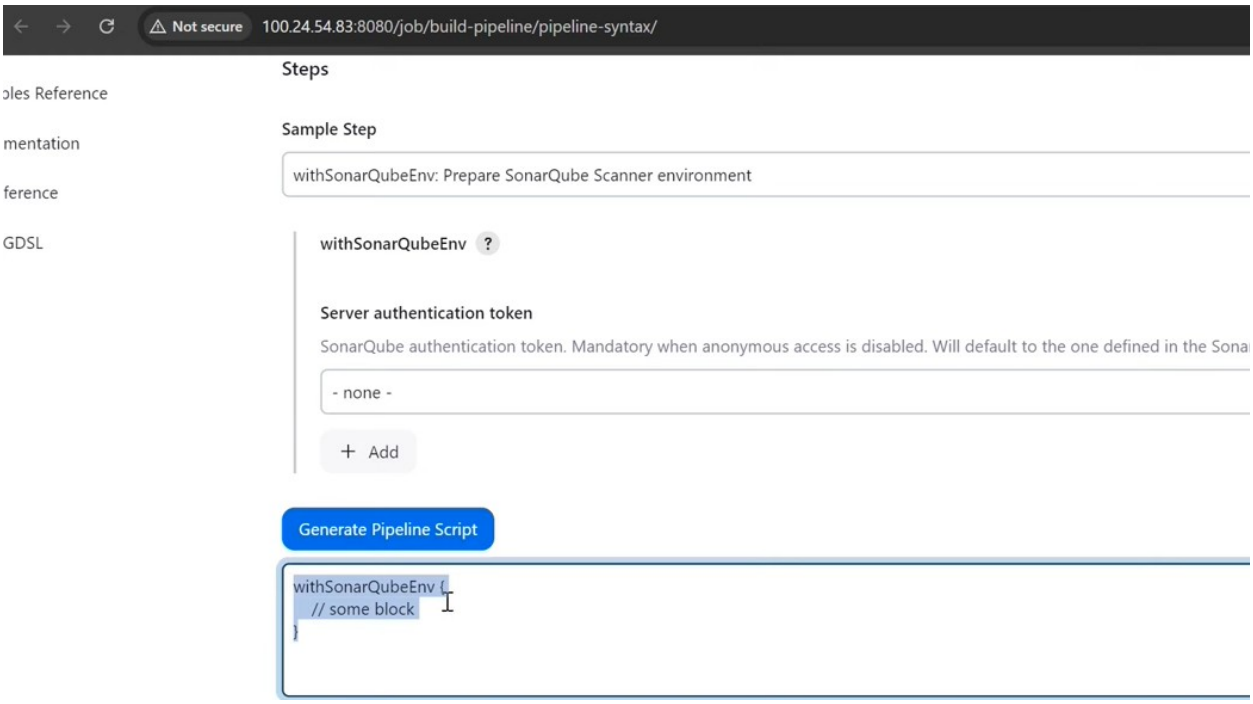


 # note no Credentials added as my repo is in Public domain

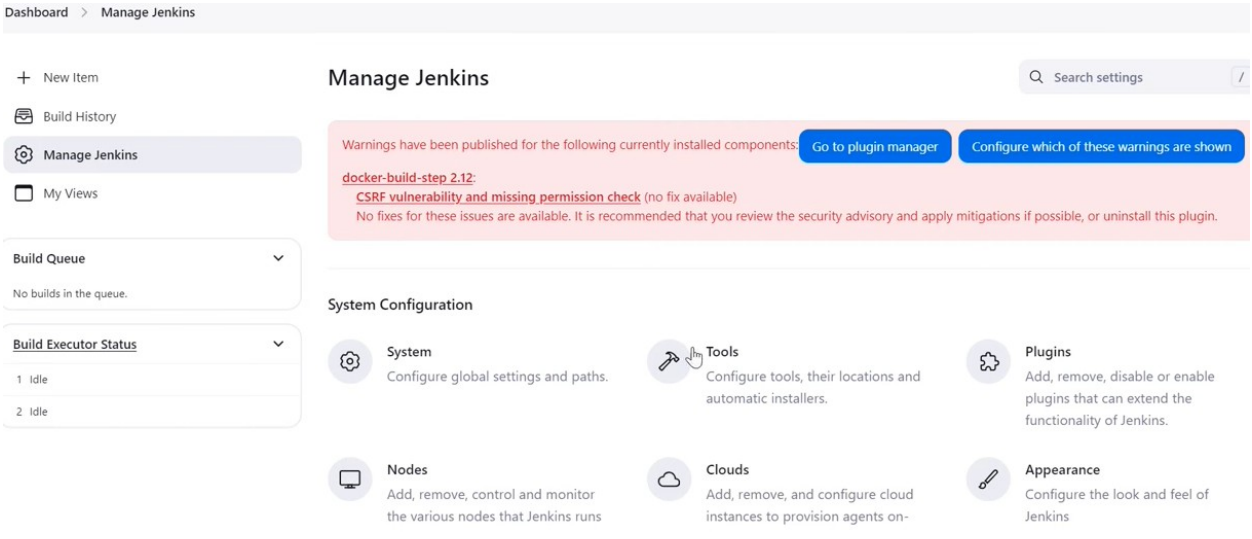

```
environment {
    SCANNER_HOME = tool 'SonarQube Scanner'
}

stages {
    stage('1. Git Checkout') {
        steps {
            git branch: 'main', url: 'https://github.com/Siddhartha082/AWS-DevOps-Final-Chapter-Project.git'
        }
    }
```

 # 2nd Stage- Static code analysis Test- SCAT – SonarQube

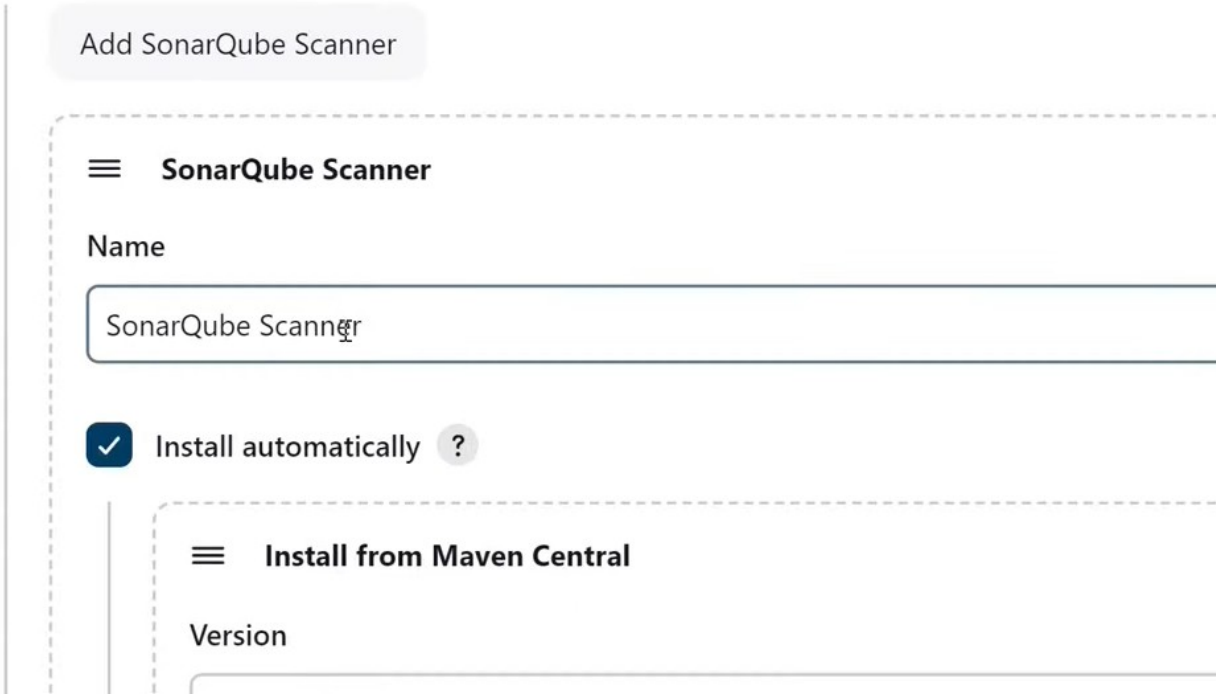

```
    stage('2. SonarQube Analysis') {
        steps {
            withSonarQubeEnv ('sonar-server') {
                sh """
                $SCANNER_HOME/bin/sonar-scanner \
                -Dsonar.projectName=amazon-prime \
                -Dsonar.projectKey=amazon-prime
                """
            }
        }
    }
```



 # go to Jenkins – tools



 # env - var



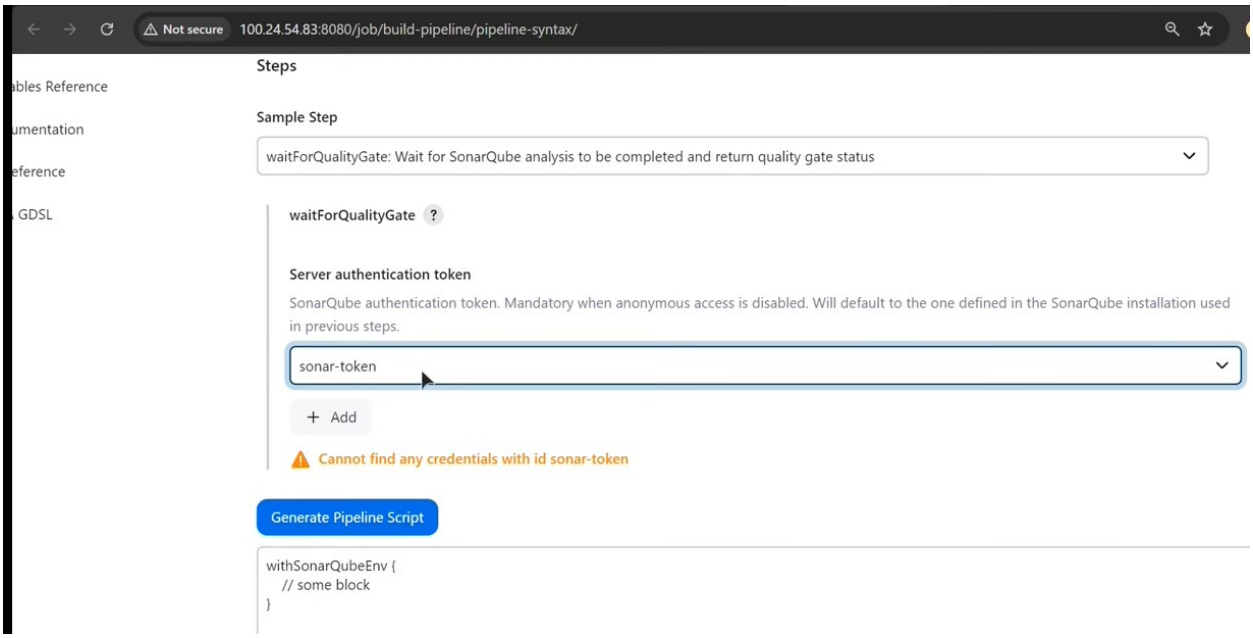
 # Env Variable Scanner

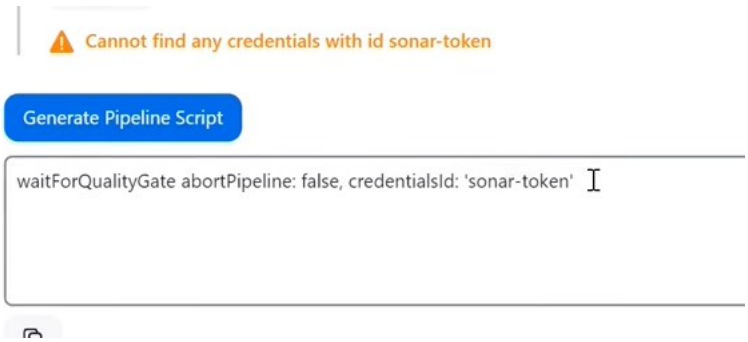
```
environment {
    SCANNER_HOME = tool 'SonarQube Scanner'
}

stages {
    stage('1. Git Checkout') {
        steps {
            git branch: 'main', url: 'https://github.com/Siddhartha082/AWS-DevOps-Final-Chapter-Project.git'
        }
    }

    stage('2. SonarQube Analysis') {
        steps {
            withSonarQubeEnv ('sonar-server') {
                sh """
                $SCANNER_HOME/bin/sonar-scanner \
                -Dsonar.projectName=amazon-prime \
                -Dsonar.projectKey=amazon-prime
                """
            }
```

 # Sonar helps in checking duplication/ Bugs in code / check security -vunlerabilities - Sonar Scanner

#3rd Stage Quality Gate - Defination & example

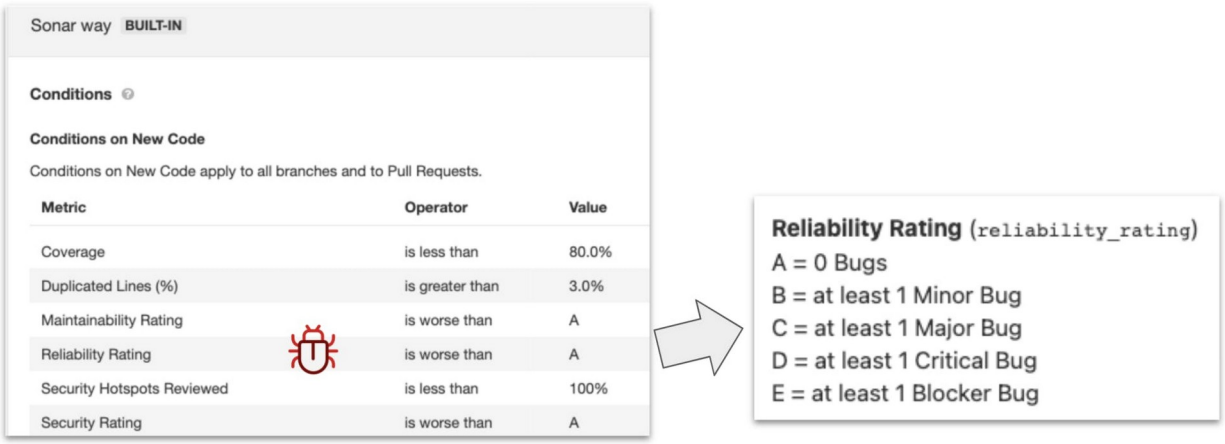
```
stage('3. Quality Gate') {
    steps {
        waitForQualityGate abortPipeline: false,
        credentialsId: 'sonar-token'
    }
}
```



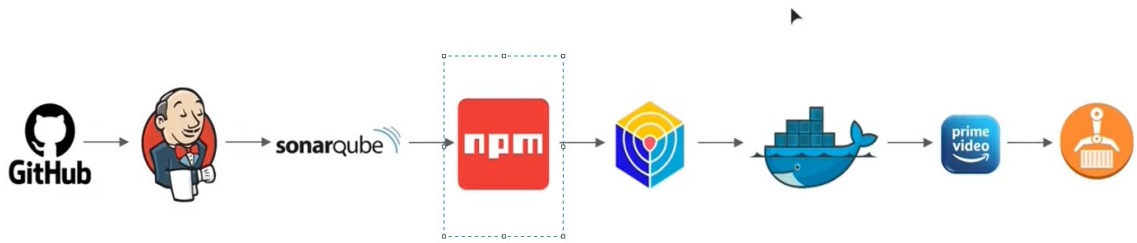


The Quality Gate

The Quality Gate lets you set your own code quality and security conditions by selecting a metric and then setting the pass/fail threshold. If any of the conditions in the QG fail, the overall QG fails and you know not to merge your code until you remedy the situation. The QG is dynamically updated so you'll know immediately if a fix gives you the 'GREEN' light! Just like with the QP, you can use the built-in Quality Gate called Sonar way or customize your own based on your team's clean, secure coding definition that we talked about earlier. An example will demonstrate how it all comes together. The graphic below shows you how the Reliability (bugs) Rating metric is calculated.

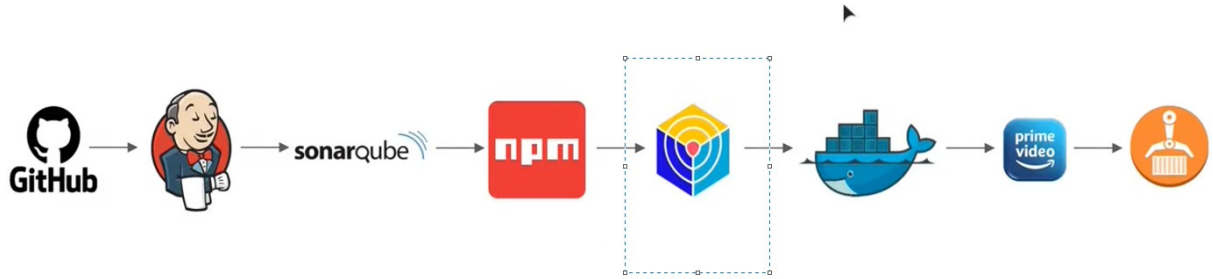


#4th Stage npm installation



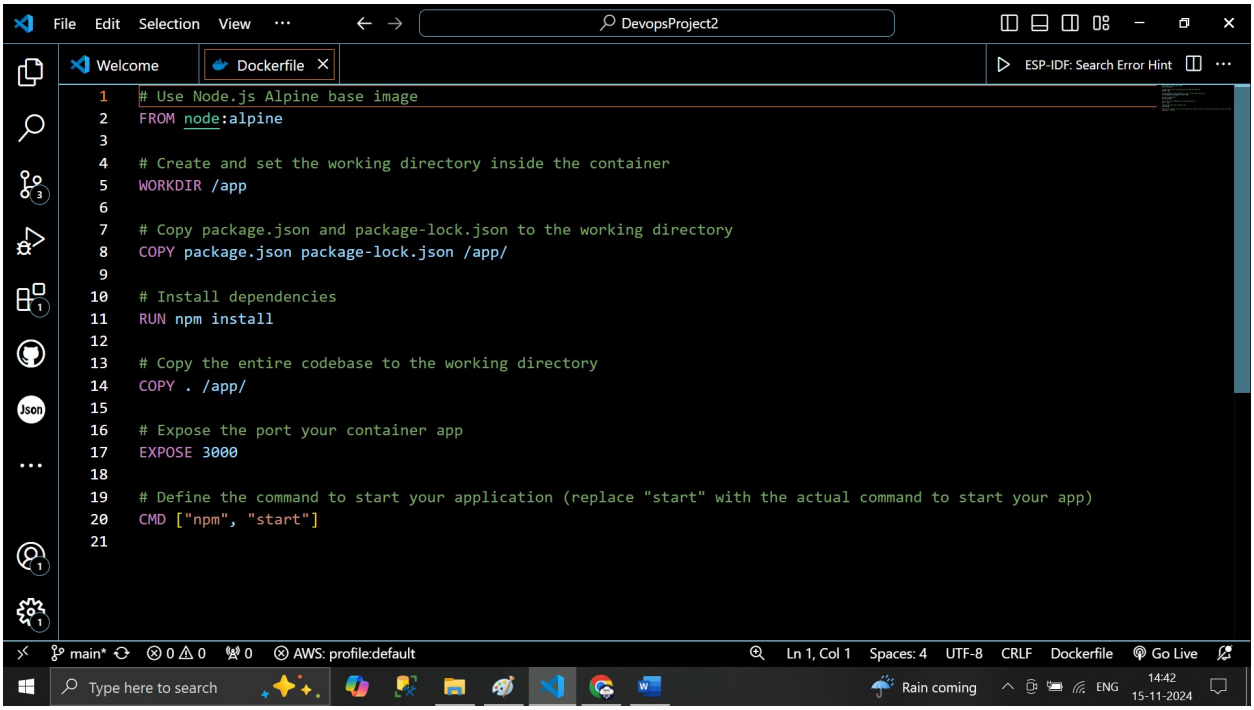

```
stage('4. Install npm') {
    steps {
        sh "npm install"
    }
}
```

#5th Stage Trivy - Scans the project for vulnerabilities.




```
stage('5. Trivy Scan') {
    steps {
        sh "trivy fs . > trivy.txt"
    }
}
```

 # 6th Stage Docker file

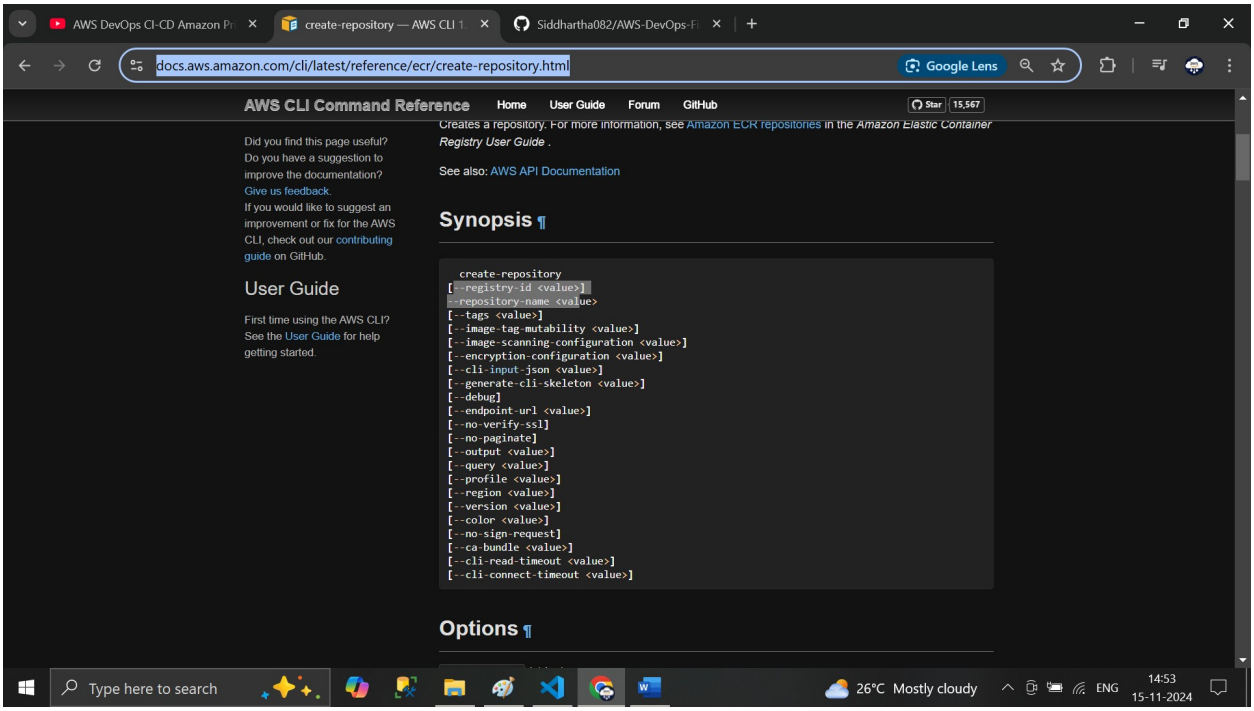


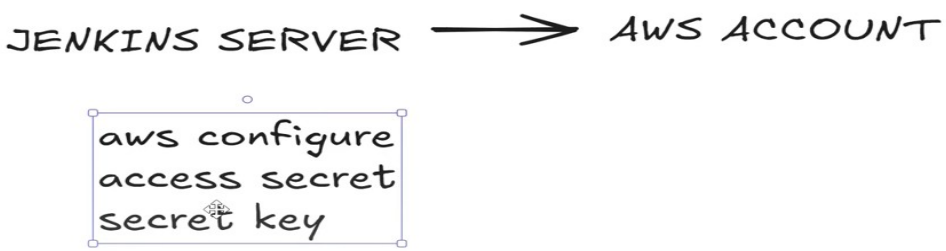
```
    stage('6. Build Docker Image') {
        steps {
            sh "docker build -t ${params.ECR_REPO_NAME} ."
        }
    }
```

```
parameters {
    string(name: 'ECR_REPO_NAME', defaultValue: 'amazon-prime', description: 'Enter repository name')
    string(name: 'AWS_ACCOUNT_ID', defaultValue: '851725192411', description: 'Enter AWS Account ID') // Added missing quote
}
```

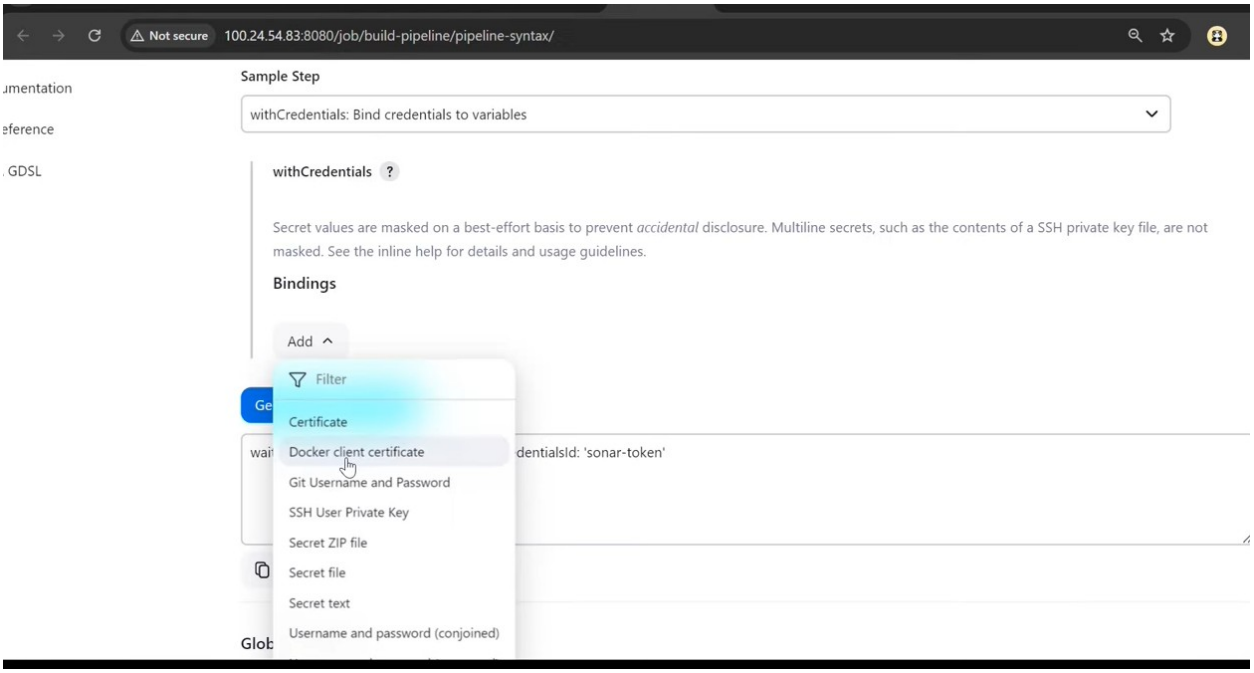
 # 7th stage – ECR Creation

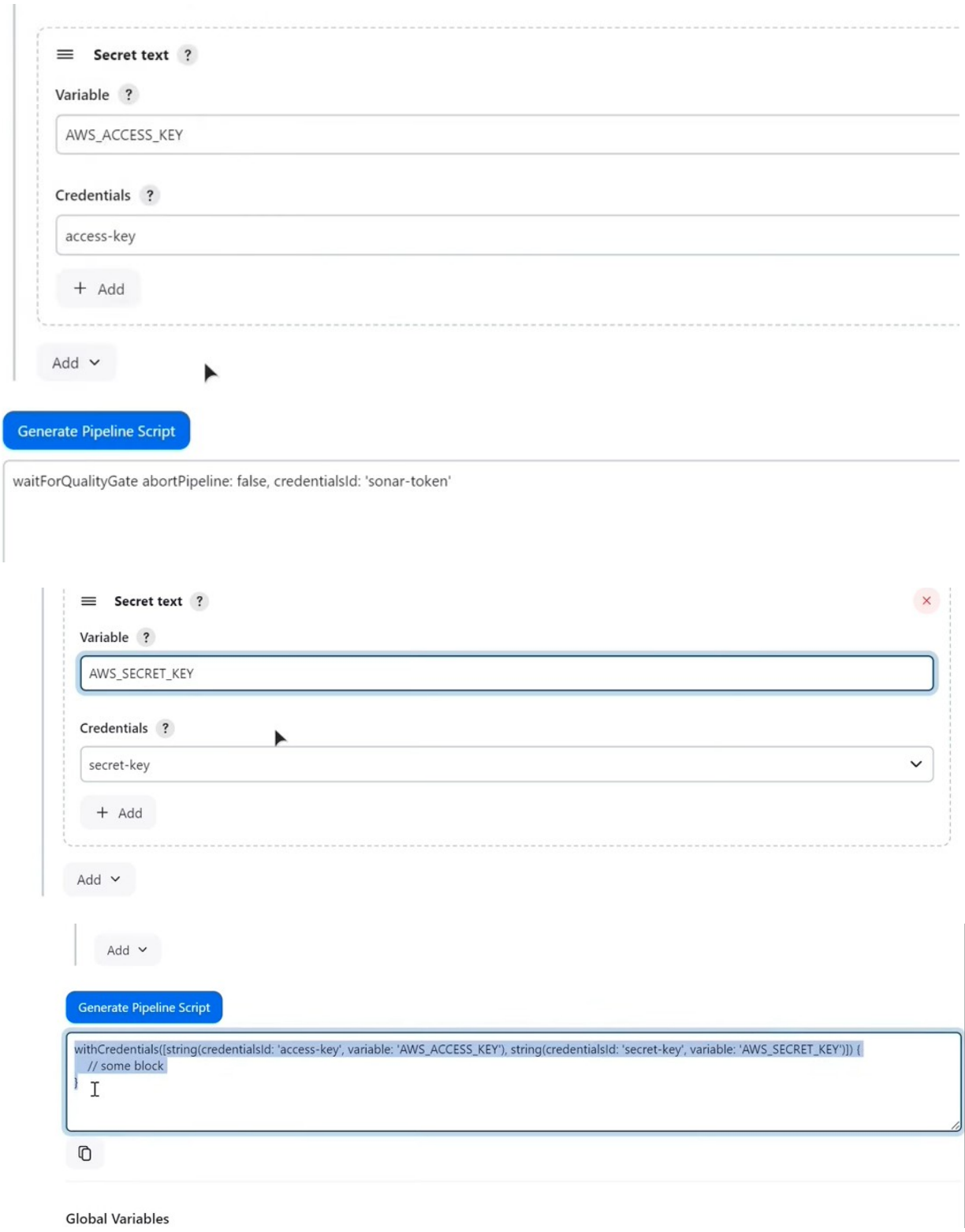
 # Ref - https://docs.aws.amazon.com/cli/latest/reference/ecr/create-repository.html





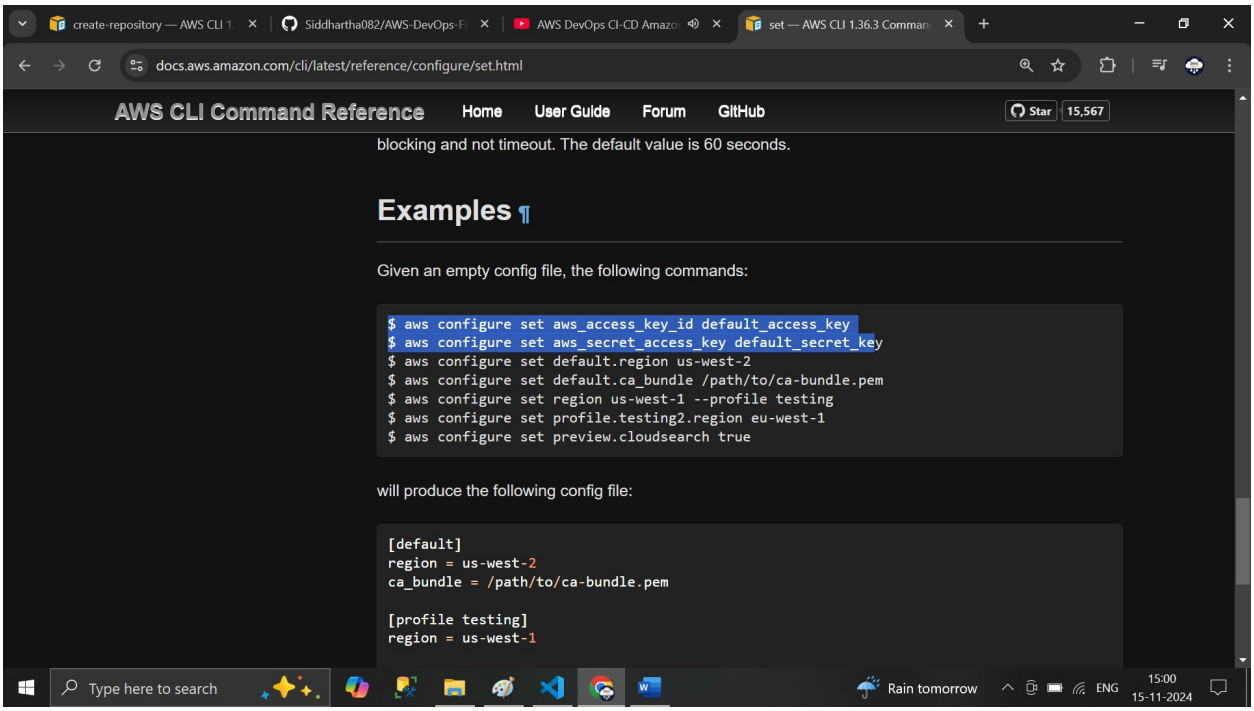
 # go to Pipeline script --set up a Connection … using credentials as key to connections





 # AWS Configure -

https://docs.aws.amazon.com/cli/latest/reference/configure/set.html



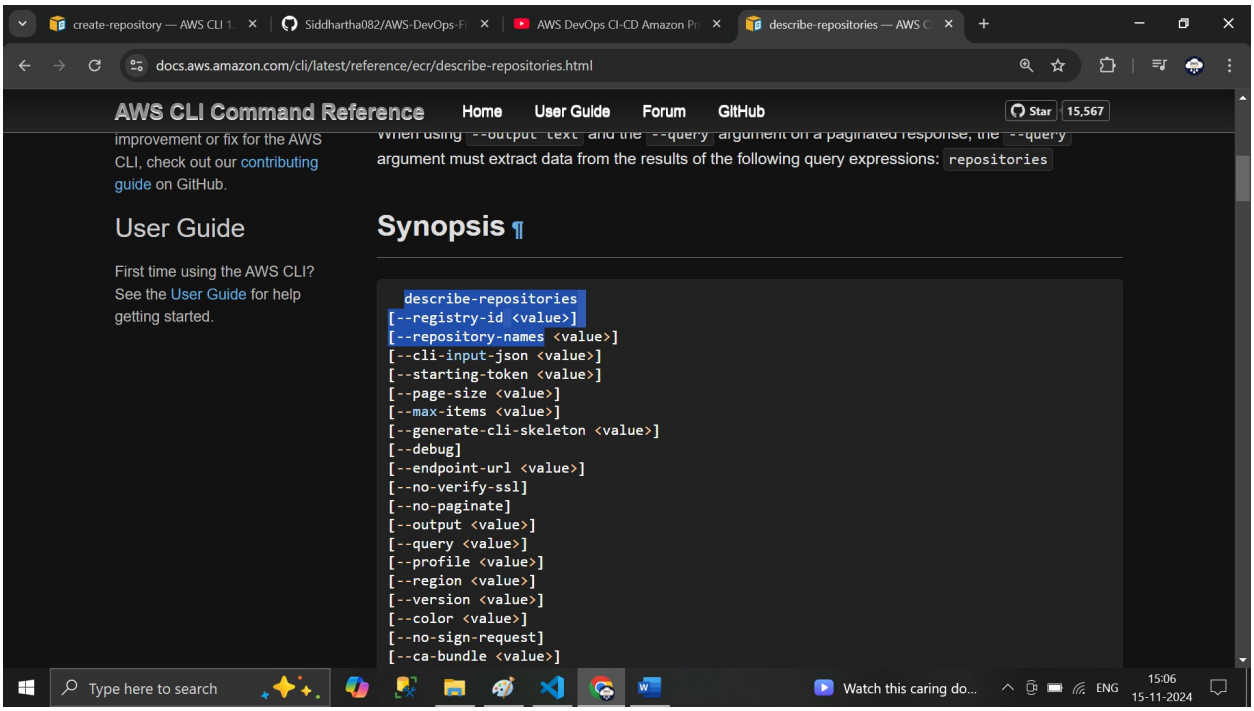

```
stage('7. Create ECR repo') {
    steps {
        withCredentials([string(credentialsId: 'access-key', variable: 'AWS_ACCESS_KEY'),
                         string(credentialsId: 'secret-key', variable: 'AWS_SECRET_KEY')]) {
            sh """
            aws configure set aws_access_key_id $AWS_ACCESS_KEY
            aws configure set aws_secret_access_key $AWS_SECRET_KEY
            aws ecr describe-repositories --repository-names ${params.ECR_REPO_NAME} --region us-east-1 || \
            aws ecr create-repository --repository-name ${params.ECR_REPO_NAME} --region us-east-1
            """
        }
    }
}
```

 # AWS ECR describe repo # if the ECR repo already exsisted then new repo will not be created

Else

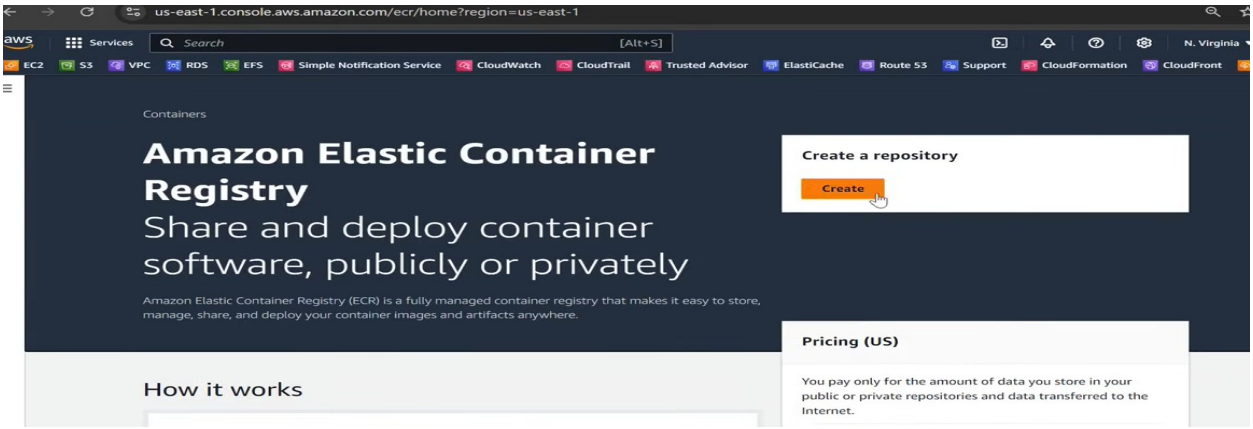
4th Line of the code .. new ECR repo will be created …… @ us-east-1 region

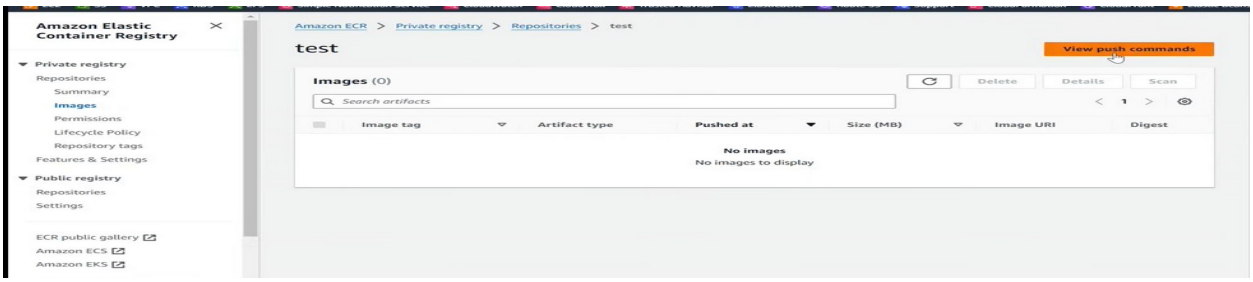
https://docs.aws.amazon.com/cli/latest/reference/ecr/describe-repositories.html



8th Stage – Login into ECR & Tagging the image(Docker)

- **Go to AWS Console- ECR -create - ECR—**





 # Mac/Linux

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry authentication.

