SHRIRAM KARPOORA SUNDRA PANDIAN
DEVSECOPS
TOOLS: JENKINS, SONAR, SNYK, AWS, ZAP, DOCKER, KUBERNETES
INSTRUCTOR: SHIKHAR VERMA

SHRIRAM: I have created this document with step by step guide on mastering the devsecops and it's tools. I got a great instructor for this course and followed his guidance. I am sharing this document for educational purpose and other to learn devsecops and be better at cybersecurity.

DEVSECOPS
DevSecOps integrates security practices into this pipeline, ensuring that security is treated as a shared responsibility throughout the software development lifecycle. By embedding security checks and compliance validations early and frequently, DevSecOps reduces vulnerabilities and enhances application resilience.

CI/CD
A **CI/CD pipeline** (Continuous Integration/Continuous Deployment) is a set of automated processes that enable developers to integrate code changes, test them, and deploy applications seamlessly. It ensures rapid delivery of high-quality software by automating repetitive tasks such as testing, building, and deployment.

AWS
AWS provides an excellent platform for implementing CI/CD pipelines and DevSecOps due to its scalability, comprehensive suite of services, and deep integrations. AWS services like **CodePipeline**, **CodeBuild**, and **CodeDeploy** streamline pipeline automation, while **AWS Security Hub**, **GuardDuty**, and **IAM** offer robust security capabilities. Additionally, AWS's global infrastructure ensures high availability and low-latency deployments, making it a top choice for organizations looking to optimize their development and security workflows.

Enough introduction let's dirty our hands through work.
Starting with Maven, what is maven?
Maven is a build automation and project management tool primarily used in Java-based projects. It is part of the **Apache Software Foundation** and simplifies the process of managing project dependencies, building, packaging, and deploying applications. Maven uses an **XML file (pom.xml)** to describe the project's dependencies, build process, and configuration, enabling consistent and repeatable builds.

MAVEN

1. Kick start the amazon ec2 instance (Amazon Linux) and open using putty or any remote client handler you like.
2. Go to /opt folder and install java 11 and maven through the following tutorial.
3. Install Java 11 or any version of your choice.
   https://docs.aws.amazon.com/corretto/latest/corretto-11-ug/amazon-linux-install.html

```
sudo yum install java-11-amazon-corretto-headless
```

Option 2: Install the headful Amazon Corretto 11:

```
sudo yum install java-11-amazon-corretto
```

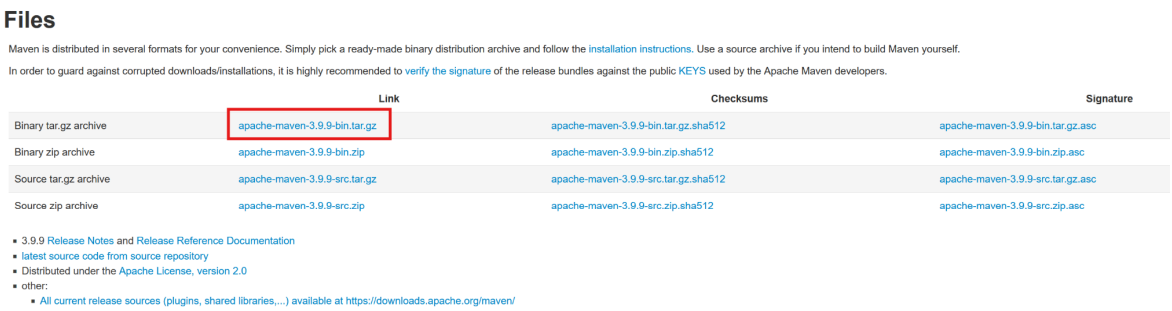
Option 3: Install the JDK for Amazon Corretto 11 (Amazon Linux 2023 only):

```
sudo yum install java-11-amazon-corretto-devel
```

Option 4: Install the JMods for Amazon Corretto 11 (Amazon Linux 2023 only):

```
sudo yum install java-11-amazon-corretto-jmods
```

4. Next step install Maven by visiting the Maven website and download the binary source of tar.gz file. https://maven.apache.org/download.cgi



Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.9.9-bin.tar.gz	apache-maven-3.9.9-bin.tar.gz.sha512	apache-maven-3.9.9-bin.tar.gz.asc
Binary zip archive	apache-maven-3.9.9-bin.zip	apache-maven-3.9.9-bin.zip.sha512	apache-maven-3.9.9-bin.zip.asc
Source tar.gz archive	apache-maven-3.9.9-src.tar.gz	apache-maven-3.9.9-src.tar.gz.sha512	apache-maven-3.9.9-src.tar.gz.asc
Source zip archive	apache-maven-3.9.9-src.zip	apache-maven-3.9.9-src.zip.sha512	apache-maven-3.9.9-src.zip.asc

- 3.9.9 Release Notes and Release Reference Documentation
- latest source code from source repository
- Distributed under the Apache License, version 2.0
- other:
  - All current release sources (plugins, shared libraries,...) available at https://downloads.apache.org/maven/

5. Extract the zip using tar -xvf “filename” and extract the folder.
6. Add it to the path through this command “export PATH=$PATH:/opt/apache-maven(filename)/bin
7. Once java 11 and maven is successfully installed go to the root folder again.
8. We are going to use the following command for maven utilization which compiles and takes care of the project.

   - `validate` - validate the project is correct and all necessary information is available
   - `compile` - compile the source code of the project
   - `test` - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
   - `package` - take the compiled code and package it in its distributable format, such as a JAR.
   - `verify` - run any checks on results of integration tests to ensure quality criteria are met
   - `install` - install the package into the local repository, for use as a dependency in other projects locally
   - `deploy` - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

9. In the root folder take any java project and clone it here using git clone “project.git” command.
10. Once you get the repository. Navigate inside the directory using the cd command and type “maven validate” to start the maven or initialize it on the git folder.
11. Then type “maven package” which does all three actions at once (validate, compile, and test). Once you are done with the command you can see the target folder.

```
[root@ip-172-31-81-176 springboot-hello]# ls -ltr
total 16
drwxr-xr-x. 3 root root   18 Nov 16 19:04 src
-rw-r--r--. 1 root root  731 Nov 16 19:04 azure-pipelines.yml
-rw-r--r--. 1 root root  325 Nov 16 19:04 README.md
-rw-r--r--. 1 root root   95 Nov 16 19:04 Dockerfile
-rw-r--r--. 1 root root 1596 Nov 16 19:18 pom.xml
drwxr-xr-x. 6 root root  159 Nov 16 19:18 target
```

12. Go inside the target folder and you can see .gz file has been created which can be used for the application or be can call it as the application build.
13. Go back to the root directory using cd ~ and find the hidden directories through “ls -ltr” and navigate to .m2/repository and you can the dependencies for the maven project.

```
[root@ip-172-31-81-176 .m2]# cd repository
[root@ip-172-31-81-176 repository]# ls
aopalliance               ch           commons-cli      io      log4j
asm                       classworlds  commons-io       javax   net
backport-util-concurrent  com          commons-logging  junit   org
```

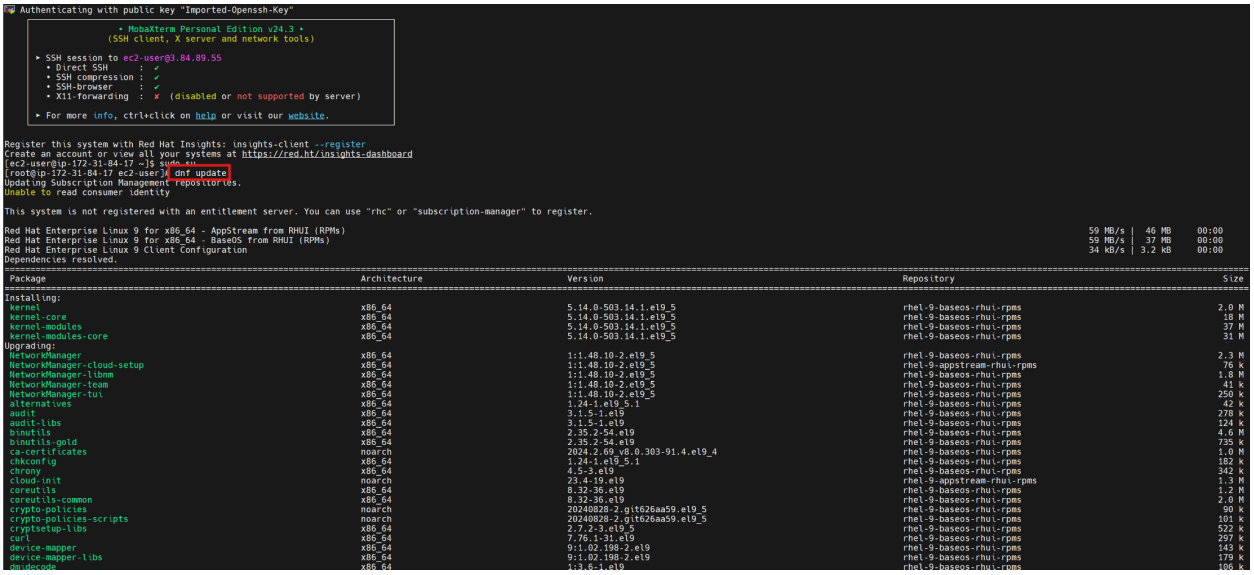
14. Whatever you make changes to the code in the future to get the fresh and clean bundle from maven simply run “maven clean package”.

SAST (Static Application Security Testing)
SonarQube and Sonarcloud
SonarQube is an open-source platform used for **continuous code quality inspection** and **static code analysis**. It helps developers and teams improve code quality and maintainability by detecting code smells, bugs, security vulnerabilities, and technical debt. SonarQube supports a wide range of programming languages and integrates seamlessly with CI/CD pipelines to automate code reviews.

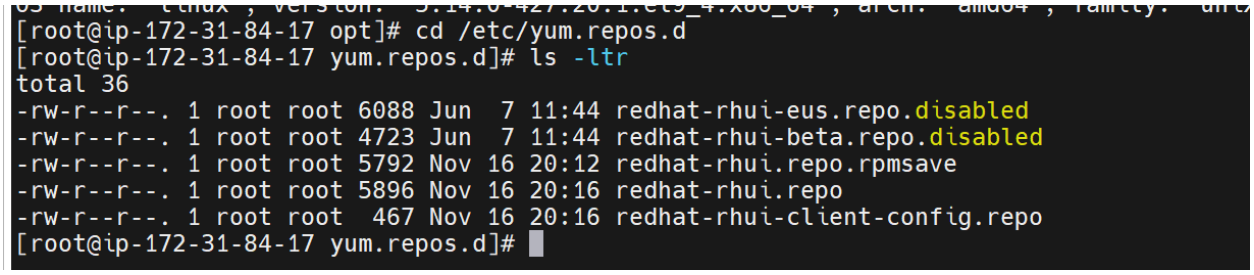
To Test the source code in the early stage of the process while the continuous integration happens. We can say simply we will test the static code before it pushed into deployment.

Before going deep into the SAST lets start the jenkins server which is the heart of the CI Integration part.

Go to AWS and start the red hat linux with t2.medium for better speed and update the system using dnf update.

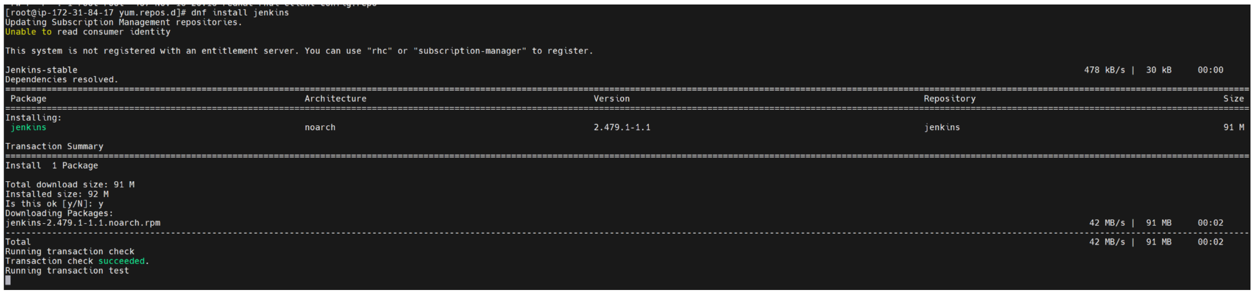


- Install maven like above steps and time to install jenkins
- Go to this folder on linux where you can see all the repositories where you can install the jenkins repository.



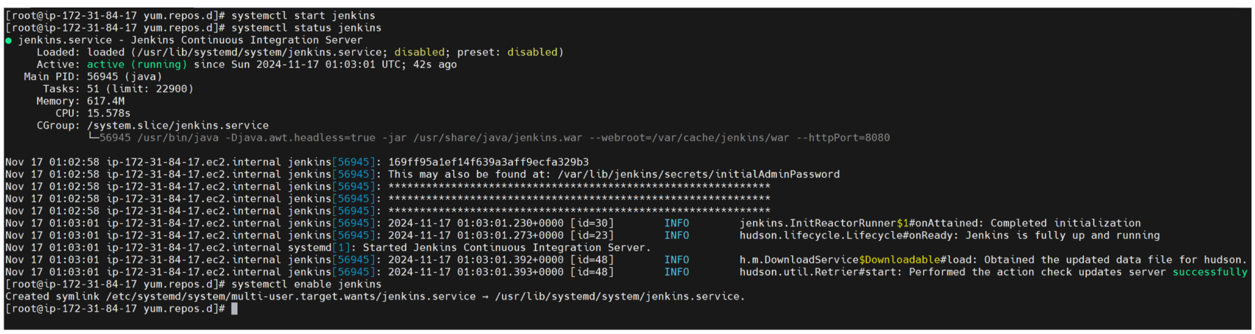
- https://pkg.jenkins.io/redhat-stable/ Navigate to this site and install the first two commands.

- Now let's install the jenkins by entering this command: "dnf install jenkins"



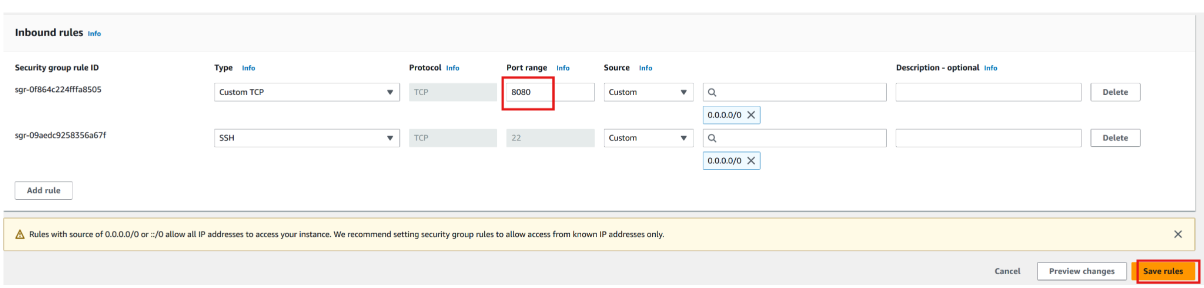
What is Jenkins?
Jenkins is an open-source automation server used for **continuous integration (CI)** and **continuous delivery (CD)** in software development. It enables developers to automate the building, testing, and deployment of applications, ensuring a faster and more reliable software delivery process. Jenkins is highly extensible with hundreds of plugins to integrate with various tools and technologies in the software development lifecycle.

- Start jenkins using the command “systemctl start jenkins”.

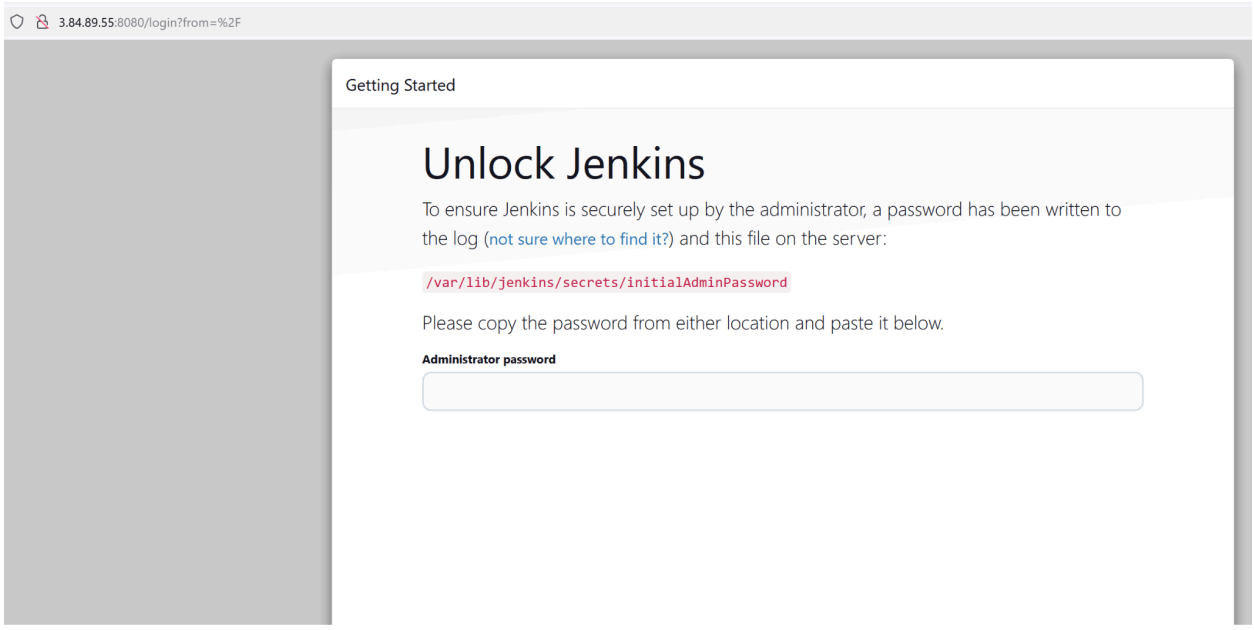


Make sure you enable the jenkins as well. (We use enable to make sure the service starts automatically when it boots up by creating symbolic links).