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 851725192411.dkr.ecr.us-east-1.amazonaws.com
```

Note: if you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch, see the instructions here. You can skip this step if your image has already been built:

```
docker build -t test .
```

3. After the build is completed, tag your image so you can push the image to this repository:

```
docker tag test:latest 851725192411.dkr.ecr.us-east-1.amazonaws.com/test:latest
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push 851725192411.dkr.ecr.us-east-1.amazonaws.com/test:latest
```

 # windows

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS TOOLS for PowerShell:

```
(Get-ECRLoginCommand).Password | docker login --username AWS --password-stdin 851725192411.dkr.ecr.us-east-1.amazonaws.com
```

Note: if you receive an error using the AWS TOOLS for PowerShell, make sure that you have the latest version of the AWS TOOLS for PowerShell and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch, see the instructions here. You can skip this step if your image has already been built:

```
docker build -t test .
```

3. After the build is completed, tag your image so you can push the image to this repository:

```
docker tag test:latest 851725192411.dkr.ecr.us-east-1.amazonaws.com/test:latest
```

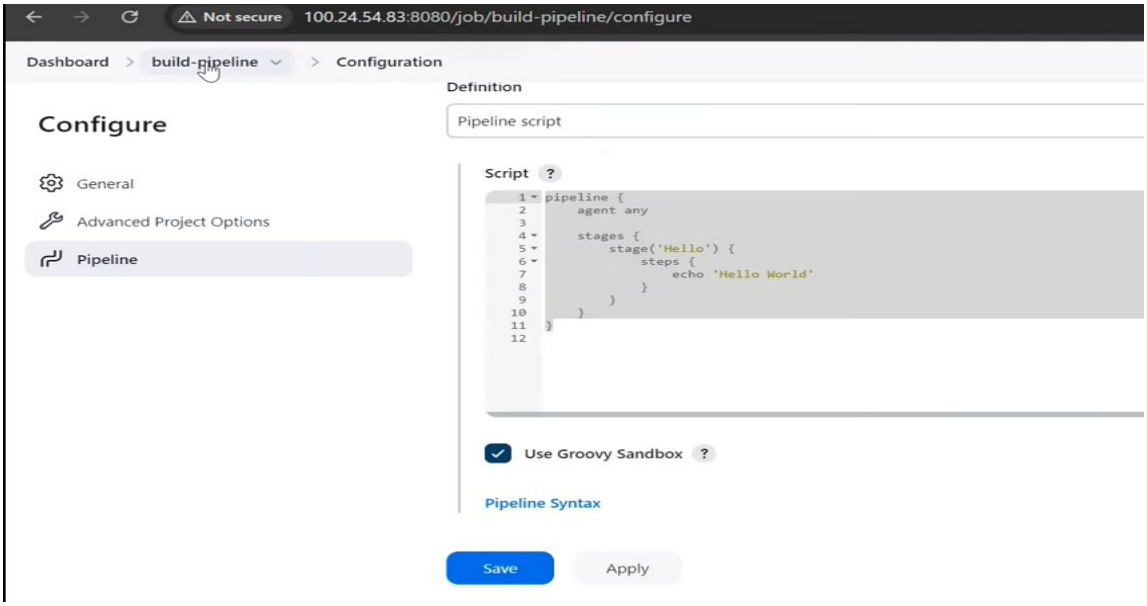
4. Run the following command to push this image to your newly created AWS repository:

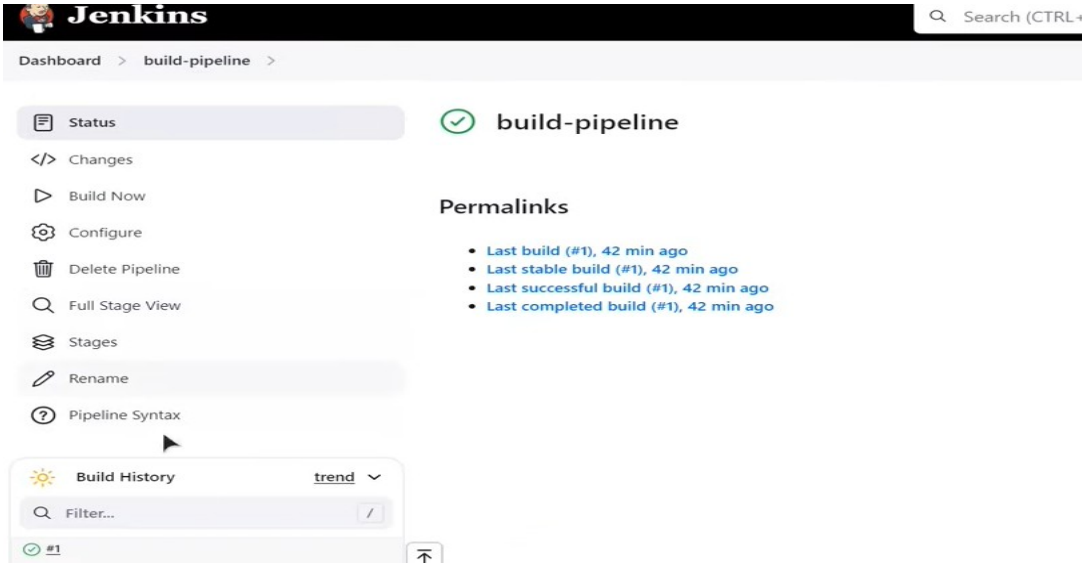
```
docker push 851725192411.dkr.ecr.us-east-1.amazonaws.com/test:latest
```

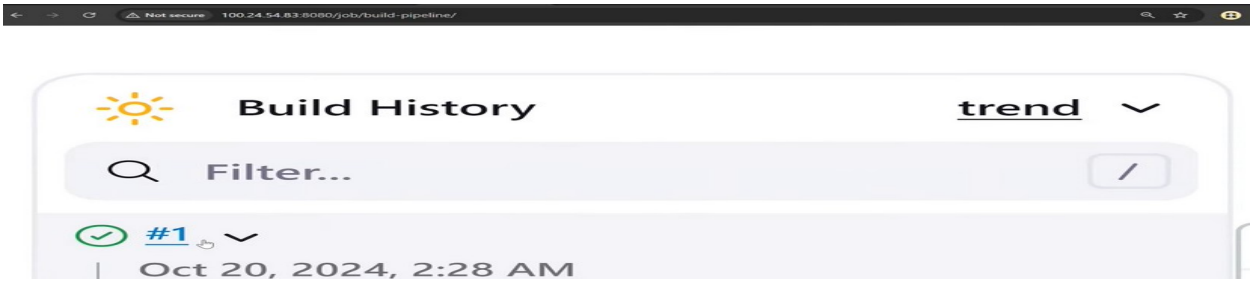
```
stage('8. Login to ECR & tag image') {
    steps {
        withCredentials([string(credentialsId: 'access-key', variable: 'AWS_ACCESS_KEY'),
                         string(credentialsId: 'secret-key', variable: 'AWS_SECRET_KEY')]) {
            sh """
            aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin ${params.AWS_ACCOUNT_ID}.dkr.ecr.us-east-1.amazonaws.com
            docker tag ${params.ECR_REPO_NAME} ${params.AWS_ACCOUNT_ID}.dkr.ecr.us-east-1.amazonaws.com/${params.ECR_REPO_NAME}:${BUILD_NUMBER}
            docker tag ${params.ECR_REPO_NAME} ${params.AWS_ACCOUNT_ID}.dkr.ecr.us-east-1.amazonaws.com/${params.ECR_REPO_NAME}:latest
            """
        }
    }
}
```

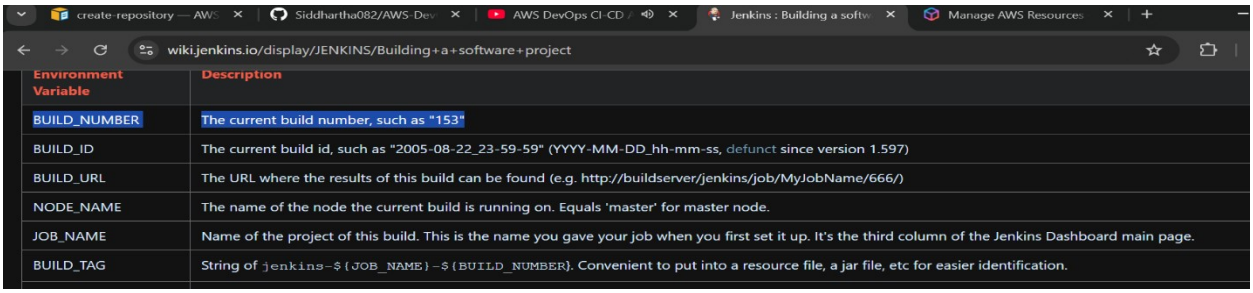
 # Build_Number plays a vital role in jenkins that Signifyes the Versions of images stored

https://wiki.jenkins.io/display/JENKINS/Building+a+software+project

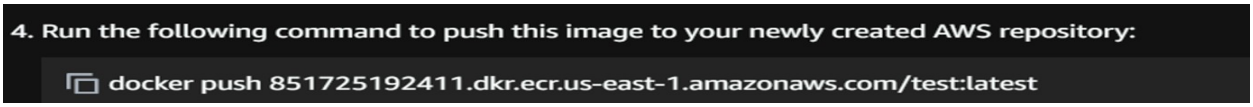






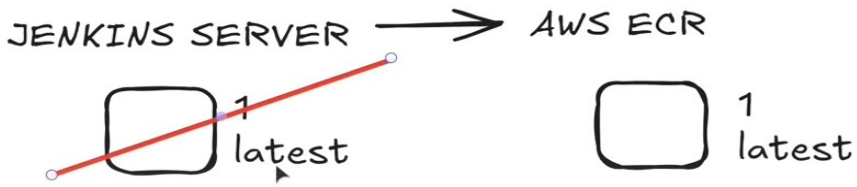


#9th Stage : Push to AWS ECR: Tags and pushes the Docker image to ECR.




```
stage('9. Push image to ECR') {
    steps {
        withCredentials([string(credentialsId: 'access-key', variable: 'AWS_ACCESS_KEY'),
                         string(credentialsId: 'secret-key', variable: 'AWS_SECRET_KEY')]) {
            sh """
            docker push ${params.AWS_ACCOUNT_ID}.dkr.ecr.us-east-1.amazonaws.com/${params.ECR_REPO_NAME}:${BUILD_NUMBER}
            docker push ${params.AWS_ACCOUNT_ID}.dkr.ecr.us-east-1.amazonaws.com/${params.ECR_REPO_NAME}:latest
            """
        }
    }
```

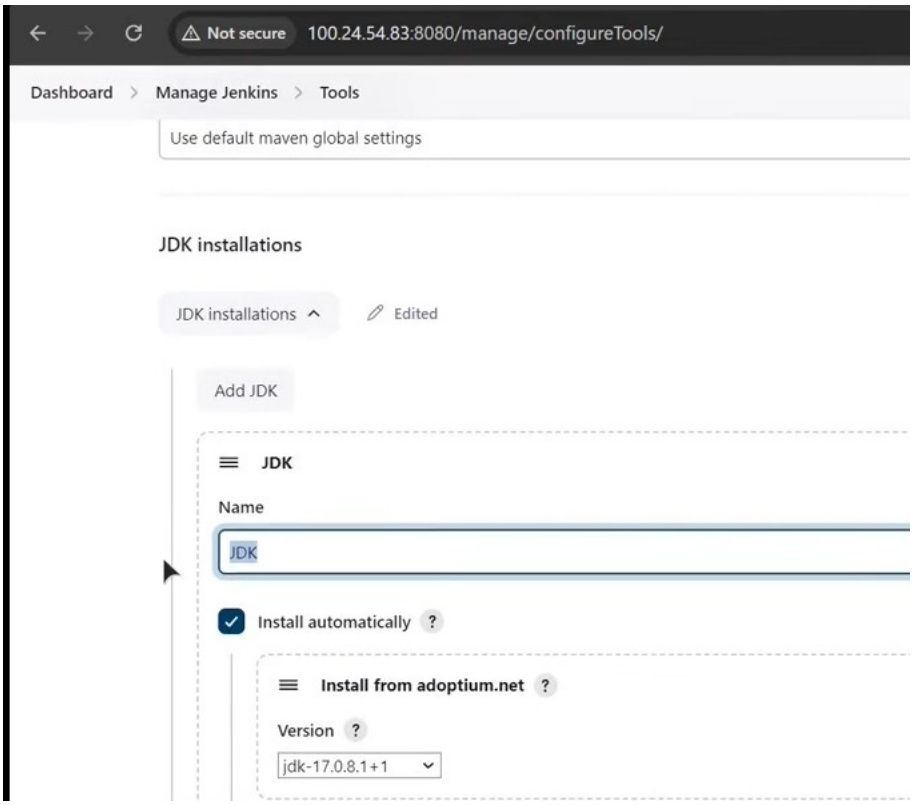
 # Note once image is Pushed in AWS ECR , We need to Clean up the images stored in Jenkins to remove unneccessry memory space

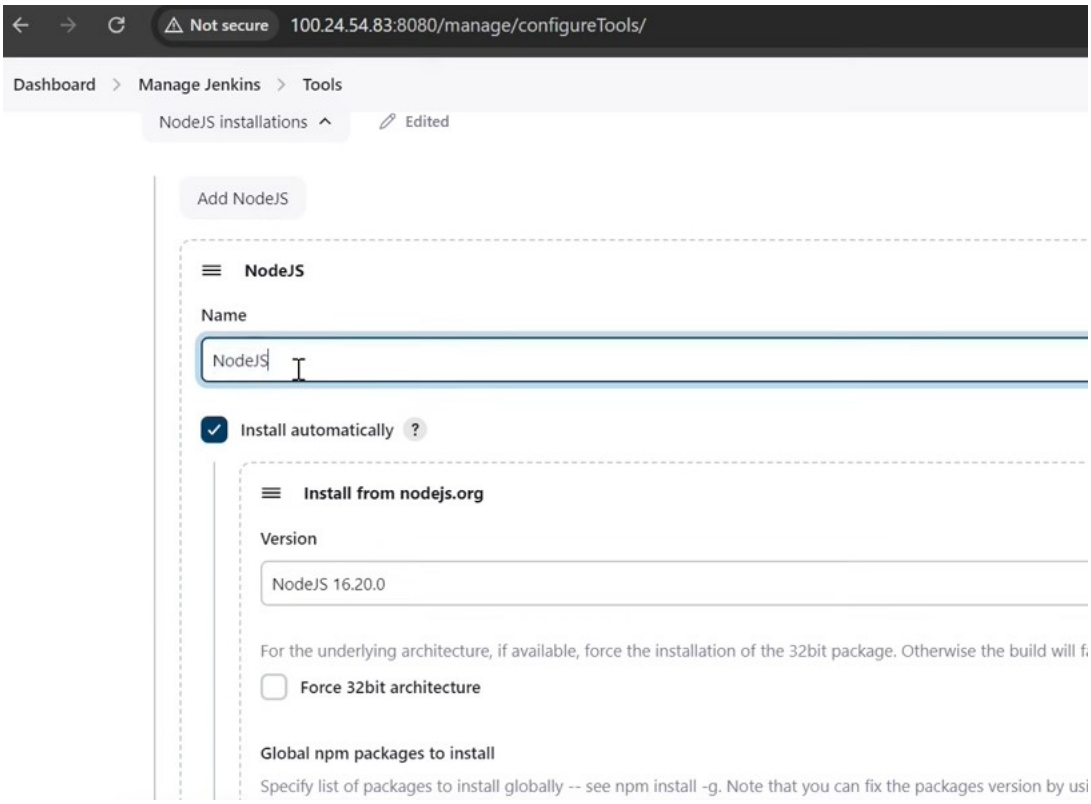


 # 10th Final Stage Clean up images in Jenkins

```
stage('10. Cleanup Images') {
    steps {
        sh """
        docker rmi ${params.AWS_ACCOUNT_ID}.dkr.ecr.us-east-1.amazonaws.com/${params.ECR_REPO_NAME}:${BUILD_NUMBER}
        docker rmi ${params.AWS_ACCOUNT_ID}.dkr.ecr.us-east-1.amazonaws.com/${params.ECR_REPO_NAME}:latest
docker images
        """
    }
}
```

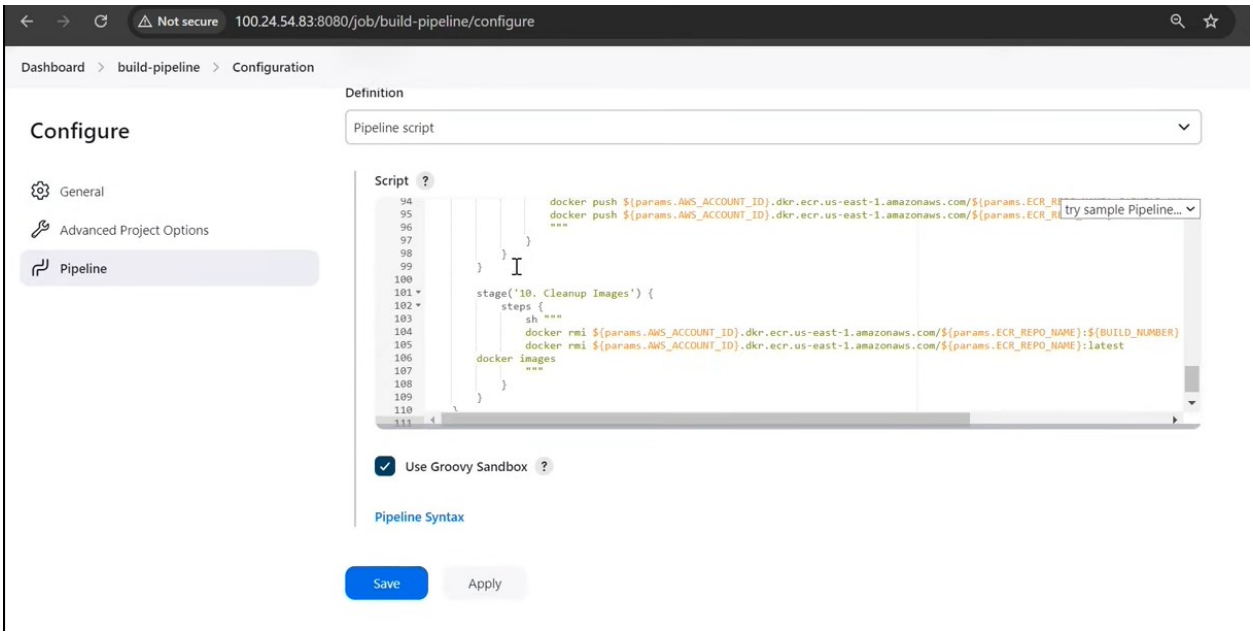
 # tools


Not secure
100.24.54.83:8080/manage/configureTools/
Dashboard
Manage Jenkins
Tools
Use default maven global settings
JDK installations
JDK installations
Edited
Add JDK
JDK
Name
JDK
Install automatically
Install from adoptium.net
Version
jdk-17.0.8.1+1



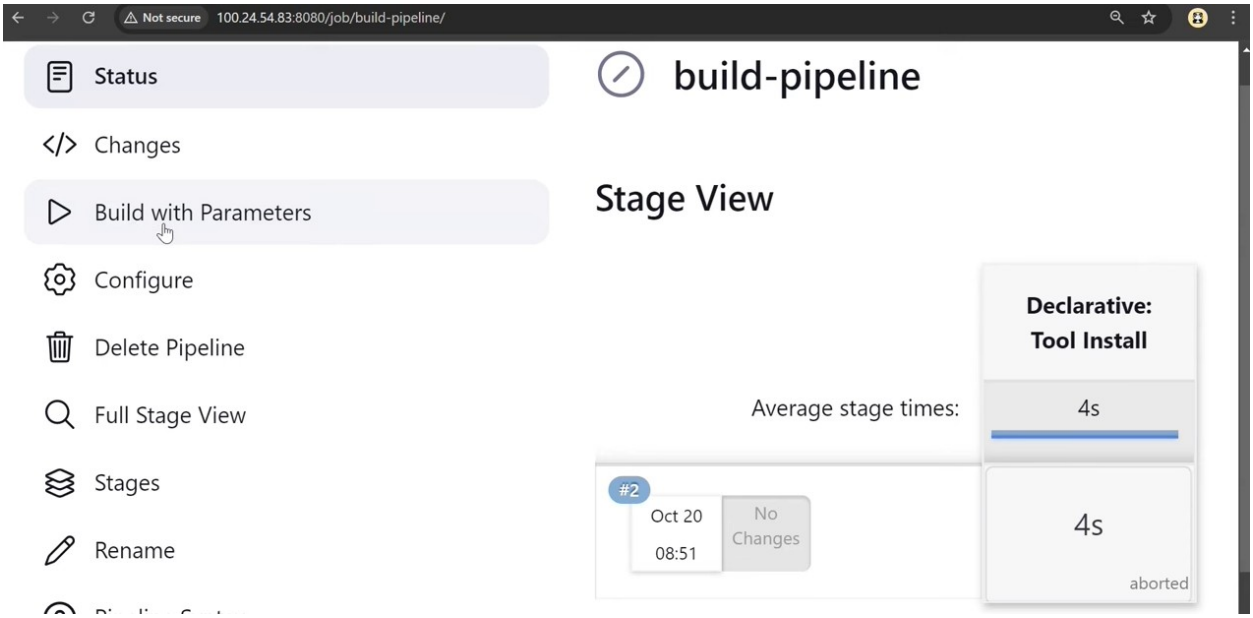

```
pipeline {
    agent any

    parameters {
        string(name: 'ECR_REPO_NAME', defaultValue: 'amazon-prime', description: 'Enter repository name')
        string(name: 'AWS_ACCOUNT_ID', defaultValue: '851725192411', description: 'Enter AWS Account ID') // Added missing quote
    }

    tools {
        jdk 'JDK'
        nodejs 'NodeJS'
    }
```

 # Combined all the stages .. @ once .. Paste it on Jenkins Pipeline Script

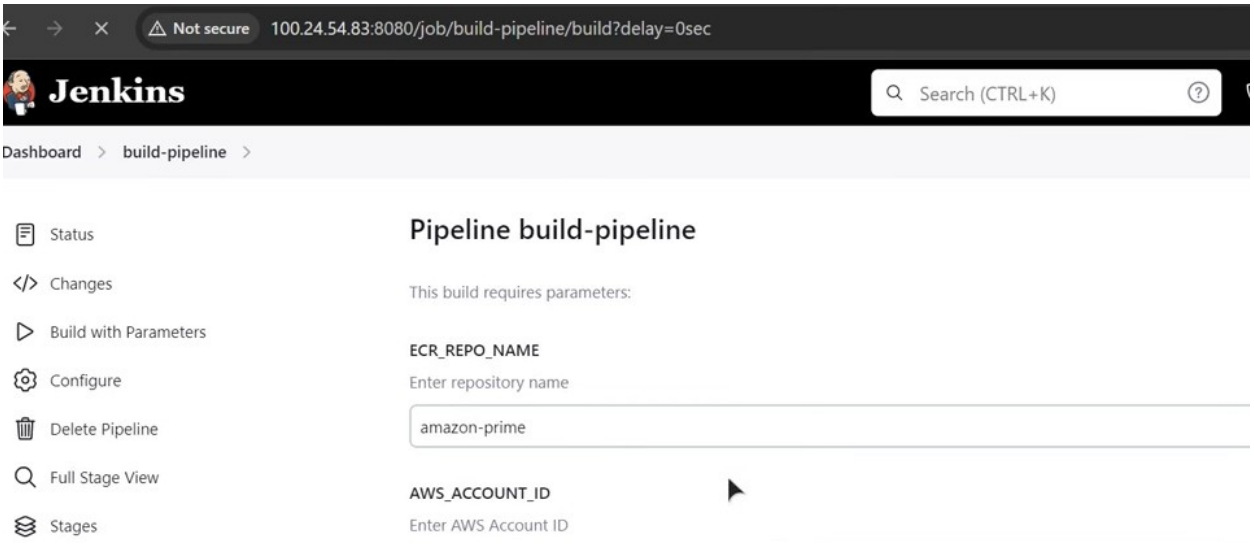


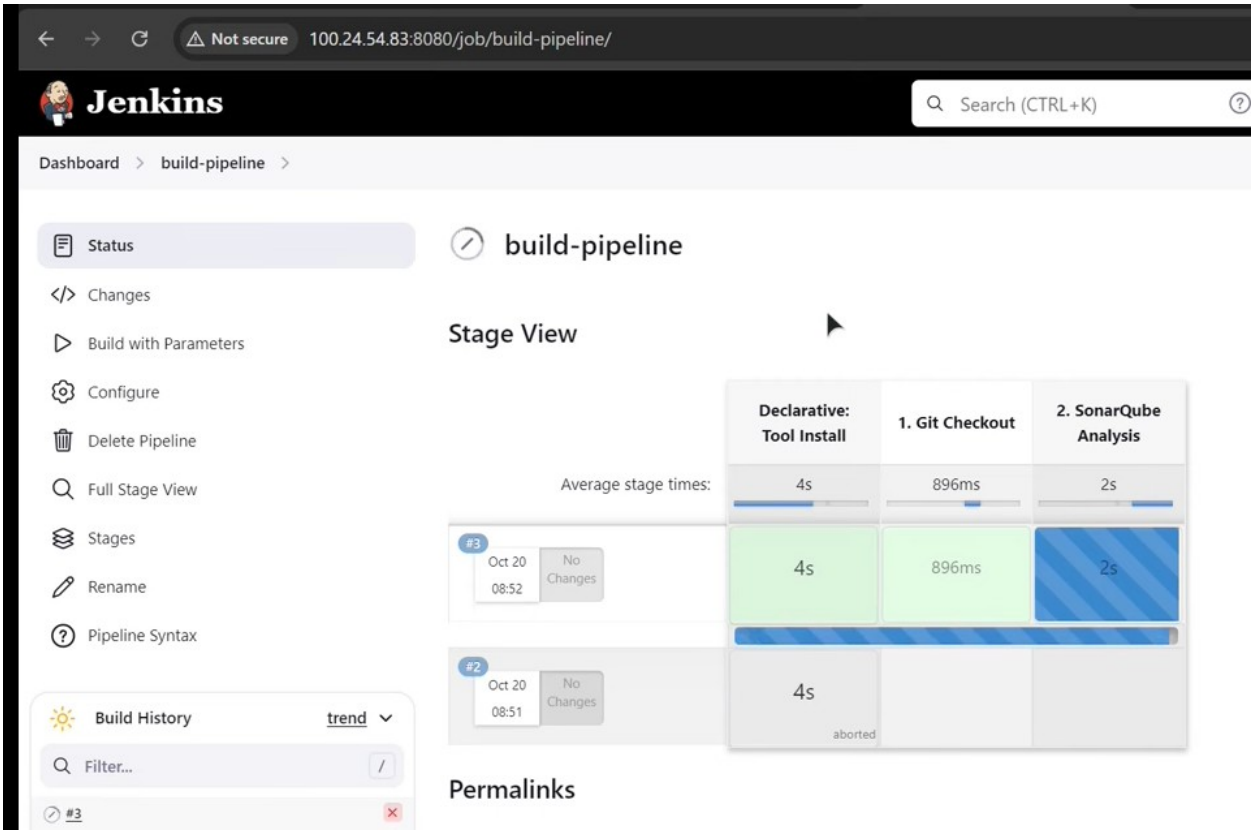
 # save it

 # PART 1 – CI -PIPELINE ….

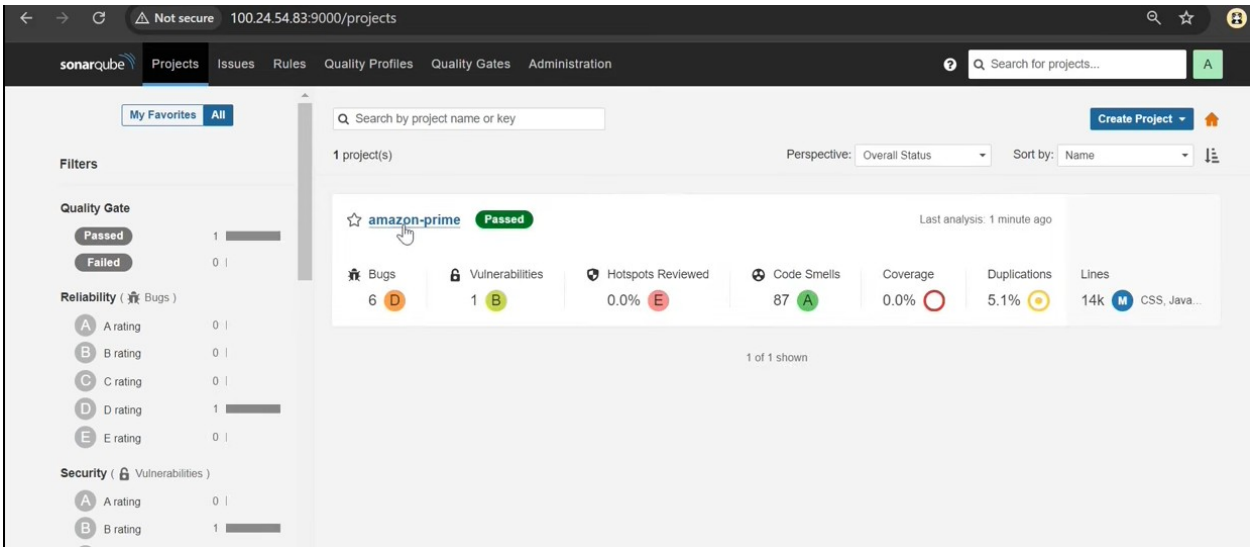


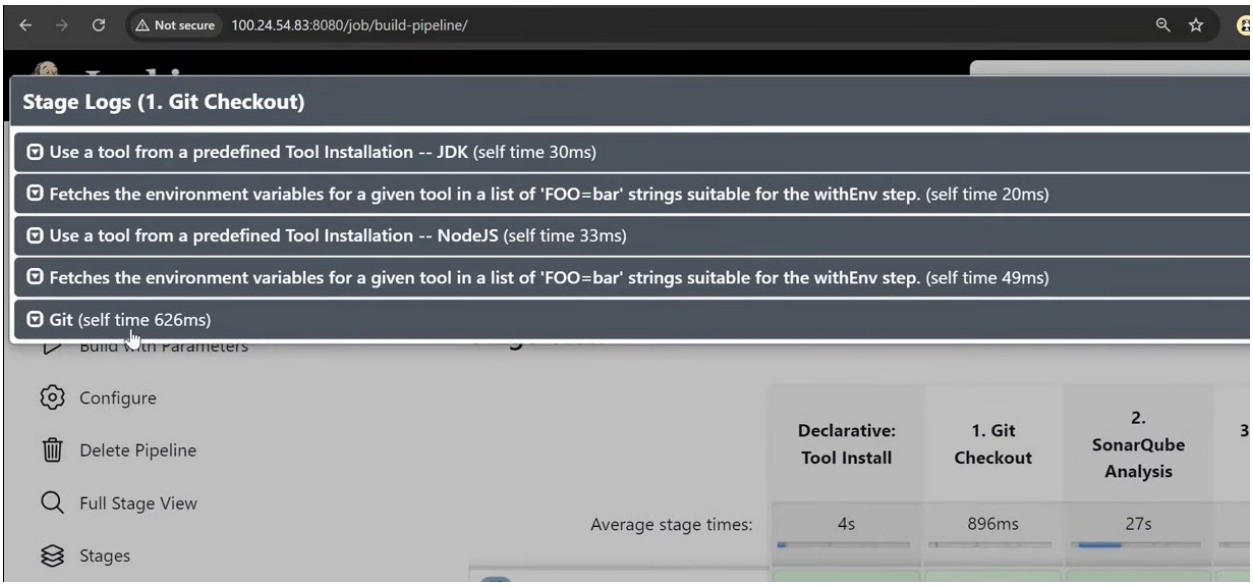
 # Click Build with parameter



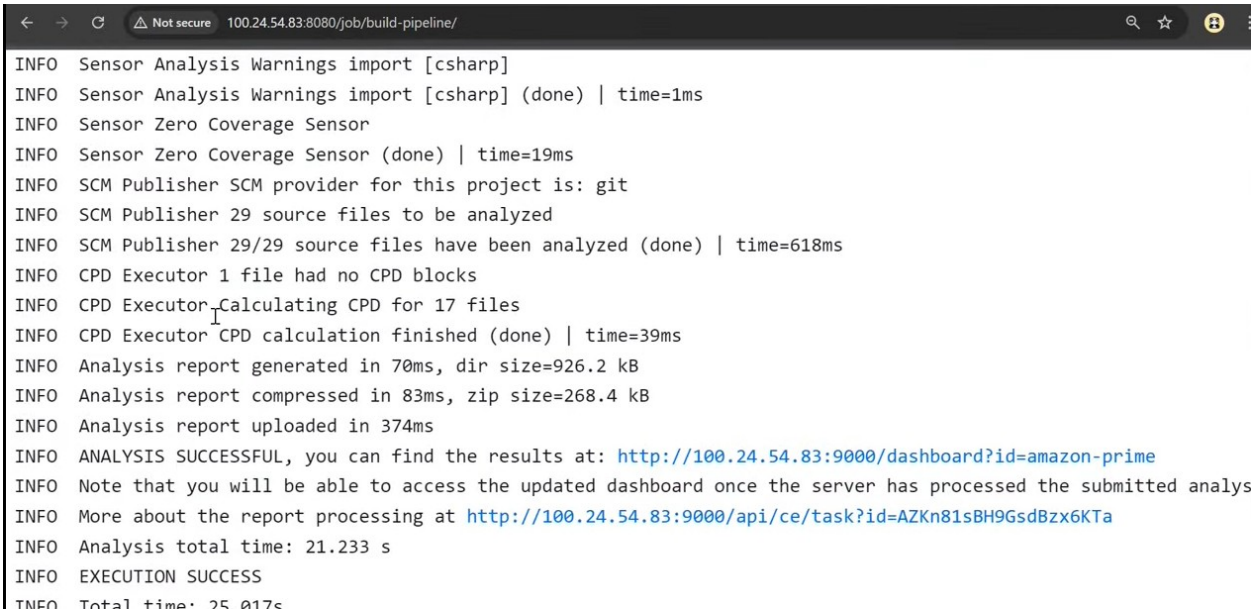


 # in SonarQube .. Projects

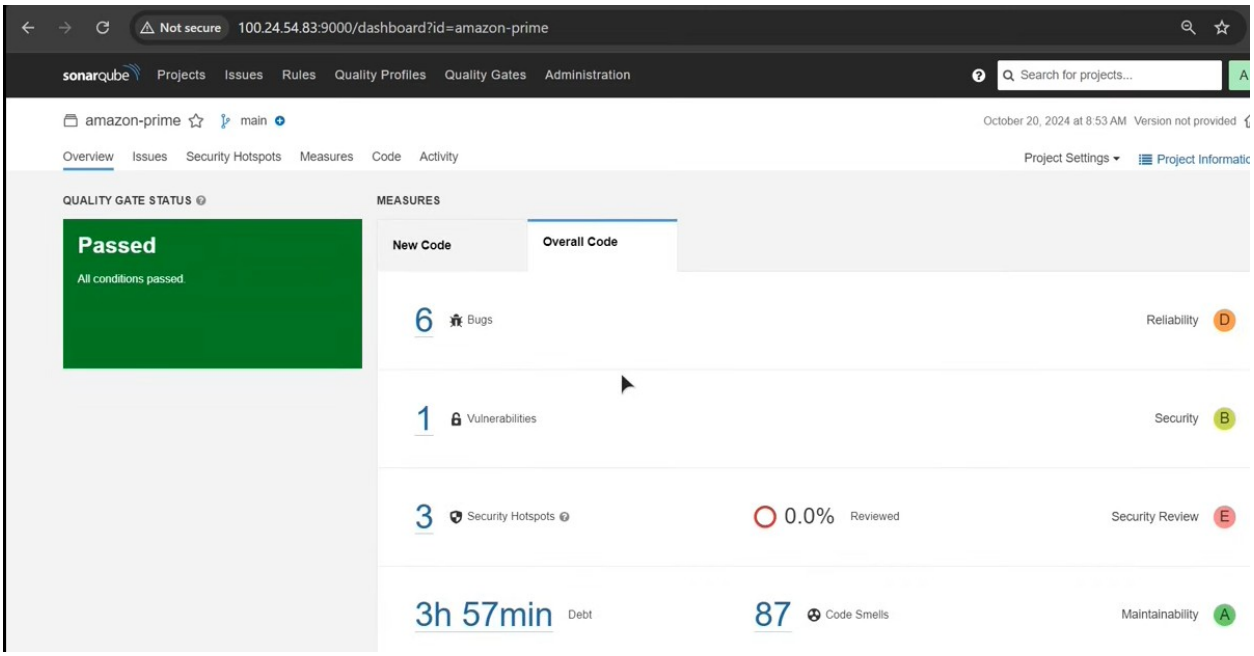




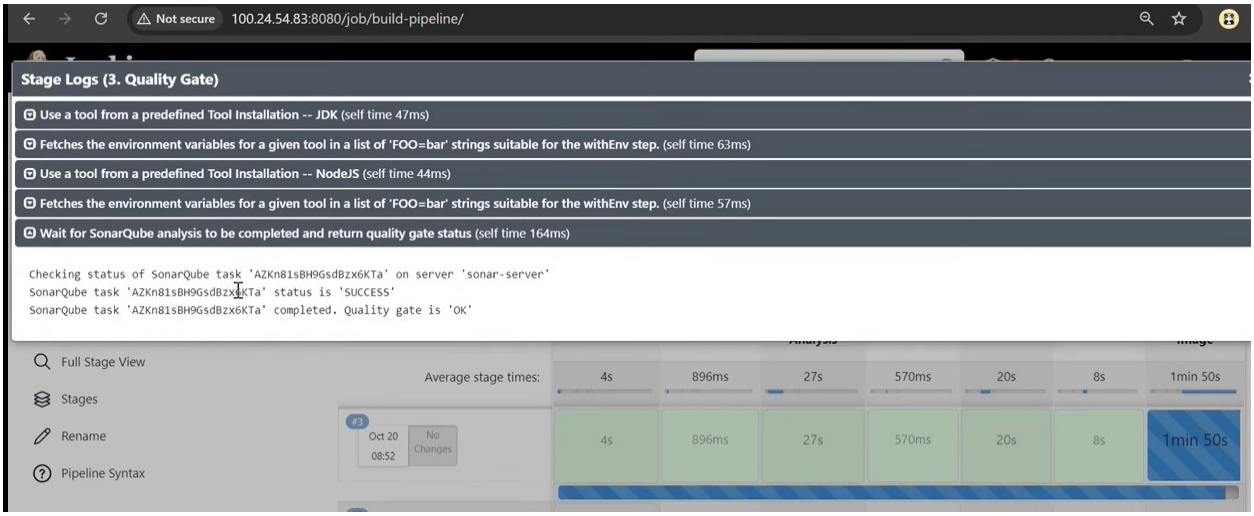
 # SonarQube Analysis



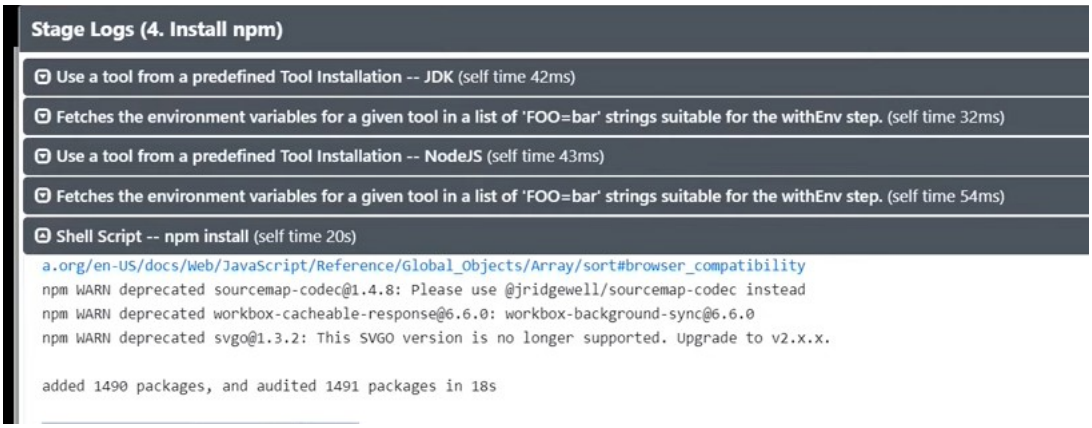
 # Go to SonarQube .. click on amazon-prime



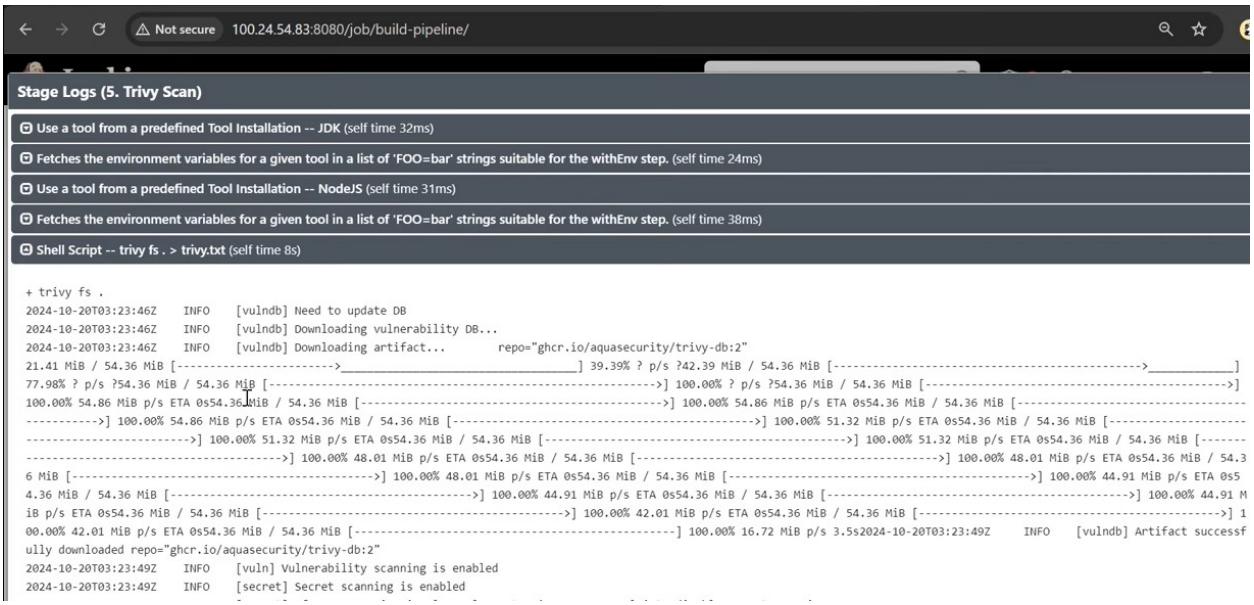
 # quality-gate



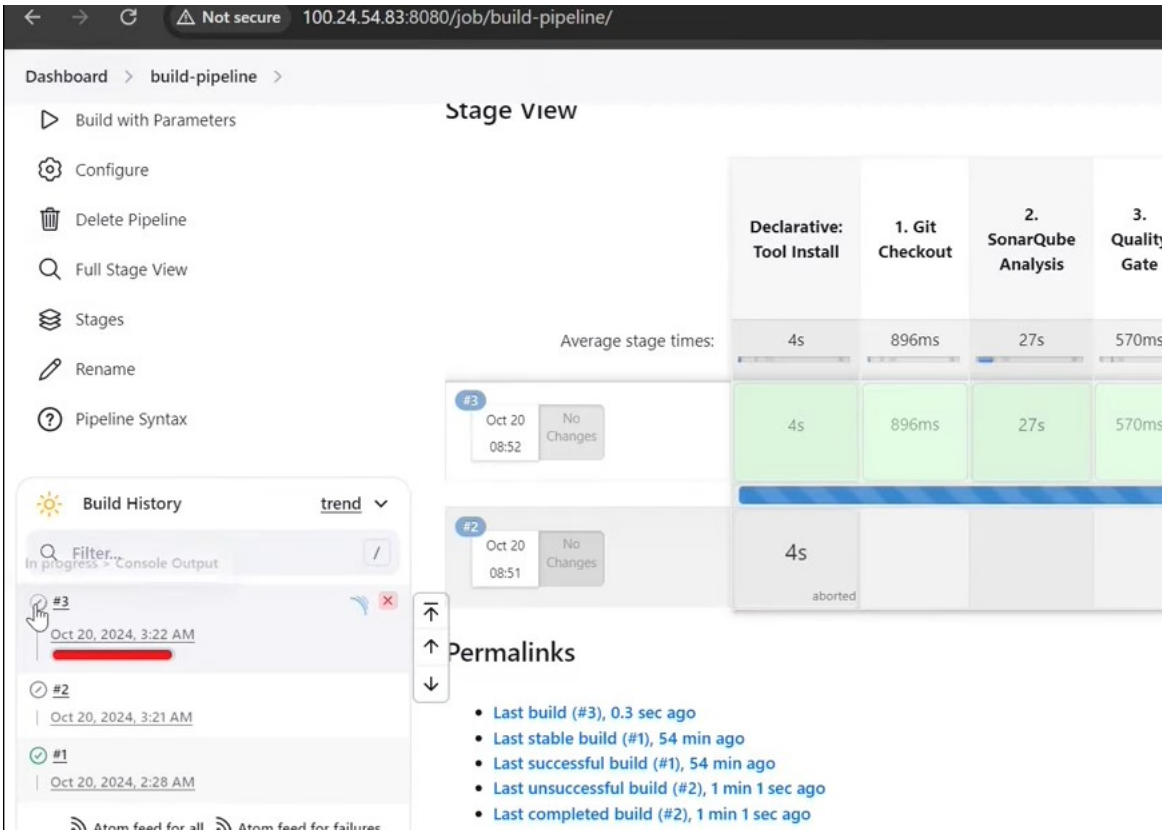
 # npm insalled



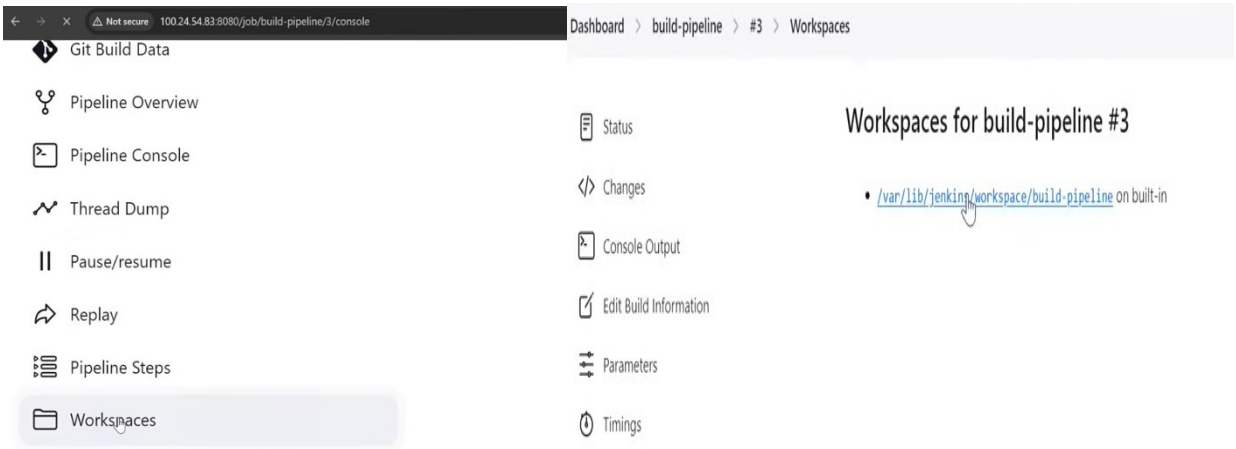
 # trivy scan

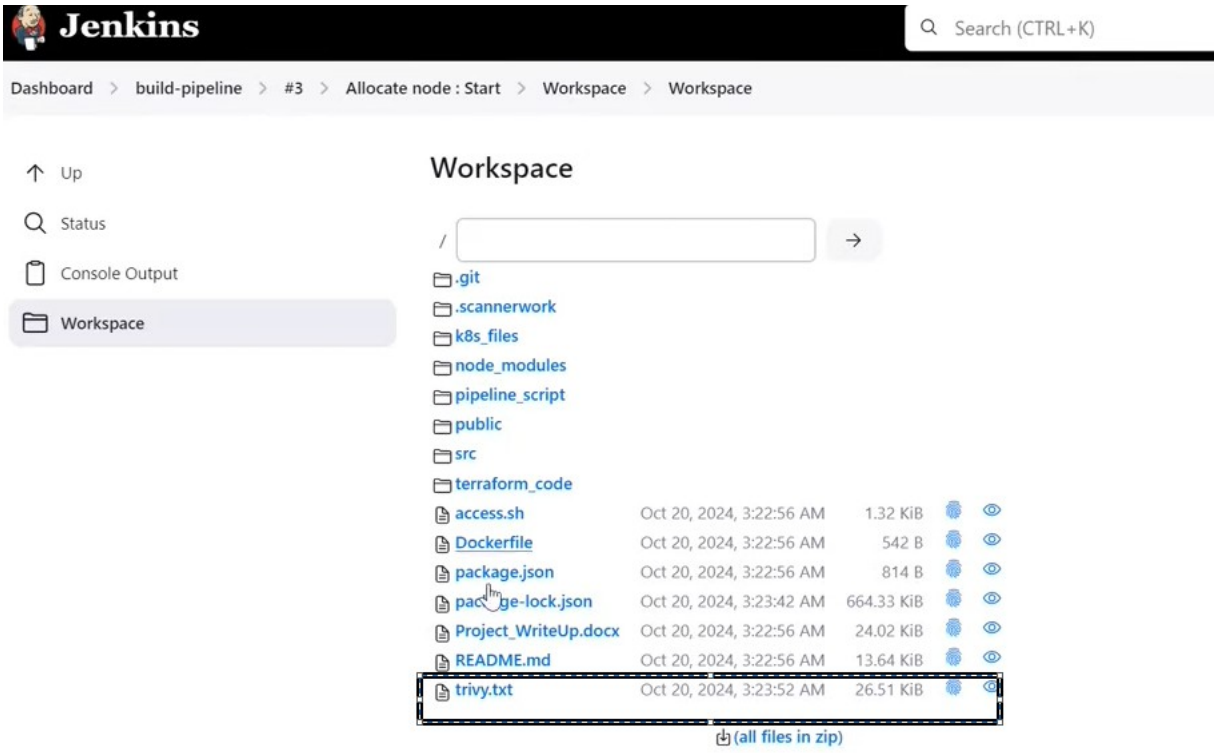


 # trivy.txt .. exsistence Click on # (Build_Number)

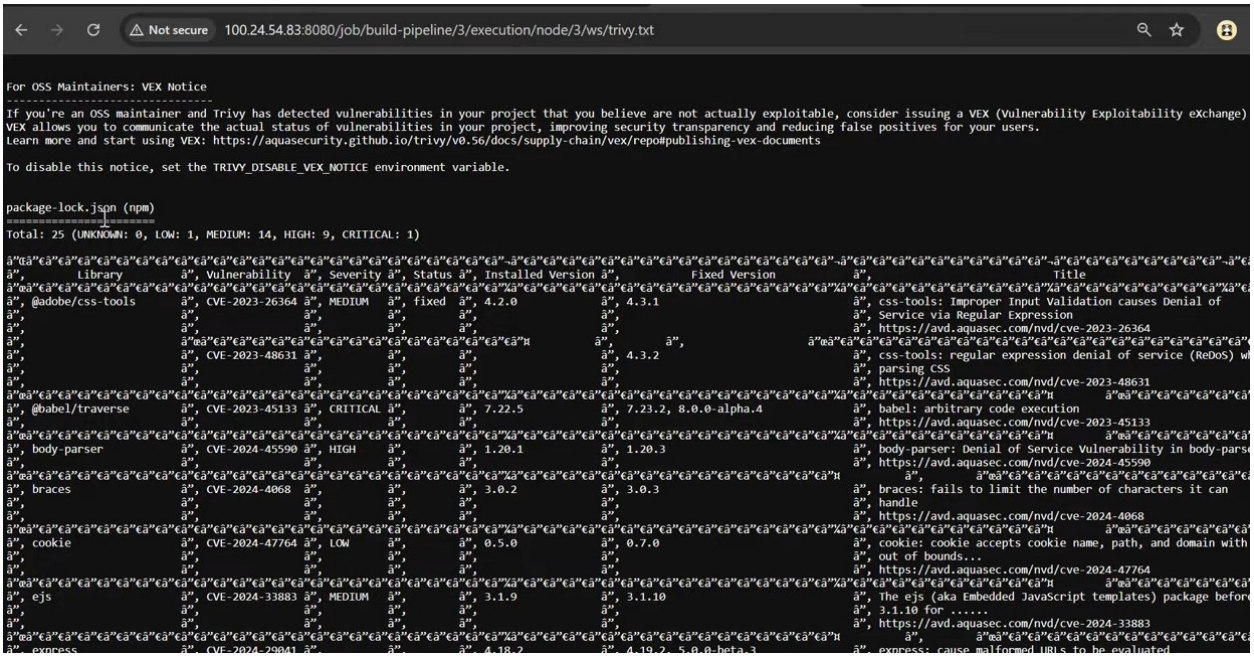

Not secure 100.24.54.83:8080/job/build-pipeline/
Dashboard > build-pipeline >
Build with Parameters
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
Build History trend
Filter...
In progress > Console Output
#3
Oct 20, 2024, 3:22 AM
#2
Oct 20, 2024, 3:21 AM
#1
Oct 20, 2024, 2:28 AM
Stage View
Declarative: Tool Install
1. Git Checkout
2. SonarQube Analysis
3. Quality Gate
Average stage times:
4s
896ms
27s
570ms
#3
Oct 20 08:52
No Changes
#2
Oct 20 08:51
No Changes
aborted
Permalinks
Last build (#3), 0.3 sec ago
Last stable build (#1), 54 min ago
Last successful build (#1), 54 min ago
Last unsuccessful build (#2), 1 min 1 sec ago
Last completed build (#2), 1 min 1 sec ago

 # Workspaces

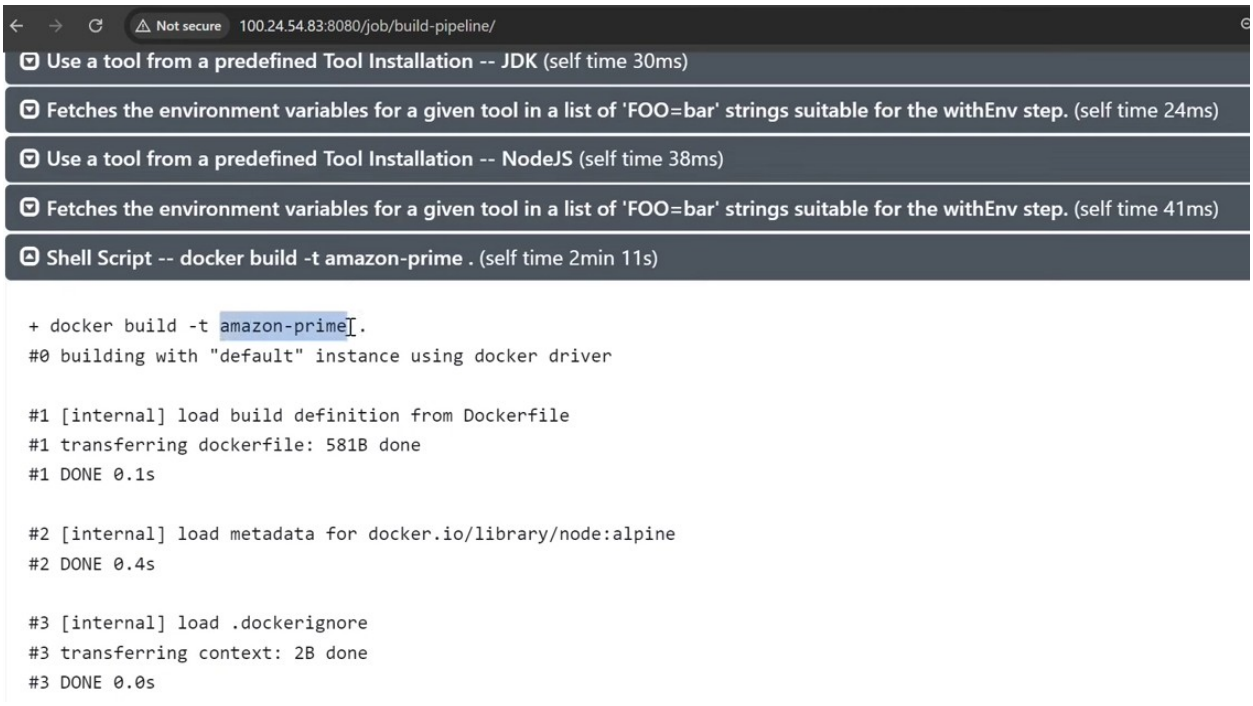

Not secure 100.24.54.83:8080/job/build-pipeline/3/console
Git Build Data
Pipeline Overview
Pipeline Console
Thread Dump
Pause/resume
Replay
Pipeline Steps
Workspaces
Dashboard > build-pipeline > #3 > Workspaces
Status
Changes
Console Output
Edit Build Information
Parameters
Timings
Workspaces for build-pipeline #3
/var/lib/jenkins/workspace/build-pipeline on built-in



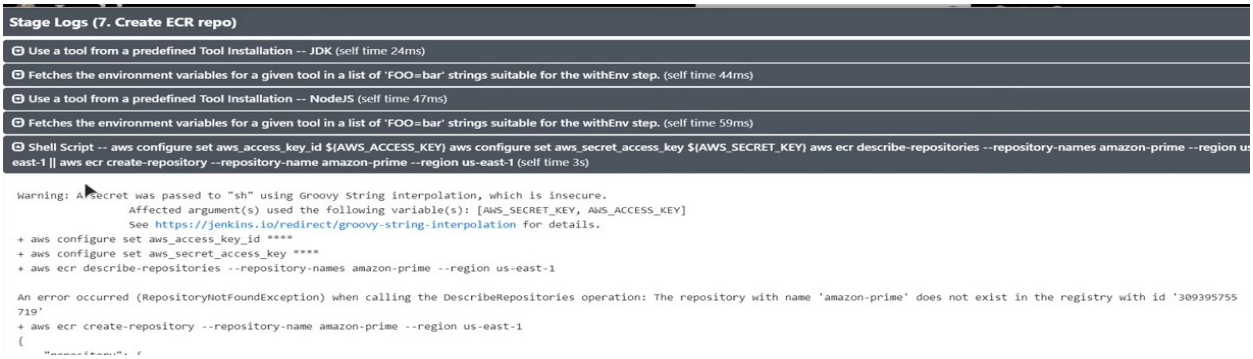
 # Open – encrypted



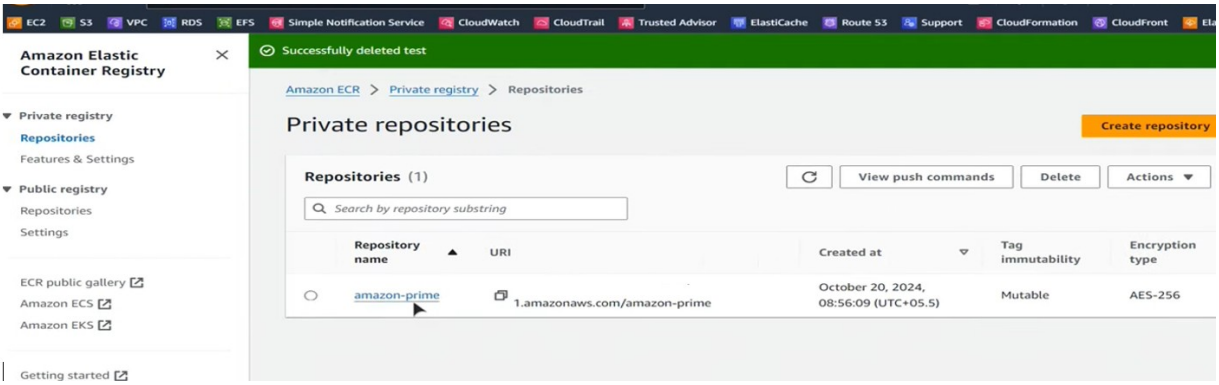
 # docker img



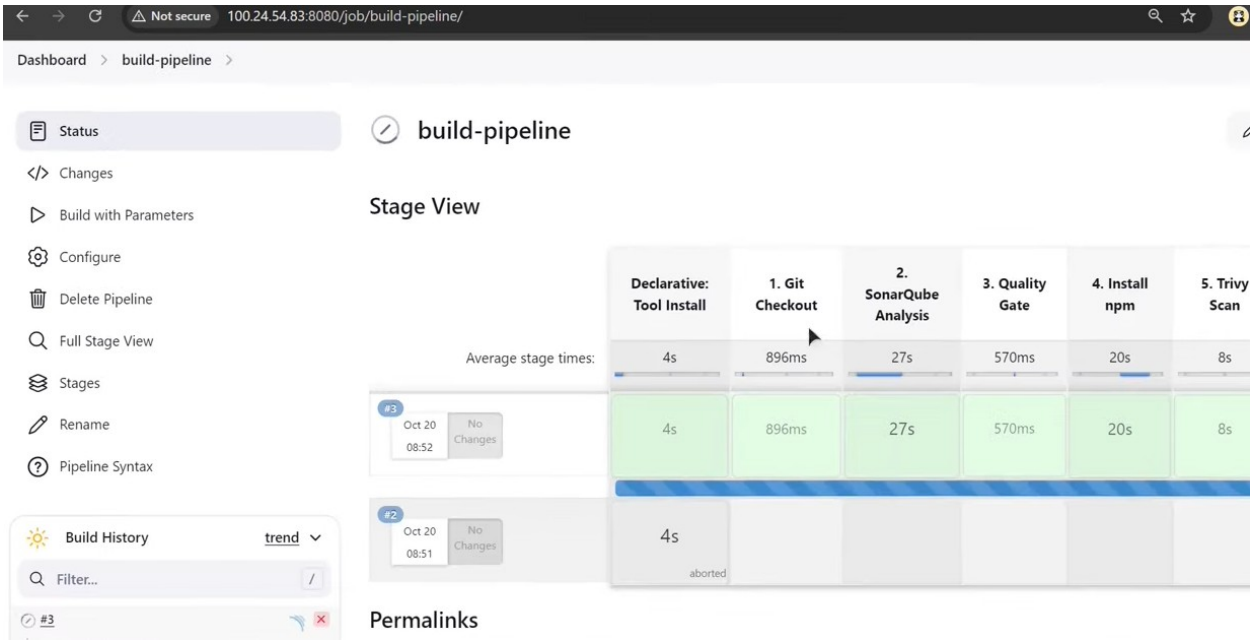
 # ECR created



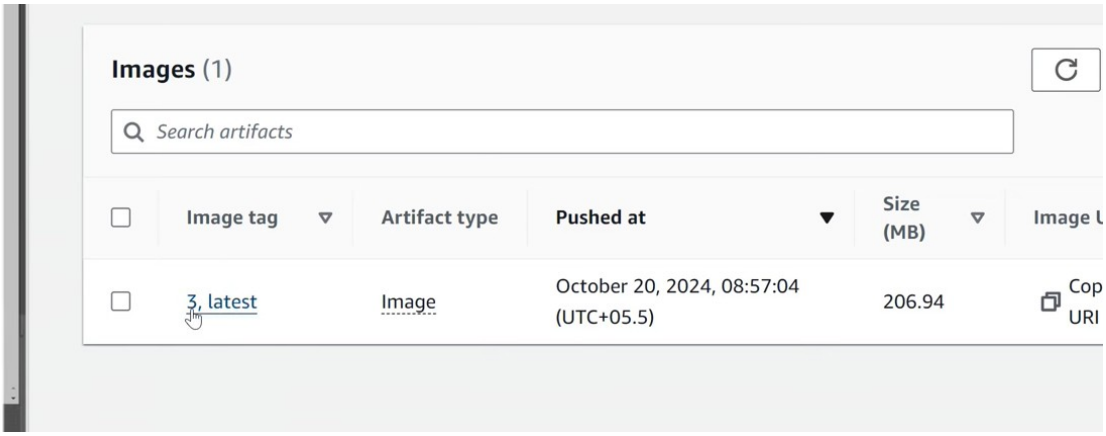
 # AWS ECR



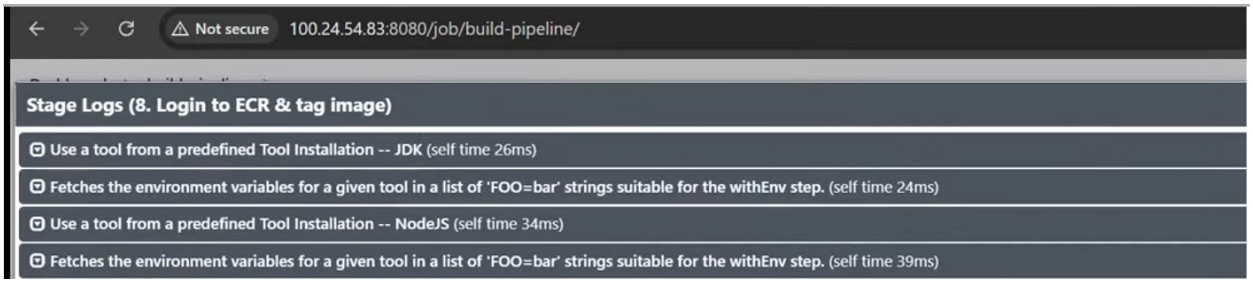
 # note Build # 3 .. here a new Repo will be created in AWS ECR with Build # 3 .. Test it Out



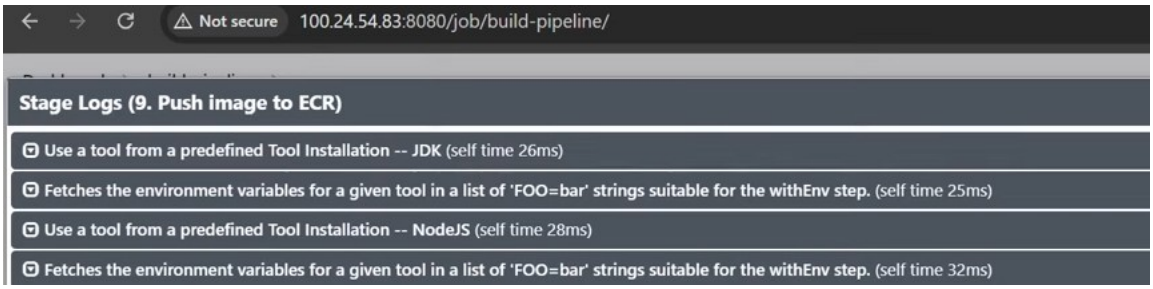
#

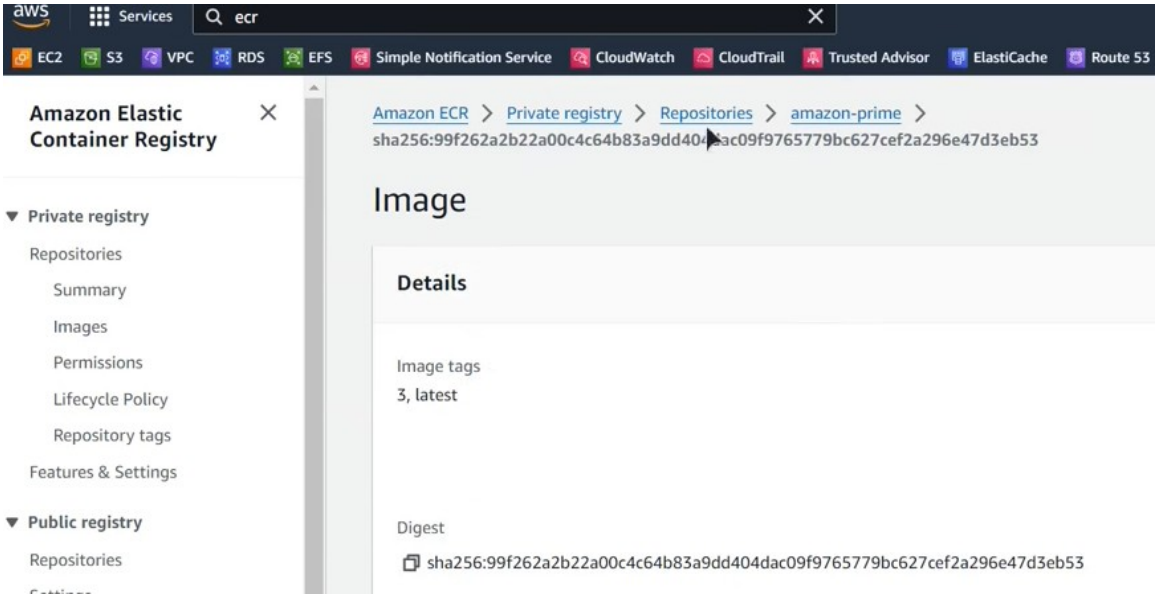


 # Login into ECR Tag Repo

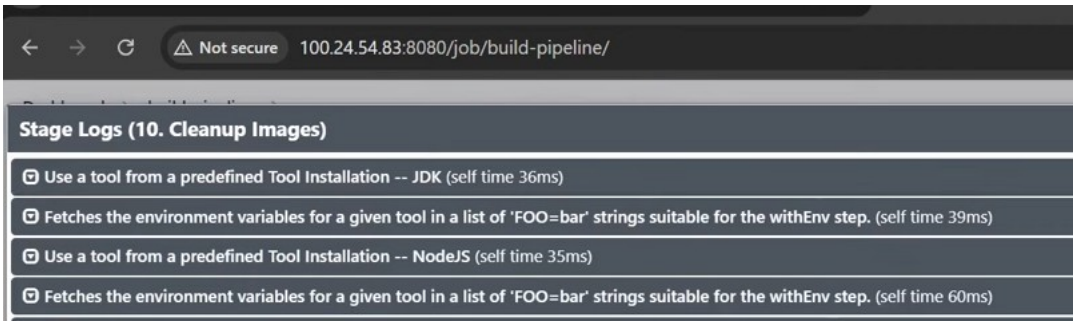


 # Push img to ECR





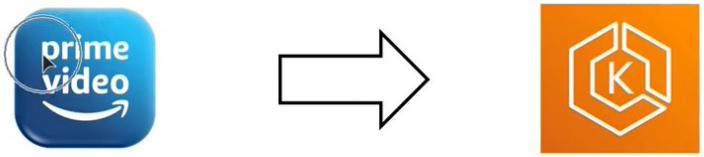
 # final – imges clean up

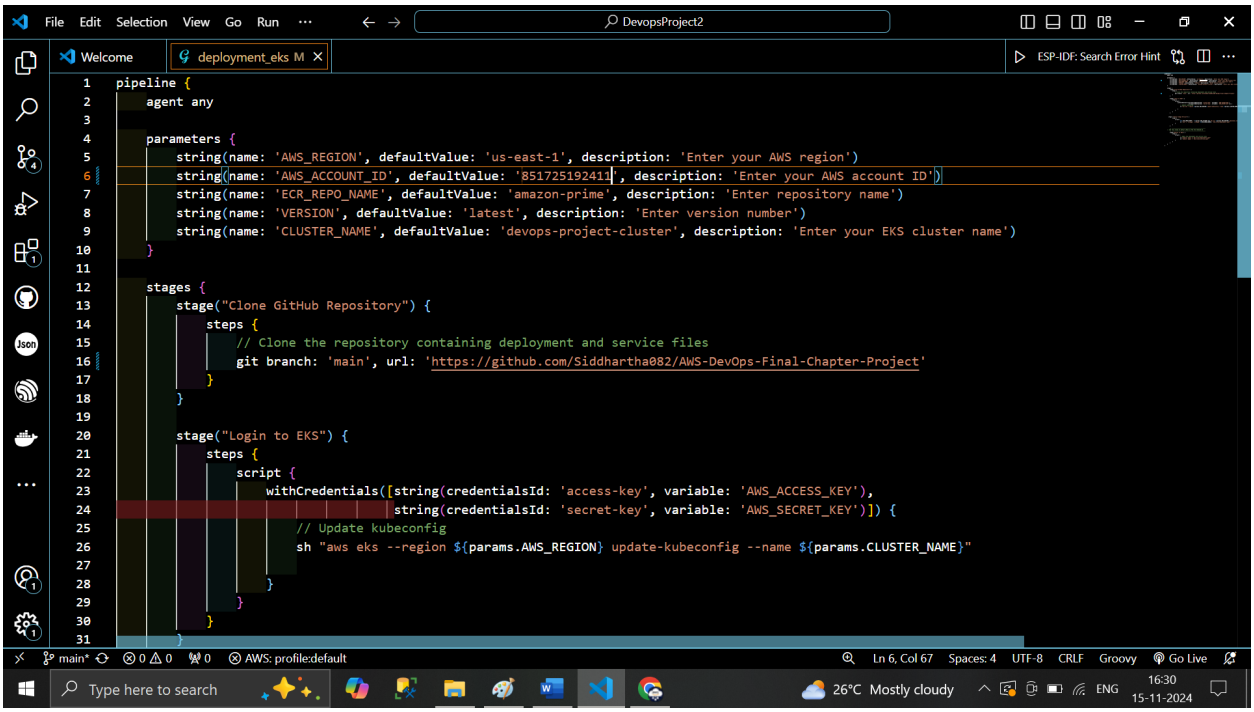


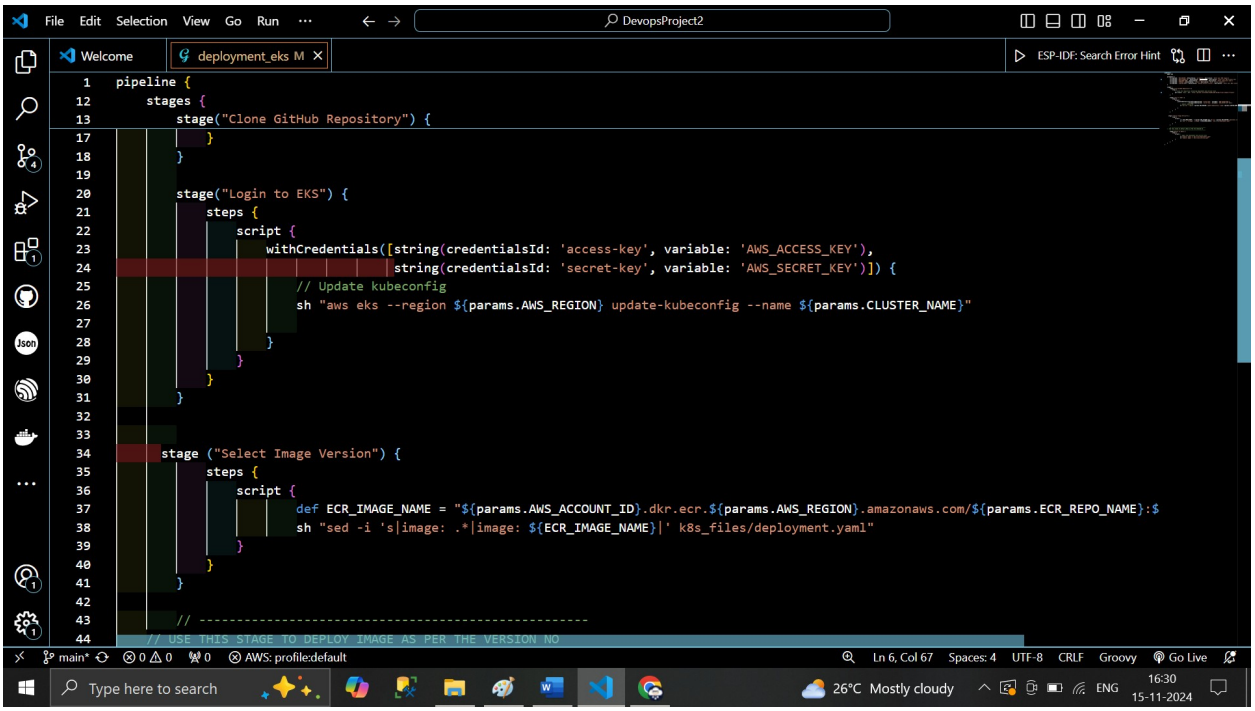