- We need to make sure that port 8080 is running properly so that jenkins can run. In aws configure inbound rules for you ec2 instance to allow port 8080.



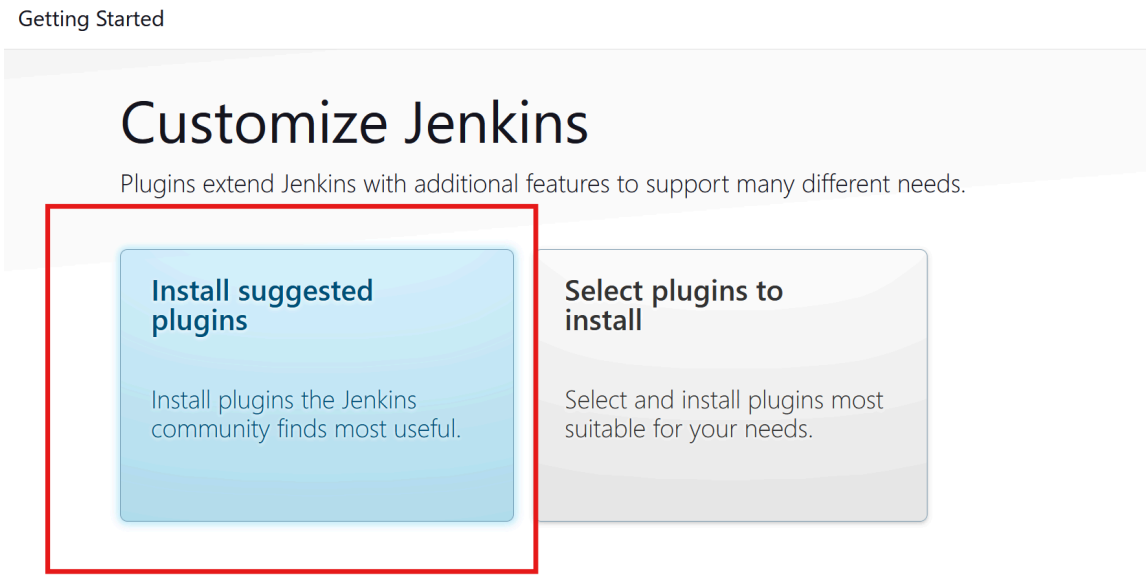
- At port 8080 with your public ip you should see the webpage of jenkins.

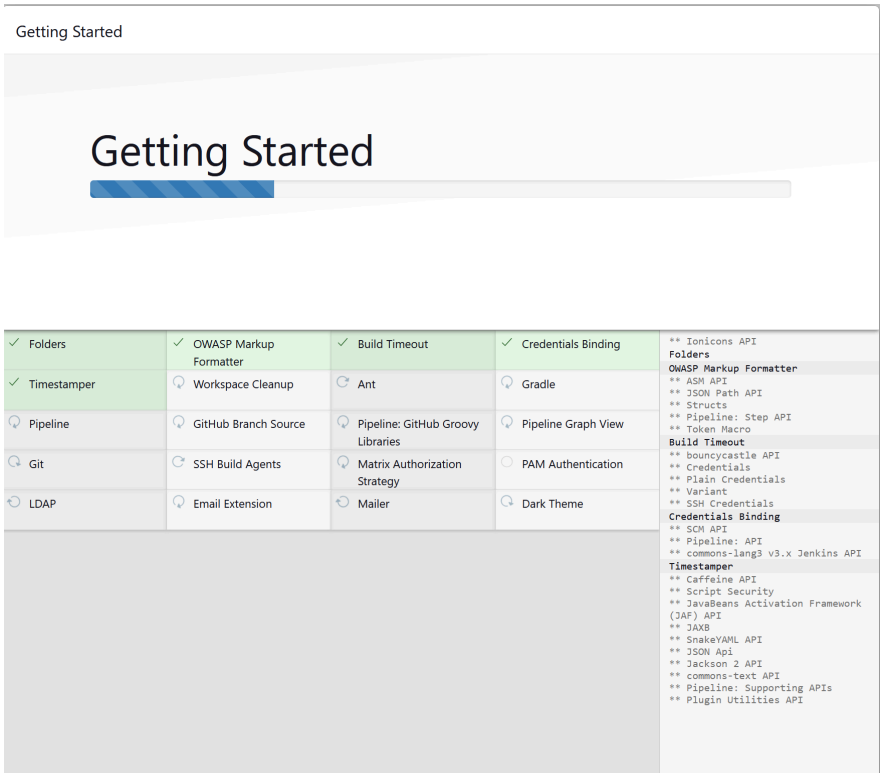


- As above it mentioned that password is on the following path, navigate to that path and copy the password to unlock.

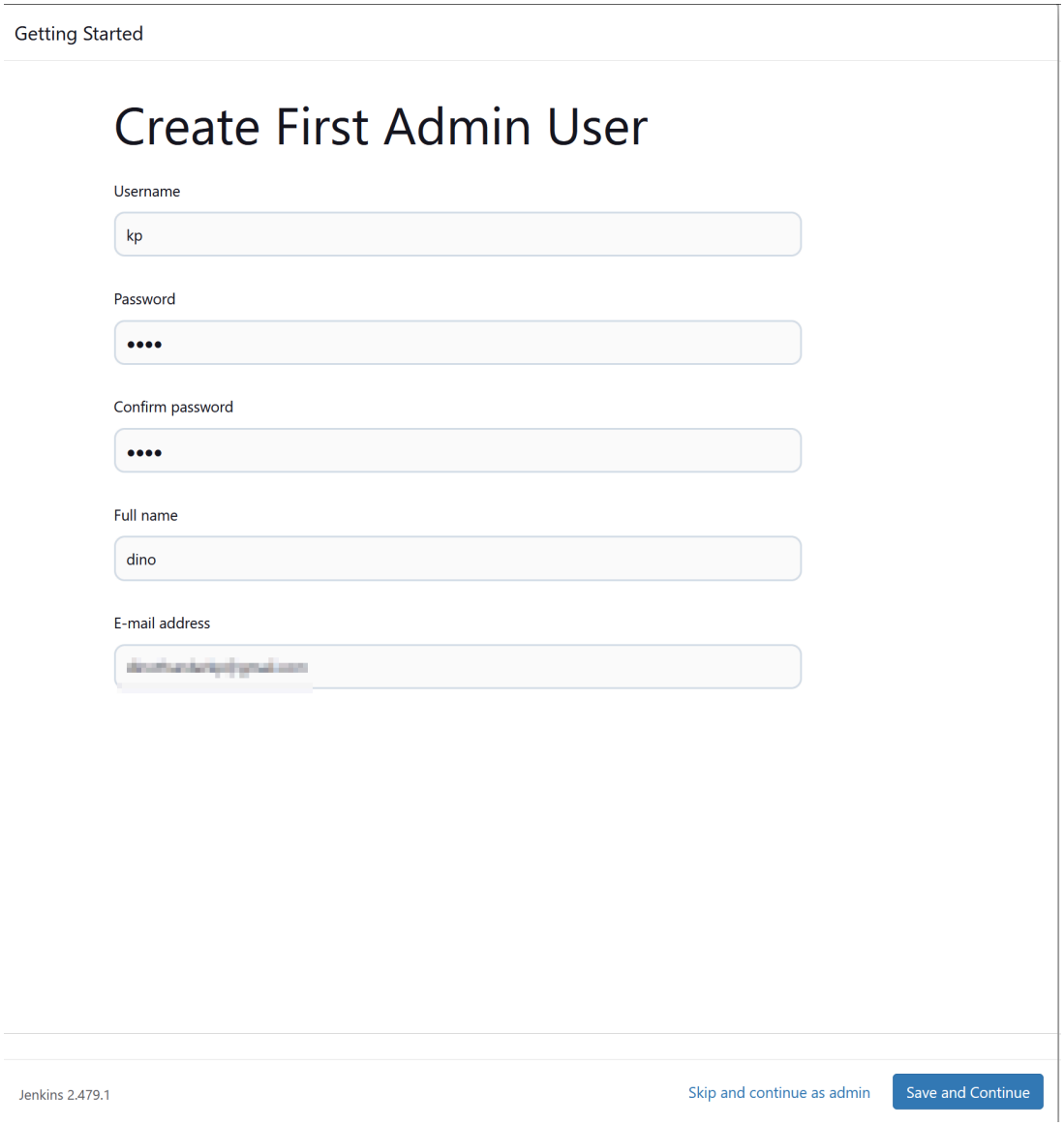
```
[root@ip-172-31-84-17 ~]# cat /var/lib/jenkins/secrets/initialAdminPassword
169ff95a1ef14f639a3aff9ecfa329b3
```

- Install the suggested plugins.

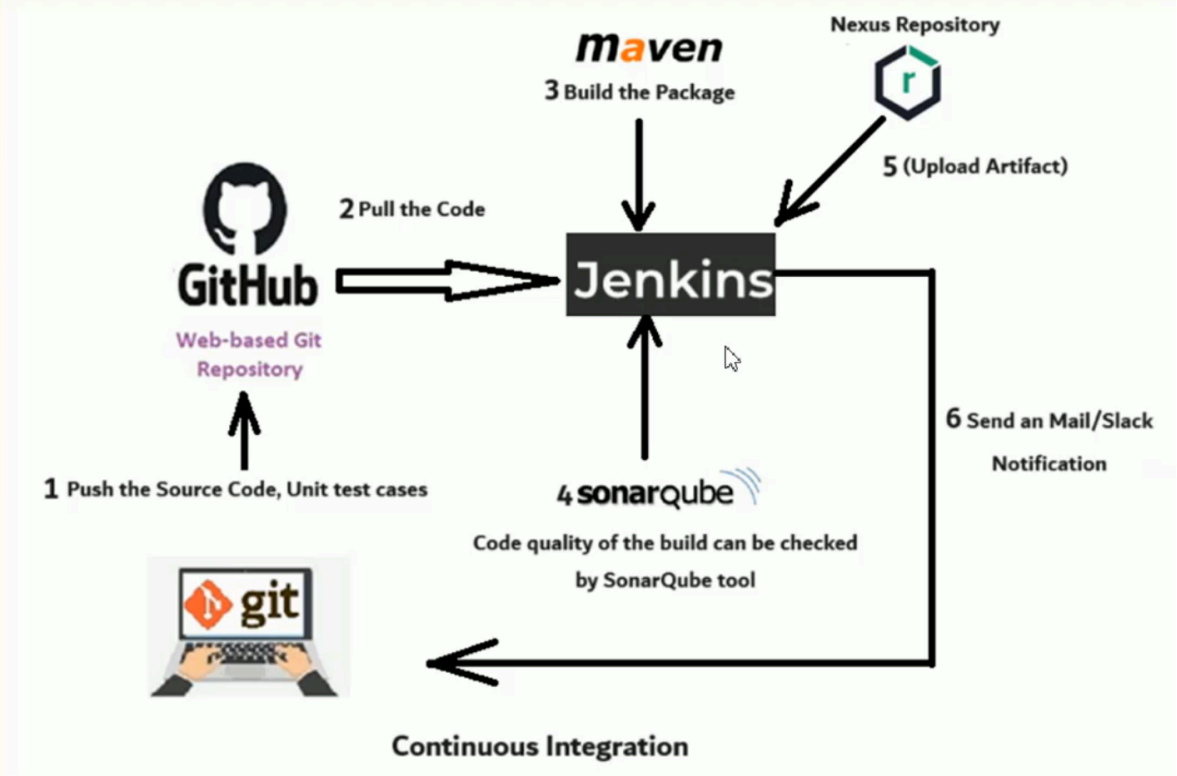




- Add your details for jenkins and save & continue.

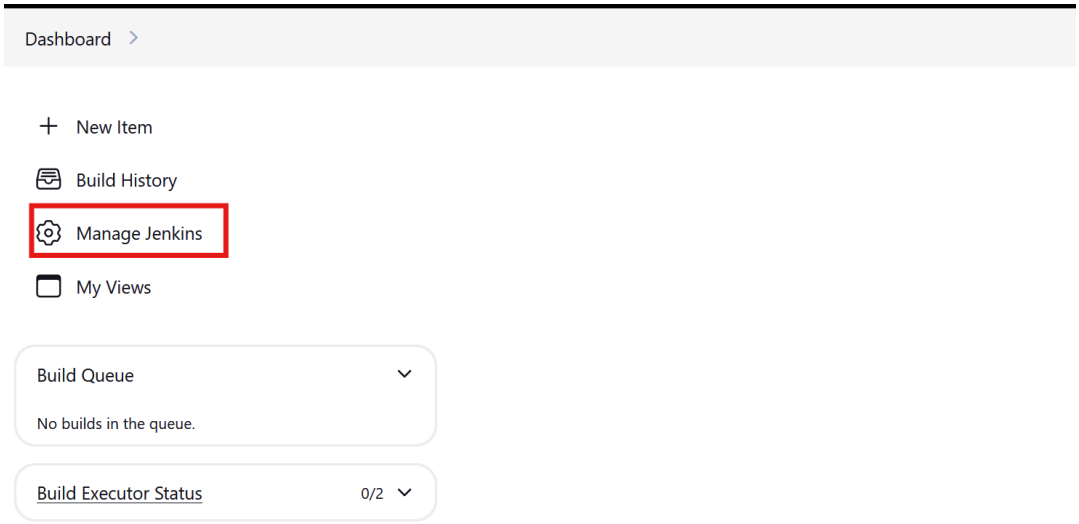


- Now it’s time to integrate maven, sonarqube and everything with jenkins like shown in the diagram.

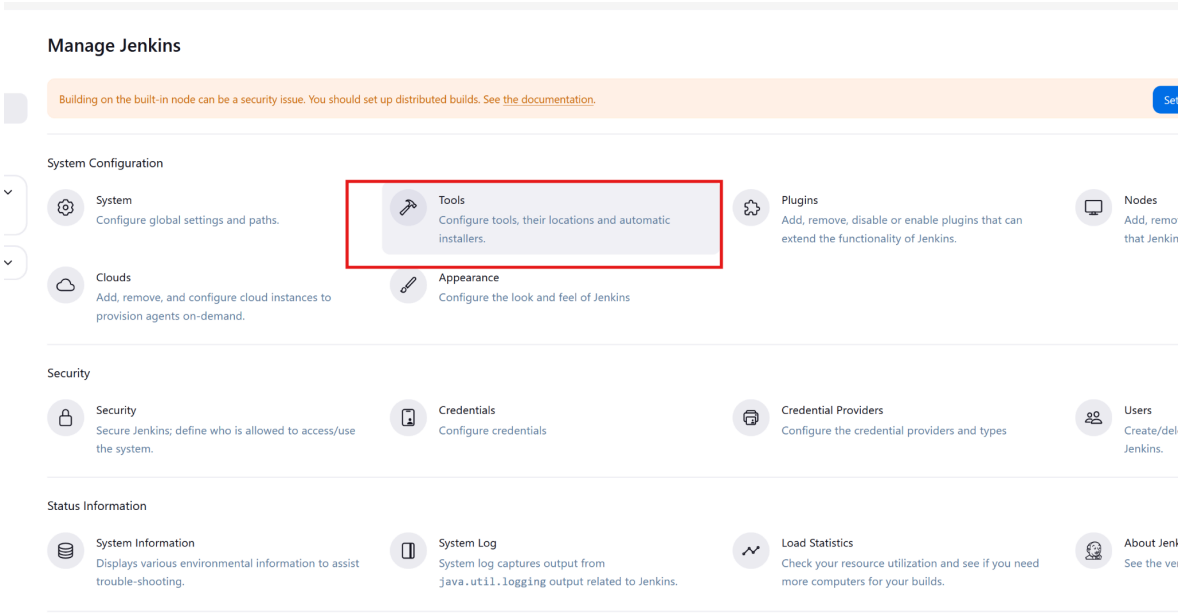


Maven Integration

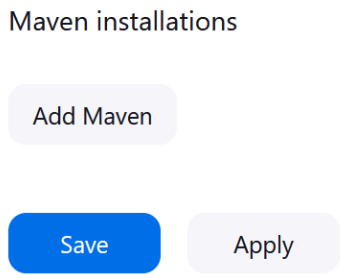
- Let’s integrate maven with jenkins.
- Click on manage jenkins.



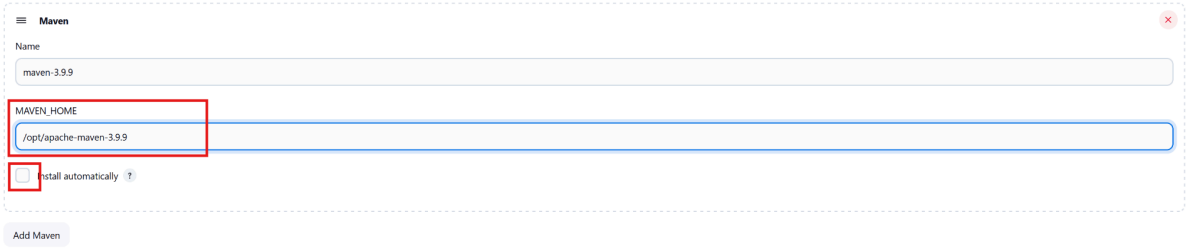
- Now click on tools.



- Go down and add maven.