```
+ docker images
REPOSITORY        TAG             IMAGE ID       CREATED              SIZE
amazon-prime      latest          24de7cd09bb2   About a minute ago   710MB
grafana/grafana   latest          2a1d608bb4df   2 weeks ago          461MB
sonarqube         lts-community   e88d8cc07644   2 weeks ago          605MB
prom/prometheus   latest          4022a502929b   7 weeks ago          275MB
```

 # 2nd PART DEPLOYMENT ---- CD PART

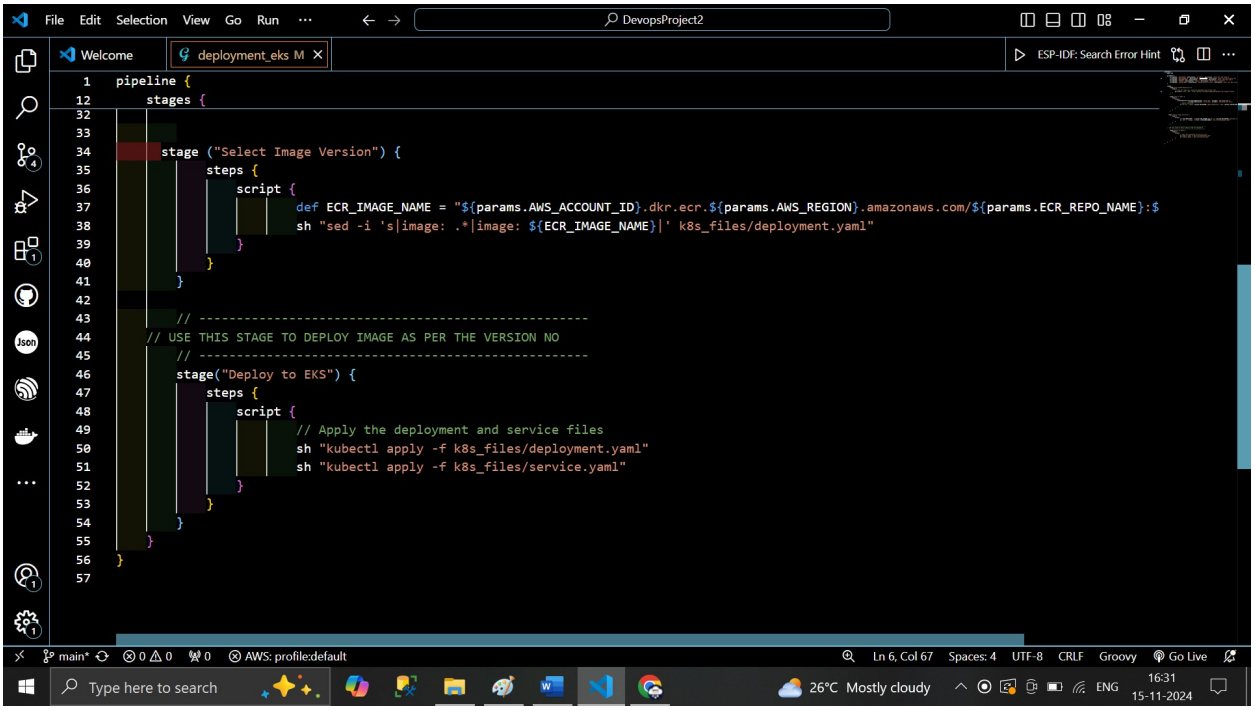
RUN JENKINS RELEASE PIPELINE





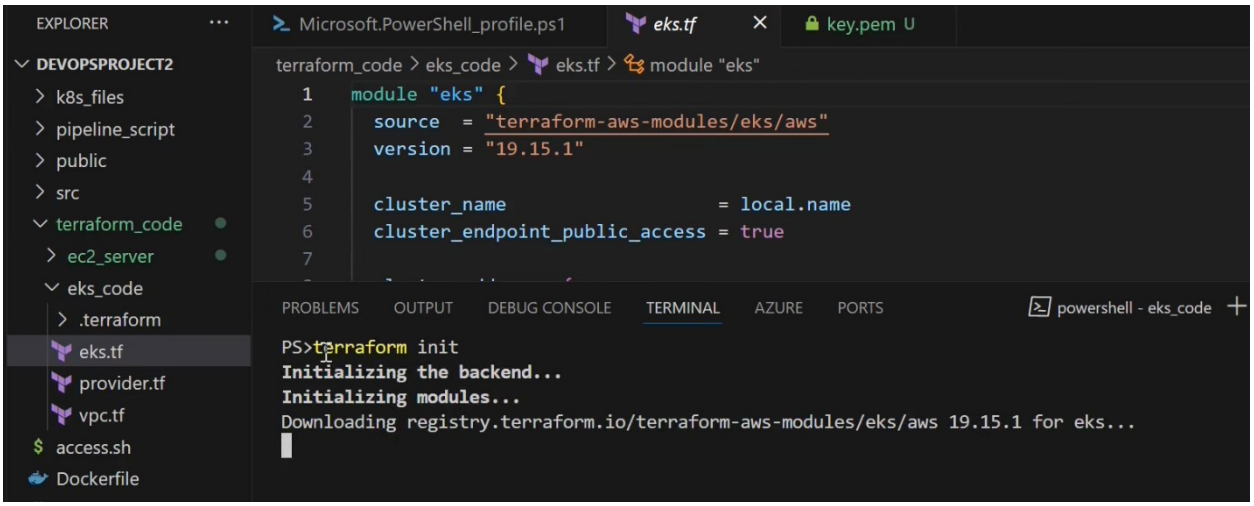


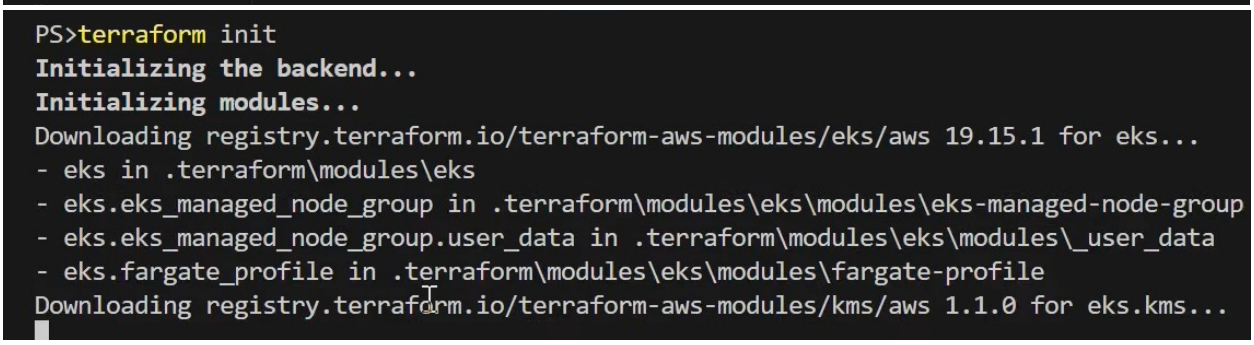
```
pipeline {
    stages {
        stage("Clone GitHub Repository") {
        }
        }

        stage("Login to EKS") {
            steps {
                script {
                    withCredentials([string(credentialsId: 'access-key', variable: 'AWS_ACCESS_KEY'),
                                     string(credentialsId: 'secret-key', variable: 'AWS_SECRET_KEY')]) {
                        // Update kubeconfig
                        sh "aws eks --region ${params.AWS_REGION} update-kubeconfig --name ${params.CLUSTER_NAME}"

                    }
                }
            }
        }


    stage ("Select Image Version") {
            steps {
                script {
                    def ECR_IMAGE_NAME = "${params.AWS_ACCOUNT_ID}.dkr.ecr.${params.AWS_REGION}.amazonaws.com/${params.ECR_REPO_NAME}:$
                    sh "sed -i 's|image: .*|image: ${ECR_IMAGE_NAME}|' k8s_files/deployment.yaml"
                }
            }
        }

        // ------------------------------------------------------
    // USE THIS STAGE TO DEPLOY IMAGE AS PER THE VERSION NO
```

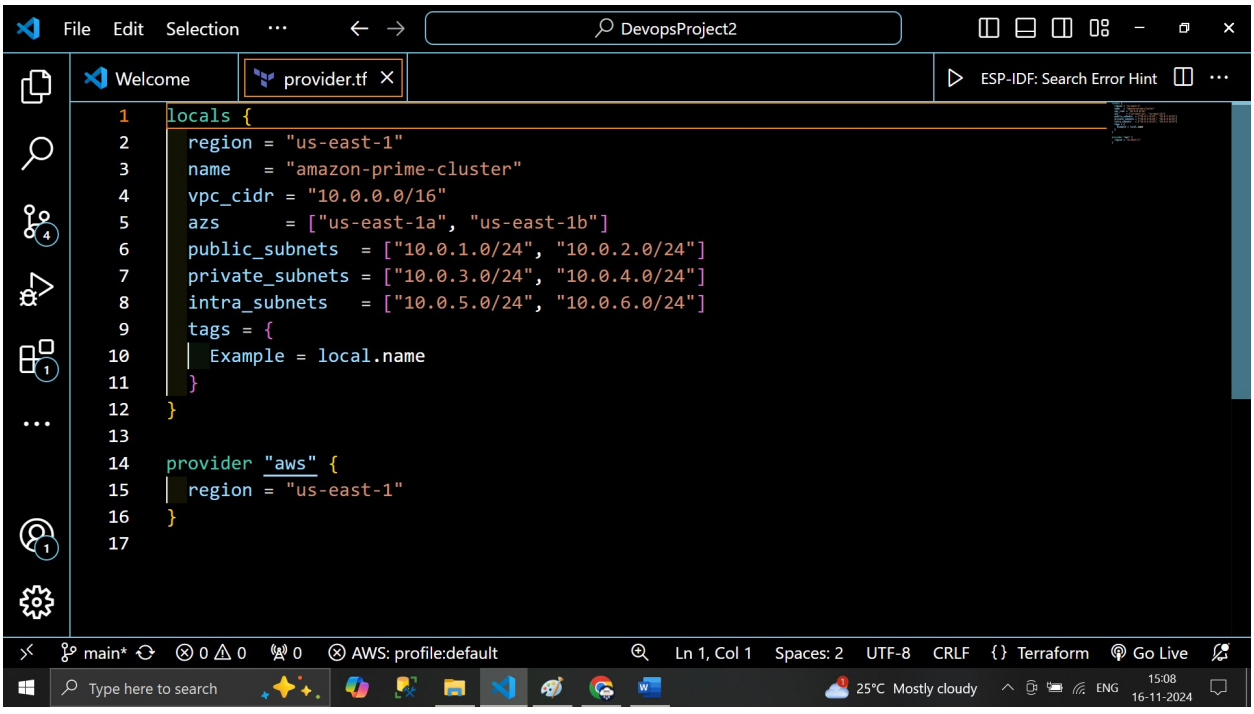


```
pipeline {
    stages {


    stage ("Select Image Version") {
            steps {
                script {
                    def ECR_IMAGE_NAME = "${params.AWS_ACCOUNT_ID}.dkr.ecr.${params.AWS_REGION}.amazonaws.com/${params.ECR_REPO_NAME}:$
                    sh "sed -i 's|image: .*|image: ${ECR_IMAGE_NAME}|' k8s_files/deployment.yaml"
                }
            }
        }

        // ------------------------------------------------------
    // USE THIS STAGE TO DEPLOY IMAGE AS PER THE VERSION NO
        // ------------------------------------------------------
        stage("Deploy to EKS") {
            steps {
                script {
                    // Apply the deployment and service files
                    sh "kubectl apply -f k8s_files/deployment.yaml"
                    sh "kubectl apply -f k8s_files/service.yaml"
                }
            }
        }
    }
}
```

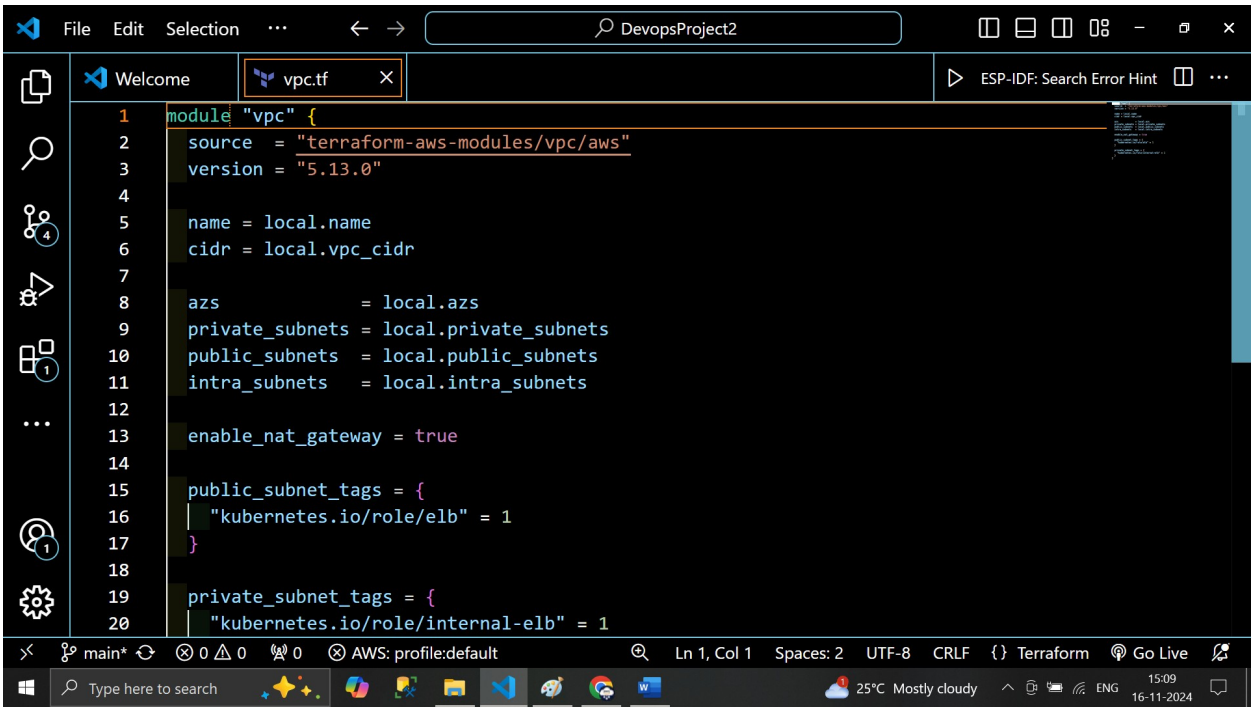
 # Go to eks.ft file in terraform folder ...

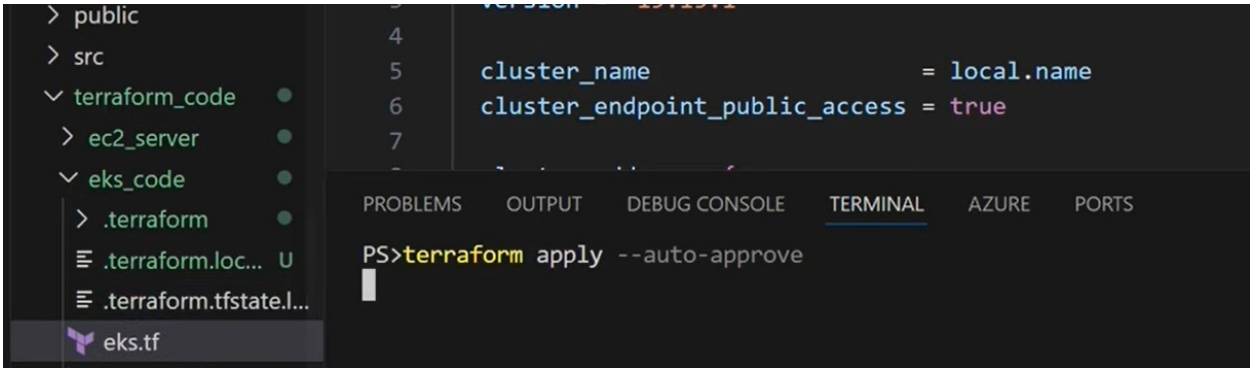

EXPLORER
Microsoft.PowerShell_profile.ps1
eks.tf
key.pem U
DEVOPSPROJECT2
k8s_files
pipeline_script
public
src
terraform_code
ec2_server
eks_code
.terraform
eks.tf
provider.tf
vpc.tf
access.sh
Dockerfile
terraform_code > eks_code > eks.tf > module "eks"
module "eks" {
source = "terraform-aws-modules/eks/aws"
version = "19.15.1"
cluster_name = local.name
cluster_endpoint_public_access = true
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
AZURE
PORTS
powershell - eks_code
PS>terraform init
Initializing the backend...
Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/eks/aws 19.15.1 for eks...


PS>terraform init
Initializing the backend...
Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/eks/aws 19.15.1 for eks...
- eks in .terraform\modules\eks
- eks.eks_managed_node_group in .terraform\modules\eks\modules\eks-managed-node-group
- eks.eks_managed_node_group.user_data in .terraform\modules\eks\modules_user_data
- eks.fargate_profile in .terraform\modules\eks\modules\fargate-profile
Downloading registry.terraform.io/terraform-aws-modules/kms/aws 1.1.0 for eks.kms...

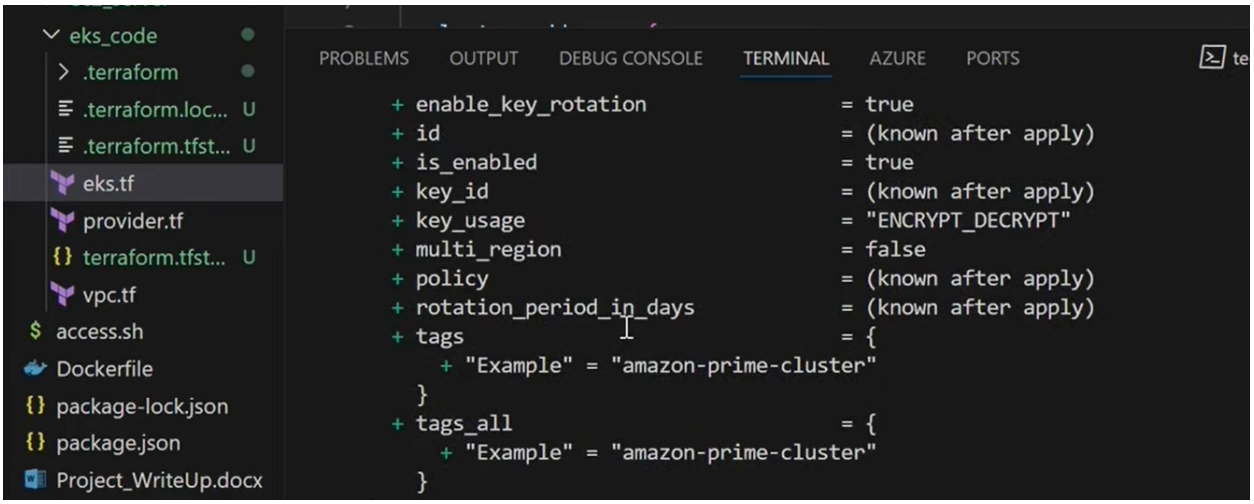

File
Edit
Selection
DevopsProject2
Welcome
provider.tf
ESP-IDF: Search Error Hint
locals {
region = "us-east-1"
name = "amazon-prime-cluster"
vpc_cidr = "10.0.0.0/16"
azs = ["us-east-1a", "us-east-1b"]
public_subnets = ["10.0.1.0/24", "10.0.2.0/24"]
private_subnets = ["10.0.3.0/24", "10.0.4.0/24"]
intra_subnets = ["10.0.5.0/24", "10.0.6.0/24"]
tags = {
Example = local.name
}
}
provider "aws" {
region = "us-east-1"
}
main*
AWS: profile:default
Ln 1, Col 1
Spaces: 2
UTF-8
CRLF
Terraform
Go Live
Type here to search
25°C Mostly cloudy
ENG
15:08
16-11-2024

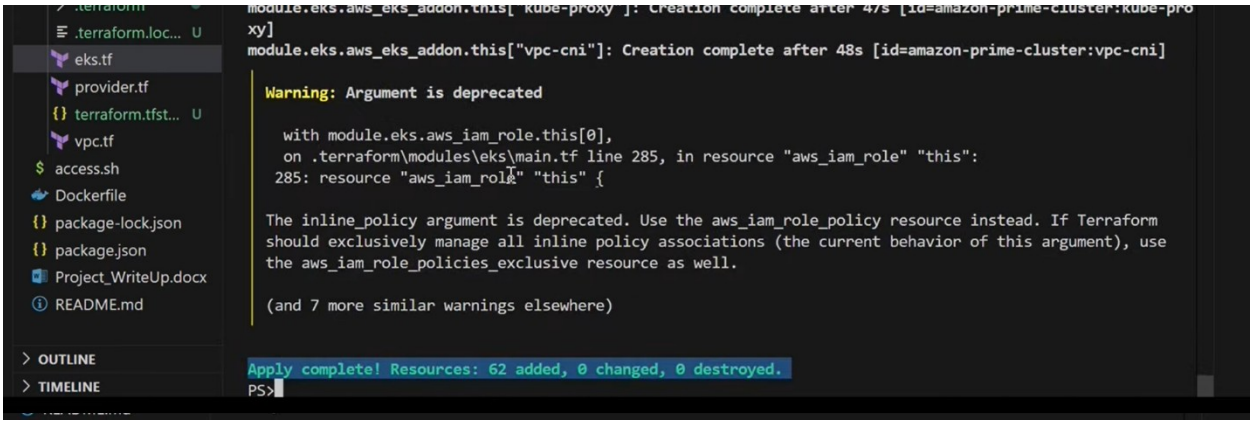
 # VPC

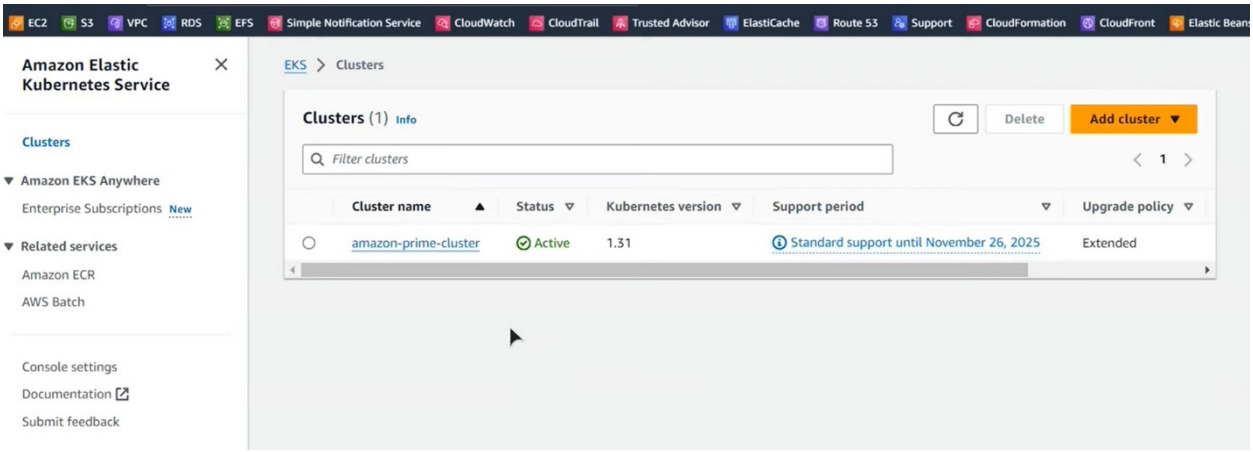


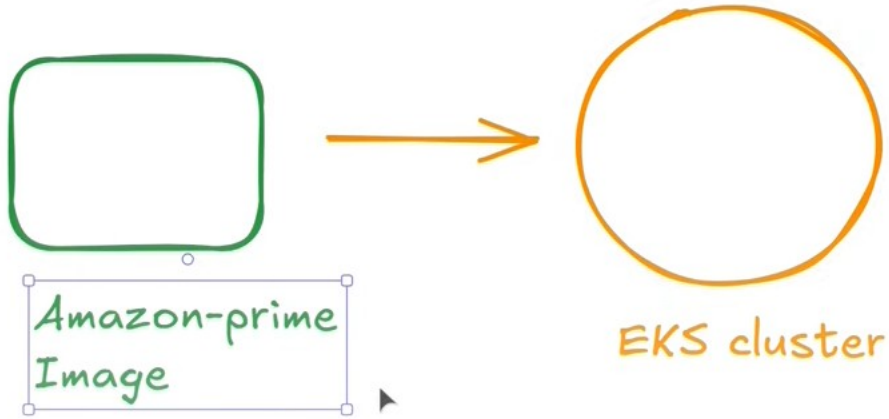


 # Eks Cluster & nodegroup

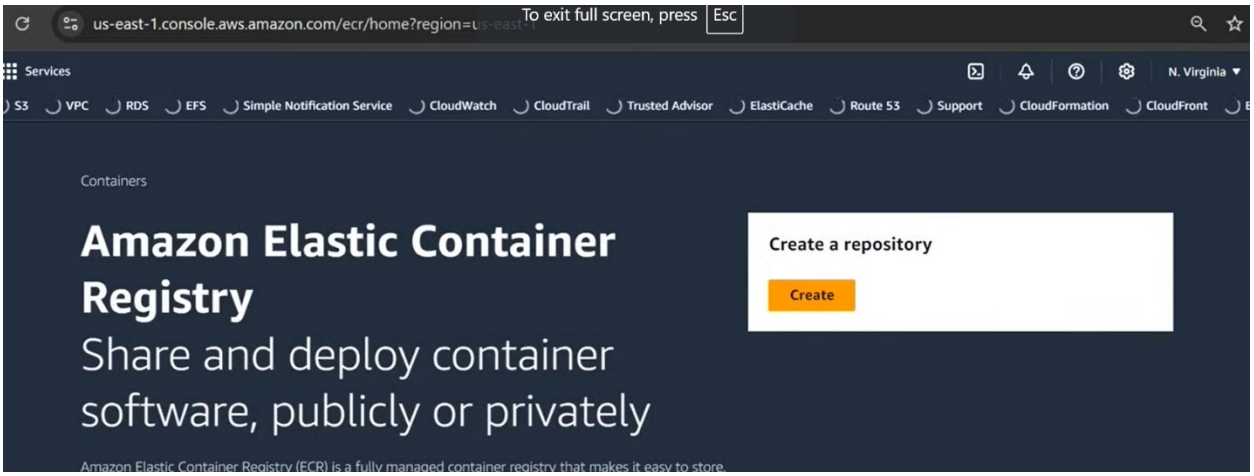


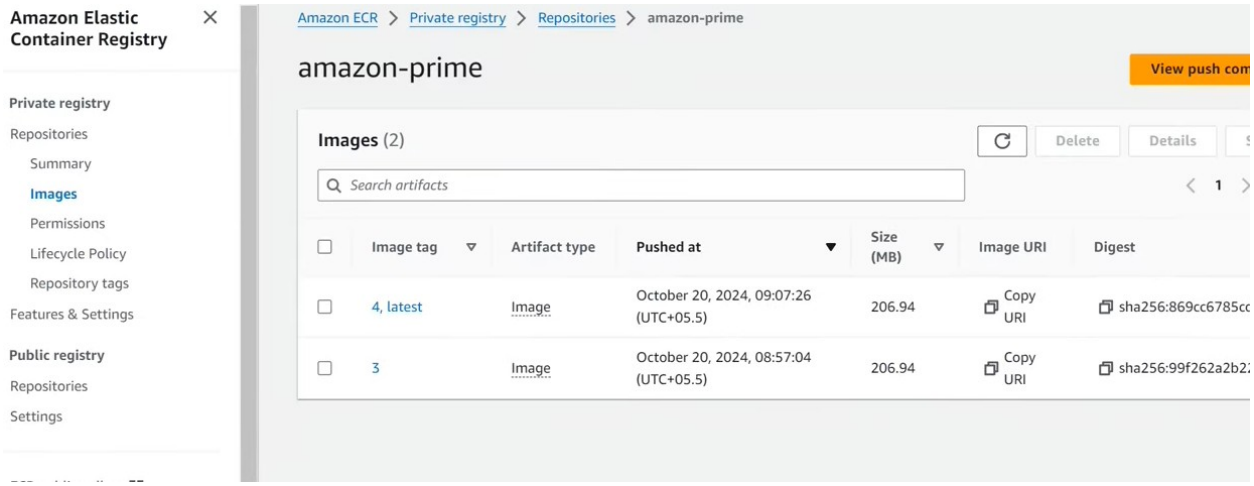

.terraform.loc... U
eks.tf
provider.tf
terraform.tfst... U
vpc.tf
access.sh
Dockerfile
package-lock.json
package.json
Project_WriteUp.docx
README.md
OUTLINE
TIMELINE
xy]
module.eks.aws_eks_addon.this["vpc-cni"]: Creation complete after 48s [id=amazon-prime-cluster:vpc-cni]
Warning: Argument is deprecated
with module.eks.aws_iam_role.this[0],
on .terraform\modules\eks\main.tf line 285, in resource "aws_iam_role" "this":
285: resource "aws_iam_role" "this" {
The inline_policy argument is deprecated. Use the aws_iam_role_policy resource instead. If Terraform should exclusively manage all inline policy associations (the current behavior of this argument), use the aws_iam_role_policies_exclusive resource as well.
(and 7 more similar warnings elsewhere)
Apply complete! Resources: 62 added, 0 changed, 0 destroyed.
PS>

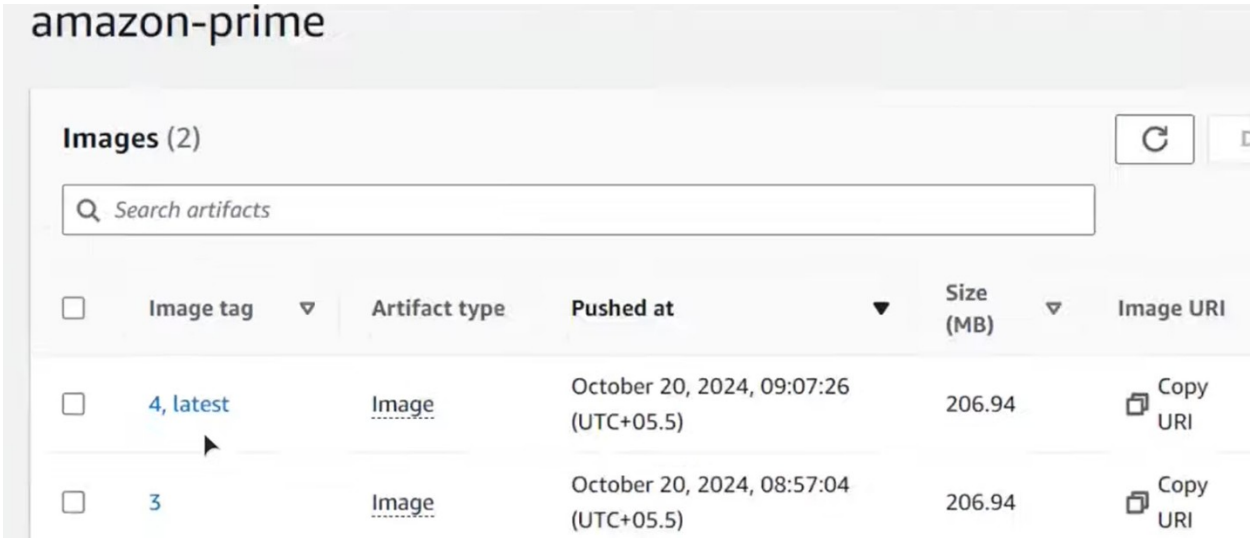

EC2 S3 VPC RDS EFS Simple Notification Service CloudWatch CloudTrail Trusted Advisor ElastiCache Route 53 Support CloudFormation CloudFront
Amazon Elastic Kubernetes Service
Clusters
Amazon EKS Anywhere
Enterprise Subscriptions New
Related services
Amazon ECR
AWS Batch
Console settings
Documentation
Submit feedback
EKS > Clusters
Clusters (1) Info
Delete
Add cluster
Filter clusters
Cluster name
Status
Kubernetes version
Support period
Upgrade policy
amazon-prime-cluster
Active
1.31
Standard support until November 26, 2025
Extended

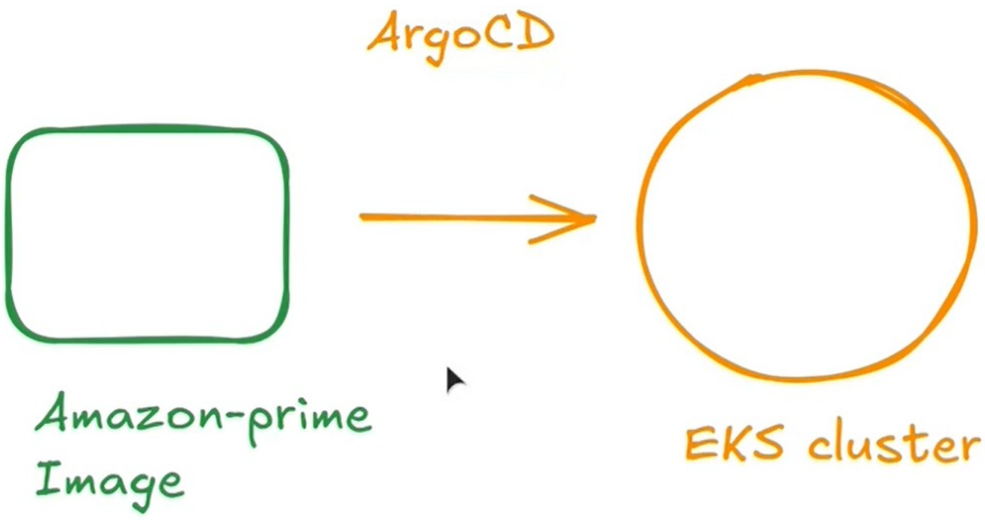

Amazon-prime Image
EKS cluster

 # Check Repo


To exit full screen, press Esc
us-east-1.console.aws.amazon.com/ecr/home?region=
Services
N. Virginia
S3 VPC RDS EFS Simple Notification Service CloudWatch CloudTrail Trusted Advisor ElastiCache Route 53 Support CloudFormation CloudFront
Containers
Amazon Elastic Container Registry
Share and deploy container software, publicly or privately
Amazon Elastic Container Registry (ECR) is a fully managed container registry that makes it easy to store
Create a repository
Create


Amazon Elastic Container Registry
Private registry
Repositories
Summary
Images
Permissions
Lifecycle Policy
Repository tags
Features & Settings
Public registry
Repositories
Settings
Amazon ECR > Private registry > Repositories > amazon-prime
amazon-prime
View push com
Images (2)
Delete
Details
Search artifacts
1
Image tag
Artifact type
Pushed at
Size (MB)
Image URI
Digest
4, latest
Image
October 20, 2024, 09:07:26 (UTC+05.5)
206.94
Copy URI
sha256:869cc6785cc
3
Image
October 20, 2024, 08:57:04 (UTC+05.5)
206.94
Copy URI
sha256:99f262a2b2


amazon-prime
Images (2)
Search artifacts
Image tag
Artifact type
Pushed at
Size (MB)
Image URI
4, latest
Image
October 20, 2024, 09:07:26 (UTC+05.5)
206.94
Copy URI
3
Image
October 20, 2024, 08:57:04 (UTC+05.5)
206.94
Copy URI



 # Deployment Pipeline

```
pipeline {
    agent any

    environment {
        KUBECTL = '/usr/local/bin/kubectl'
    }

    parameters {
        string(name: 'CLUSTER_NAME', defaultValue: 'amazon-prime-cluster', description: 'Enter your EKS cluster n
    }

    stages {
        stage("Login to EKS") {
            steps {
                script {
                    withCredentials([string(credentialsId: 'access-key', variable: 'AWS_ACCESS_KEY'),
                                     string(credentialsId: 'secret-key', variable: 'AWS_SECRET_KEY')]) {
                        sh "aws eks --region us-east-1 update-kubeconfig --name ${params.CLUSTER_NAME}"
                    }
```

 # install helm chart for Prometheus + Grafana

```
stage("Configure Prometheus & Grafana") {
    steps {
        script {
            sh """
            helm repo add stable https://charts.helm.sh/stable || true
            helm repo add prometheus-community https://prometheus-community.github.io/helm-charts || true
            # Check if namespace 'prometheus' exists
            if kubectl get namespace prometheus > /dev/null 2>&1; then
                # If namespace exists, upgrade the Helm release
                helm upgrade stable prometheus-community/kube-prometheus-stack -n prometheus
            else
                # If namespace does not exist, create it and install Helm release
                kubectl create namespace prometheus
                helm install stable prometheus-community/kube-prometheus-stack -n prometheus
            fi
            kubectl patch svc stable-kube-prometheus-sta-prometheus -n prometheus -p '{"spec": {"type": "
            kubectl patch svc stable-grafana -n prometheus -p '{"spec": {"type": "LoadBalancer"}}'
            """
```

ACCESS GRAFANA, PROMETHEUS, ARGOCD

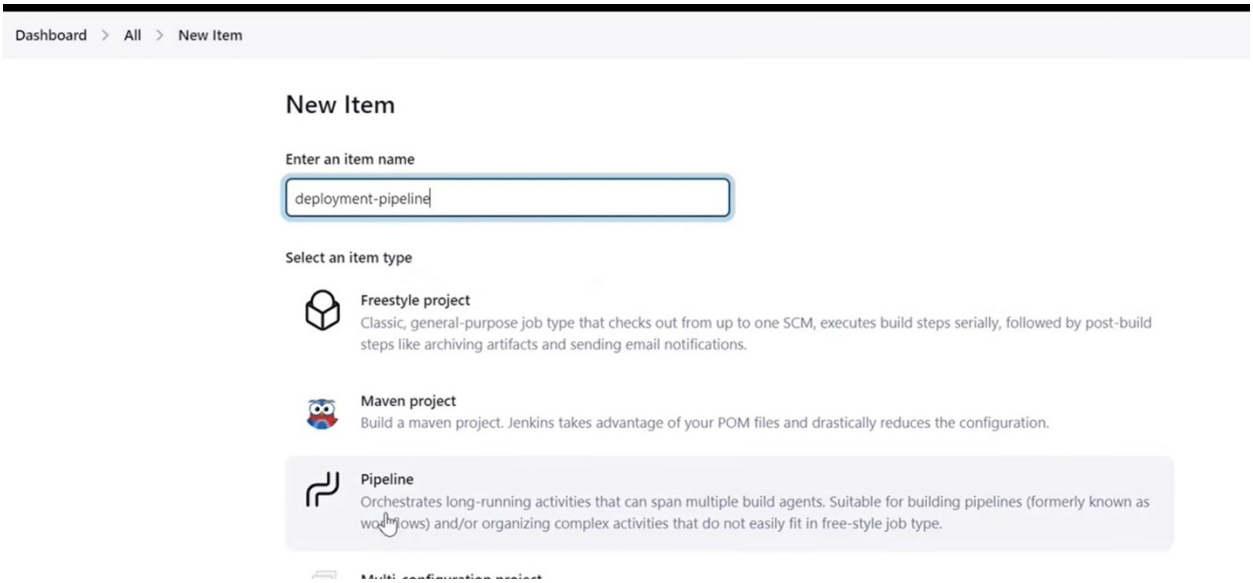
 # Apply the ArgoCd manifest… Change the type from Nodeport to LoadBalancer

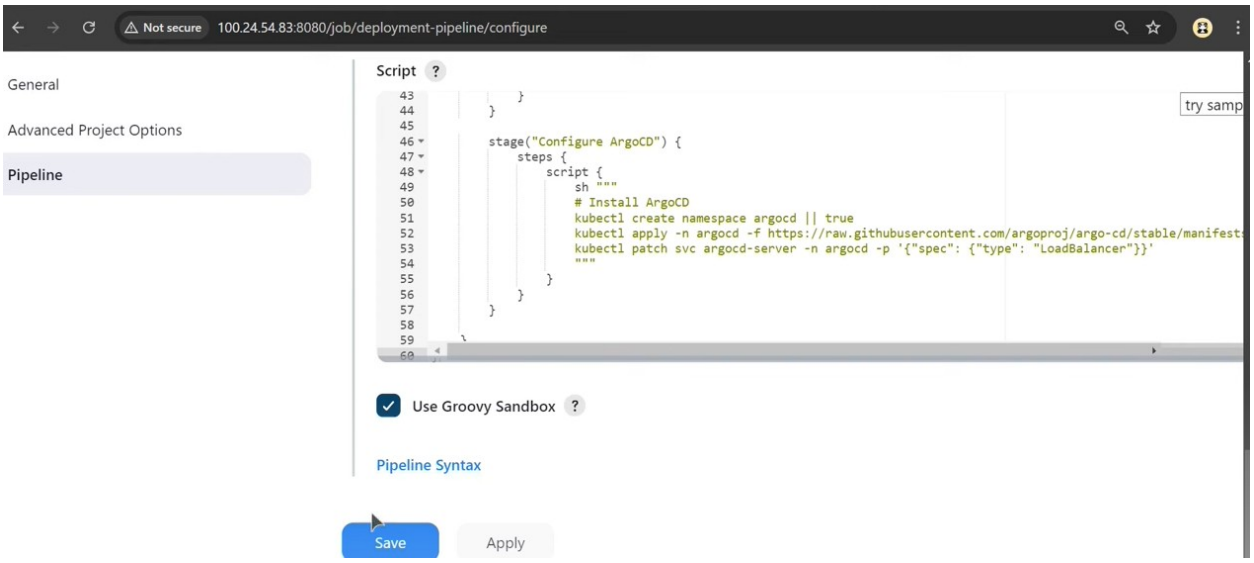
```
    stage("Configure ArgoCD") {
        steps {
            script {
                sh """
                # Install ArgoCD
                kubectl create namespace argocd || true
                kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifest
                kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
                """
            }
        }
    }
}
```

 # load balancer to Prometheus + Grafana

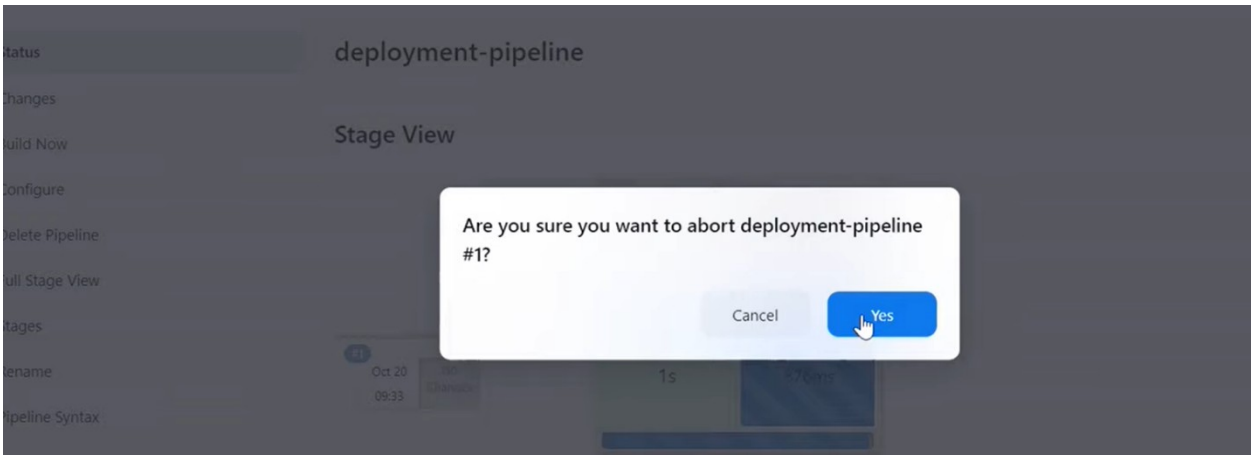
```
            fi
            kubectl patch svc stable-kube-prometheus-sta-prometheus -n prometheus -p '{"spec":{"type":"
            kubectl patch svc stable-grafana -n prometheus -p '{"spec":{"type":"LoadBalancer"}}'
            """
        }
    }
}
```

 # Go Back to Jenkins – Cp the deployment.yml file paste it on jenkins -----select Pipeline

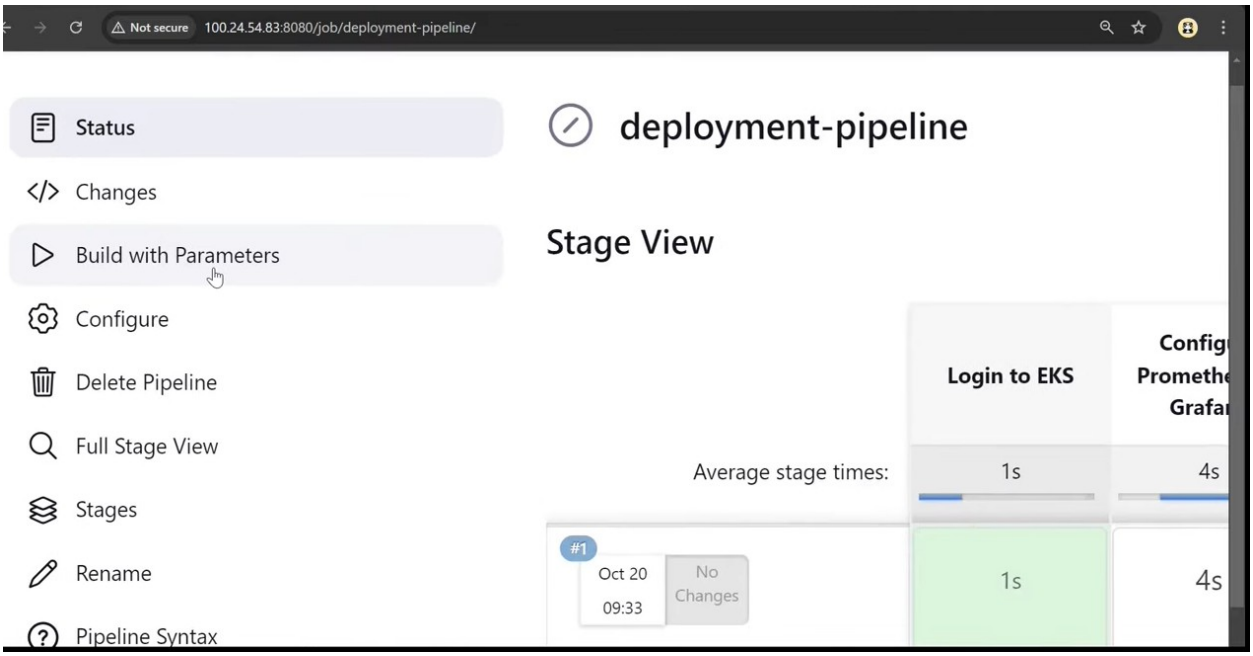


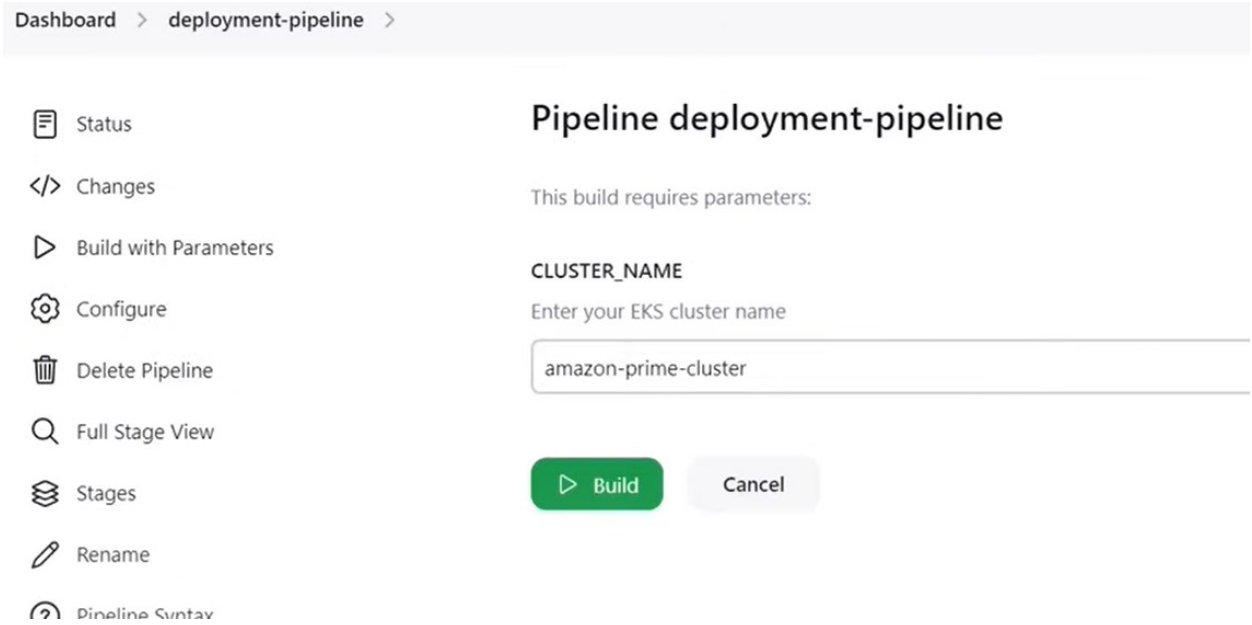


 # Abort it

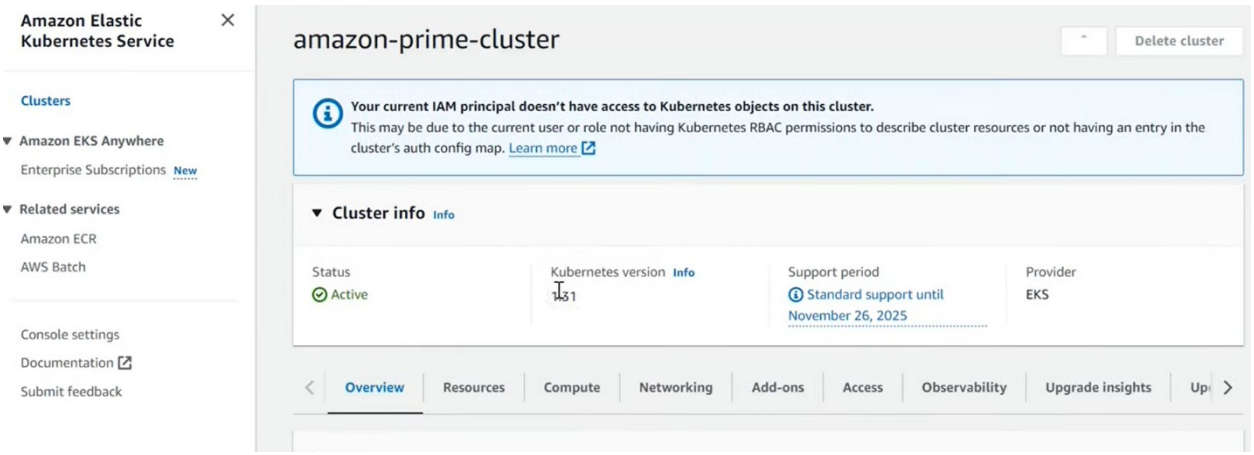


 # Again Refresh it as we need Build with Paramters

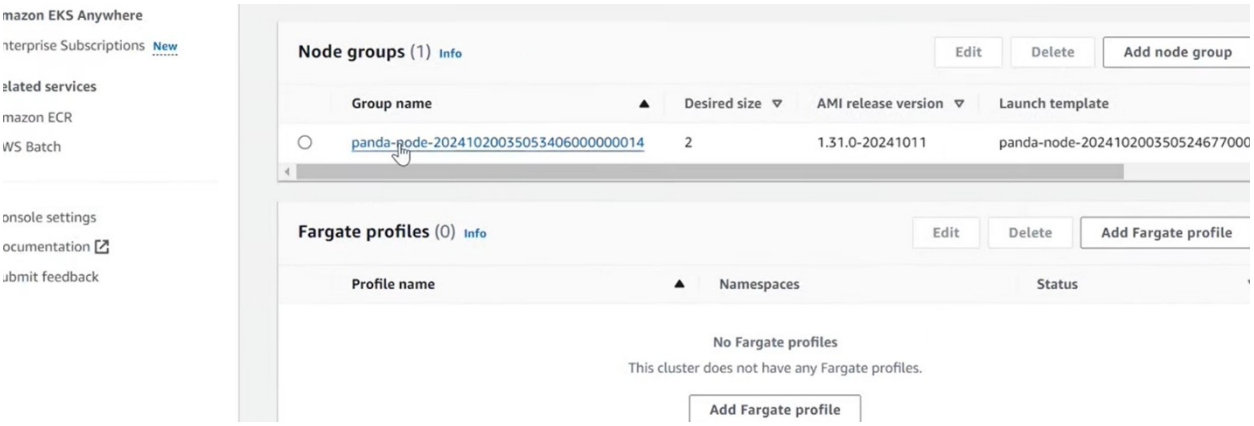

Not secure
100.24.54.83:8080/job/deployment-pipeline/
Status
Changes
Build with Parameters
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
deployment-pipeline
Stage View
Login to EKS
Average stage times:
1s
4s
#1
Oct 20
09:33
No Changes
1s
4s


Dashboard
deployment-pipeline
Status
Changes
Build with Parameters
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
Pipeline deployment-pipeline
This build requires parameters:
CLUSTER_NAME
Enter your EKS cluster name
amazon-prime-cluster
Build
Cancel

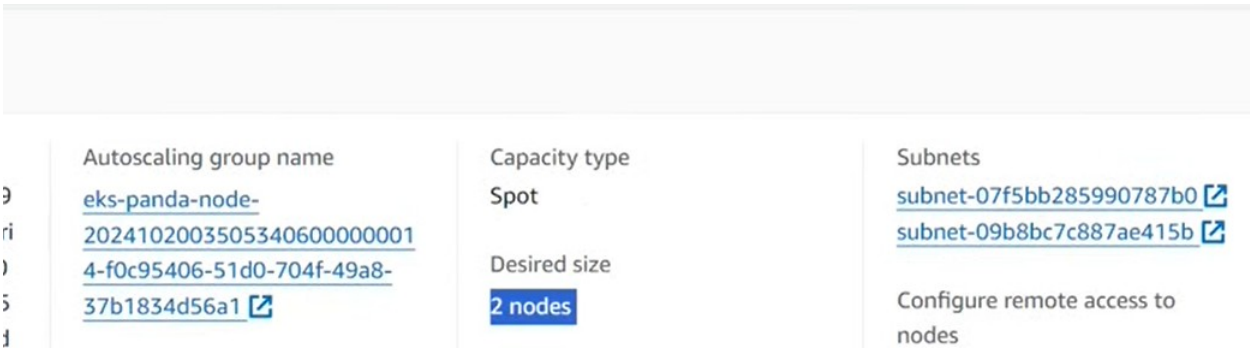
 # go to EKS



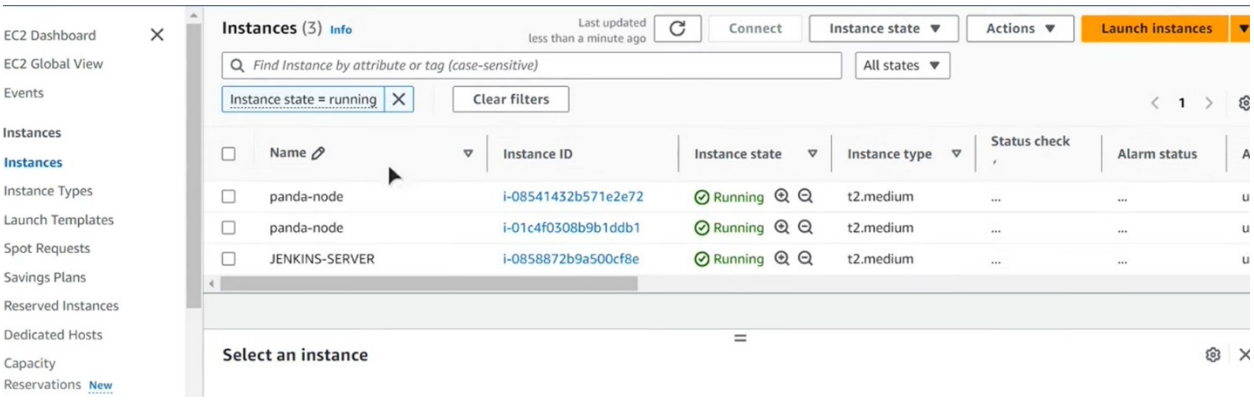
 # Click Compute



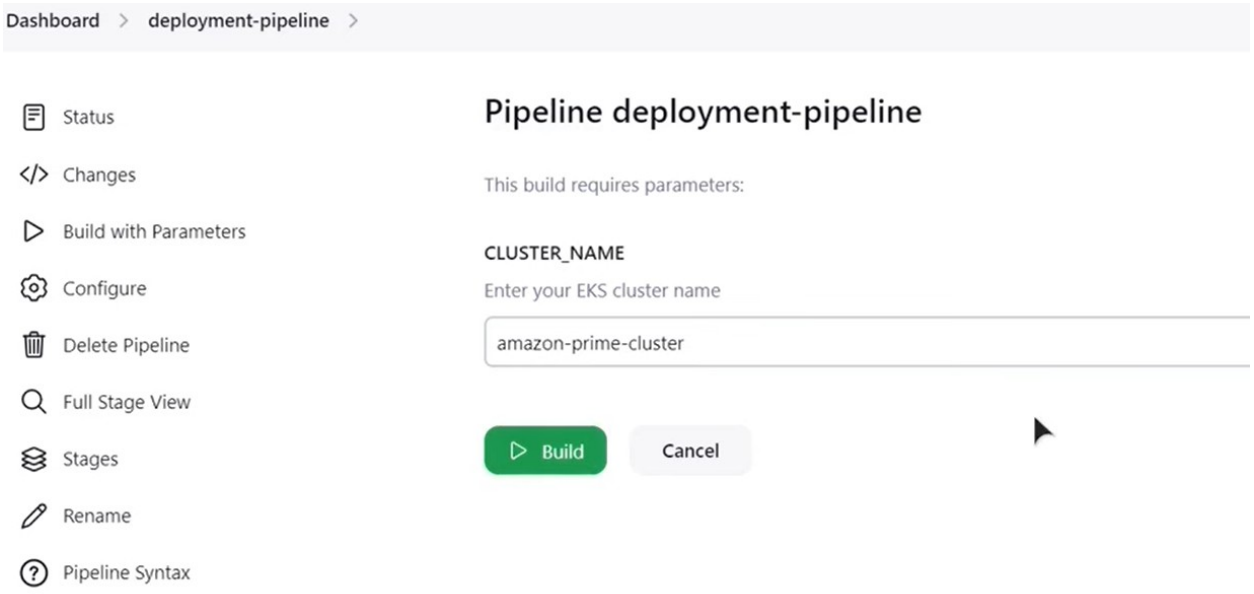
 # 2 nodes

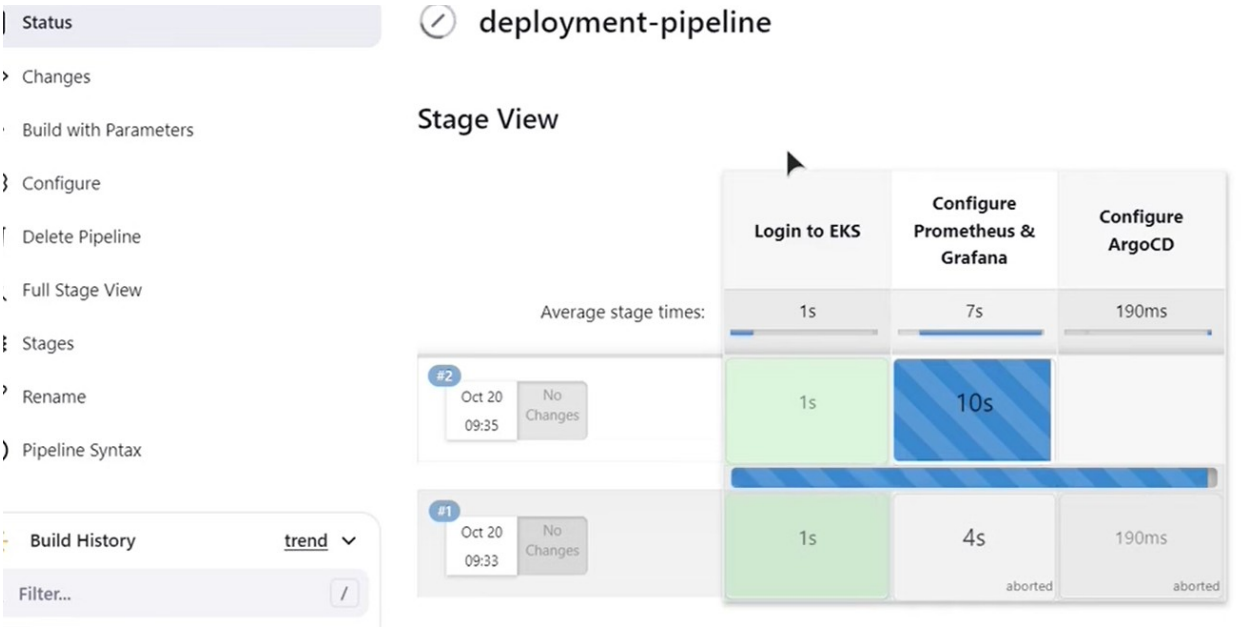


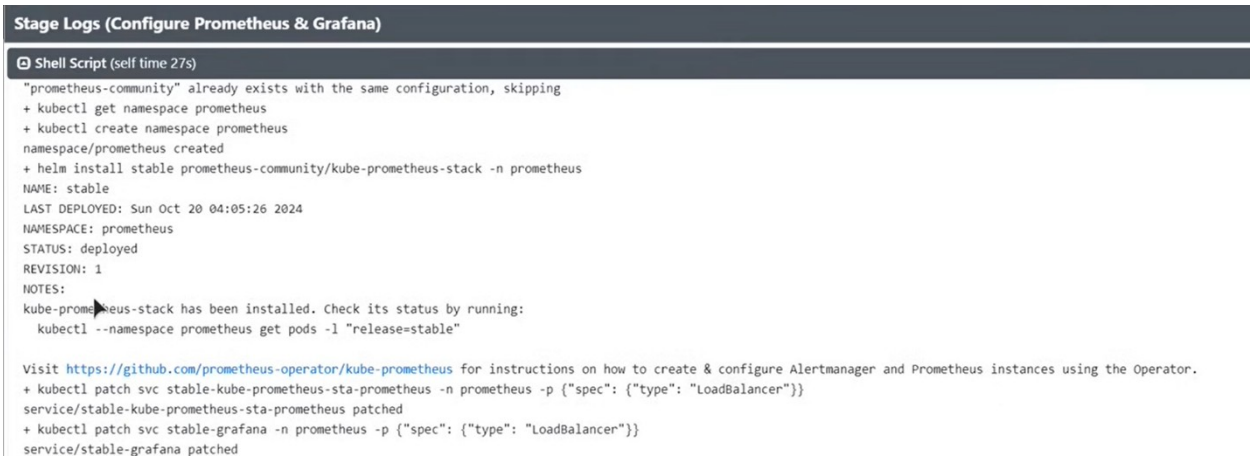
 # EC2 Created nodes help of Terraform Code

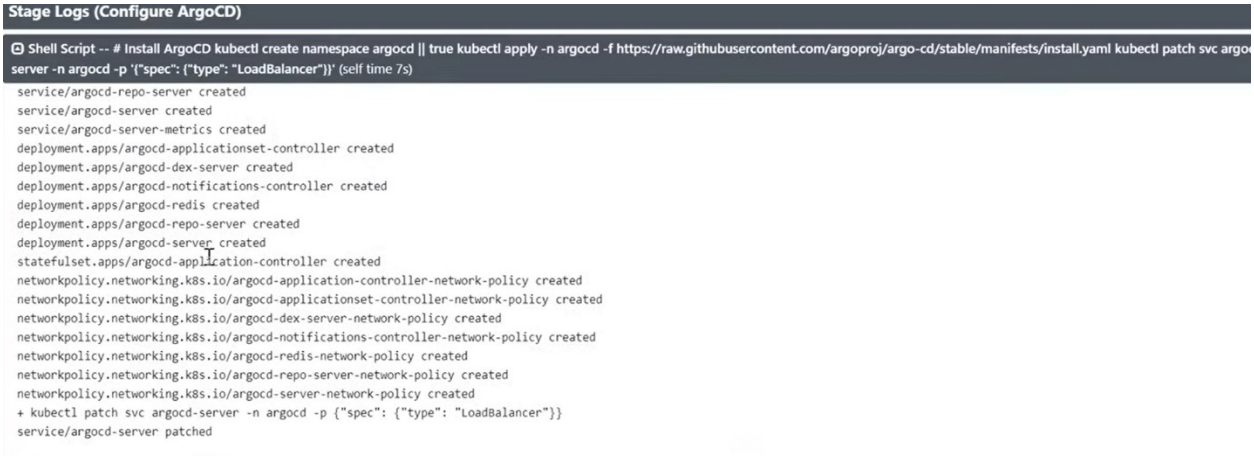

EC2 Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Instances (3) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
panda-node
i-08541432b571e2e72
Running
t2.medium
panda-node
i-01c4f0308b9b1ddb1
Running
t2.medium
JENKINS-SERVER
i-0858872b9a500cf8e
Running
t2.medium
Select an instance

 # jenkins


Dashboard
deployment-pipeline
Status
Changes
Build with Parameters
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
Pipeline deployment-pipeline
This build requires parameters:
CLUSTER_NAME
Enter your EKS cluster name
amazon-prime-cluster
Build
Cancel

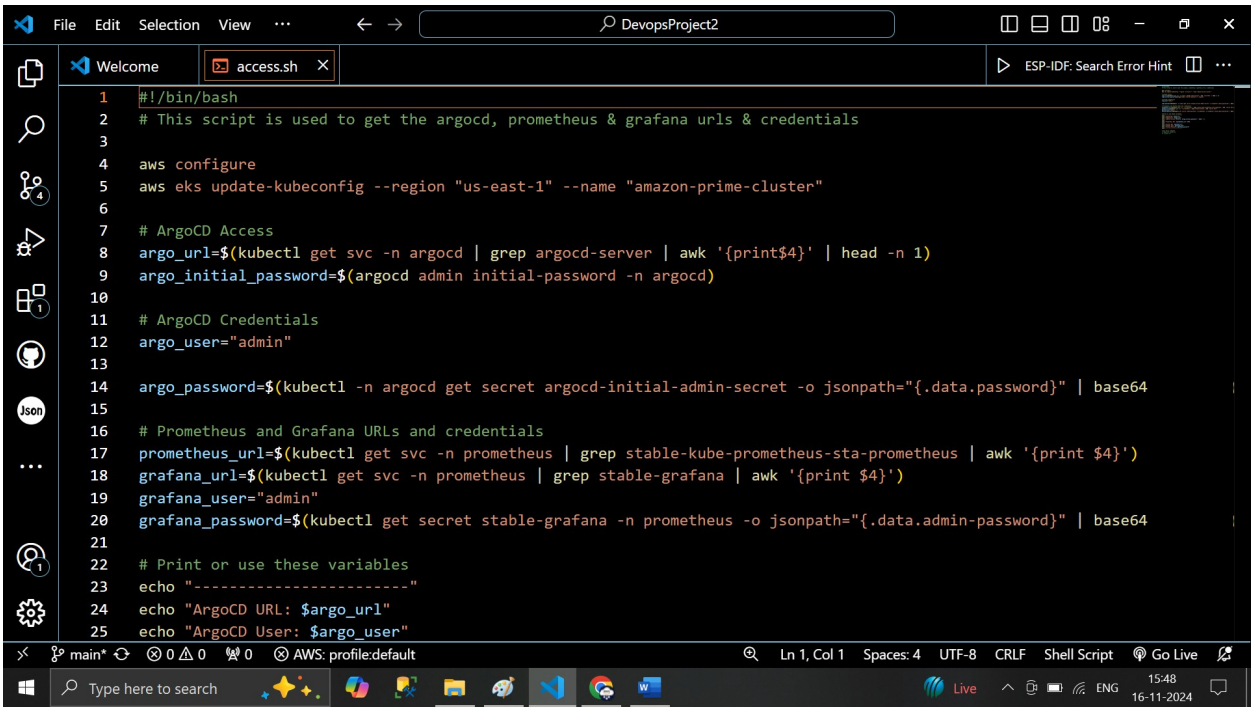


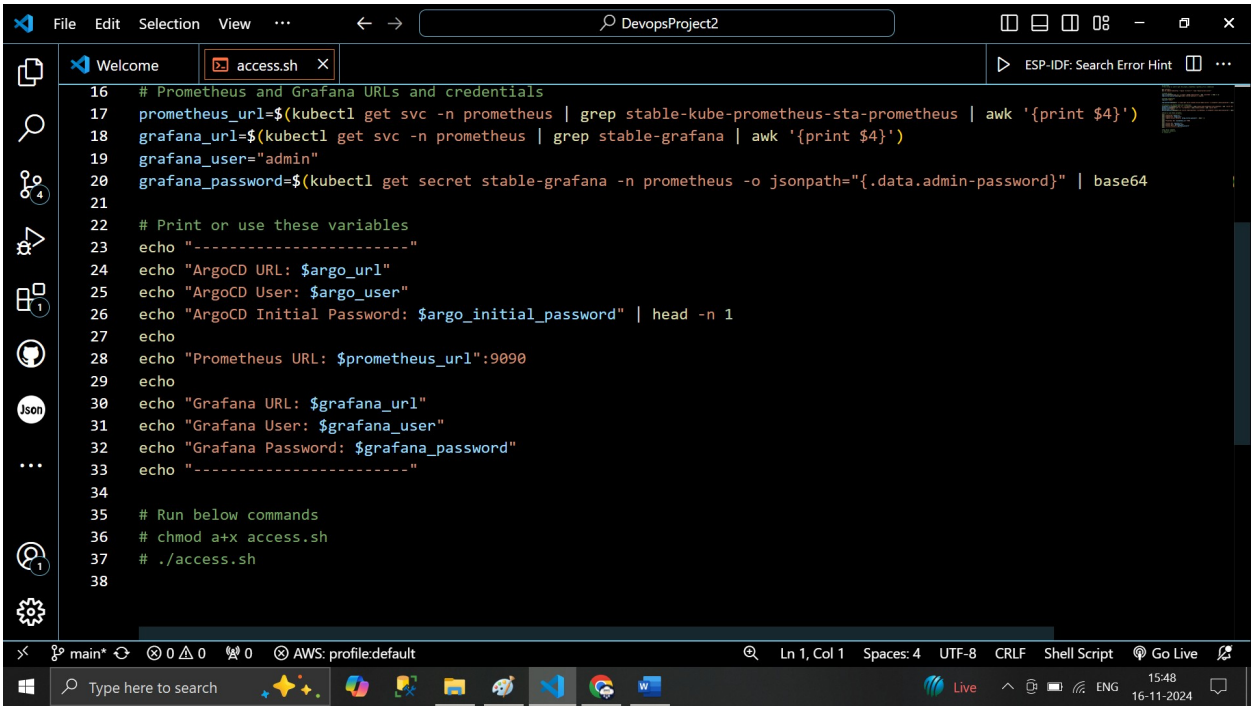




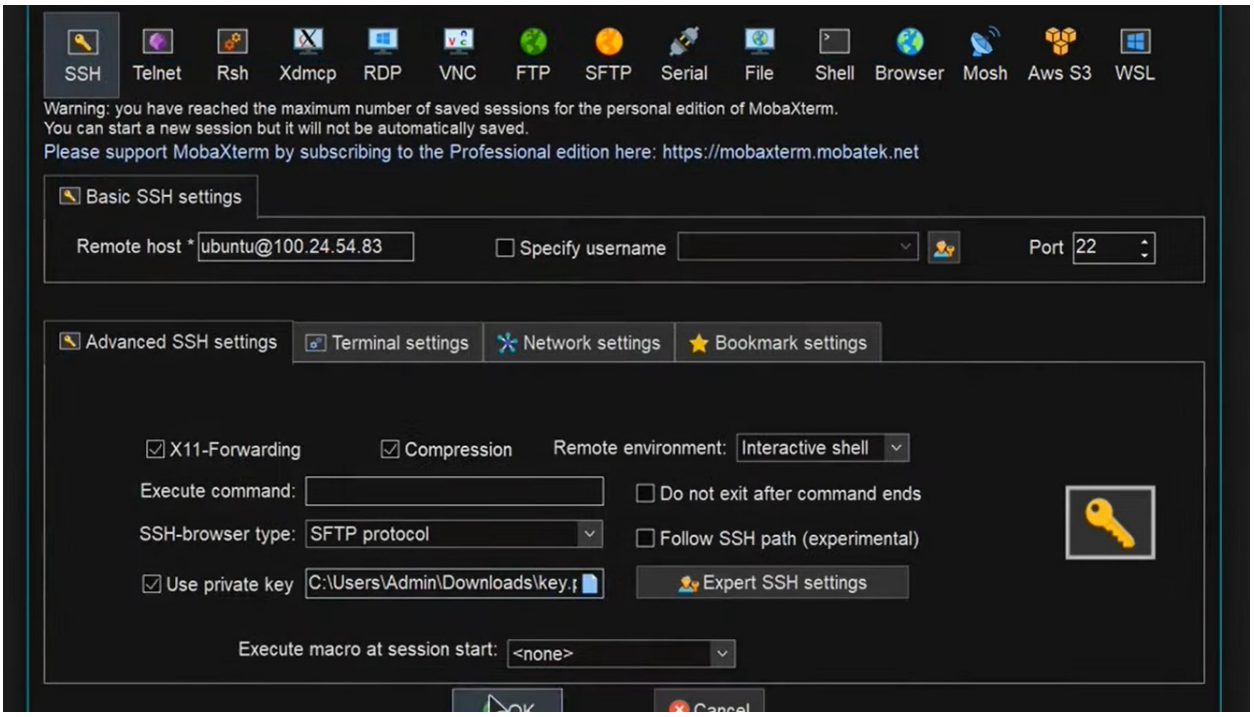
 # We need URL of ArgoCd/Grafana / Prometheus??

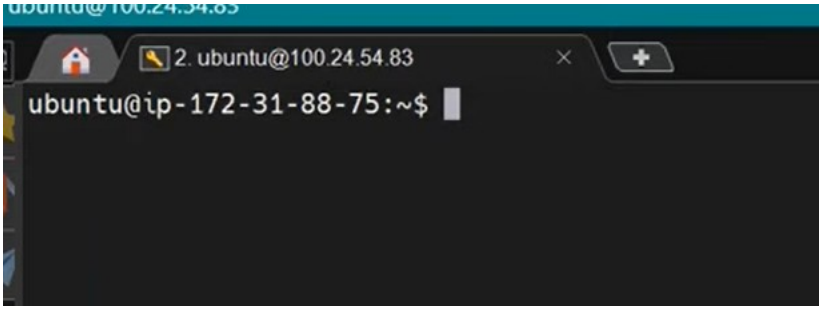
 # My_repo - https://github.com/Siddhartha082/AWS-DevOps-Final-Chapter-Project/blob/main/access.sh

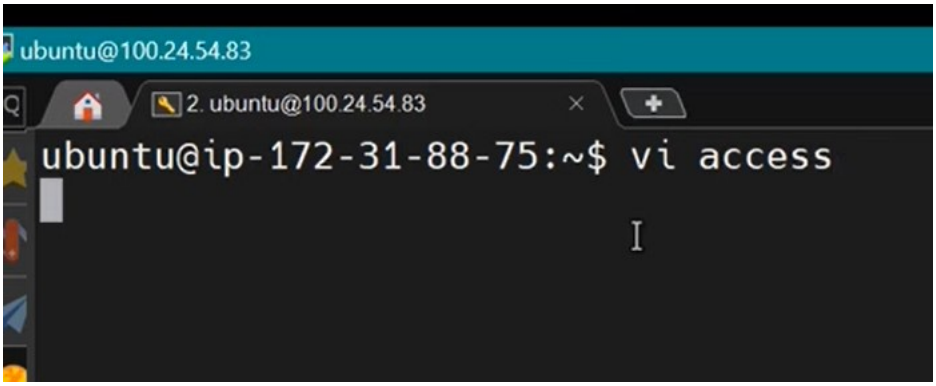




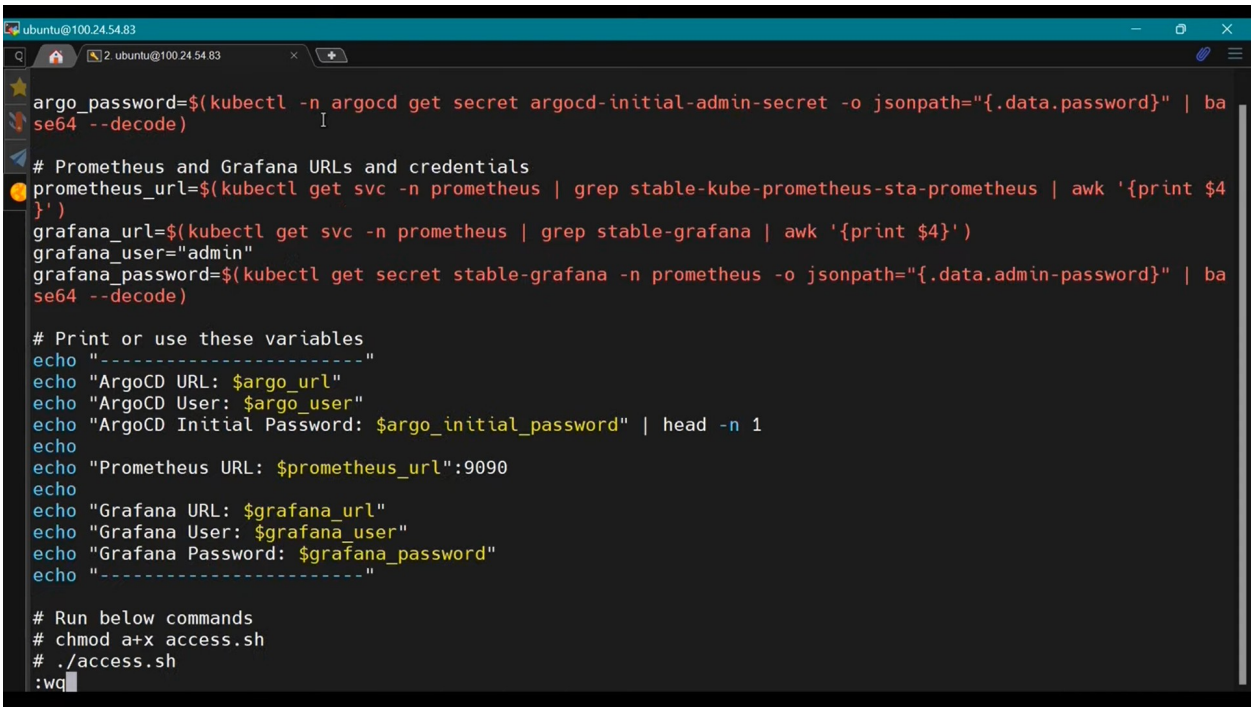
 # go to Jenkin Server

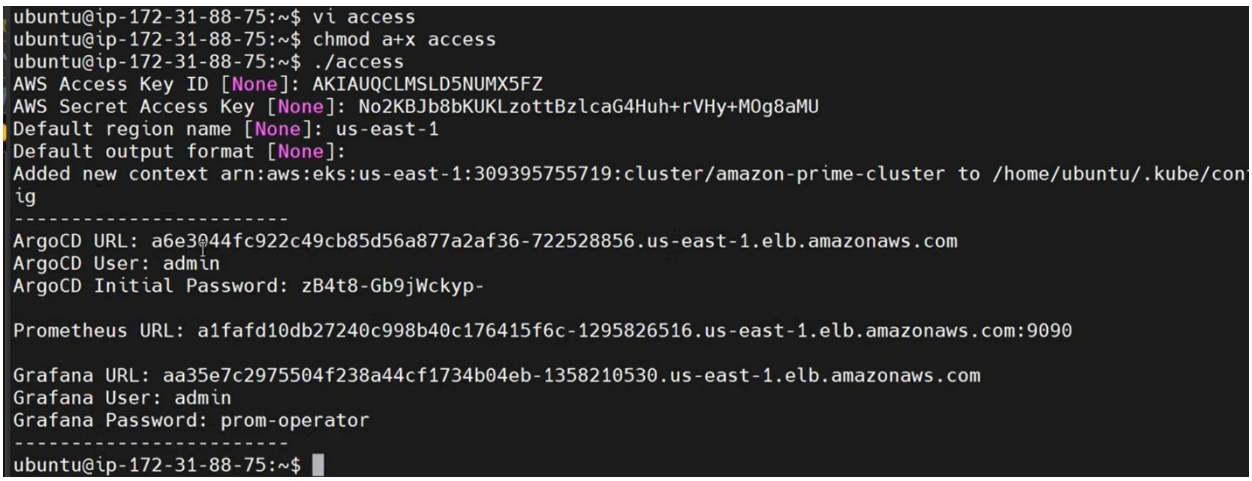




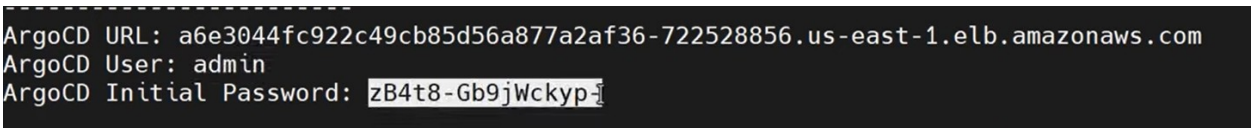


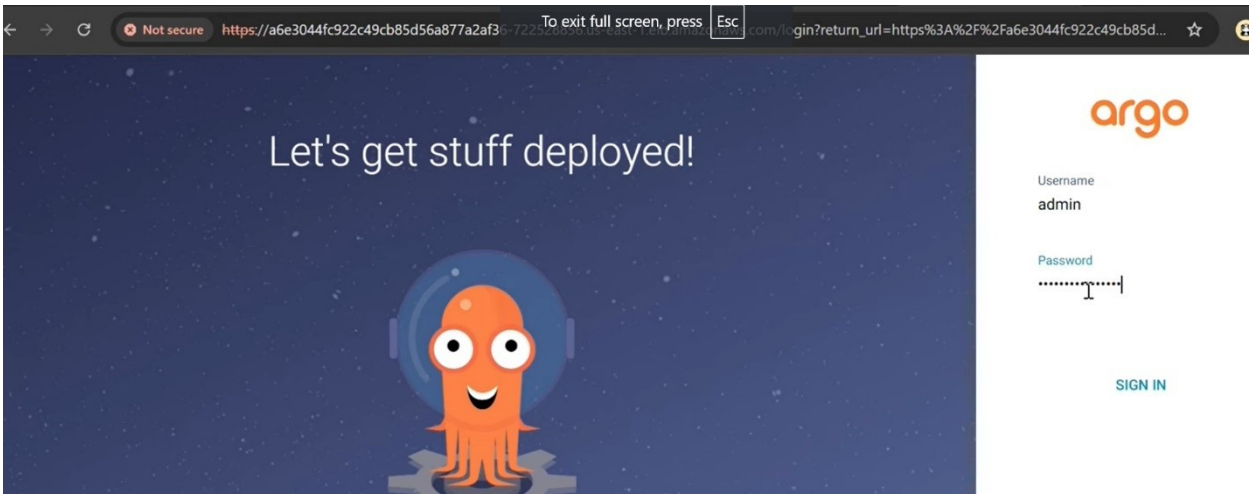
 # past the above script



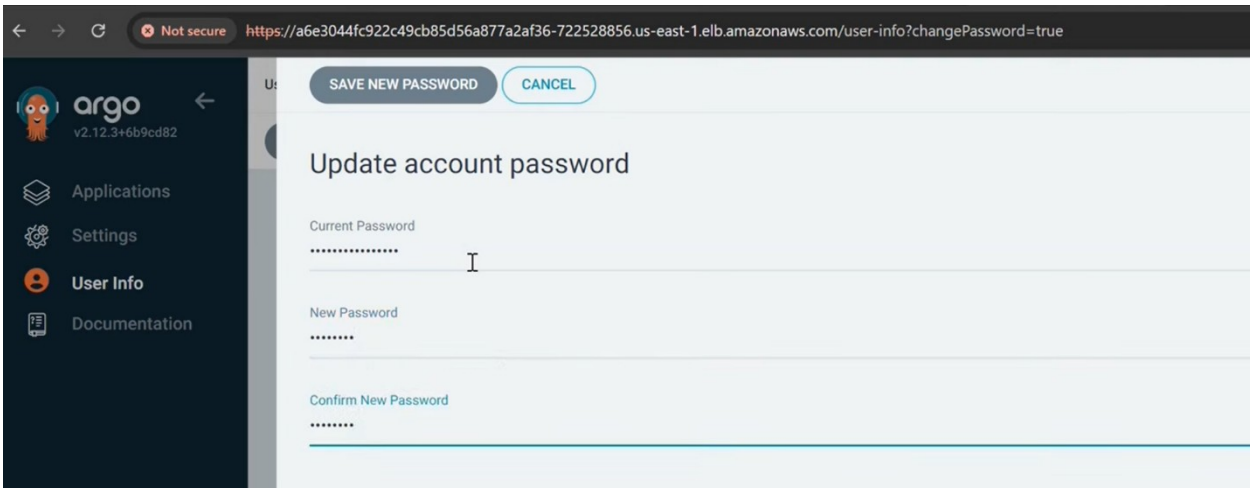


 # 1st ArgoCD access

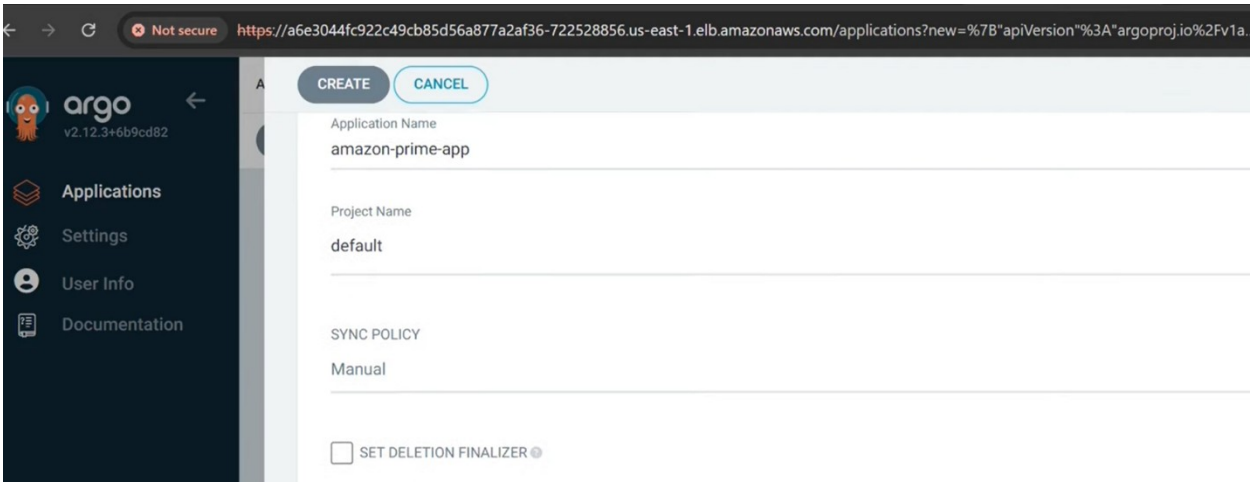




 # Change password



 # Create App



SOURCE

Repository URL

https://github.com/Siddhartha082/AWS-DevOps-Final-Chapter-Project

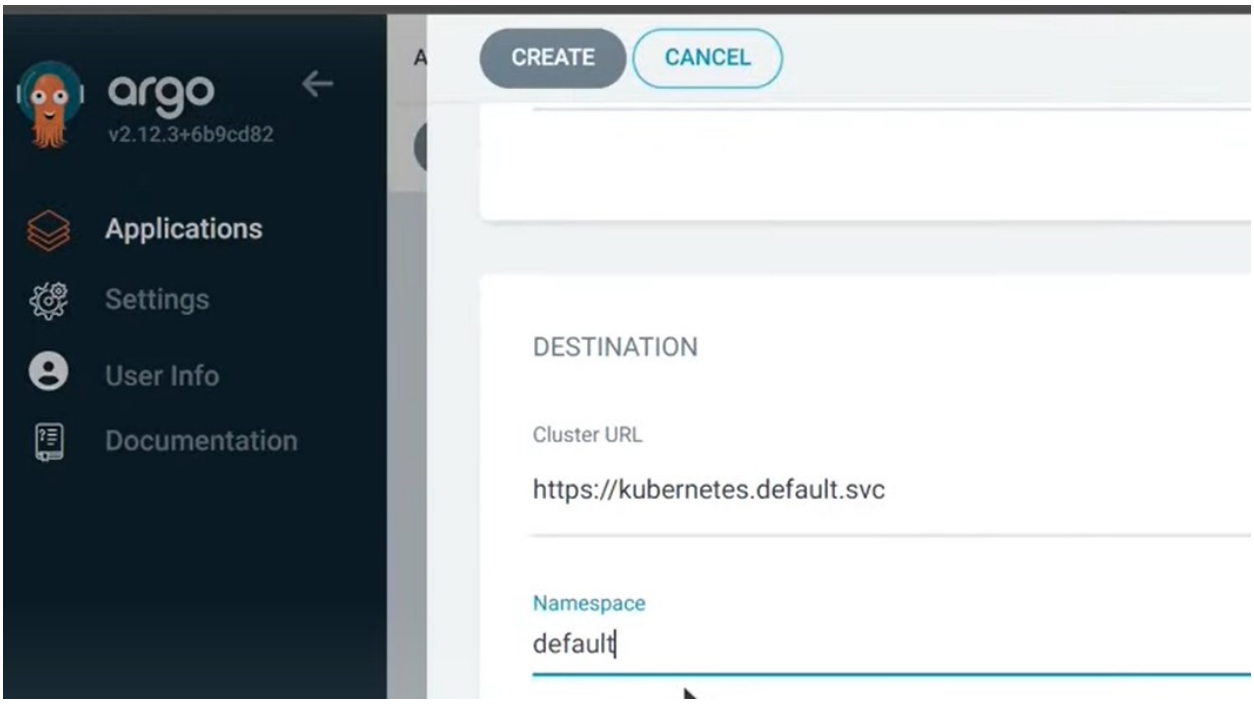
Revision

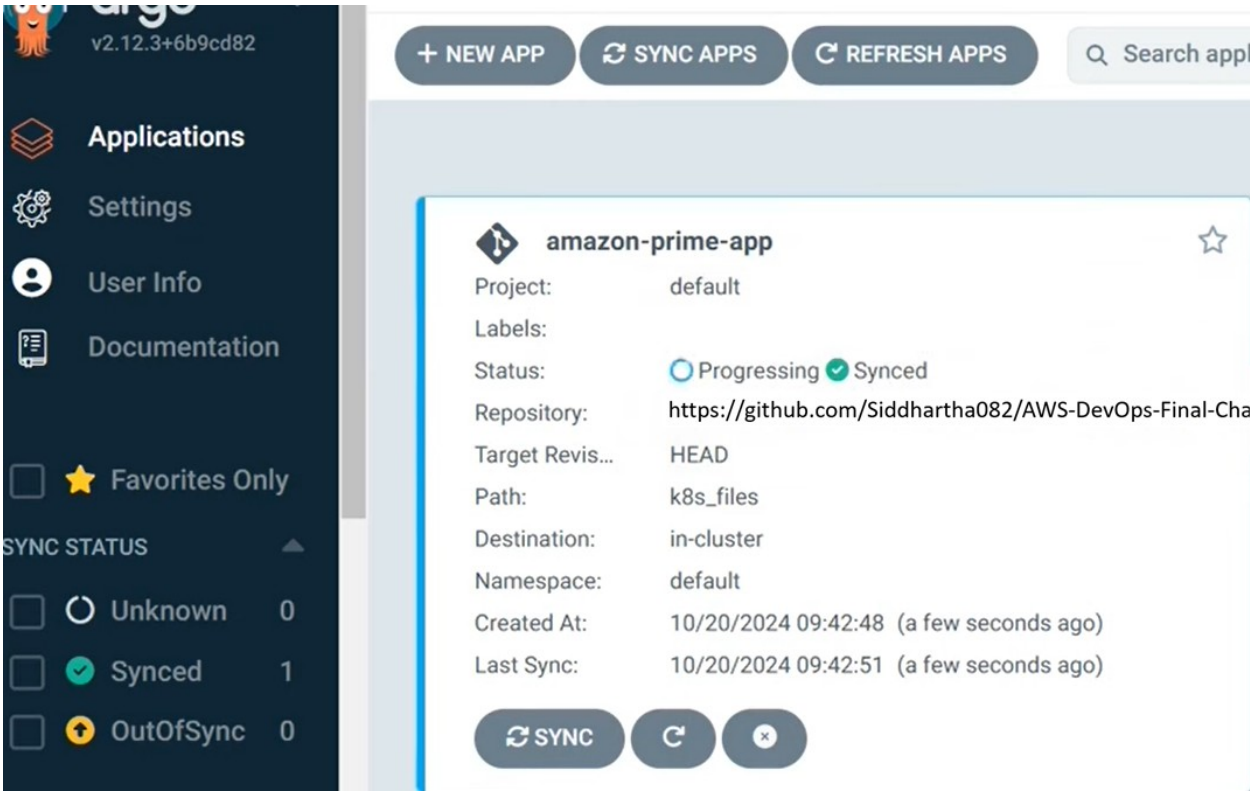
HEAD

K8s_files

 # path

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pandacloud-app
  labels:
    app: pandacloud-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pandacloud-app
  template:
    metadata:
      labels:
        app: pandacloud-app
    spec:
      containers:
      - name: pandacloud-container
        image: 851725192411.dkr.ecr.us-east-1.amazonaws.com/amazon-prime:latest
        ports:
        - containerPort: 3000
```