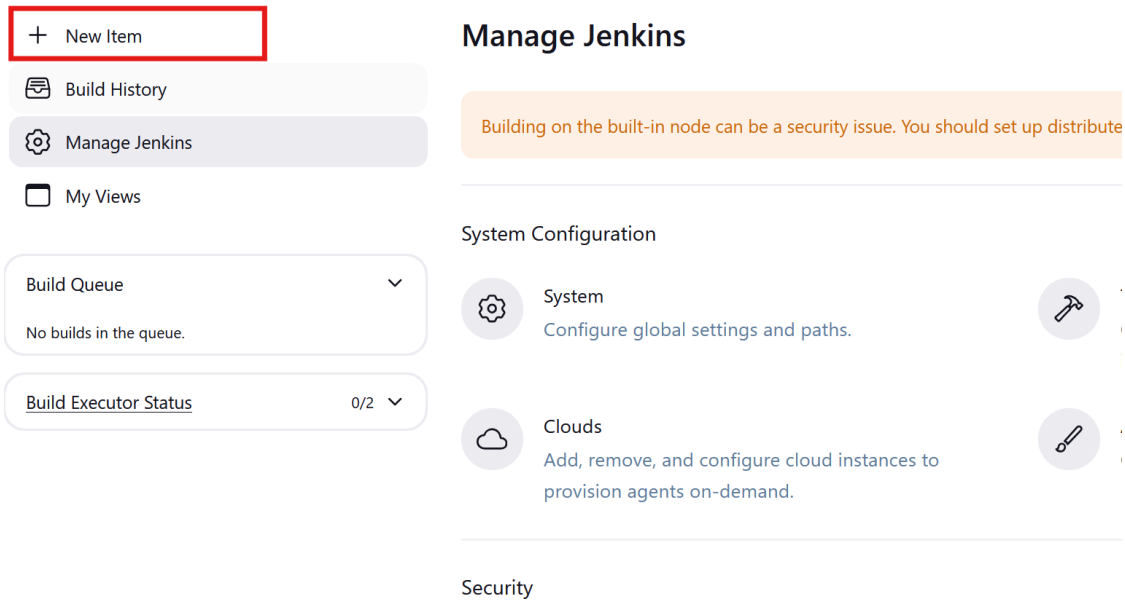


- Give any name and remove the install maven option as we already installed it and give the maven path by checking “mvn –version” on the virtual machine.

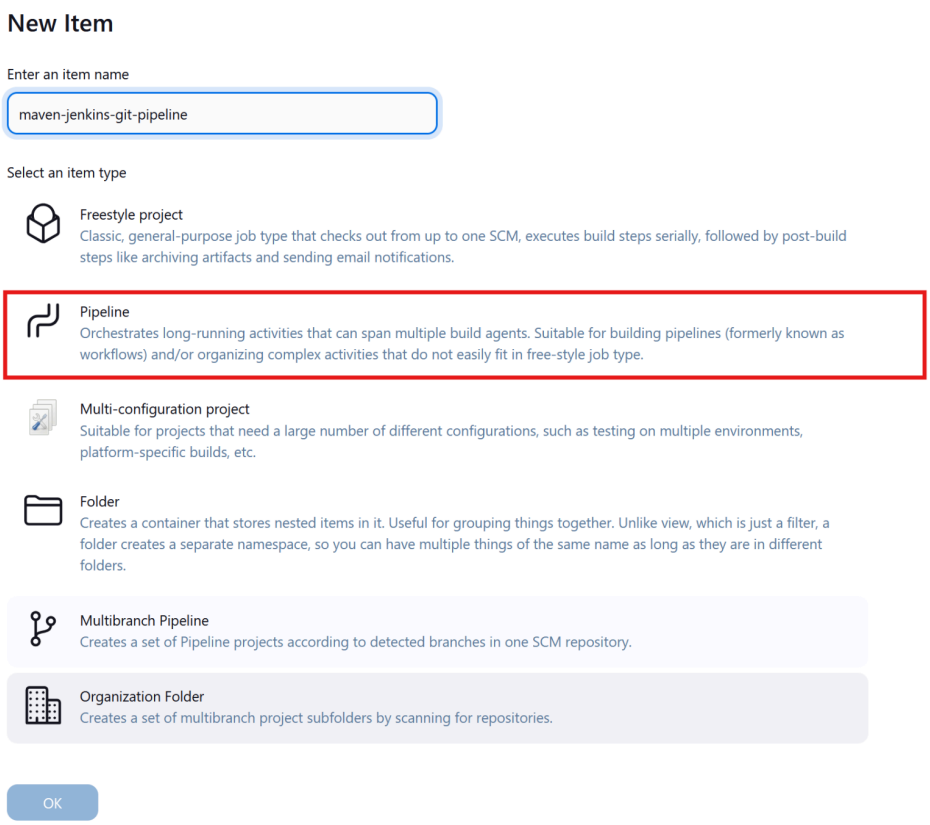



```
[root@ip-172-31-84-17 ~]# mvn --version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: /opt/apache-maven-3.9.9
Java version: 17.0.13, vendor: Red Hat, Inc., runtime: /usr/lib/jvm/java-17-openjdk-17.0.13.0.11-4.el9.x86_64
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.14.0-427.20.1.el9_4.x86_64", arch: "amd64", family: "unix"
```

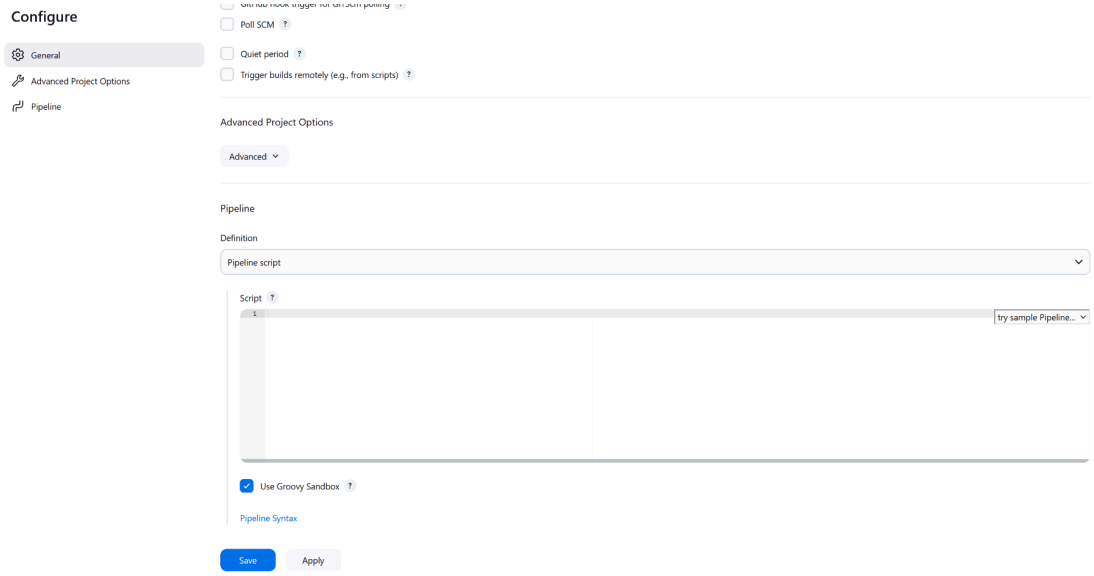
- This is the important step now lets create a declarative pipeline where we are going to write a script about on importing the git project for verifying it's status.
- Click on the new item on the left most part of the screen.



- Give any meaningful name and select pipeline and click ok.



- Go down and look for the script window.

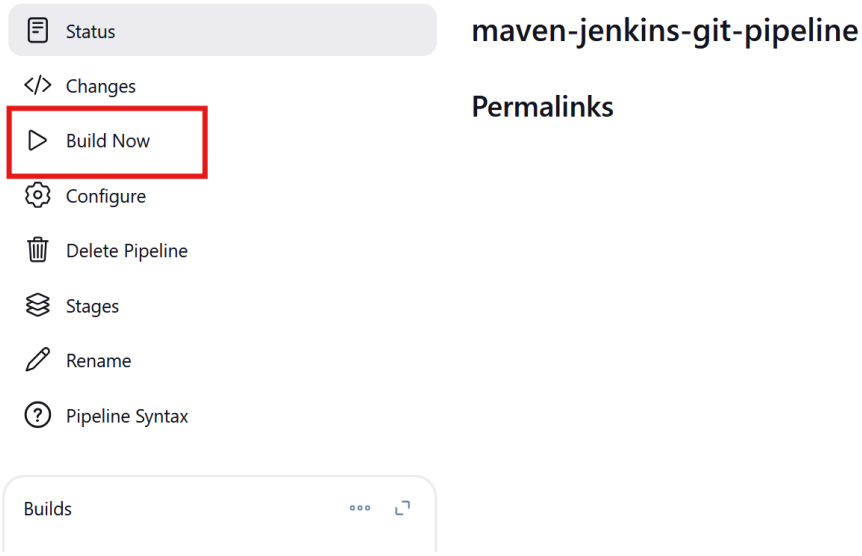


- Enter this script for declaring the github repository that has java code.

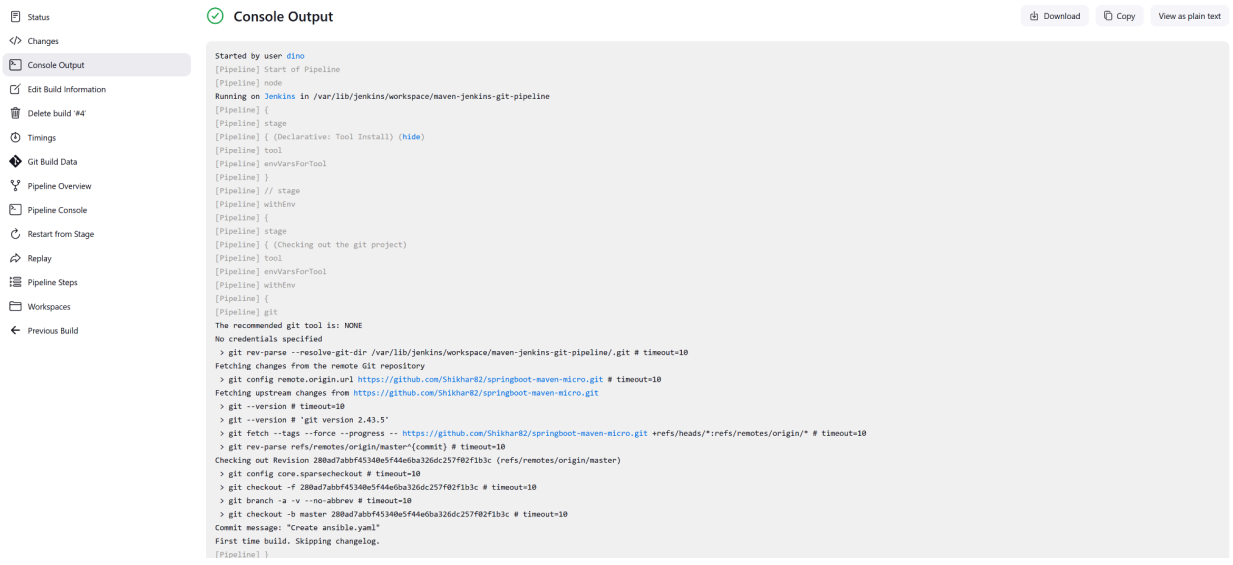

```
pipeline {
      agent any
      tools {
      maven 'maven-3.9.9'
      }
      stages {
      stage ("Checking out the git project") {
            steps {
                  git
```

```
'https://github.com/Shikhar82/springboot-maven-micro.git'
            }
      }
      }

}
```

- Click on build now after saving the script for executing it.



- Once you finish executing it, you will see the log success.



- Add the clean package script on the pipeline script like below for adding that feature.

Script ?

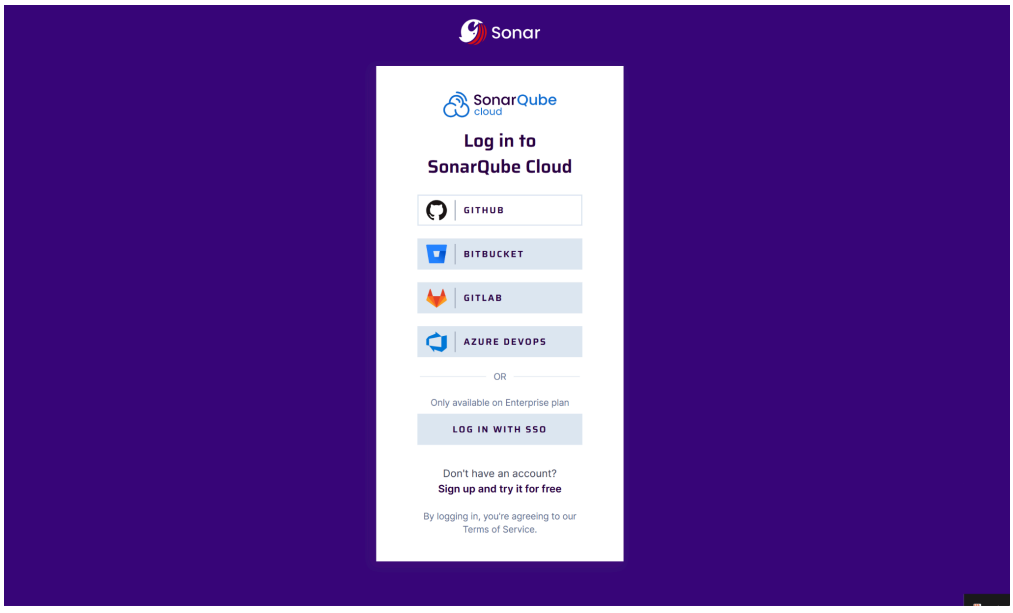
```
pipeline {
    agent any
    tools {
        maven 'maven-3.9.9'
    }
    stages {
        stage ("Checking out the git project") {
            steps {
                git 'https://github.com/Shikhar82/springboot-maven-micro.git'
            }
        }

        stage ("Building the package") {
            steps {
                sh 'mvn clean package'
            }
        }
    }
```

- Now try build again and check whether everything working fine.

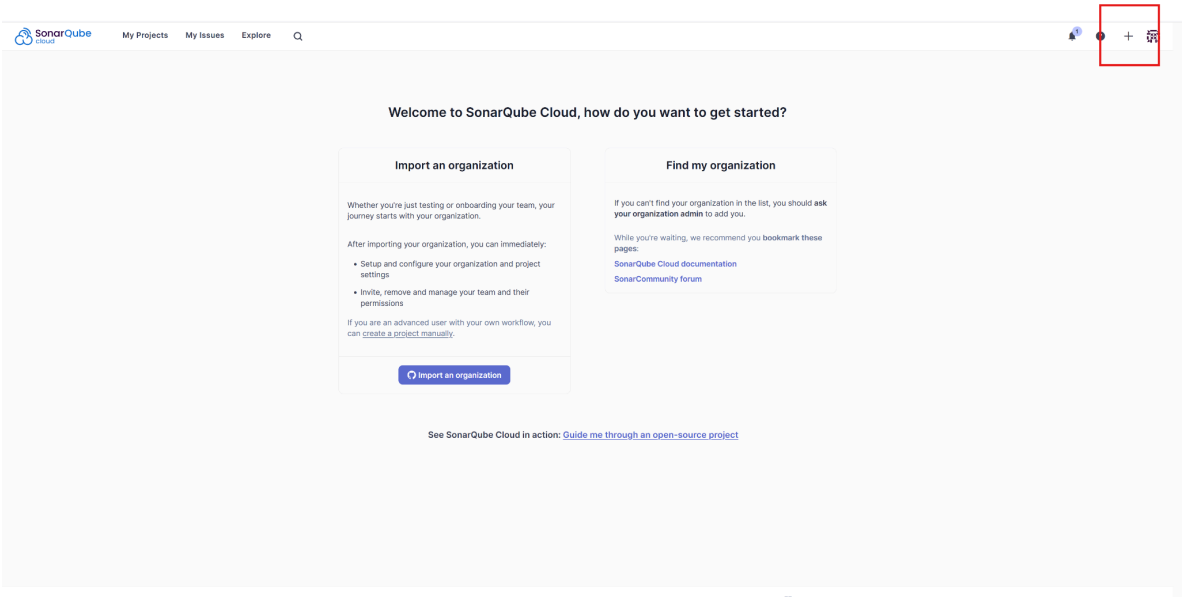
- We successfully integrated maven to create the project, but we still haven't integrated the SAST tool for checking vulnerabilities on our code. Let's integrate SAST with jenkins.

SAST Sonarcloud Integration

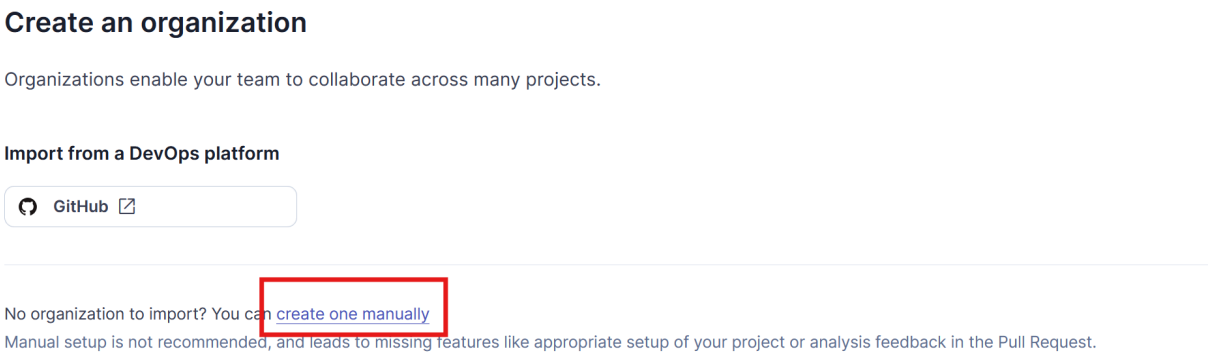
- Go to sonarcloud website and login using any credentials you have https://sonarcloud.io/login.



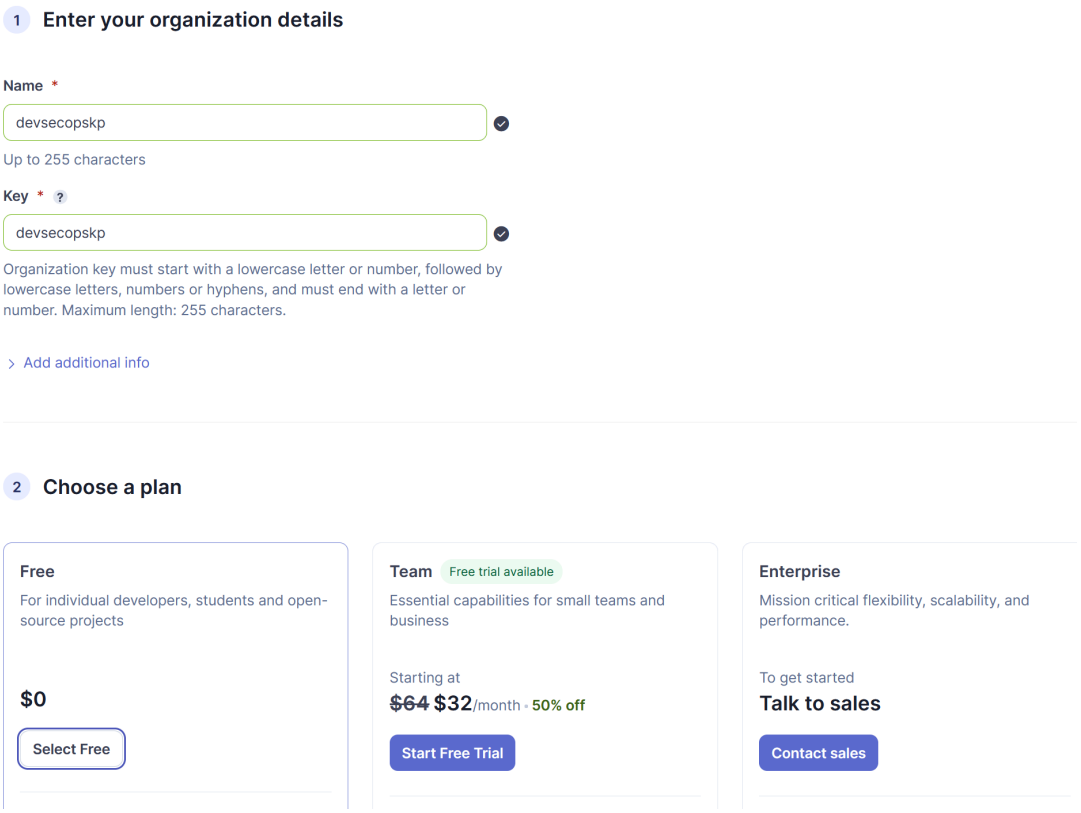
- Click on + button on the top right to create you own organization.



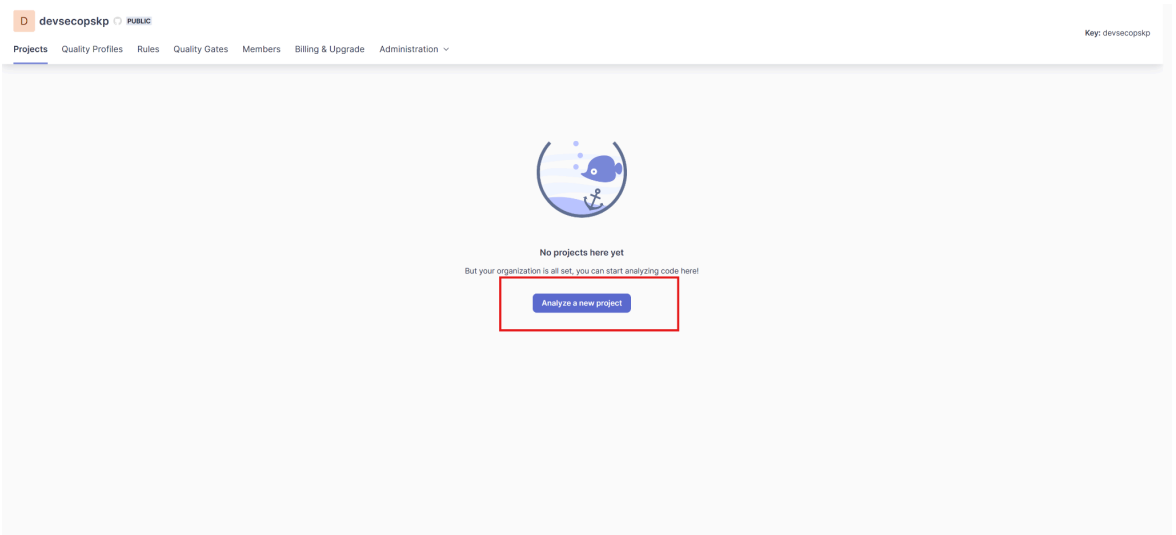
- Select create your own.

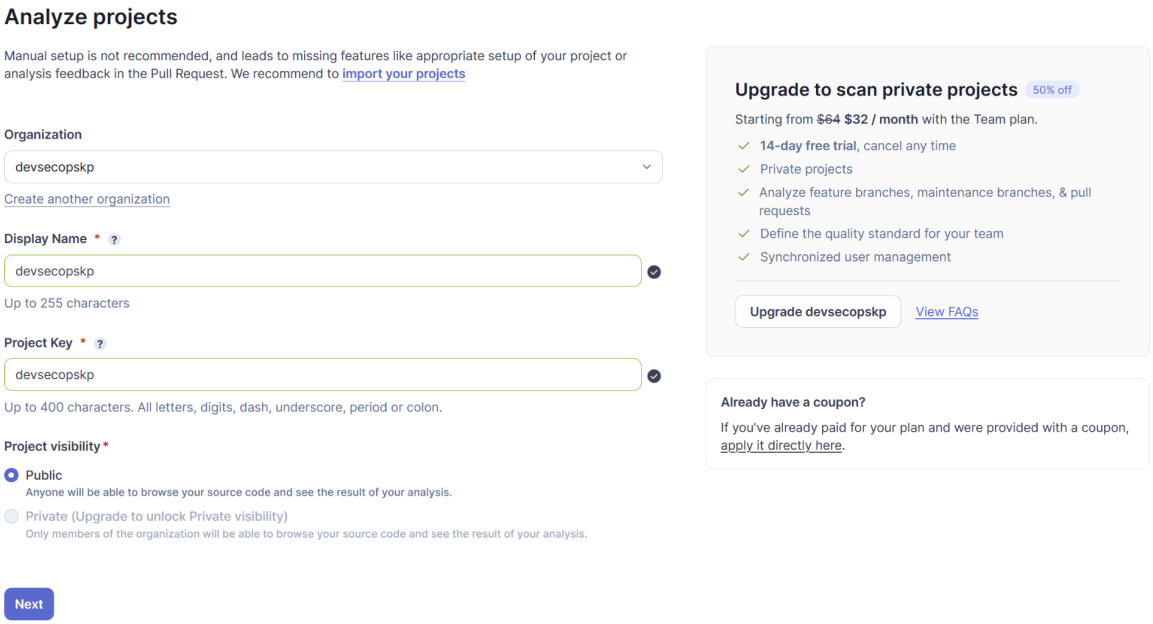


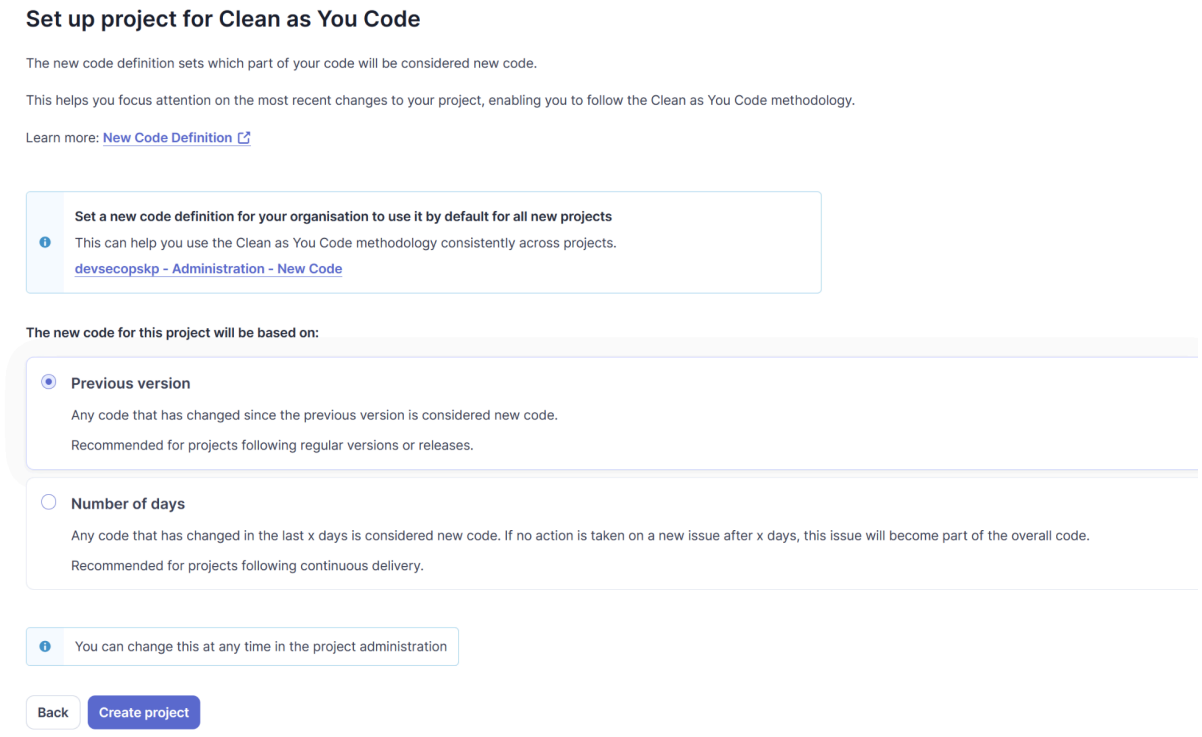
- Select any name you like for your organization and click on free plan and create your organization.



- After entering the organization windows, give analyze new project.

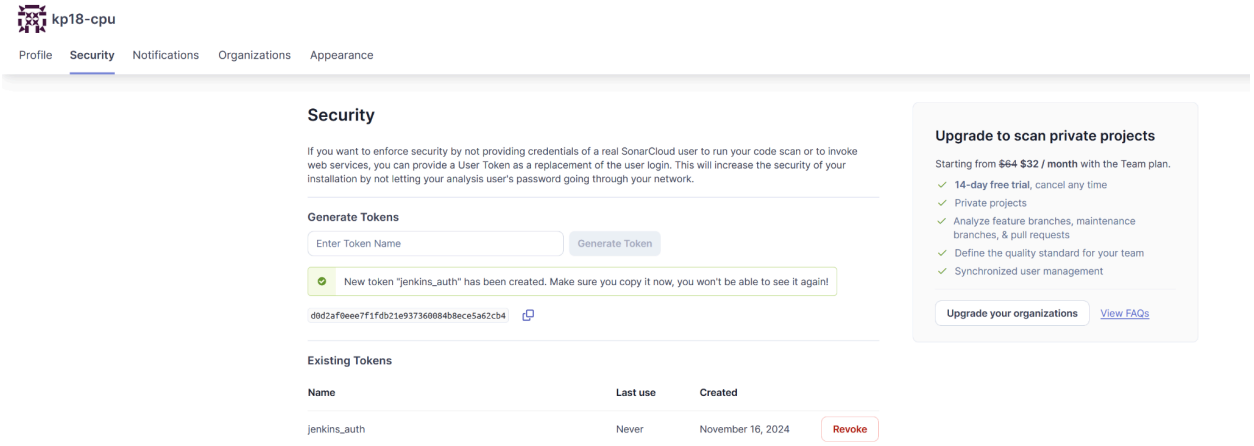






- After creating a new project.
- Before integrating sonarcloud with our jenkins pipeline we need to generate a token which is used to authenticate the jenkins server on using sonarcloud.

- Go to sonarcloud and click on myaccount>security and click on the token name and generate one.



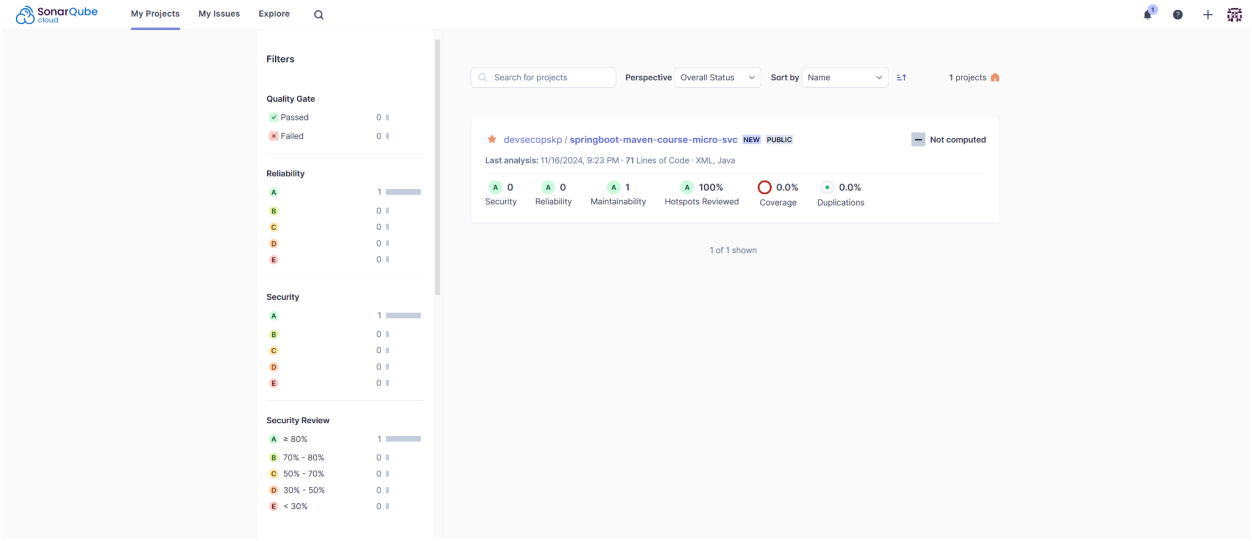
Keep the token safe, now lets go back to jenkins server and add script for integrating the sonarcloud.

- This is the script we are going to add.

```
mvn clean verify sonar:sonar -Dsonar.projectKey=devsecopskp
-Dsonar.organization=devsecopskp -Dsonar.host.url=https://sonarcloud.io
-Dsonar.token=d0d2af0eee7f1fdb21e937360084b8ece5a62cb4
```

- Script looks like this. (Make sure you remove clean package and add this for the time being)

- After build is success we can go to sonarcloud and my projet to see the results.



- Let’s do something fun, go this github and copy the url. 'https://github.com/Shikhar82/devsecops-jenkins-k8s-tf-sast-sca-sonarcloud-snyk-repo.git'

- Go to configuration and click on pipeline syntax.

- Click on git:Git and give the git url with mentioned on branch as main.

Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step

archiveArtifacts: Archive the artifacts

- archiveArtifacts: Archive the artifacts
- bat: Windows Batch Script
- build: Build a job
- catchError: Catch error and set build result to failure
- checkout: Check out from version control
- cleanWs: Delete workspace when build is done
- deleteDir: Recursively delete the current directory from the workspace
- dir: Change current directory
- echo: Print Message
- emailext: Extended Email
- emailextrecipients: Extended Email Recipients
- error: Error signal
- fileExists: Verify if file exists in workspace
- findBuildScans: Find published build scans
- fingerprint: Record fingerprints of files to track usage
- git: Git
- input: Wait for interactive input
- isUnix: Checks if running on a Unix-like node
- junit: Archive JUnit-formatted test results
- library: Load a library on the fly
- libraryResource: Load a resource file from a library
- load: Evaluate a Groovy source file into the Pipeline script

Sample Step

git: Git

git ?

Repository URL ?

https://github.com/Shikhar82/devsecops-jenkins-k8s-tf-sast-sca-sonarcloud-snyk-repo.git

Branch ?

main

Credentials ?

- none -

+ Add

Include in polling? ?

Include in changelog? ?

Generate Pipeline Script

```
git branch: 'main', url: 'https://github.com/Shikhar82/devsecops-jenkins-k8s-tf-sast-sca-sonarcloud-snyk-repo.git'
```

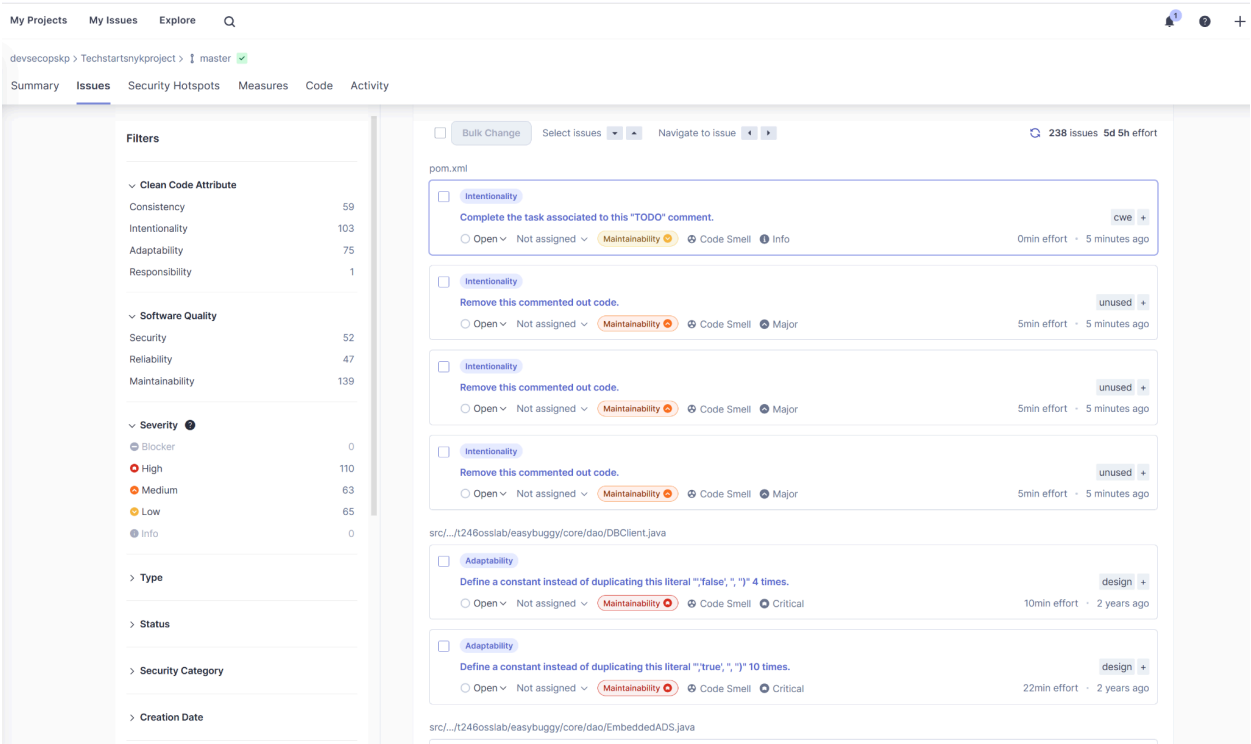
- Copy the script from generate pipeline and replace on the script.