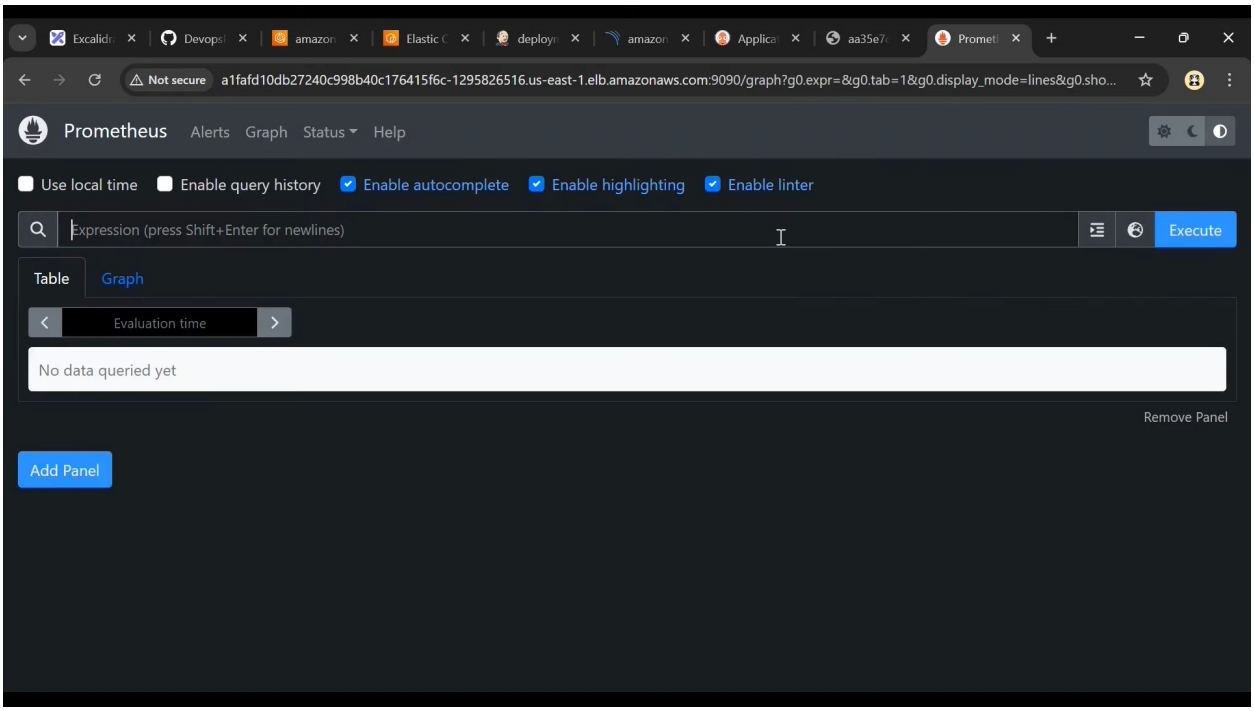
 # Go back to jenkis server - ArgoCD manifest

```
ubuntu@ip-172-31-88-75:~$ kubectl get service -n argocd
NAME                                      TYPE           CLUSTER-IP       EXTERNAL-IP
                                 PORT(S)                        AGE
argocd-applicationset-controller          ClusterIP      172.20.104.251   <none>
                                 7000/TCP,8080/TCP              7m17s
argocd-dex-server                         ClusterIP      172.20.174.198   <none>
                                 5556/TCP,5557/TCP,5558/TCP     7m17s
argocd-metrics                            ClusterIP      172.20.46.45     <none>
                                 8082/TCP                       7m17s
argocd-notifications-controller-metrics   ClusterIP      172.20.14.48     <none>
                                 9001/TCP                       7m17s
argocd-redis                              ClusterIP      172.20.47.3      <none>
                                 6379/TCP                       7m17s
argocd-repo-server                        ClusterIP      172.20.134.60    <none>
                                 8081/TCP,8084/TCP              7m17s
argocd-server                             LoadBalancer   172.20.61.25     a6e3044fc922c49cb85d56a877a2af
22528856.us-east-1.elb.amazonaws.com      80:32509/TCP,443:30251/TCP     7m17s
argocd-server-metrics                     ClusterIP      172.20.50.4      <none>
                                 8083/TCP                       7m17s

ubuntu@ip-172-31-88-75:~$
```

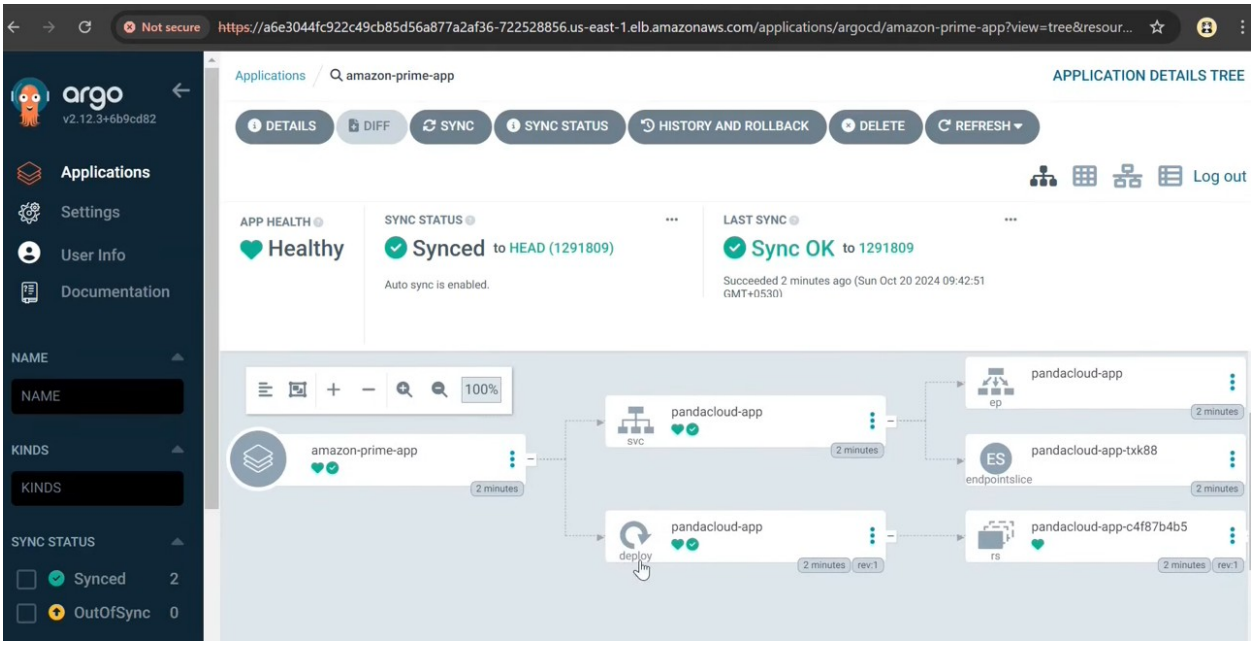
 # endpoint of Prometheus + Grafana

```
ubuntu@100.24.54.83
2. ubuntu@100.24.54.83
ubuntu@ip-172-31-88-75:~$ kubectl get service -n prometheus
NAME                                     TYPE           CLUSTER-IP       EXTERNAL-IP
                                 PORT(S)                      AGE
alertmanager-operated                    ClusterIP      None             <none>
                                 9093/TCP,9094/TCP,9094/UDP   7m44s
prometheus-operated                      ClusterIP      None             <none>
                                 9090/TCP                     7m43s
stable-grafana                           LoadBalancer   172.20.47.38     aa35e7c2975504f238a44cf1734b04eb-1
358210530.us-east-1.elb.amazonaws.com    80:31308/TCP                 7m49s
stable-kube-prometheus-sta-alertmanager  ClusterIP      172.20.156.127   <none>
                                 9093/TCP,8080/TCP            7m49s
stable-kube-prometheus-sta-operator      ClusterIP      172.20.56.107    <none>
                                 443/TCP                      7m49s
stable-kube-prometheus-sta-prometheus    LoadBalancer   172.20.8.245     a1fafd10db27240c998b40c176415f6c-1
295826516.us-east-1.elb.amazonaws.com    9090:31422/TCP,8080:31670/TCP   7m49s
stable-kube-state-metrics                ClusterIP      172.20.123.243   <none>
                                 8080/TCP                     7m49s
stable-prometheus-node-exporter          ClusterIP      172.20.196.201   <none>
                                 9100/TCP                     7m49s

ubuntu@ip-172-31-88-75:~$ ./access
AWS Access Key ID [****************X5FZ]:
AWS Secret Access Key [****************8aMU]:
Default region name [us-east-1]:
Default output format [None]:
Updated context arn:aws:eks:us-east-1:309395755719:cluster/amazon-prime-cluster in /home/ubuntu/.kube/config
------------------------
ArgoCD URL: a6e3044fc922c49cb85d56a877a2af36-722528856.us-east-1.elb.amazonaws.com
ArgoCD User: admin
ArgoCD Initial Password: zB4t8-Gb9jWckyp-

Prometheus URL: a1fafd10db27240c998b40c176415f6c-1295826516.us-east-1.elb.amazonaws.com:9090

Grafana URL: aa35e7c2975504f238a44cf1734b04eb-1358210530.us-east-1.elb.amazonaws.com
Grafana User: admin
Grafana Password: prom-operator
------------------------
ubuntu@ip-172-31-88-75:~$
```

 # Prometheus



 # Argocd



 # Loadbalancer

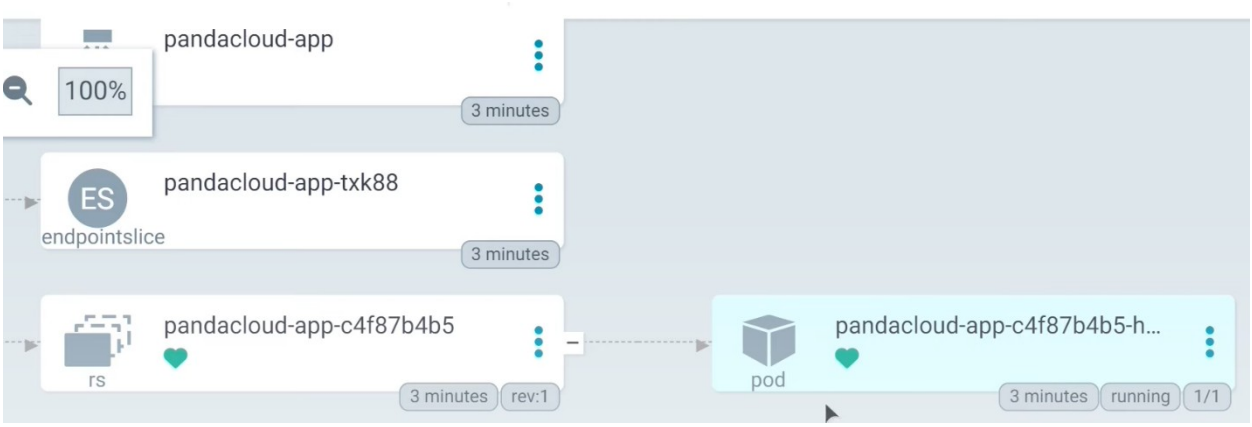
```
ubuntu@ip-172-31-88-75:~$ kubectl get service
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)          AG
E
kubernetes       ClusterIP      172.20.0.1      <none>                                                                   443/TCP          30
m
pandacloud-app   LoadBalancer   172.20.10.156   a183d4639631b4a3da83de4bea5e7980-1737361266.us-east-1.elb.amazonaws.com   3000:30261/TCP   2m
34s
```

```
ubuntu@ip-172-31-88-75:~$ kubectl get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
pandacloud-app    1/1     1            1           2m48s
ubuntu@ip-172-31-88-75:~$
```

 # Pod

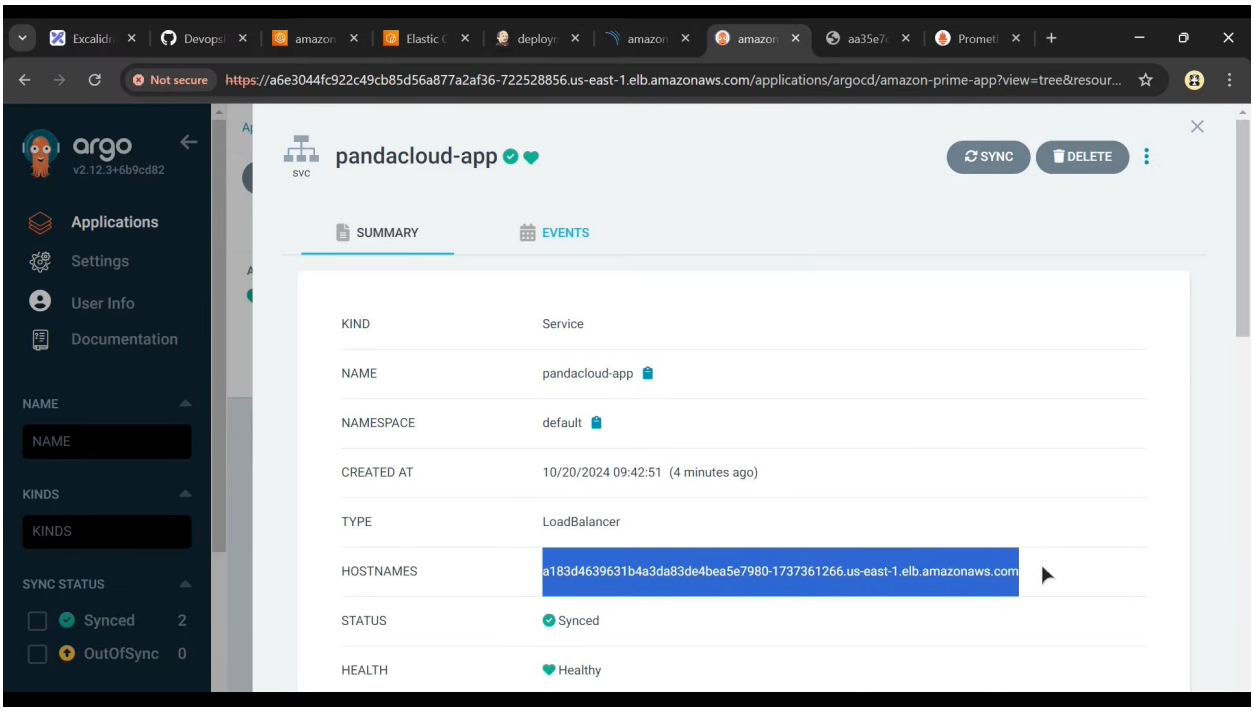
```
ubuntu@ip-172-31-88-75:~$ kubectl get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
pandacloud-app    1/1     1            1           2m48s
ubuntu@ip-172-31-88-75:~$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
pandacloud-app-c4f87b4b5-hhr6x    1/1     Running   0          3m8s
ubuntu@ip-172-31-88-75:~$
```