Script ?

```
    maven 'maven-3.9.9'
}
stages {
    stage ("Checking out the git project") {
        steps {
            git branch: 'main', url: 'https://github.com/Shikhar82/devsecops-jenkins-k8s-tf-sast-sca-sonarcloud-snyk-repo.git'
        }
    }

    stage ("Checking bugs on sonarcloud") {
        steps {
            sh 'mvn clean verify sonar:sonar -Dsonar.projectKey=devsecopskp -Dsonar.organization=devsecopskp -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=d0d2af0eee7f1fdb21e93736008
        }
    }
}

}
```

- Click on build, once the build is success go to sonarcloud for finding the vulnerabilities.

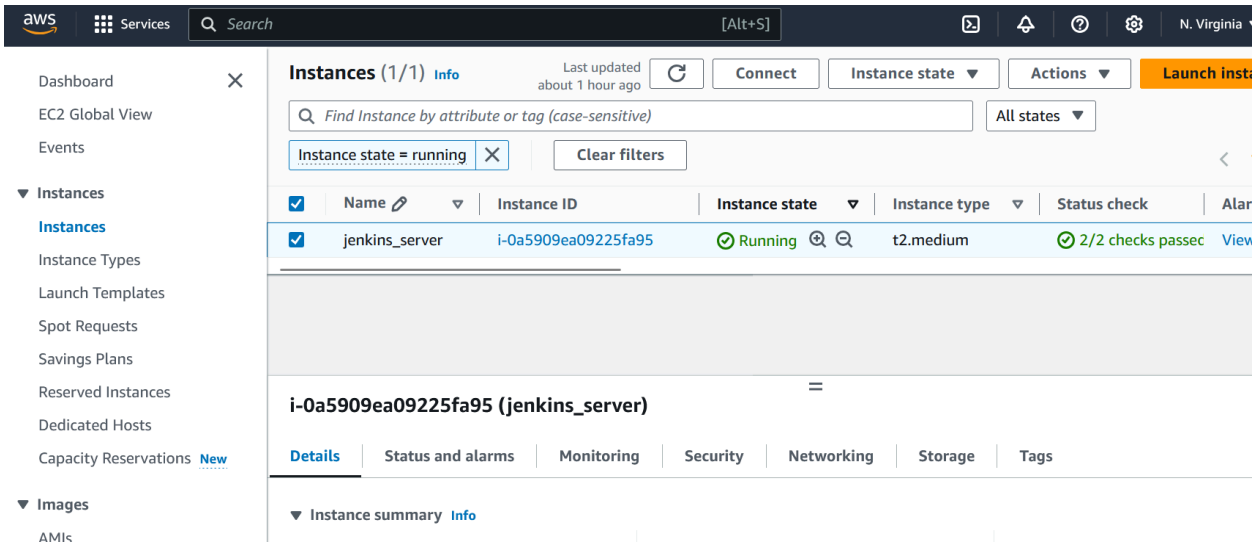
- We can see vulnerabilities and click on the result for further analysis.



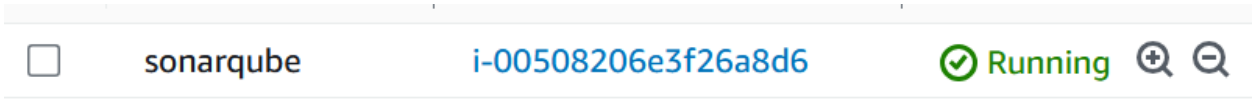
SonarQUBE (Own server instead of sonarcloud)

Create your server for static analysis

- Already we have our jenkins server running.



- Now it's time to run our sonarqube, so create a t2.medium ubuntu 22 ec2 instance for our server.



- Once you are in the ubuntu machine using any SSH client i prefer mobaxterm. Update the system.

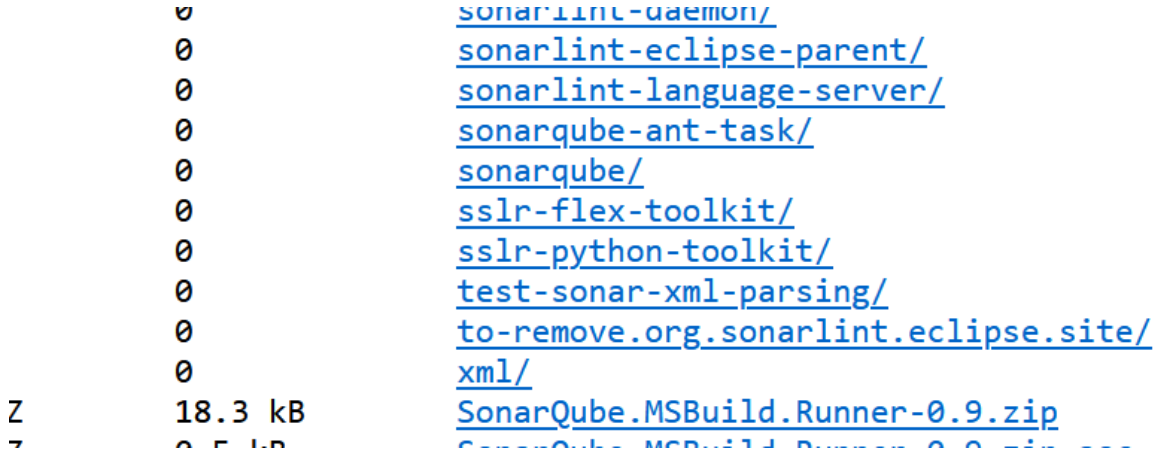
```
root@ip-172-31-81-238:/home/ubuntu# apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2152 kB]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [368 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [17.9 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [2636 kB]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [456 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 c-n-f Metadata [612 B]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1135 kB]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [266 kB]
Get:19 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [26.4 kB]
Get:20 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [43.3 kB]
Get:21 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse Translation-en [10.8 kB]
Get:22 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 c-n-f Metadata [440 B]
```

- Time to install java on the system using the command in the screenshot.

```
root@ip-172-31-81-238:/home/ubuntu# apt install openjdk-11-jdk-headless -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  alsa-topology-conf alsa-ucm-conf ca-certificates-java fontconfig-config fonts-dejavu-core java-common libasound2
  libasound2-data libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libfontconfig1 libgraphite2-3 libharfbuzz0b
  libjpeg-turbo8 libjpeg8 liblcms2-2 libpcsclite1 openjdk-11-jre-headless
Suggested packages:
  default-jre libasound2-plugins alsa-utils cups-common liblcms2-utils pcscd openjdk-11-demo openjdk-11-source libnss-mdns
  fonts-dejavu-extra fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  alsa-topology-conf alsa-ucm-conf ca-certificates-java fontconfig-config fonts-dejavu-core java-common libasound2
  libasound2-data libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libfontconfig1 libgraphite2-3 libharfbuzz0b
  libjpeg-turbo8 libjpeg8 liblcms2-2 libpcsclite1 openjdk-11-jdk-headless openjdk-11-jre-headless
0 upgraded, 21 newly installed, 0 to remove and 16 not upgraded.
Need to get 119 MB of archives.
After this operation, 268 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 alsa-topology-conf all 1.2.5.1-2 [15.5 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libasound2-data all 1.2.6.1-1ubuntu1 [19.1 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libasound2 amd64 1.2.6.1-1ubuntu1 [390 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 alsa-ucm-conf all 1.2.6.3-1ubuntu1.11 [43.6 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 java-common all 0.72build2 [6782 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libavahi-common-data amd64 0.8-5ubuntu5.2 [23.8 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libavahi-common3 amd64 0.8-5ubuntu5.2 [23.9 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libavahi-client3 amd64 0.8-5ubuntu5.2 [28.0 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libcups2 amd64 2.4.1op1-1ubuntu4.11 [263 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 fonts-dejavu-core all 2.37-2build1 [1041 kB]
```

```
root@ip-172-31-81-238:/home/ubuntu# java --version
openjdk 11.0.25 2024-10-15
OpenJDK Runtime Environment (build 11.0.25+9-post-Ubuntu-1ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.25+9-post-Ubuntu-1ubuntu122.04, mixed mode, sharing)
root@ip-172-31-81-238:/home/ubuntu#
```

- Go to binaries.sonarsource.com > distribution > sonarqube > select version 9.3.0 and copy link address.



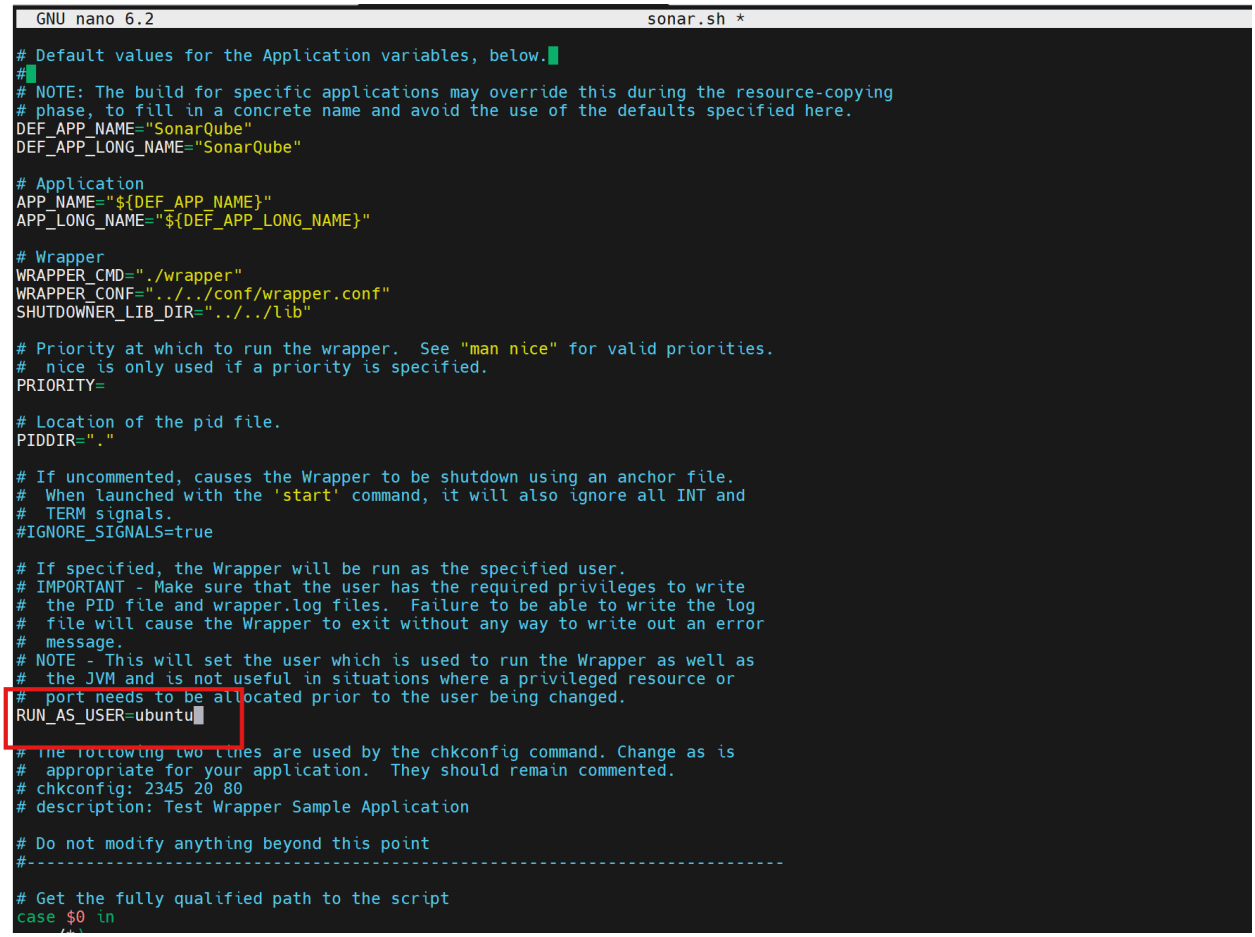
- Navigate to opt and wget to download the package of sonarqube.

```
root@ip-172-31-81-238:/home/ubuntu# cd /opt
root@ip-172-31-81-238:/opt# wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.3.0.51899.zip
--2024-11-18 16:23:59--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.3.0.51899.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 99.84.188.21, 99.84.188.45, 99.84.188.35, ...
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|99.84.188.21|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 276912542 (264M) [application/zip]
Saving to: 'sonarqube-9.3.0.51899.zip'

sonarqube-9.3.0.51899.zip         51%[============================>                       ] 135.72M  27.3MB/s    eta 5s
```

- Unzip the package of sonarqube and navigate to the /bin directory and /Linux directory as we use Ubuntu here.
- An important thing to understand is we are unable to start the sonarqube as a root, even if you try to start it, it will stop eventually.

```
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899/bin/linux-x86-64# sh sonar.sh start
Starting SonarQube...
Started SonarQube.
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899/bin/linux-x86-64# sh sonar.sh status
SonarQube is not running.
```

- Edit the sonarqube file sonar.sh and add run as user = ubuntu.



Save and exit.

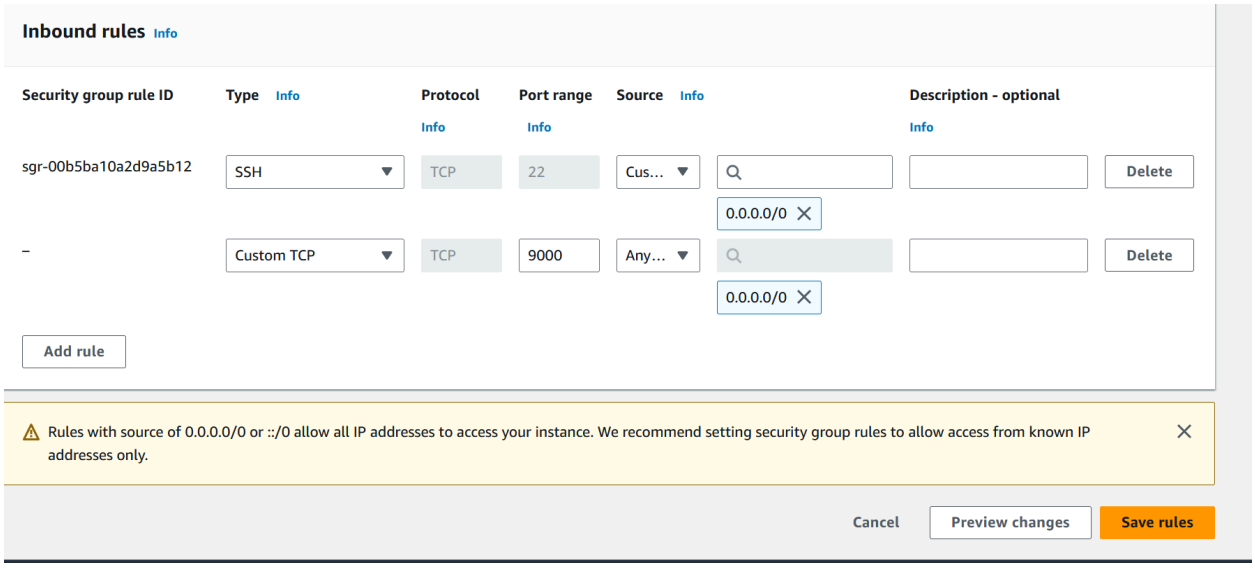
- Go back to opt directory and change the ownership to Ubuntu and check whether it's updated.

```
root@ip-172-31-81-238:/opt# chown ubuntu:ubuntu sonarqube-9.3.0.51899 -R
root@ip-172-31-81-238:/opt# ls -ltr
total 270428
drwxr-xr-x 11 ubuntu ubuntu      4096 Jan 28  2022 sonarqube-9.3.0.51899
-rw-r--r--  1 root   root   276912542 Feb 16  2022 sonarqube-9.3.0.51899.zip
root@ip-172-31-81-238:/opt#
```

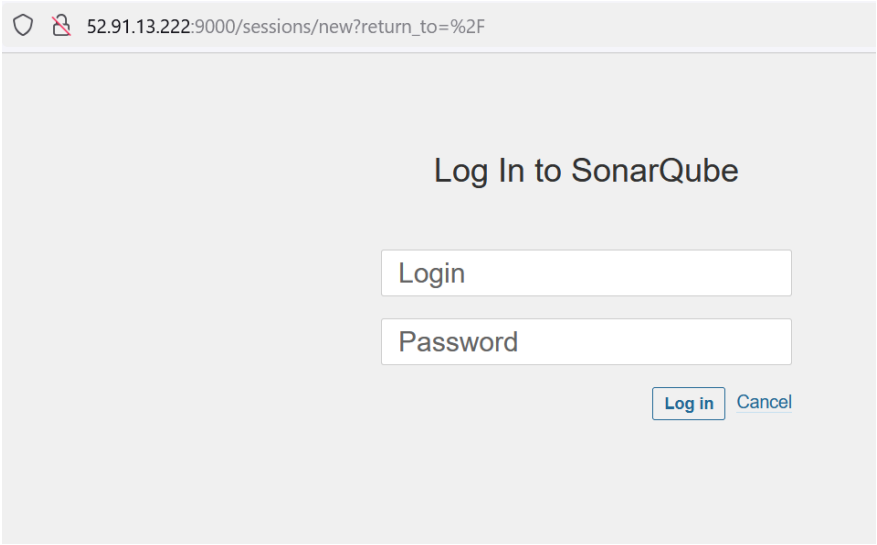
- Navigate to sonar.sh folder and start the sonarqube as you change the ownership to ubuntu.

```
drwxr-xr-x 11 ubuntu ubuntu      4096 Jan 28  2022 sonarqube-9.3.0.51899
-rw-r--r--  1 root   root   276912542 Feb 16  2022 sonarqube-9.3.0.51899.zip
root@ip-172-31-81-238:/opt# cd sonarqube-9.3.0.51899
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899# ls
COPYING  bin  conf  data  dependency-license.json  elasticsearch  extensions  lib  logs  temp  web
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899# cd bin
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899/bin# ls
jsw-license  linux-x86-64  macosx-universal-64  windows-x86-64
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899/bin# cd linux-x86-64/
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899/bin/linux-x86-64# ls
lib  sonar.sh  wrapper
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899/bin/linux-x86-64# sh sonar.sh start
Starting SonarQube...
Started SonarQube.
root@ip-172-31-81-238:/opt/sonarqube-9.3.0.51899/bin/linux-x86-64#
```

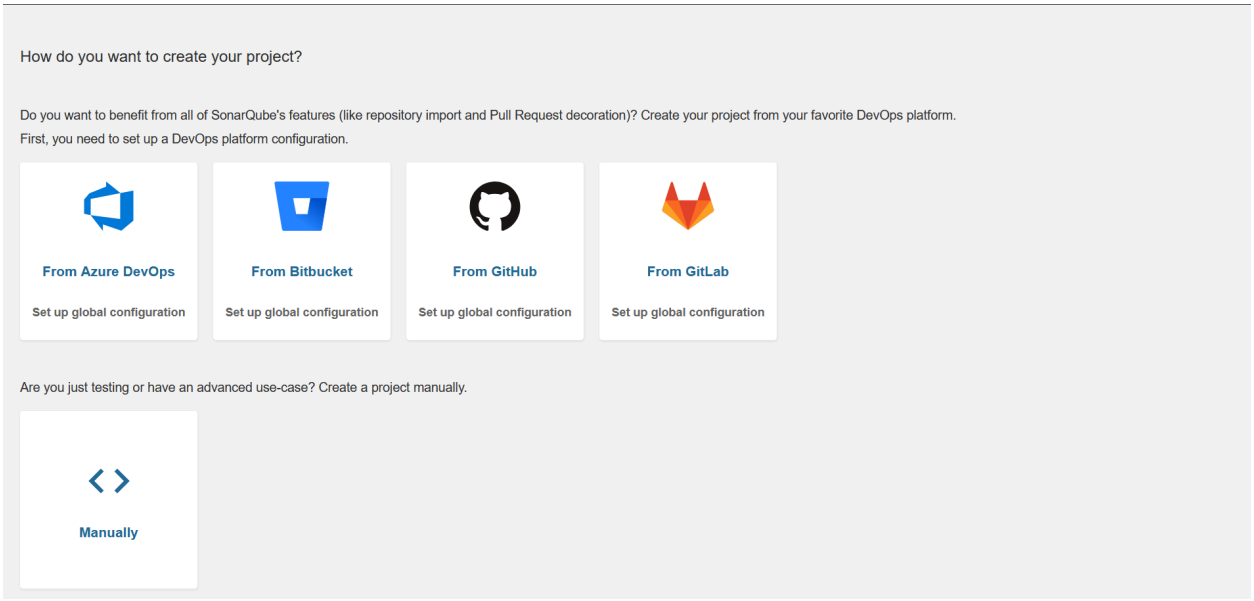
- Now let's access sonarqube on the website. Sonarqube uses port 9000, so go to aws and edit the inbound rules and add port 9000.



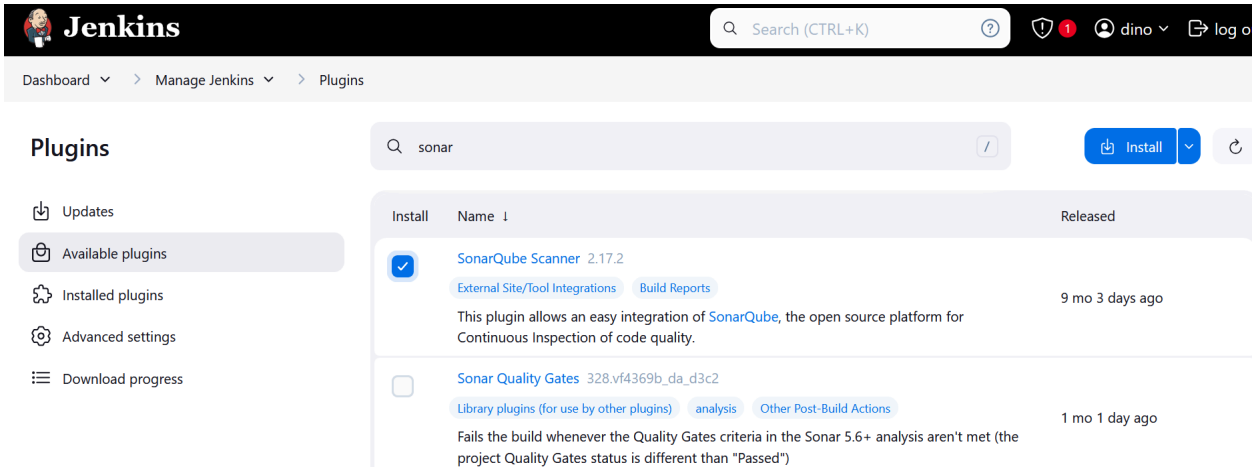
- Open the URL with port 9000 in the browser and see the login page, default password: admin:admin and change password for your convenience.



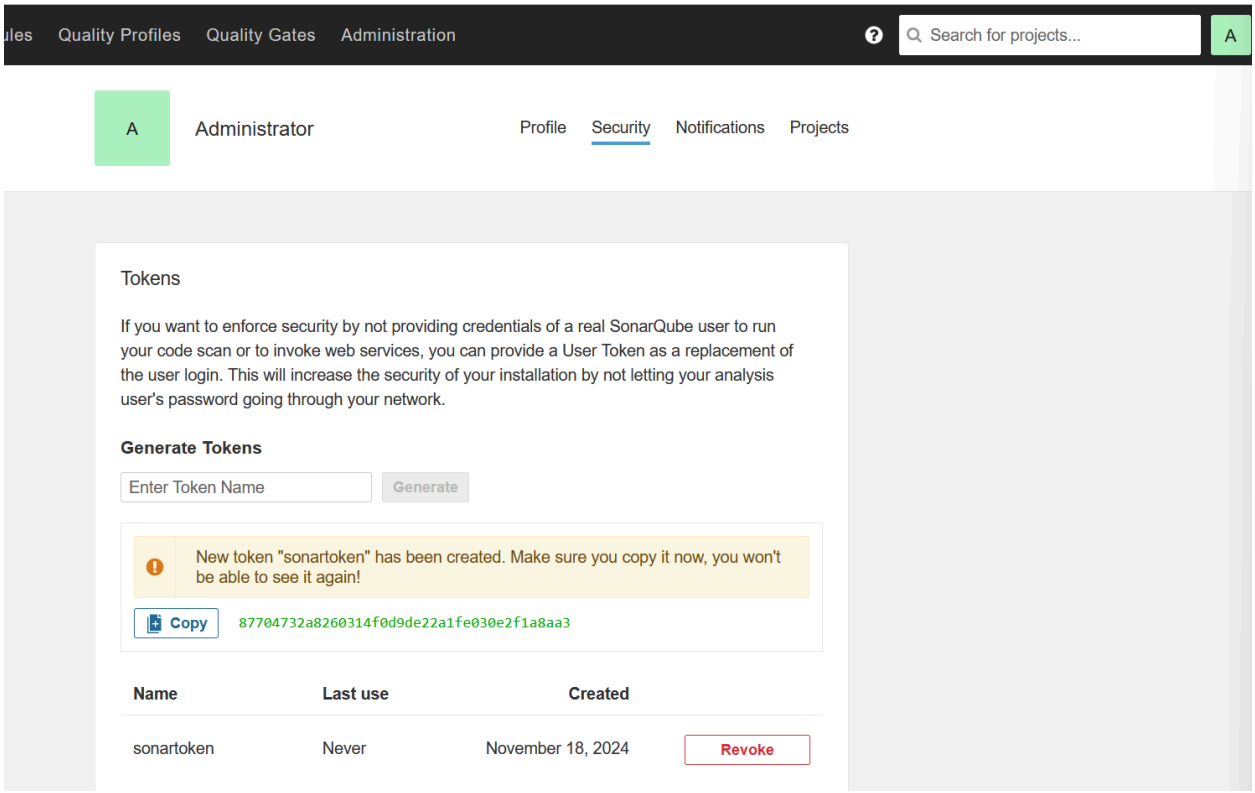
- This is how the dashboard looks like.



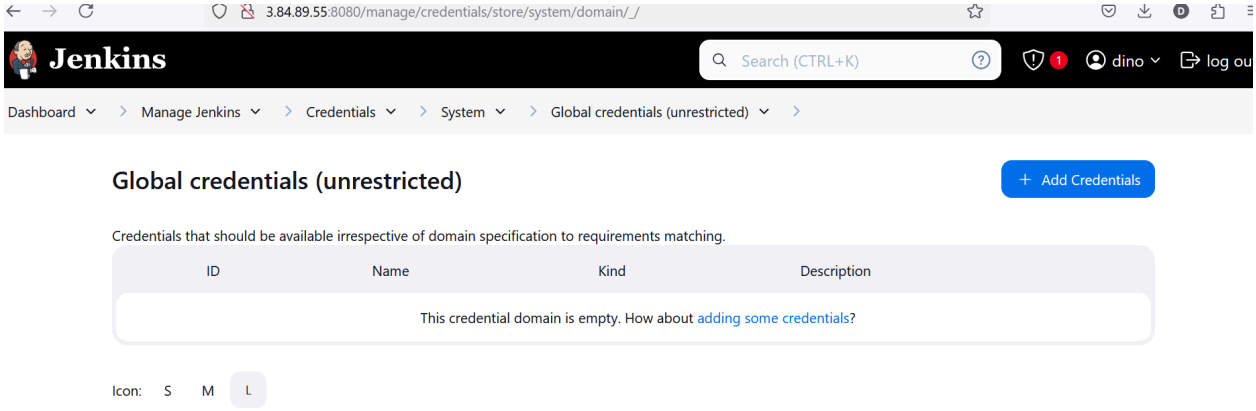
- Install the sonarqube scanner plugin before the integration.



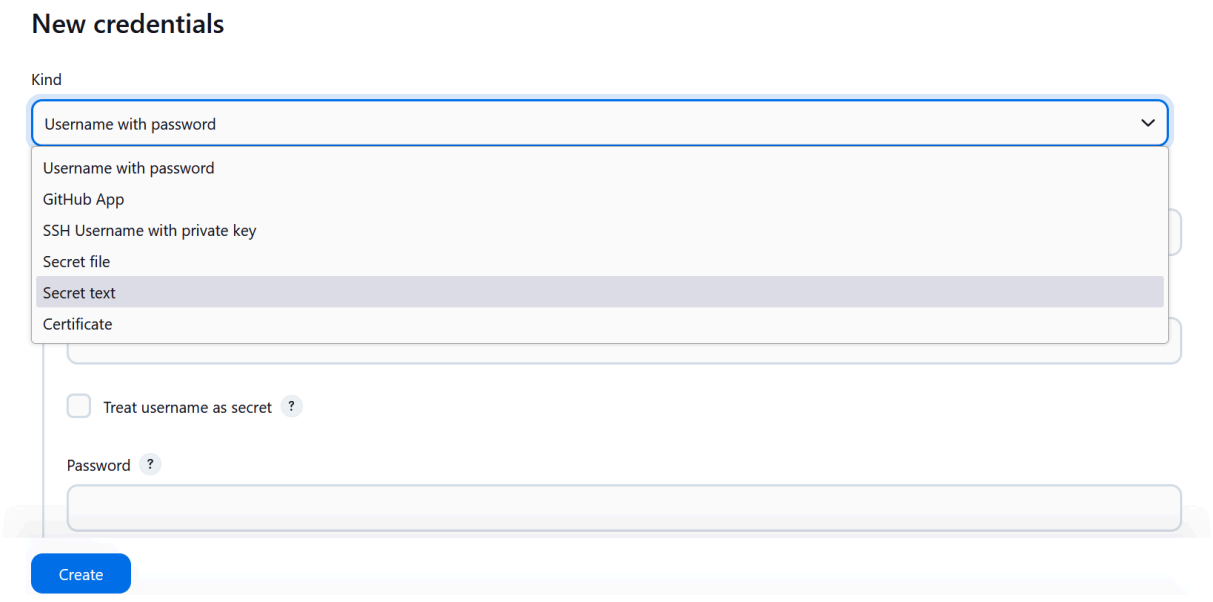
- We need to create a authentication token so that jenkins can communication with sonar.

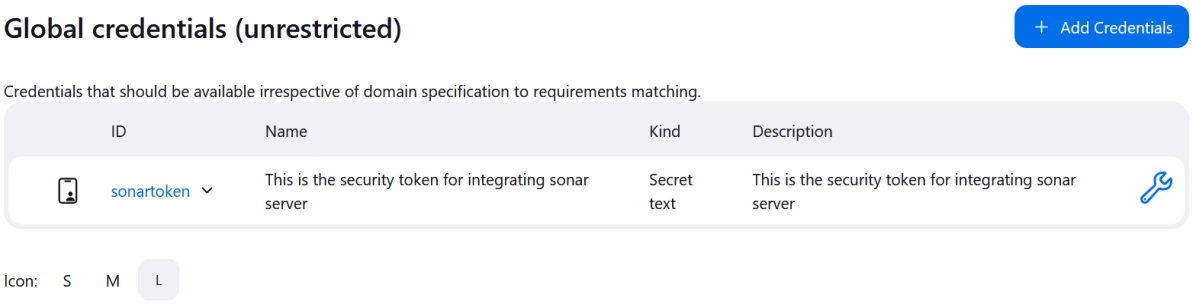


- Go back to manage jenkins > credentials > Global credentials and click on add credentials.

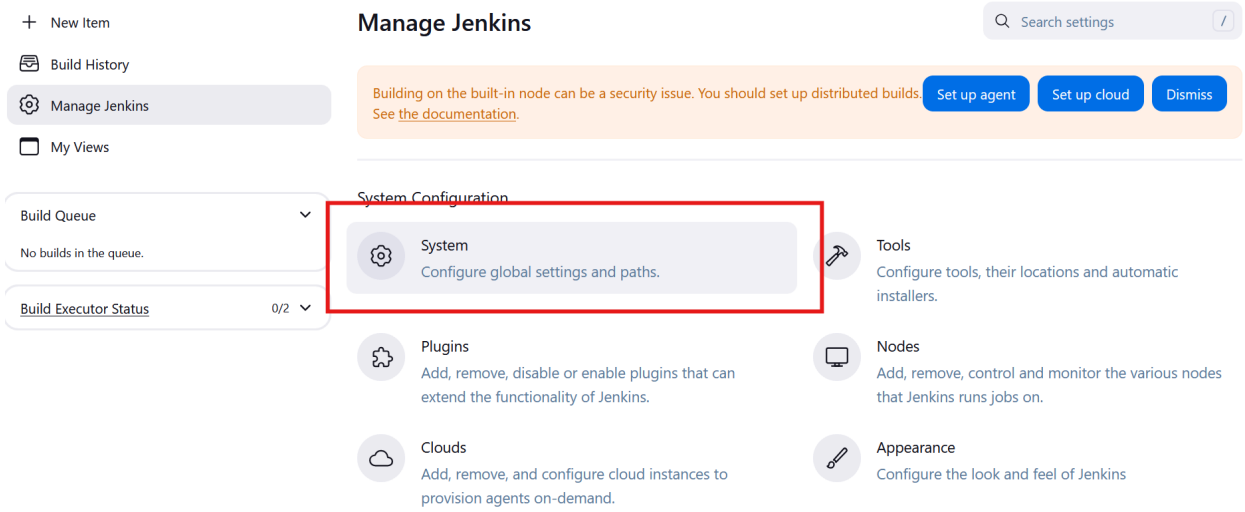


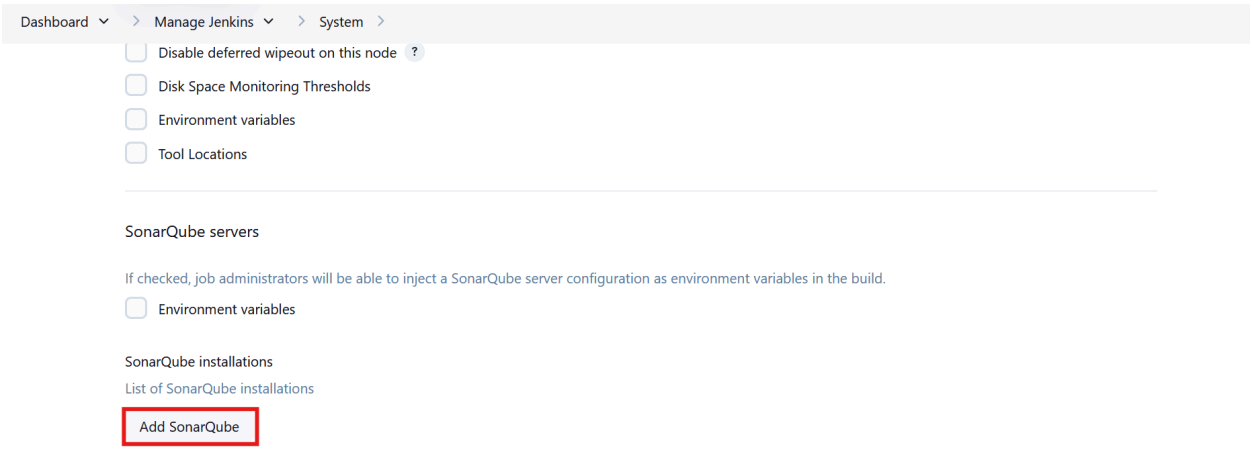
- Select secret text and give the token at