 # Graphically

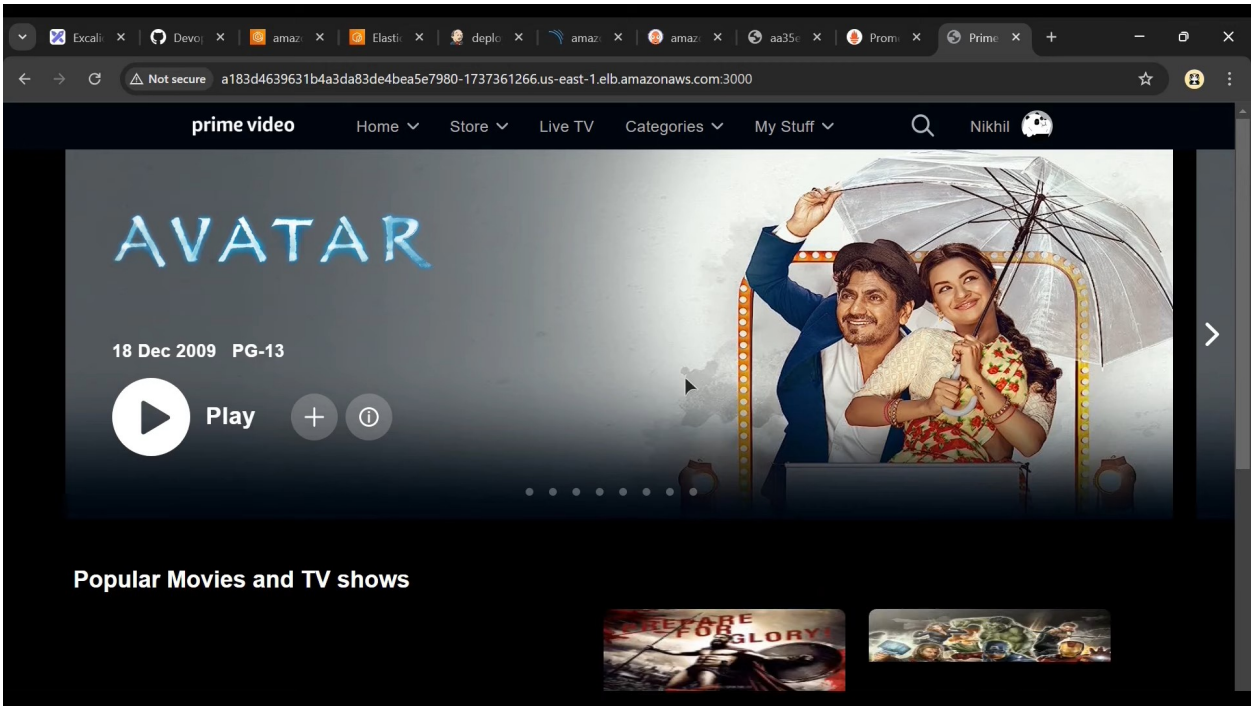


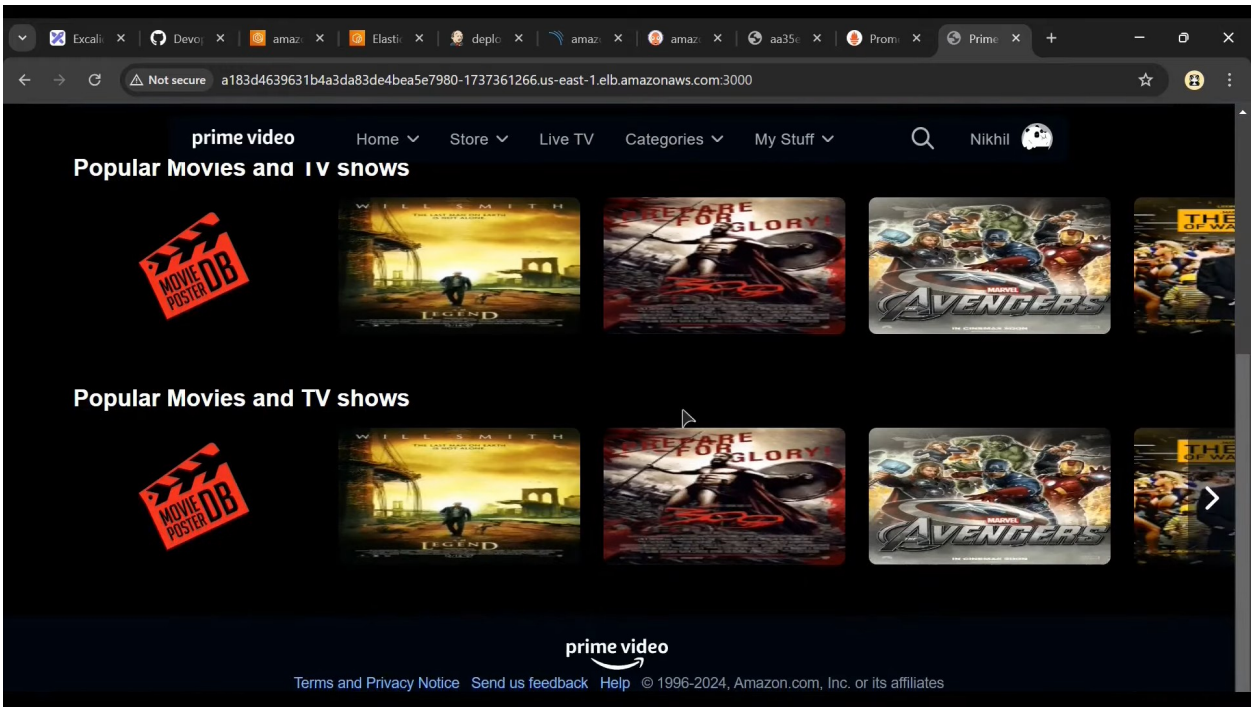
 # Endpoint

```
ubuntu@ip-172-31-88-75:~$ kubectl get service
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP
E
kubernetes       ClusterIP      172.20.0.1      <none>
m
pandacloud-app   LoadBalancer   172.20.10.156   a183d4639631b4a3da83de4bea5e7980-1737361266.us-east-1.elb.amazonaws
84s
```



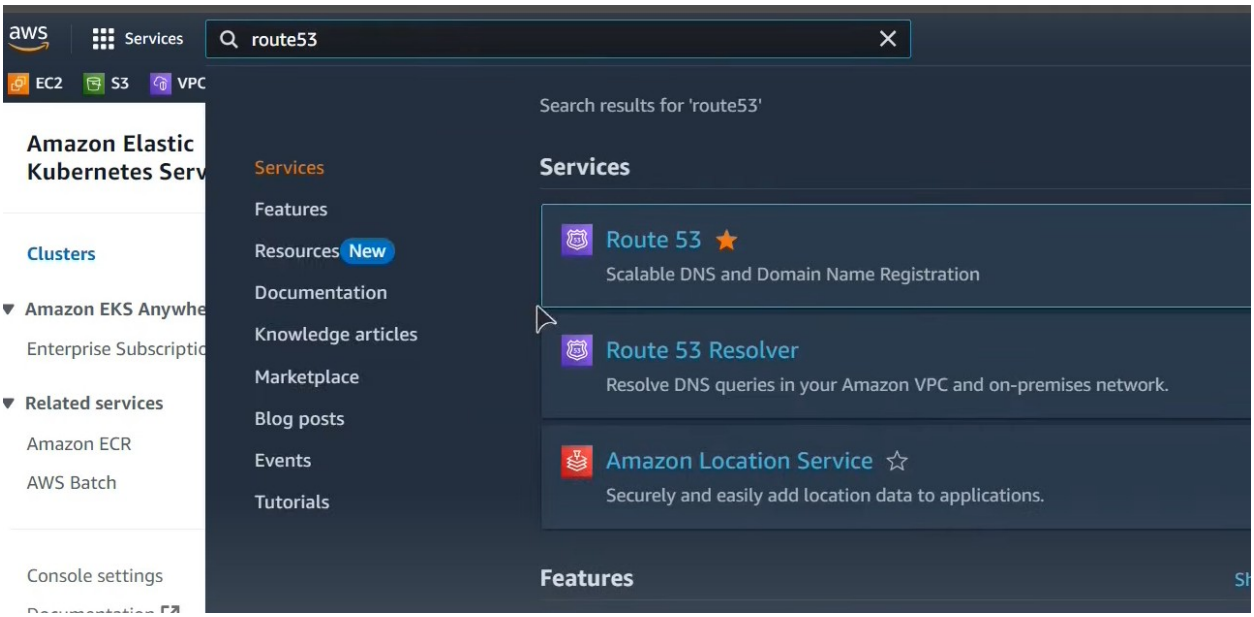
Run – Amazon Prime -app @ Port 3000

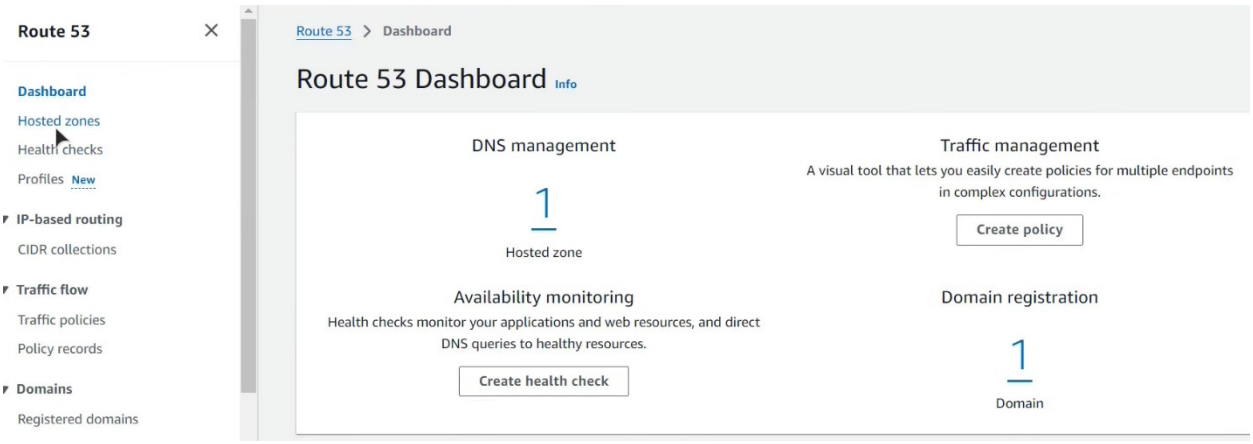




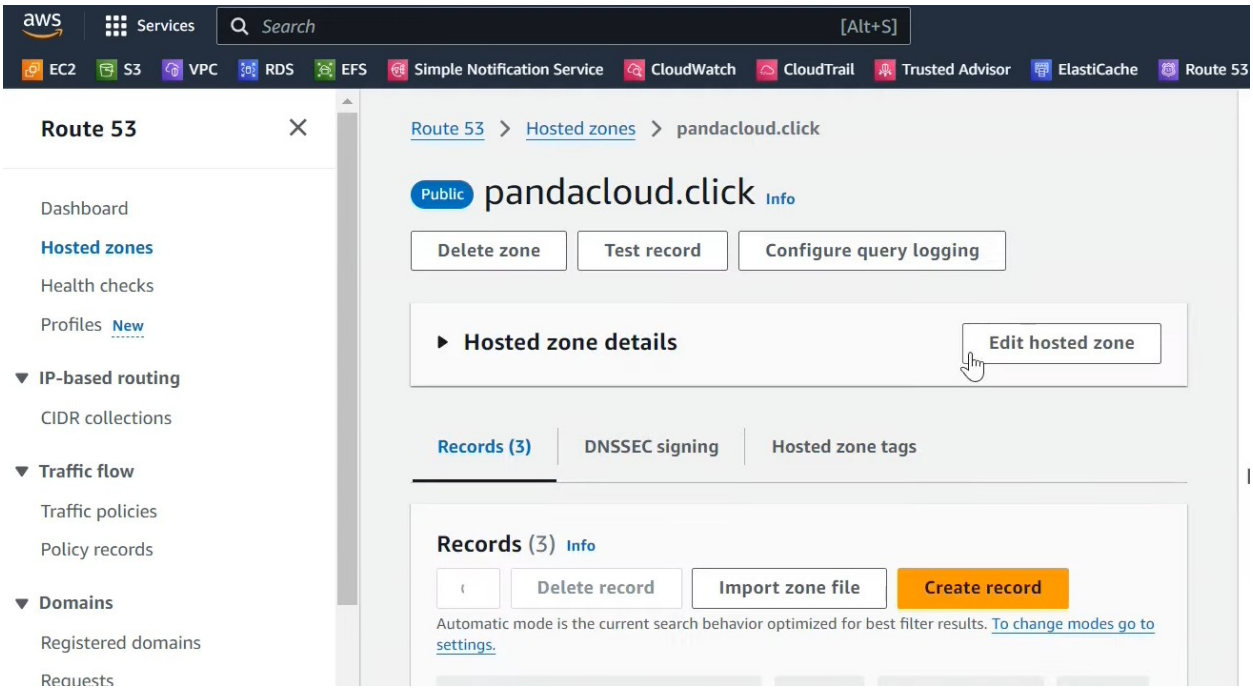
 # Route53 Server .. where Big Line of .com .. can be shorten using DNS naming

```
ArgoCD URL: a6e3044fc922c49cb85d56a877a2af36-722528856.us-east-1.elb.amazonaws.com
ArgoCD User: admin
ArgoCD Initial Password: zB4t8-Gb9jWckyp-

Prometheus URL: a1fafd10db27240c998b40c176415f6c-1295826516.us-east-1.elb.amazonaws.com:9090

Grafana URL: aa35e7c2975504f238a44cf1734b04eb-1358210530.us-east-1.elb.amazonaws.com
Grafana User: admin
Grafana Password: prom-operator
```

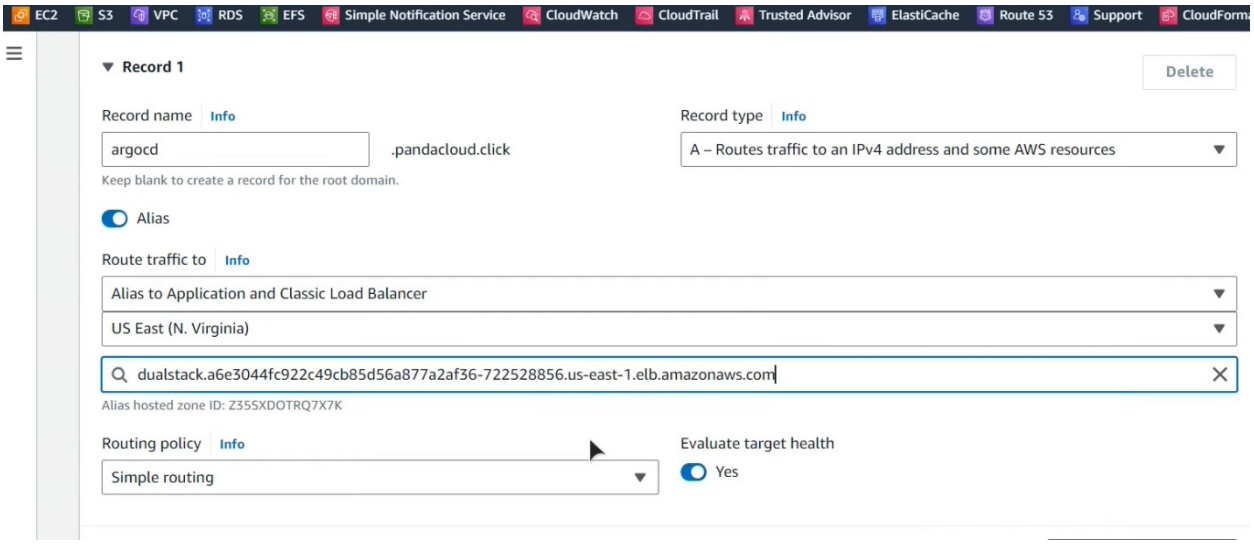


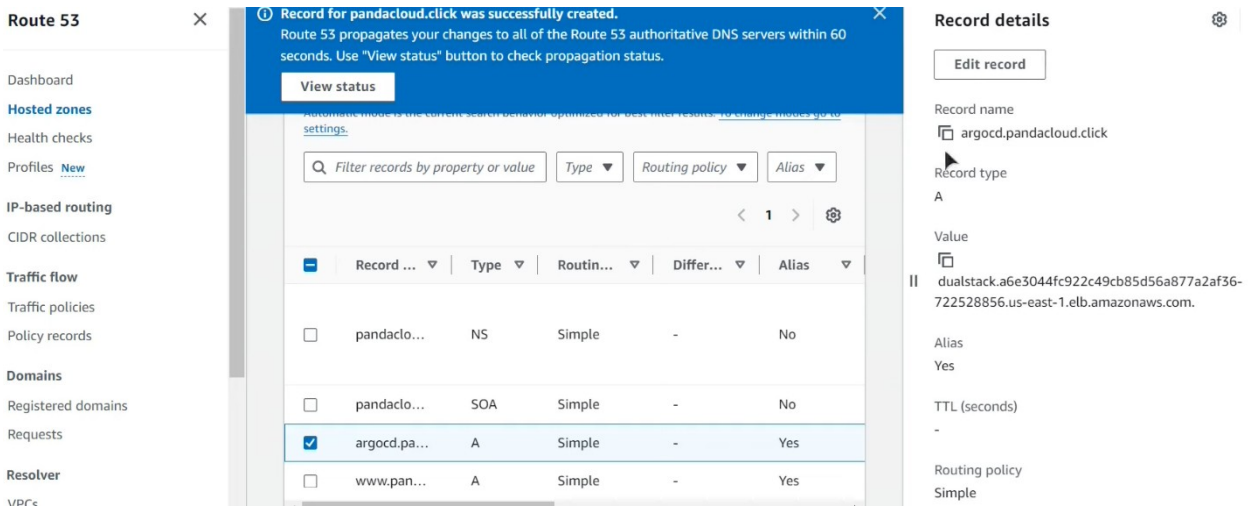


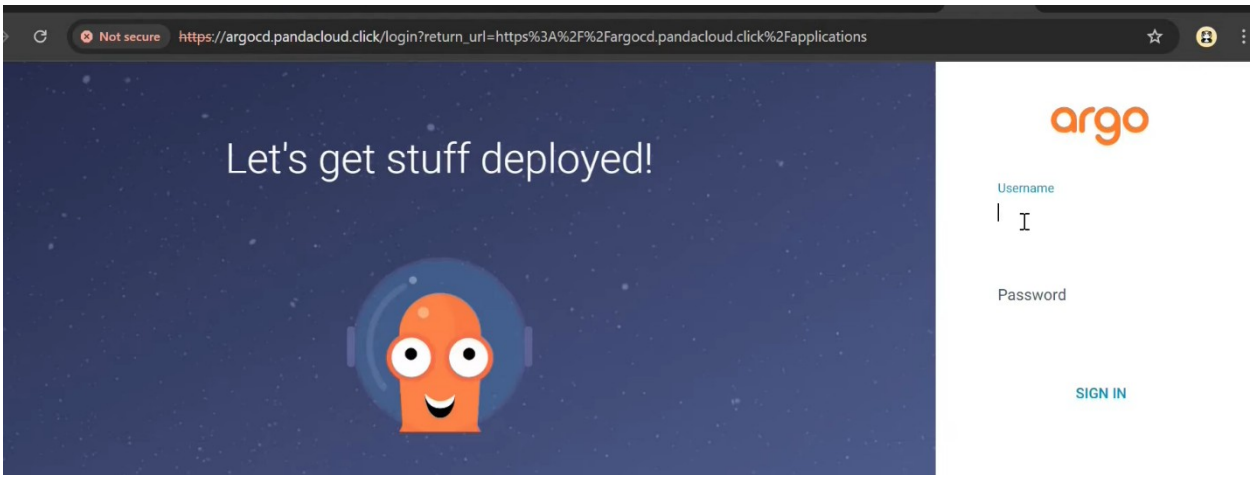
 # Create a URL



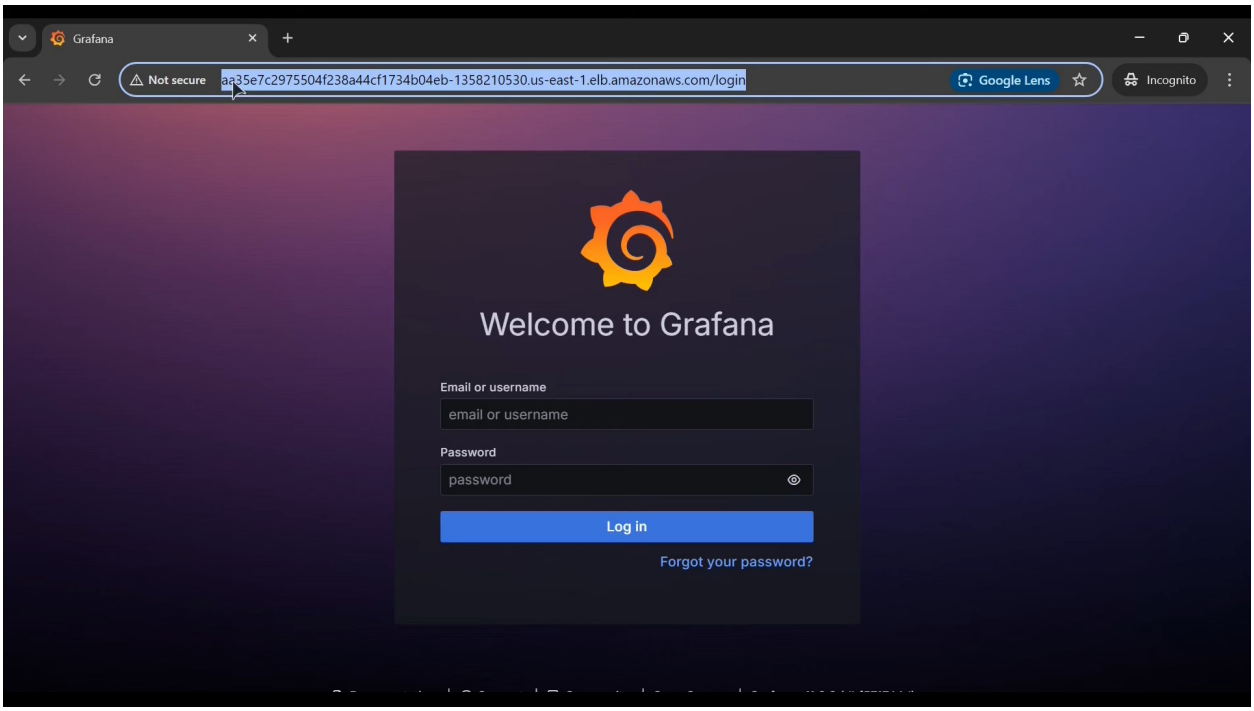
 # Create Records


EC2 S3 VPC RDS EFS Simple Notification Service CloudWatch CloudTrail Trusted Advisor ElastiCache Route 53 Support CloudForma
Record 1
Delete
Record name Info
argocd
.pandacloud.click
Keep blank to create a record for the root domain.
Record type Info
A – Routes traffic to an IPv4 address and some AWS resources
Alias
Route traffic to Info
Alias to Application and Classic Load Balancer
US East (N. Virginia)
dualstack.a6e3044fc922c49cb85d56a877a2af36-722528856.us-east-1.elb.amazonaws.com
Alias hosted zone ID: Z35SXDOTRQ7X7K
Routing policy Info
Simple routing
Evaluate target health
Yes


Route 53
Dashboard
Hosted zones
Health checks
Profiles New
IP-based routing
CIDR collections
Traffic flow
Traffic policies
Policy records
Domains
Registered domains
Requests
Resolver
Record for pandacloud.click was successfully created.
Route 53 propagates your changes to all of the Route 53 authoritative DNS servers within 60 seconds. Use "View status" button to check propagation status.
View status
Filter records by property or value
Type
Routing policy
Alias
Record ...
Type
Routin...
Differ...
Alias
pandaclo... NS Simple - No
pandaclo... SOA Simple - No
argocd.pa... A Simple - Yes
www.pan... A Simple - Yes
Record details
Edit record
Record name
argocd.pandacloud.click
Record type
A
Value
dualstack.a6e3044fc922c49cb85d56a877a2af36-722528856.us-east-1.elb.amazonaws.com.
Alias
Yes
TTL (seconds)
-
Routing policy
Simple

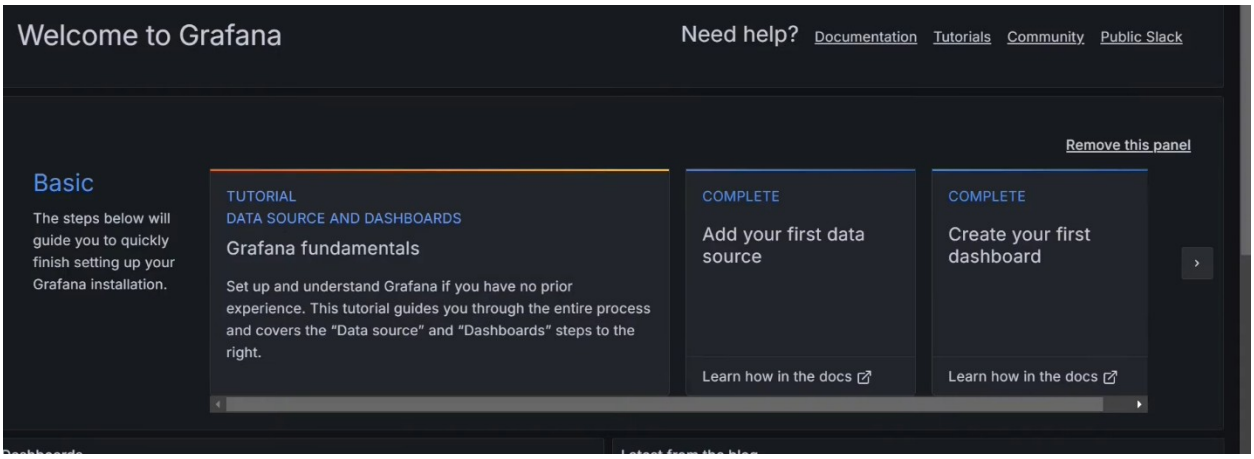

Not secure
https://argocd.pandacloud.click/login?return_url=https%3A%2F%2Fargocd.pandacloud.click%2Fapplications
Let's get stuff deployed!
argo
Username
Password
SIGN IN

 # check for Prometheus + Grafana status

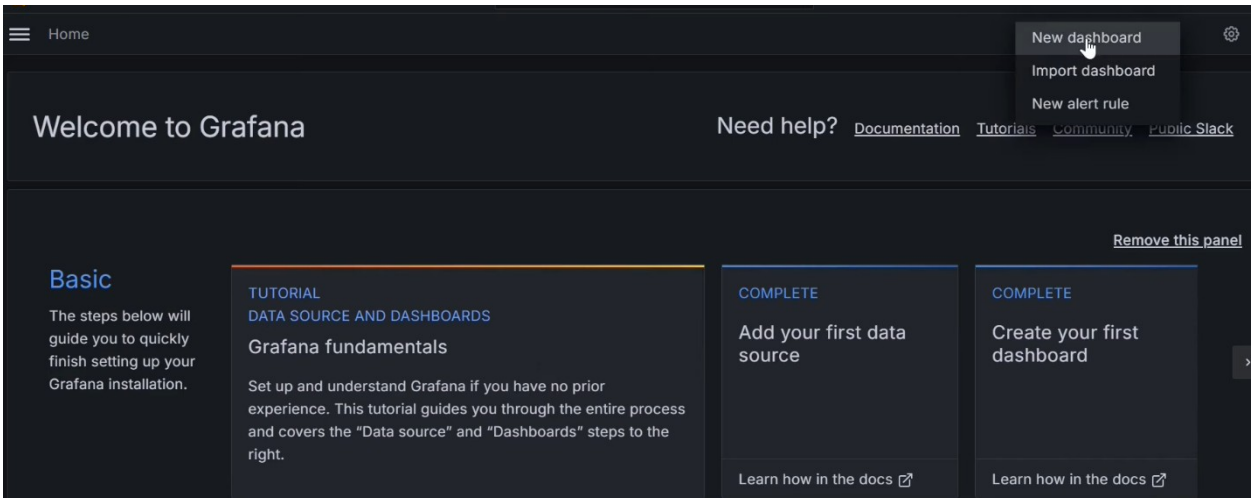



```
Grafana URL: aa35e7c2975504f238a44cf1734b04eb-1358210530.us-east-1.elb.amazonaws.com
Grafana User: admin
Grafana Password: prom-operator
```

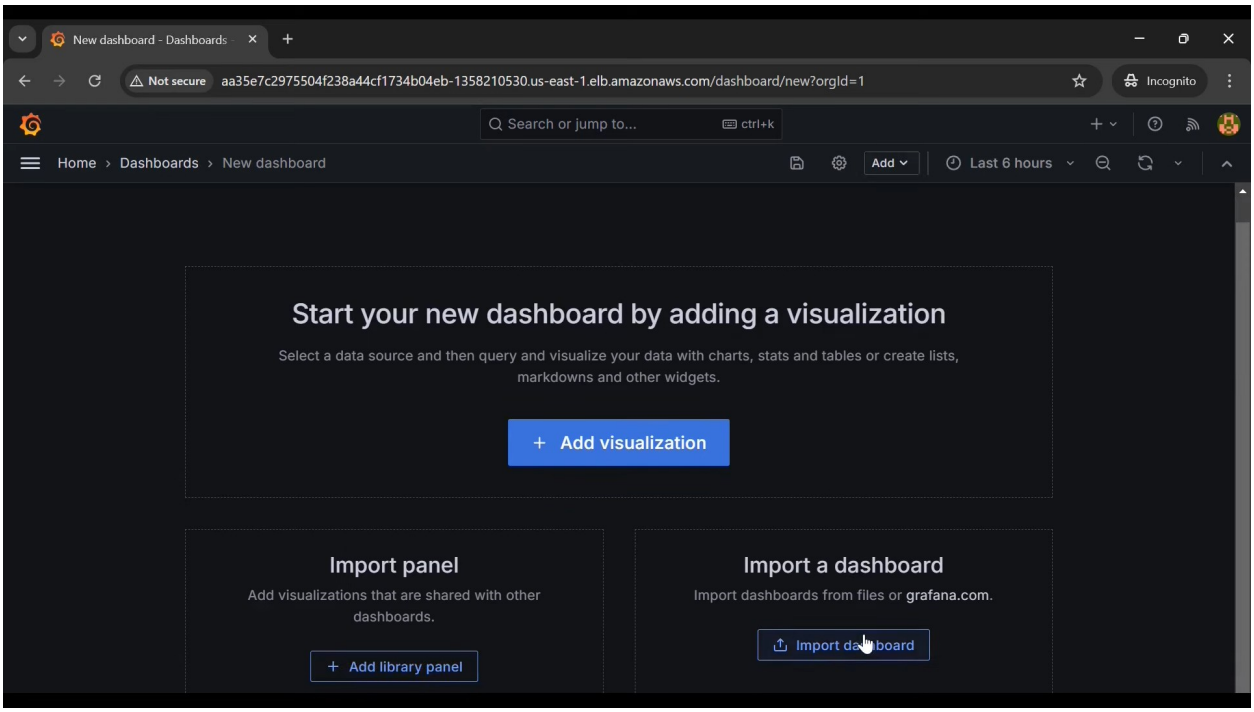
 # Note if grafana does not work … type htpp://URL…