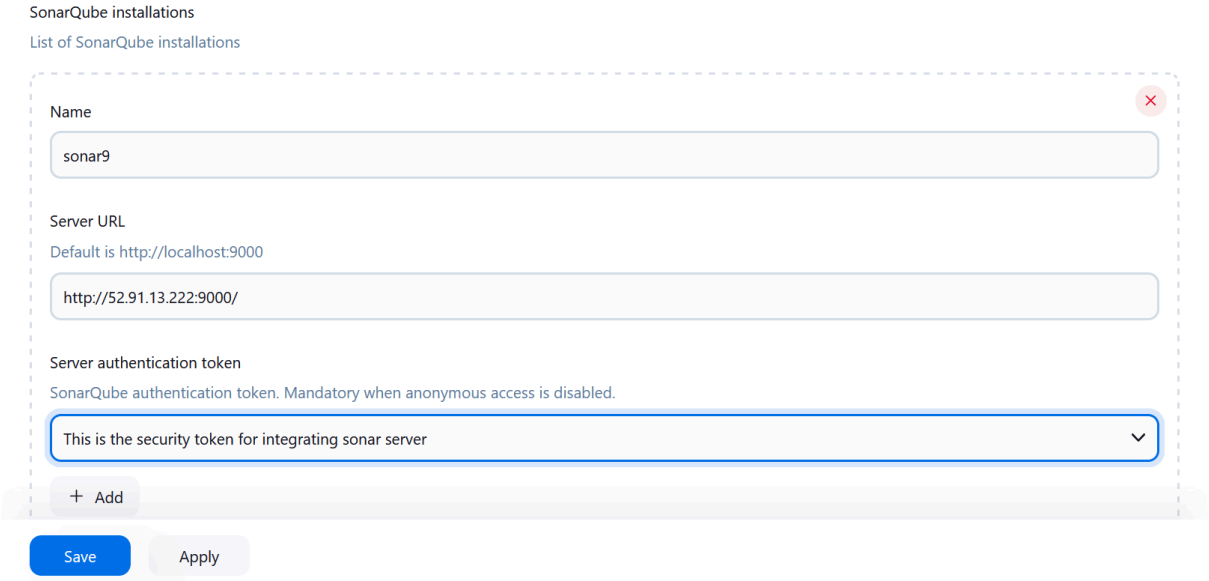


- Now let's add the sonar server on jenkins system.





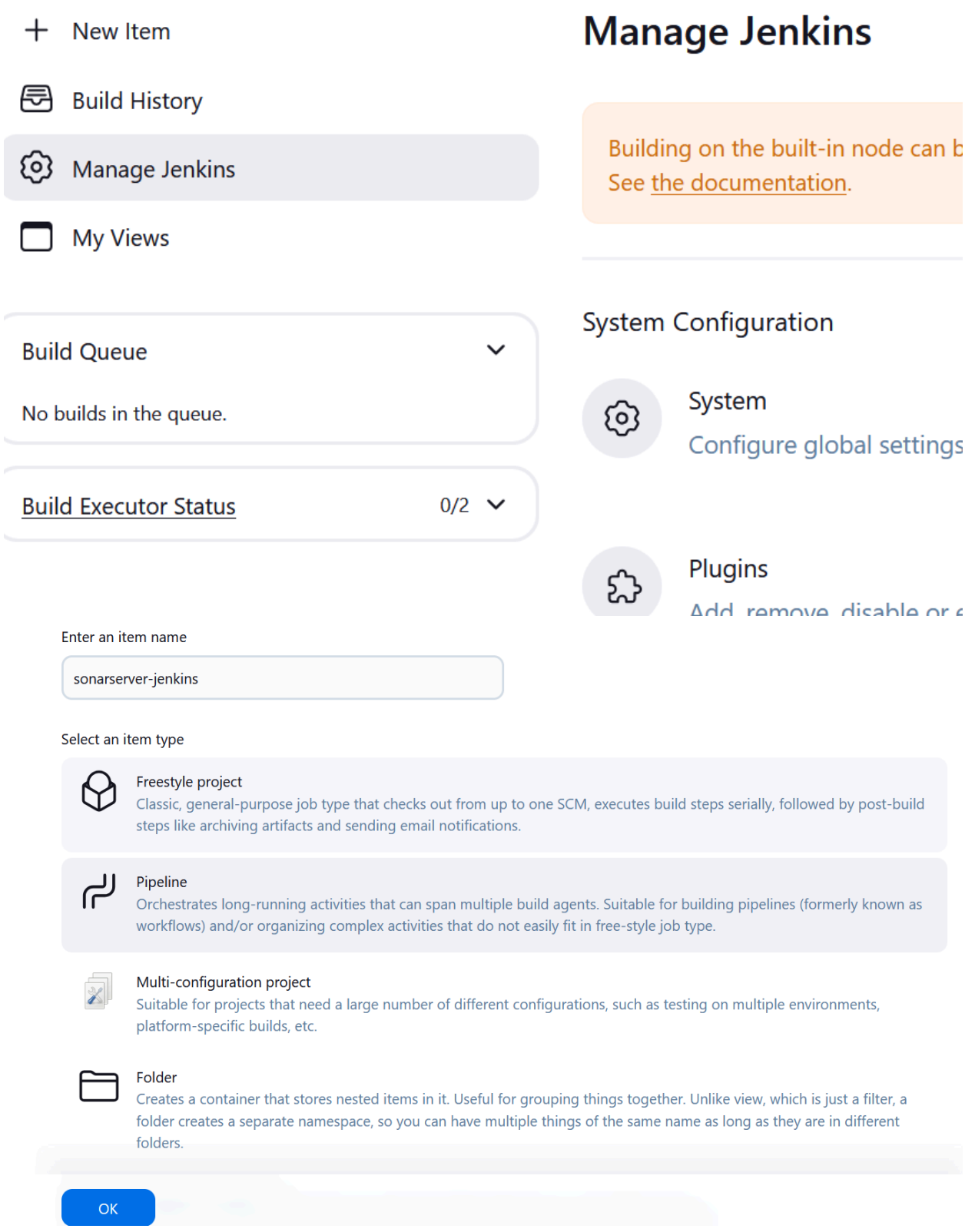
- Give any name, I have given sonar9 because we are using version 9 and give server url and select the token we gave at credentials.



Save and apply.

- Now let’s create the pipeline for completion of our sonar pipeline.

- Create a new pipeline for this one specially by selecting new item.



Click Pipeline above.

- Go to this url for getting the script which we will use in the pipeline code. https://www.jenkins.io/doc/pipeline/steps/sonar/

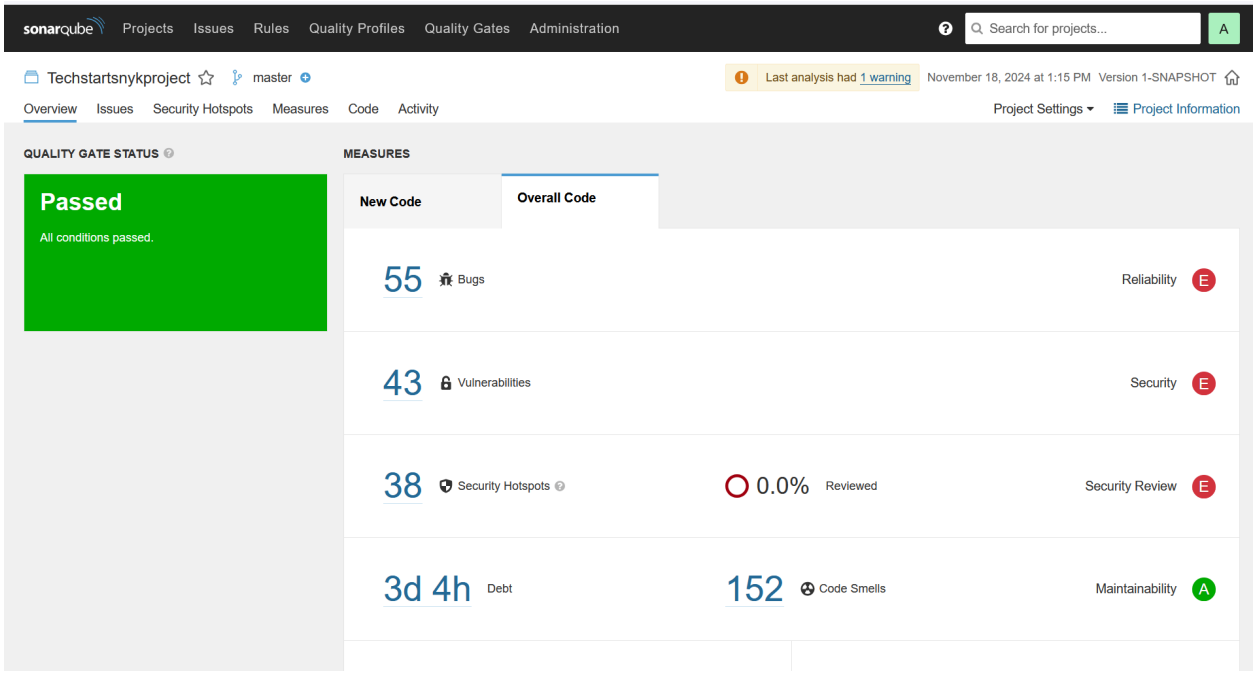
Script ?

try sample Pipeline...

```
pipeline {
    agent any
    tools {
        maven 'maven-3.9.9'
    }
    stages {
        stage ("Checking out the git project") {
            steps {
                git branch: 'main', url: 'https://github.com/Shikhar82/devsecops-jenkins-k8s-tf-sast-sca-sonarclou
            }
        }

        stage ("Build the package") {
            steps {
                sh 'mvn clean package'
            }
        }

        stage ("Integration of sonarcloud server on 9000") {
            steps {
                withSonarQubeEnv(installationName: 'sonar9', credentialsId: 'sonartoken') {
                sh 'mvn sonar:sonar'
              }
            }
        }
    }

}
```

Here for installationName is the name you configured for sonar on jenkins system and credentialsID is the name that you used for configuring the sonarqube token on jenkins credentials option.

- Now save the code and select build Now to look for vulnerabilities.
- Once the build is done, go the webpage where your sonarcloud server is runninga and open the projects to view the vulnerabilities and changes you need to make.

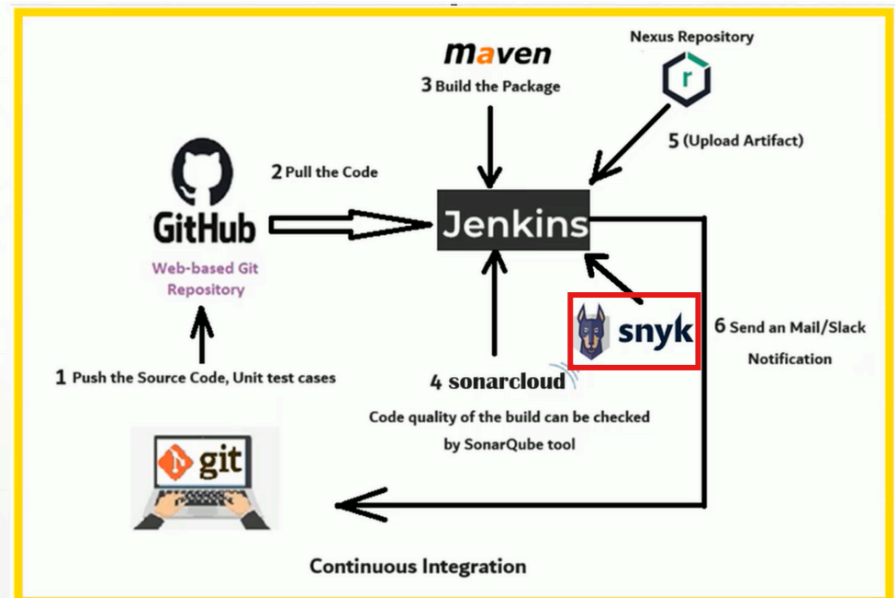


Snyk

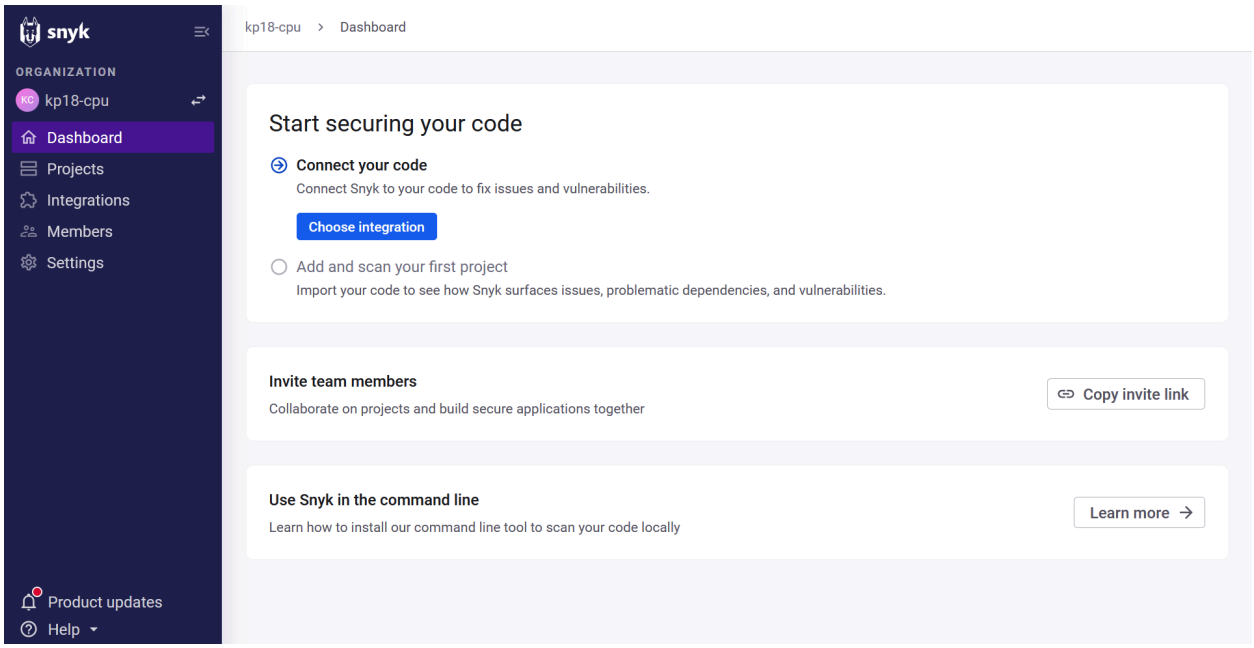
Tool to find the vulnerabilities and fix it automatically at code level.
Types:

1. Synk Code: SAST Tool
2. Synk opensource: Software composition analysis. (Find vulnerabilities in third party libraries) like log4j.
3. Synk container: Finding vulnerabilities in kubernetes and container technologies.
4. Synk is Infrastructure as a code: Finding vulnerabilities on AWS, Terraform and other IAAC Platforms.

Integrating Snyk with Jenkins



- Lets create the account on snyk.io, I would suggest logging in using github account.

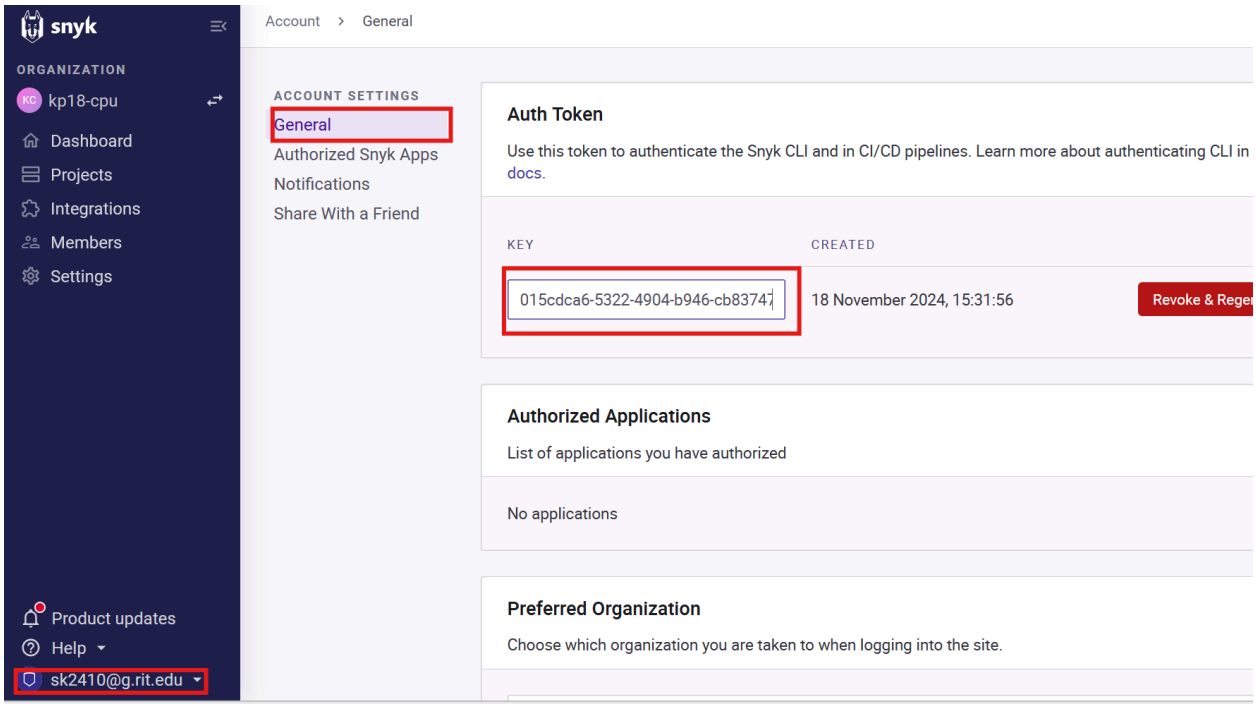


You will see the dashboard like this.