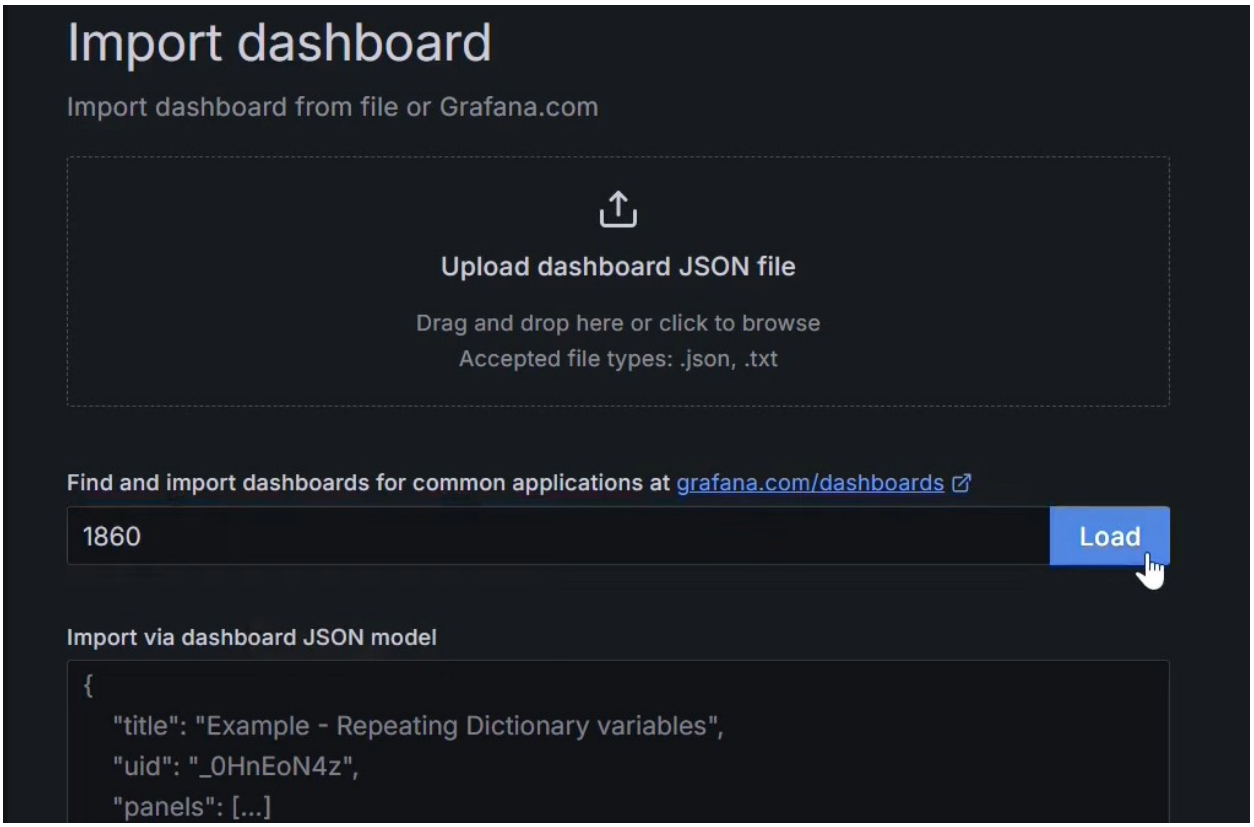


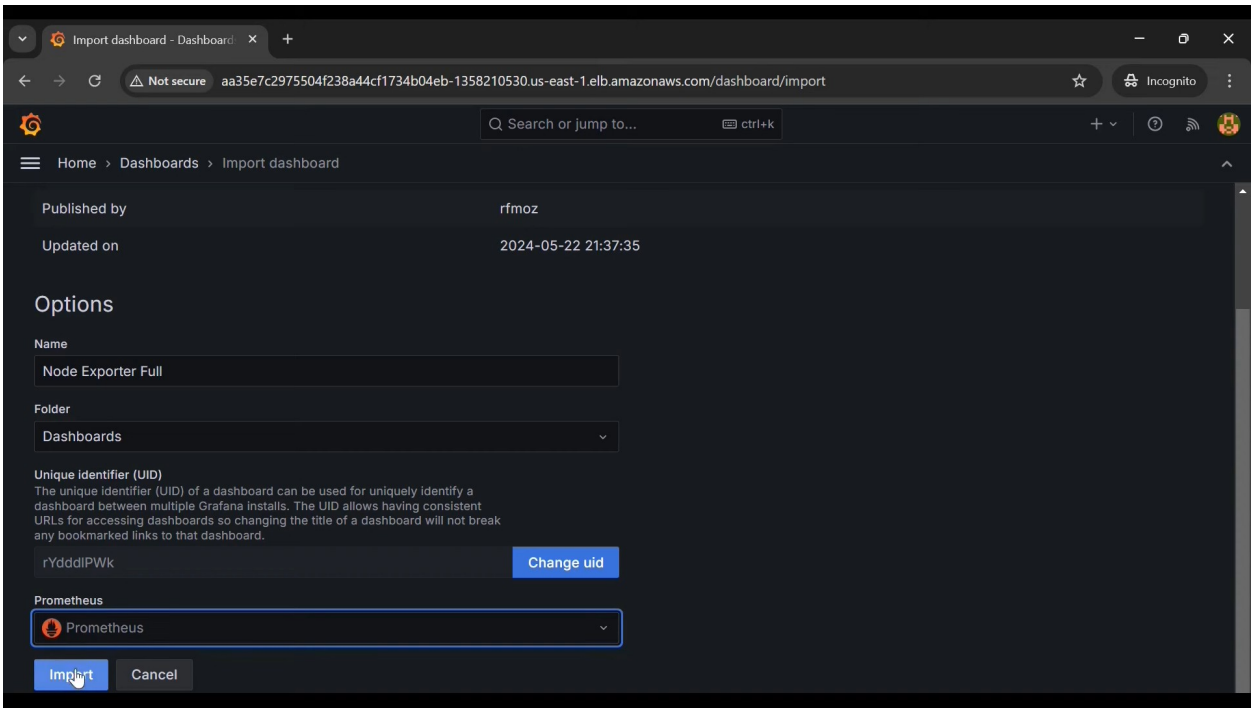
 # Create Custome Dashboard



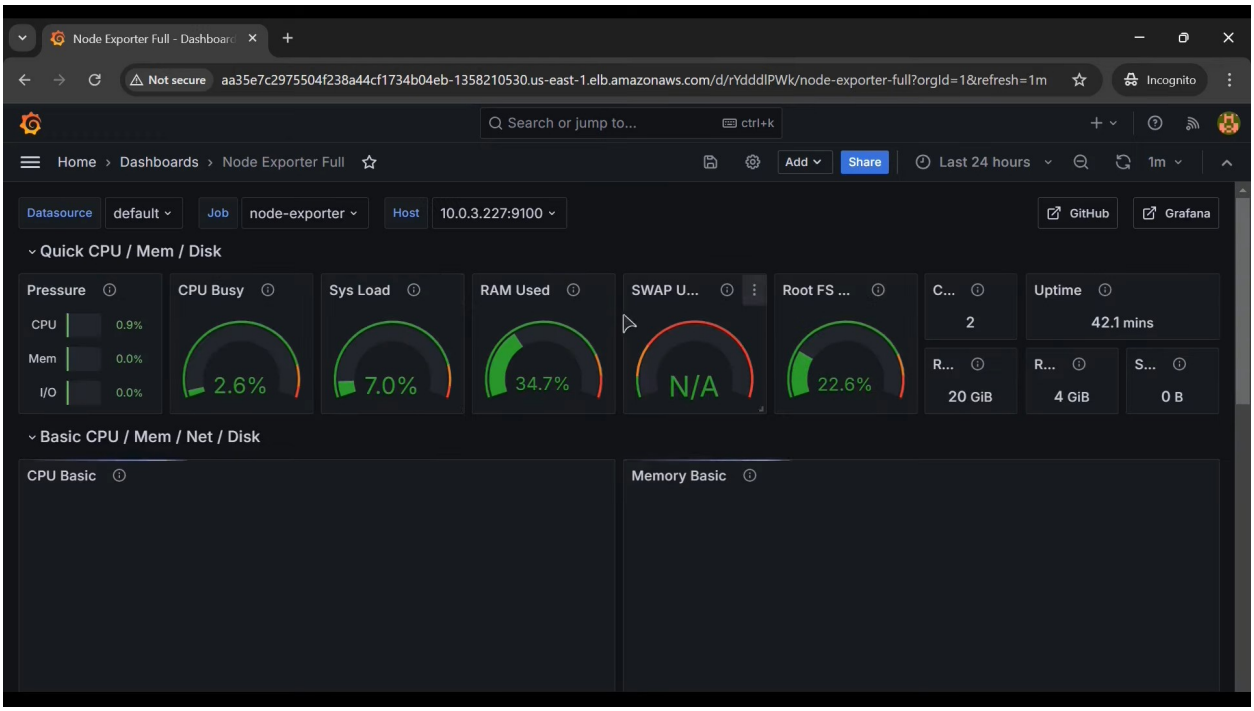
 # Click on impart Dashboard



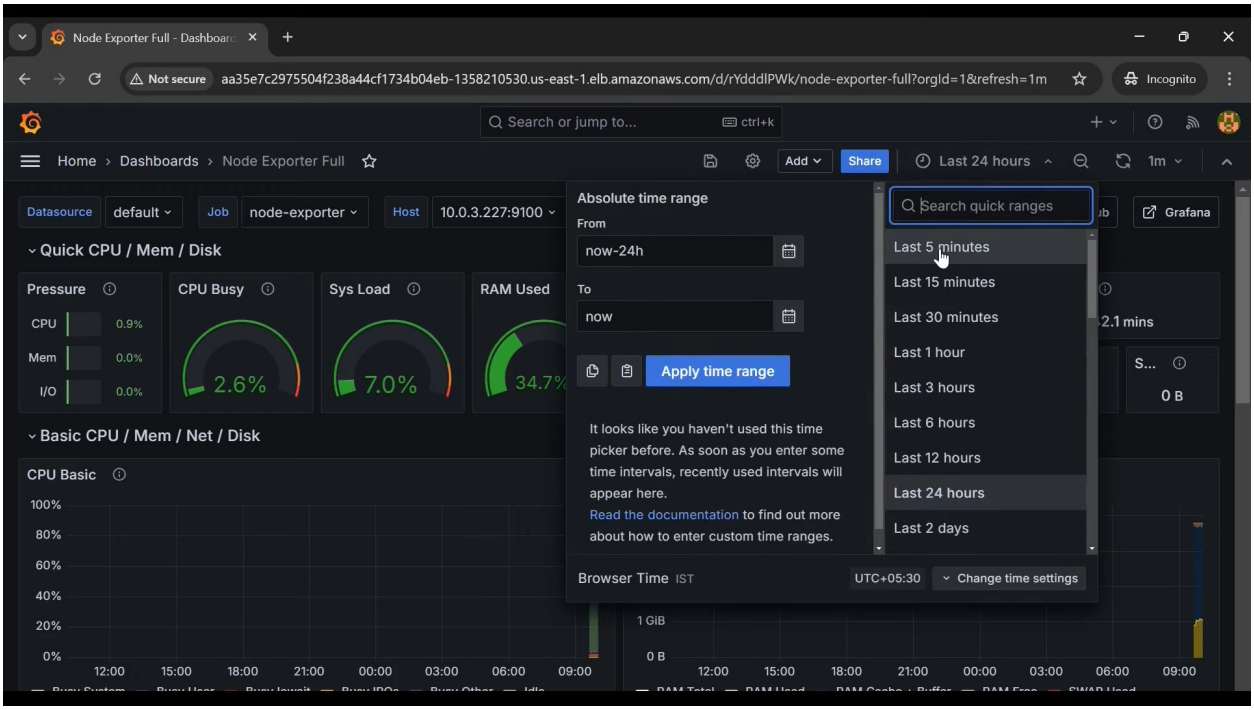




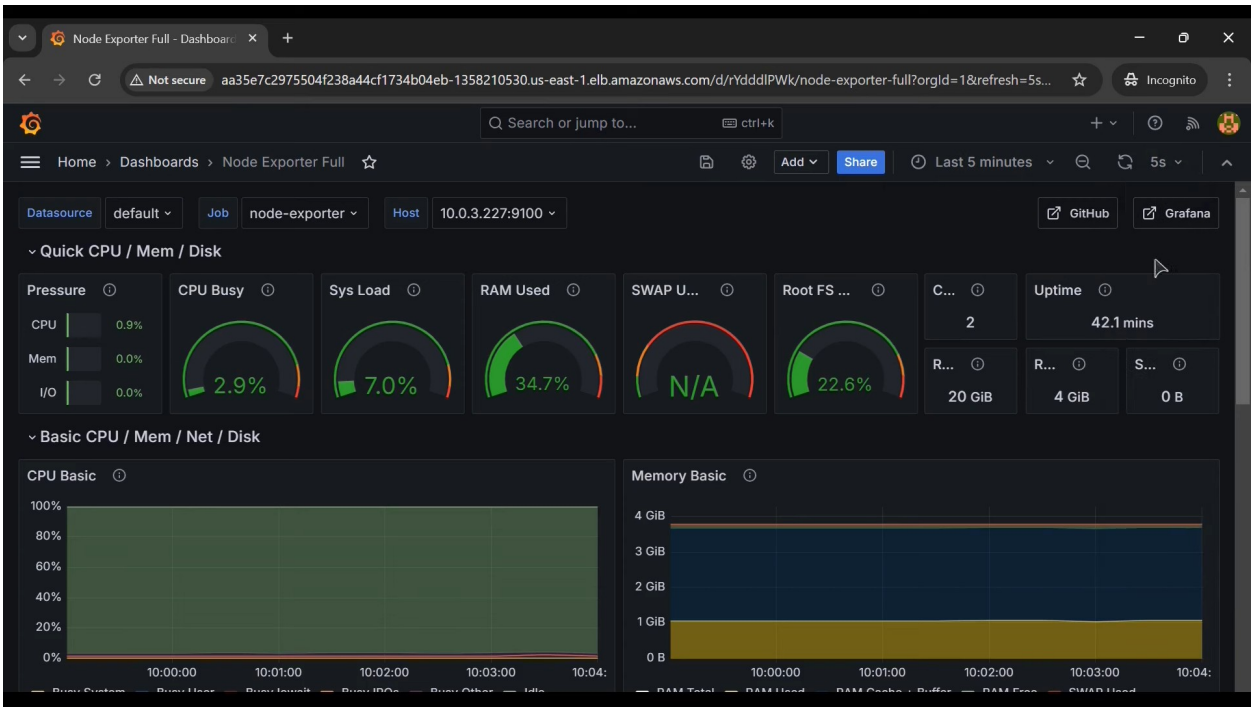
 # check CPU/RAM Utilization



 # last 5 mins



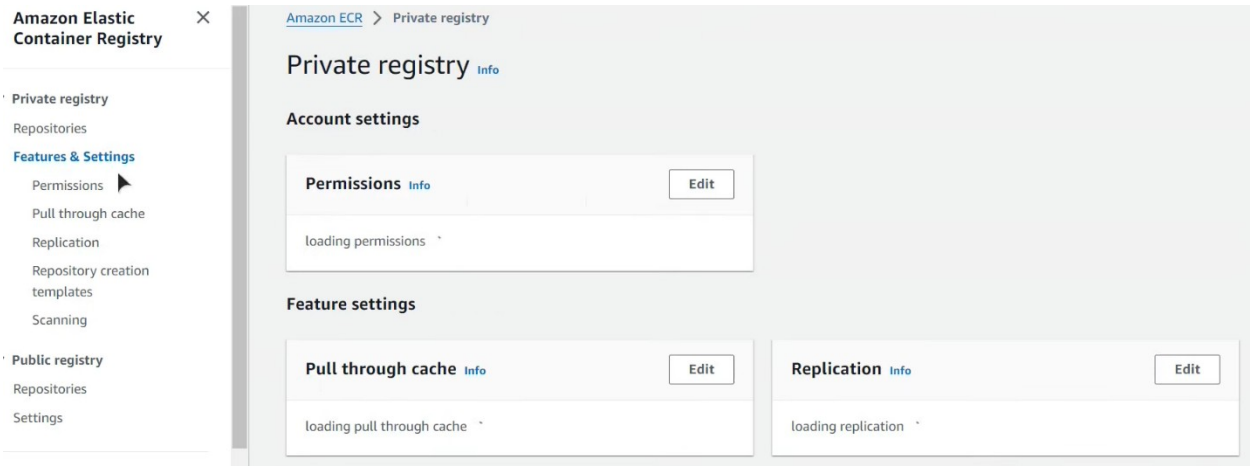
 # Refresh 5 sec

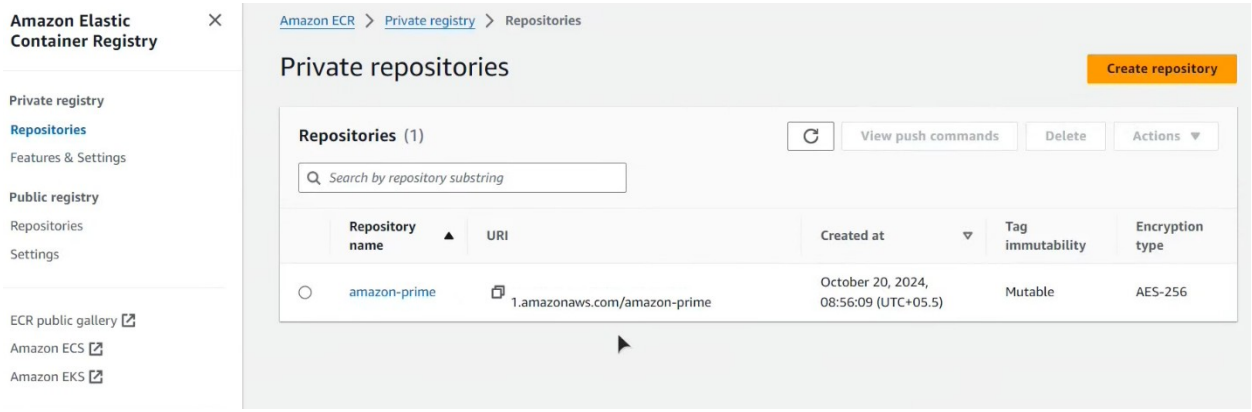


 # this is how we Monitor Grafana Dashboard to check status of Pods

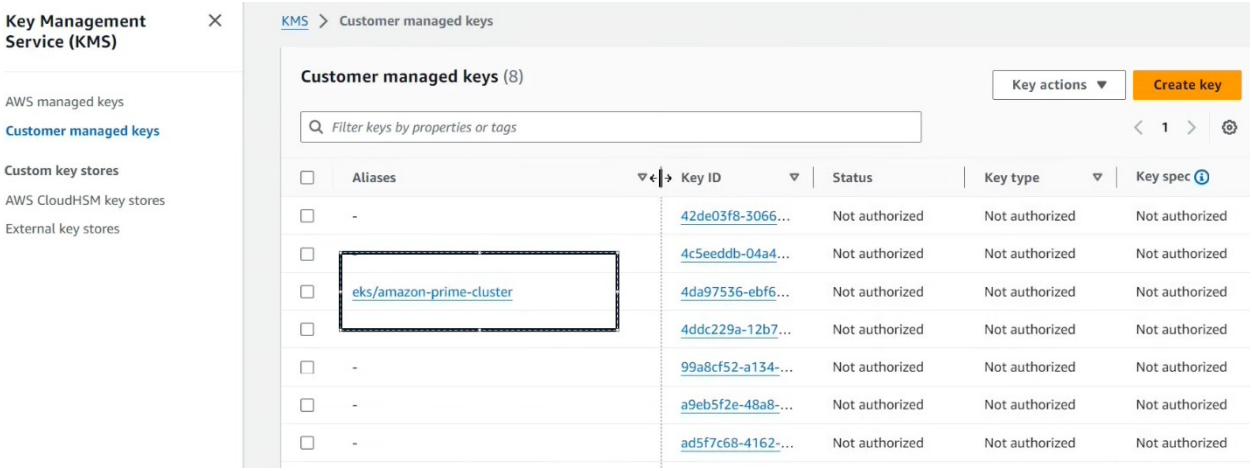
 # cleanup

 # AWS KMS will be created default in EKS Cluster … We need to remove it





 # AWS KMS Key –



 # cleanup pipeline

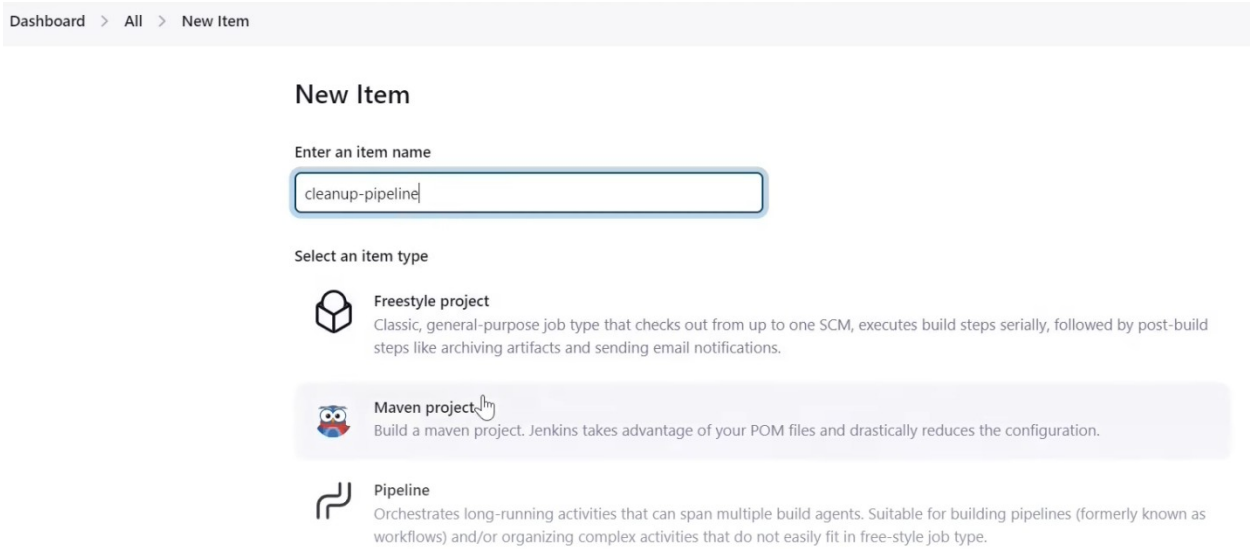
Welcome | cleanup_pipeline

```groovy
pipeline {
    agent any

    environment {
        KUBECTL = '/usr/local/bin/kubectl'
    }

    parameters {
        string(name: 'CLUSTER_NAME', defaultValue: 'amazon-prime-cluster', description: 'Enter your EKS cluster name')
    }

    stages {

        stage("Login to EKS") {
            steps {
                script {
                    withCredentials([string(credentialsId: 'access-key', variable: 'AWS_ACCESS_KEY'),
                    string(credentialsId: 'secret-key', variable: 'AWS_SECRET_KEY')]) {
                    sh "aws eks --region us-east-1 update-kubeconfig --name ${params.CLUSTER_NAME}"
                    }
                }
            }
        }
```

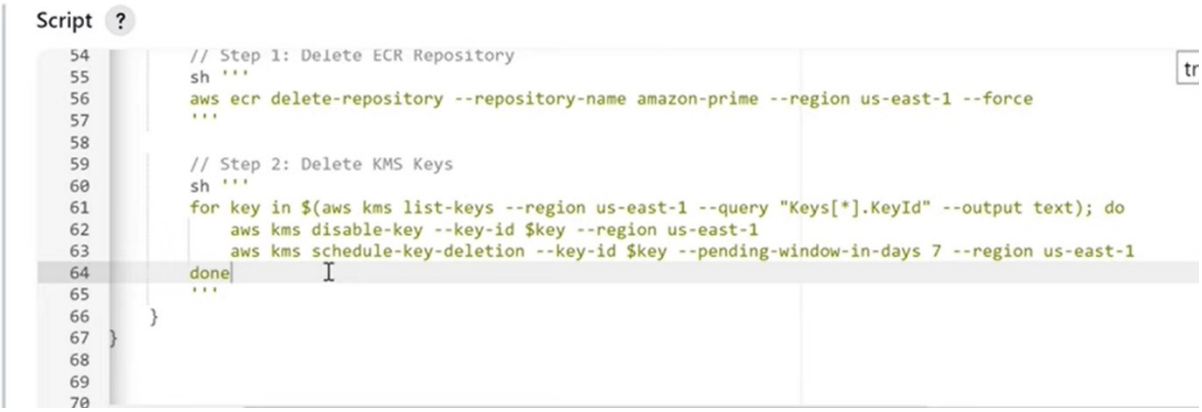
```groovy
    stage('Cleanup K8s Resources') {
        steps {
            script {
                // Step 1: Delete services and deployments
                sh 'kubectl delete svc kubernetes || true'
                sh 'kubectl delete deploy pandacloud-app || true'
                sh 'kubectl delete svc pandacloud-app || true'

                // Step 2: Delete ArgoCD installation and namespace
                sh 'kubectl delete -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml || true'
                sh 'kubectl delete namespace argocd || true'

                // Step 3: List and uninstall Helm releases in prometheus namespace
                sh 'helm list -n prometheus || true'
                sh 'helm uninstall kube-stack -n prometheus || true'

                // Step 4: Delete prometheus namespace
                sh 'kubectl delete namespace prometheus || true'

                // Step 5: Remove Helm repositories
                sh 'helm repo remove stable || true'
                sh 'helm repo remove prometheus-community || true'
```

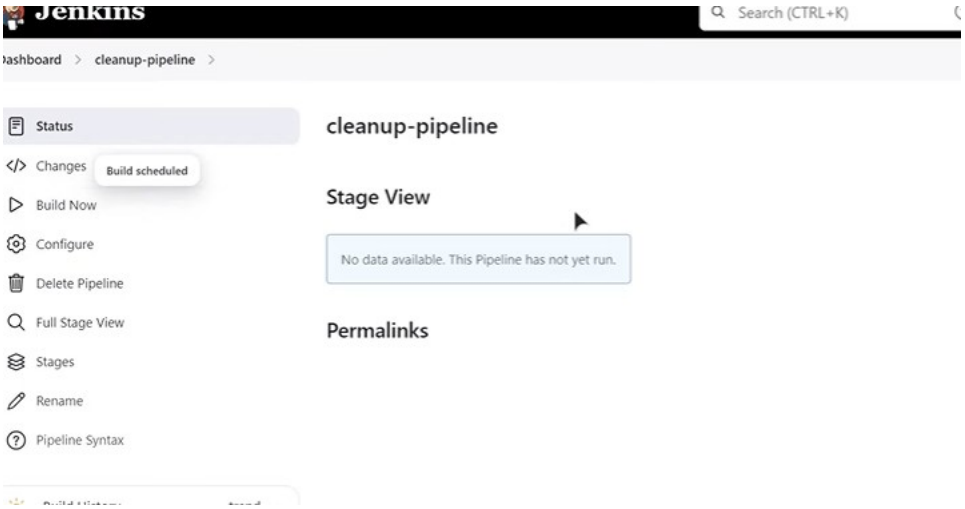
```groovy
stage('Delete ECR Repository and KMS Keys') {
    steps {
        script {
            // Step 1: Delete ECR Repository
            sh '''
            aws ecr delete-repository --repository-name amazon-prime --region us-east-1 --force
            '''

            // Step 2: Delete KMS Keys
            sh '''
            for key in $(aws kms list-keys --region us-east-1 --query "Keys[*].KeyId" --output text); do
                aws kms disable-key --key-id $key --region us-east-1
                aws kms schedule-key-deletion --key-id $key --pending-window-in-days 7 --region us-east-1
            done
            '''
        }
```

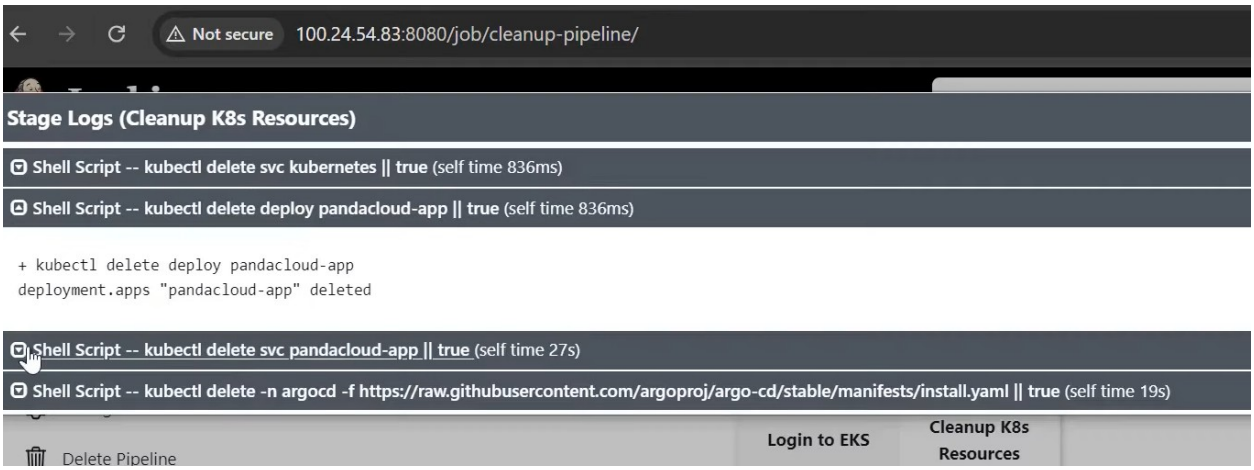
 # Jenkins

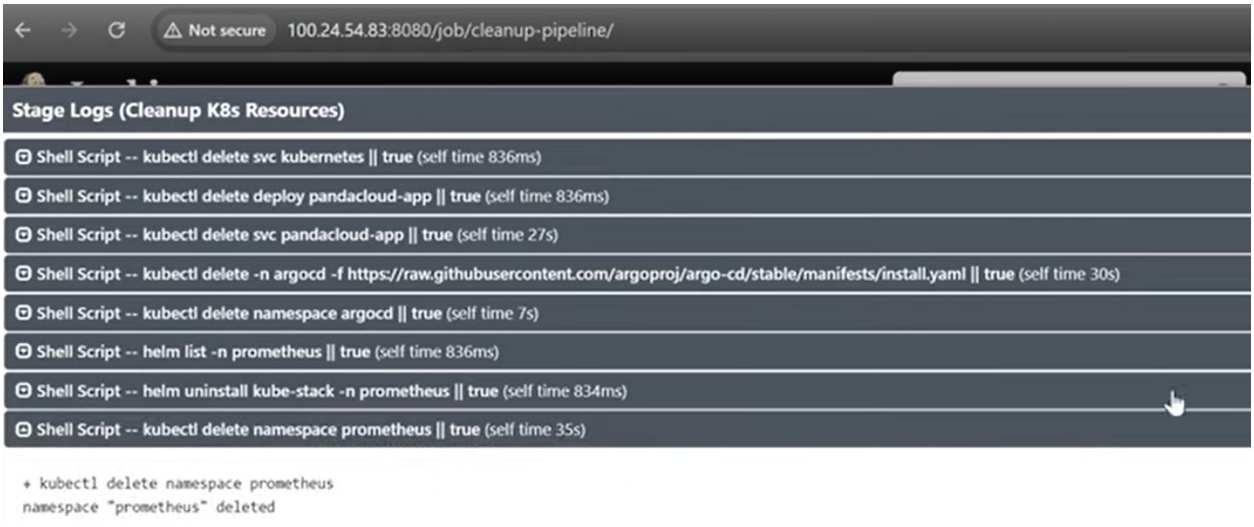


 # this will Clean up exsisting + default Keys

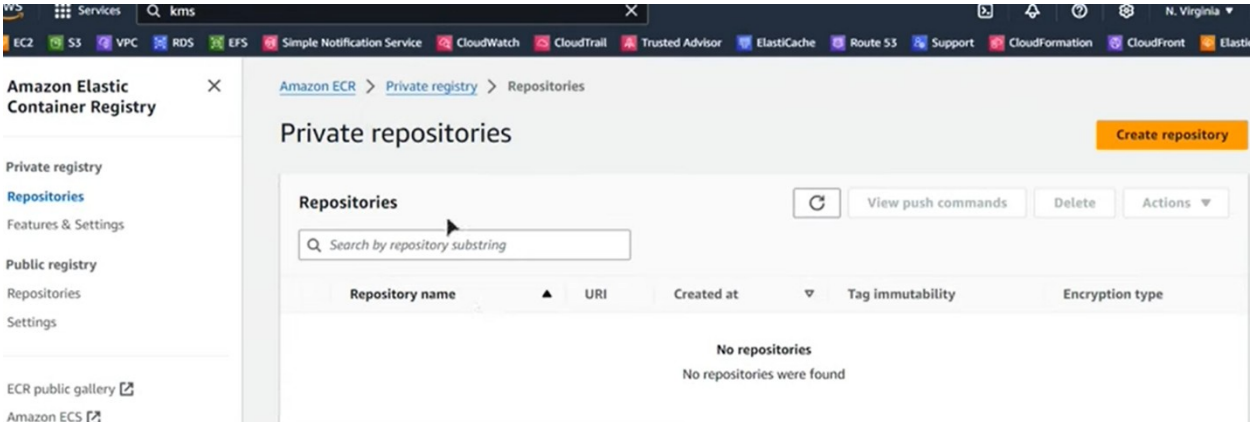




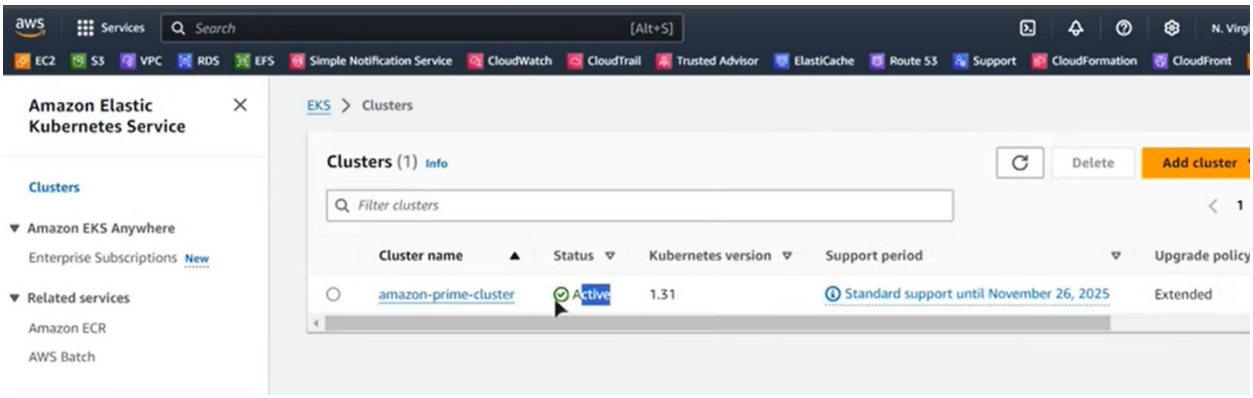




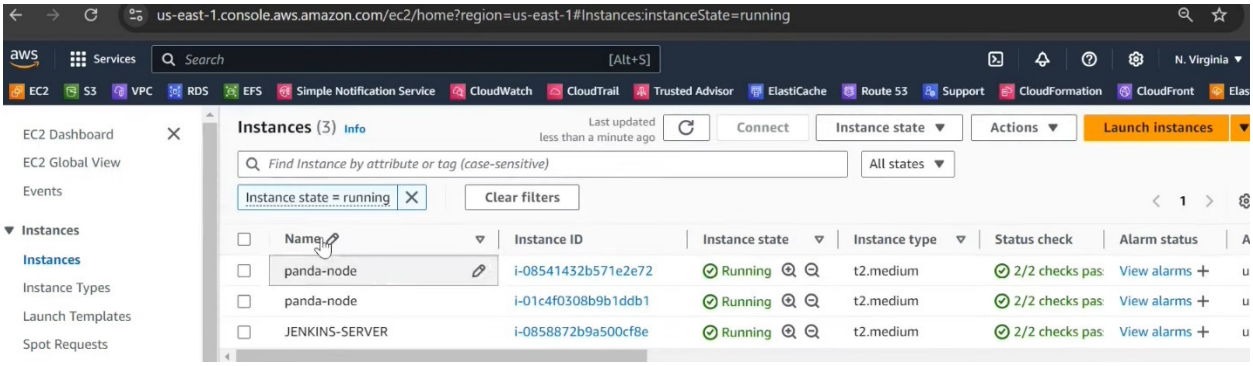
 # ECR Repo cleaned up



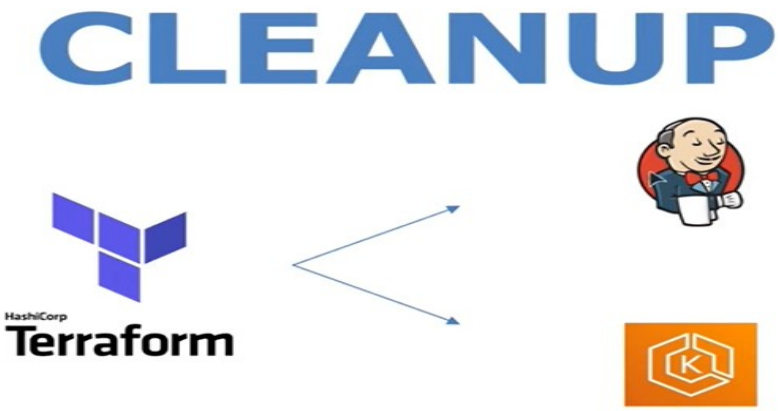
 # EKS Still Running



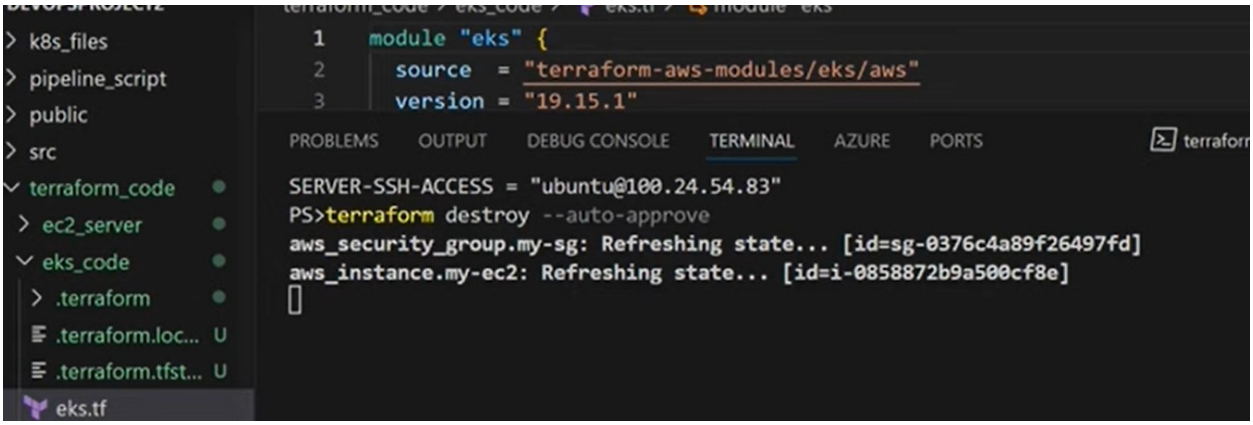
 # Even EC2 – server stil running



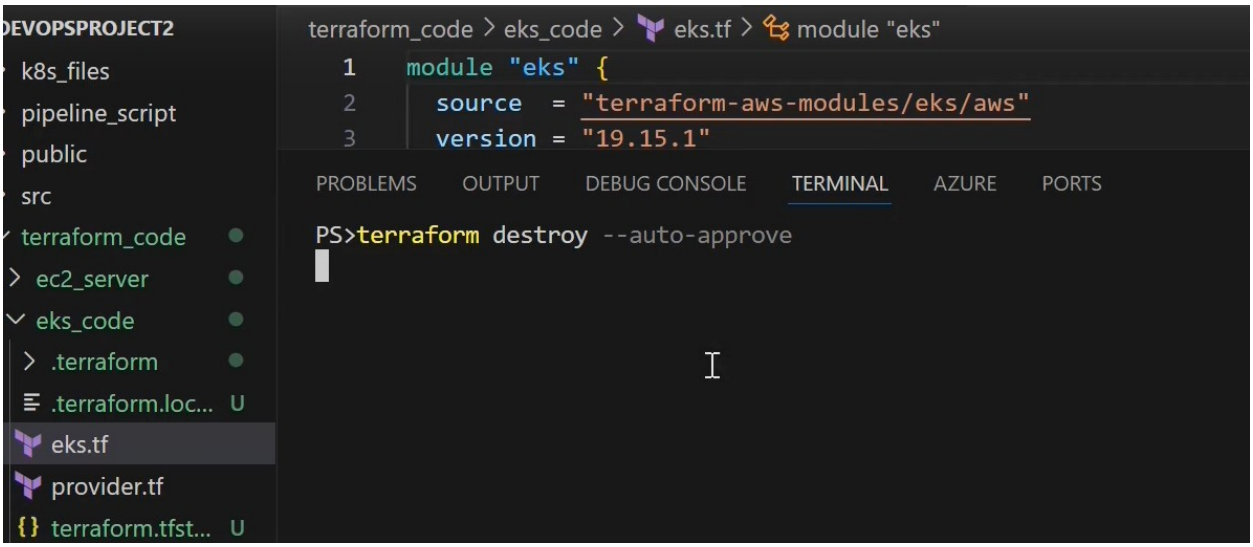
 # 2nd part Clean up Jenkins + EKS using Terraform Destroy

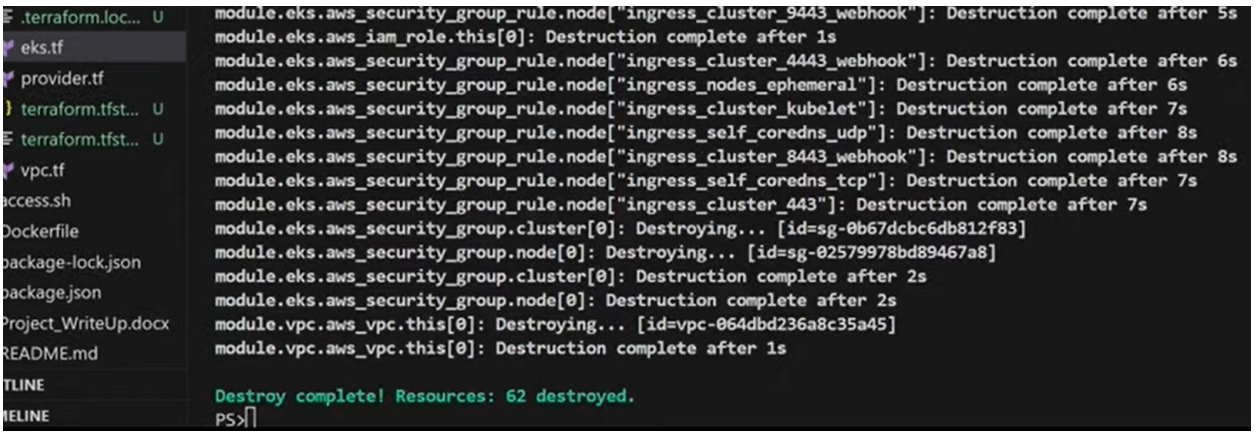


 # EC2 instances

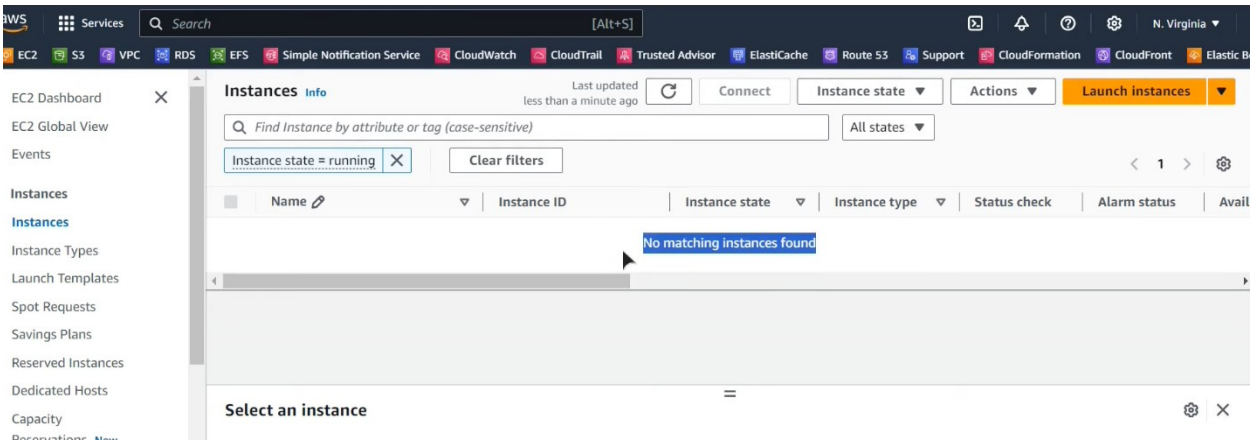


 # EKS Server





 # AWS EC2 – no instances



 # No Cluster EKS