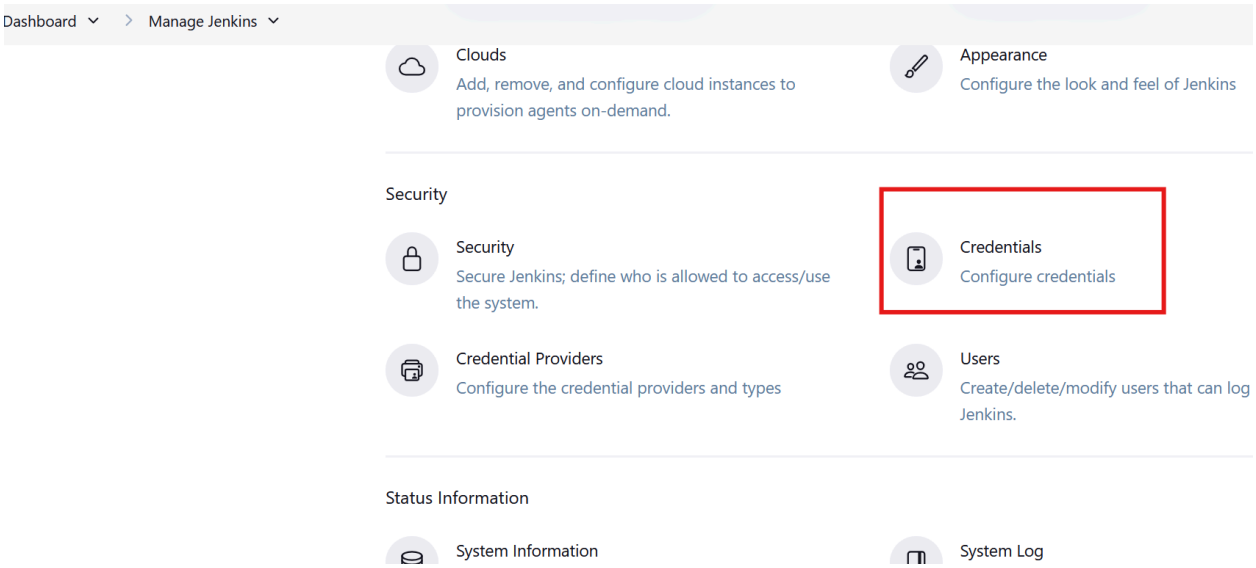
- Important note, whatever code we are going to analyse, we are going to use maven and snyk integration and in that pom.xml file we need to have this plugin script for sure.

```
-->
    <plugin>
      <groupId>io.snyk</groupId>
      <artifactId>snyk-maven-plugin</artifactId>
      <version>2.2.0</version>
      <inherited>false</inherited>
      <configuration>
      </configuration>
    </plugin>
<!-- Snyk Changes for SCA Ends Here -->
```

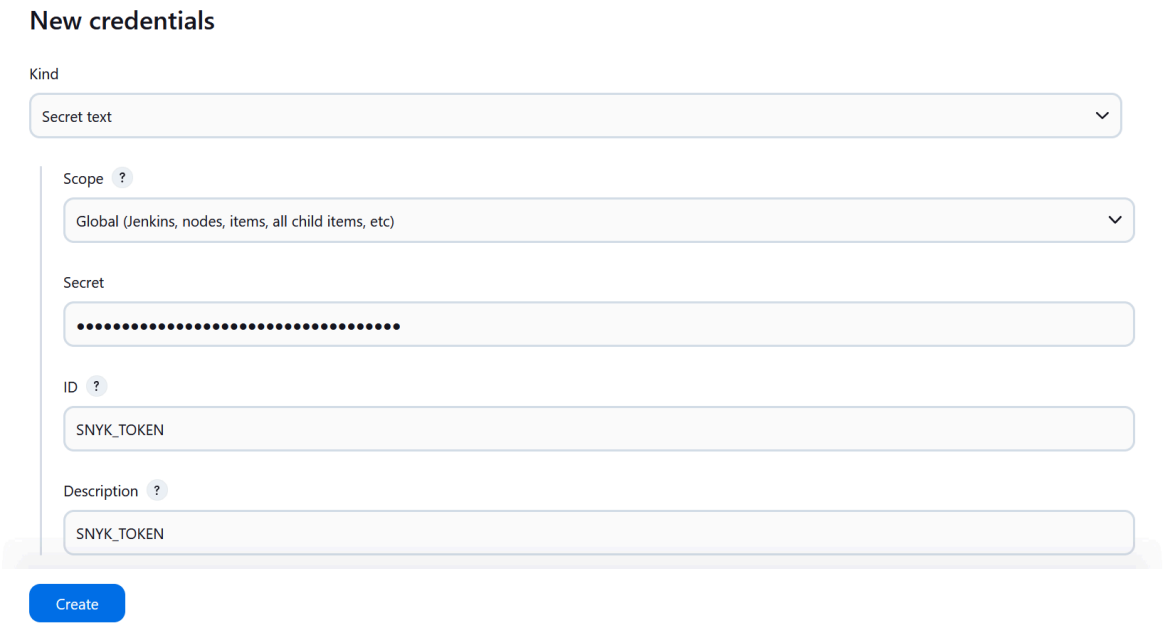
- We need to have snyk token for using that and for authenticating it with jenkins. You will find the auth token on synk website, click on your name in the left bottom.



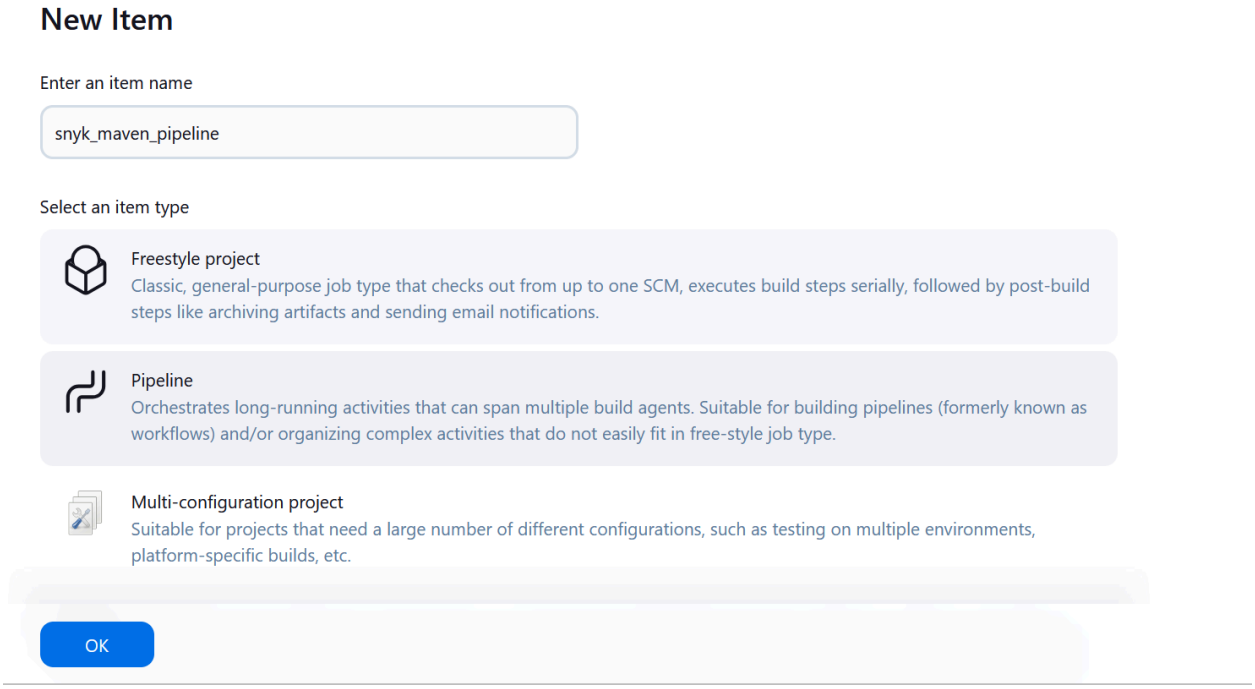
Keep this token safe, now navigate to manage jenkins on the jenkins server.



- Try to add the credentials and enter secret text with the token value and mention the ID and description which we will use it on the pipeline.



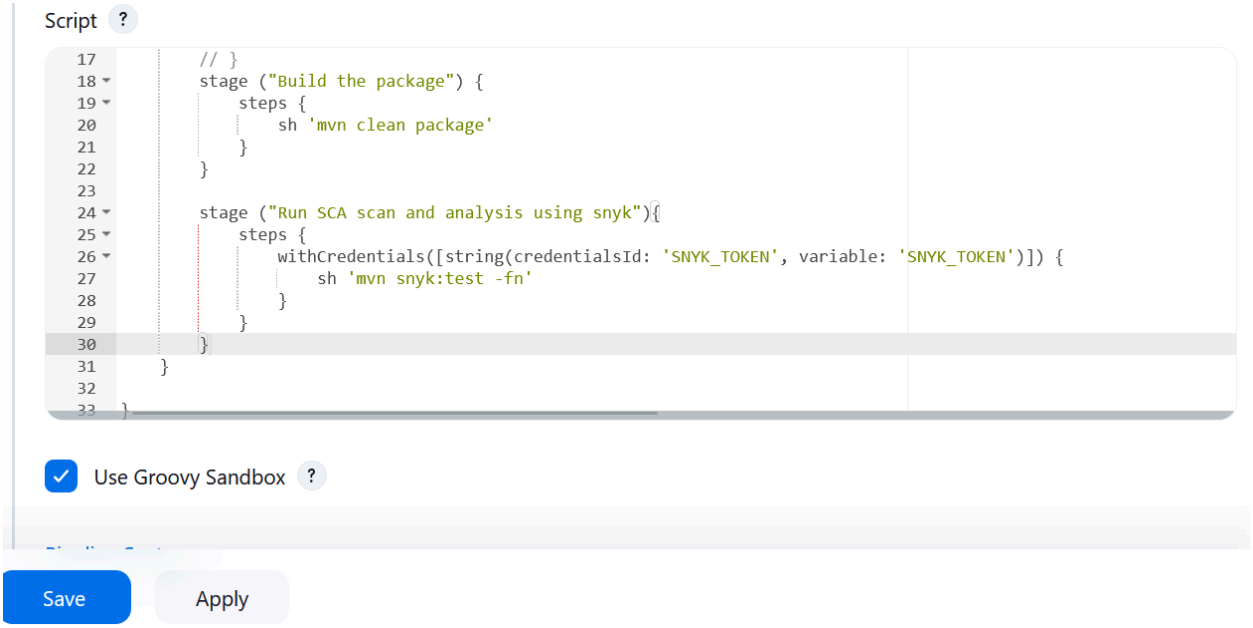
- Now let’s create a new pipeline.



- Use the following script in the pipeline, make sure of the indentation.

```
pipeline {
      agent any
      tools {
      maven 'maven-3.9.9'
      }
      stages {
      stage ("Checking out the git project") {
            steps {
                  git branch: 'main', url:
'https://github.com/Shikhar82/devsecops-jenkins-k8s-tf-sast-sca-sonarcloud-
snyk-repo.git'
            }
      }

      stage ("Checking bugs on sonarcloud") {
            steps {
                  sh 'mvn clean verify sonar:sonar
-Dsonar.projectKey=devsecopskp -Dsonar.organization=devsecopskp
-Dsonar.host.url=https://sonarcloud.io
-Dsonar.token=d0d2af0eee7f1fdb21e937360084b8ece5a62cb4'
            }
      }
```

```
      stage ("Run SCA scan and analysis using snyk"){
            steps {
                  withCredentials([string(credentialsId: 'SNYK_TOKEN',
variable: 'SNYK_TOKEN')]) {
                  sh 'mvn snyk:test -fn'
                  }
            }
      }
      }

}
```



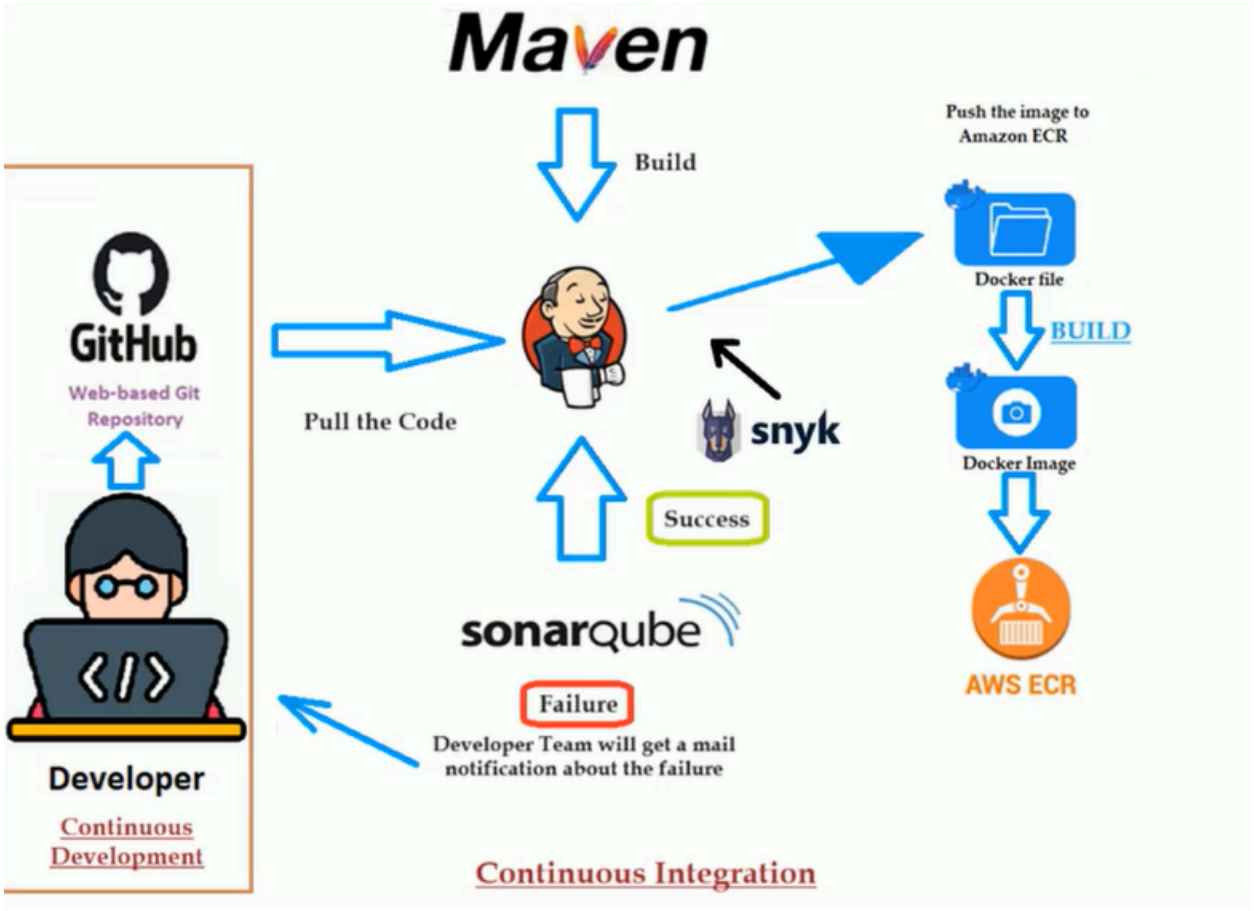
Once you are done with the script apply and save.

```
[INFO] Tested 35 dependencies for known issues, found 51 issues, 51 vulnerable paths.
[INFO]
[INFO]
[INFO] Issues to fix by upgrading:
[INFO]
[INFO]   Upgrade mysql:mysql-connector-java@5.1.25 to mysql:mysql-connector-java@8.0.28 to fix
[INFO]   ✗ Improper Access Control [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-MYSQL-31449] in
mysql:mysql-connector-java@5.1.25
[INFO]     introduced by mysql:mysql-connector-java@5.1.25
[INFO]   ✗ Improper Authorization [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-MYSQL-2386864] in
mysql:mysql-connector-java@5.1.25
[INFO]     introduced by mysql:mysql-connector-java@5.1.25
[INFO]   ✗ XML External Entity (XXE) Injection [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-
MYSQL-1766958] in mysql:mysql-connector-java@5.1.25
[INFO]     introduced by mysql:mysql-connector-java@5.1.25
[INFO]   ✗ Privilege Escalation [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-MYSQL-174574] in
mysql:mysql-connector-java@5.1.25
[INFO]     introduced by mysql:mysql-connector-java@5.1.25
[INFO]   ✗ Arbitrary Code Execution [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-MYSQL-31580] in
mysql:mysql-connector-java@5.1.25
[INFO]     introduced by mysql:mysql-connector-java@5.1.25
[INFO]   ✗ SQL Injection [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-MYSQL-451460] in
mysql:mysql-connector-java@5.1.25
[INFO]     introduced by mysql:mysql-connector-java@5.1.25
[INFO]   ✗ Access Control Bypass [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-MYSQL-451464] in
mysql:mysql-connector-java@5.1.25
[INFO]     introduced by mysql:mysql-connector-java@5.1.25
```

We can see all the vulnerabilities and severity on the console output which is report of the pipeline.

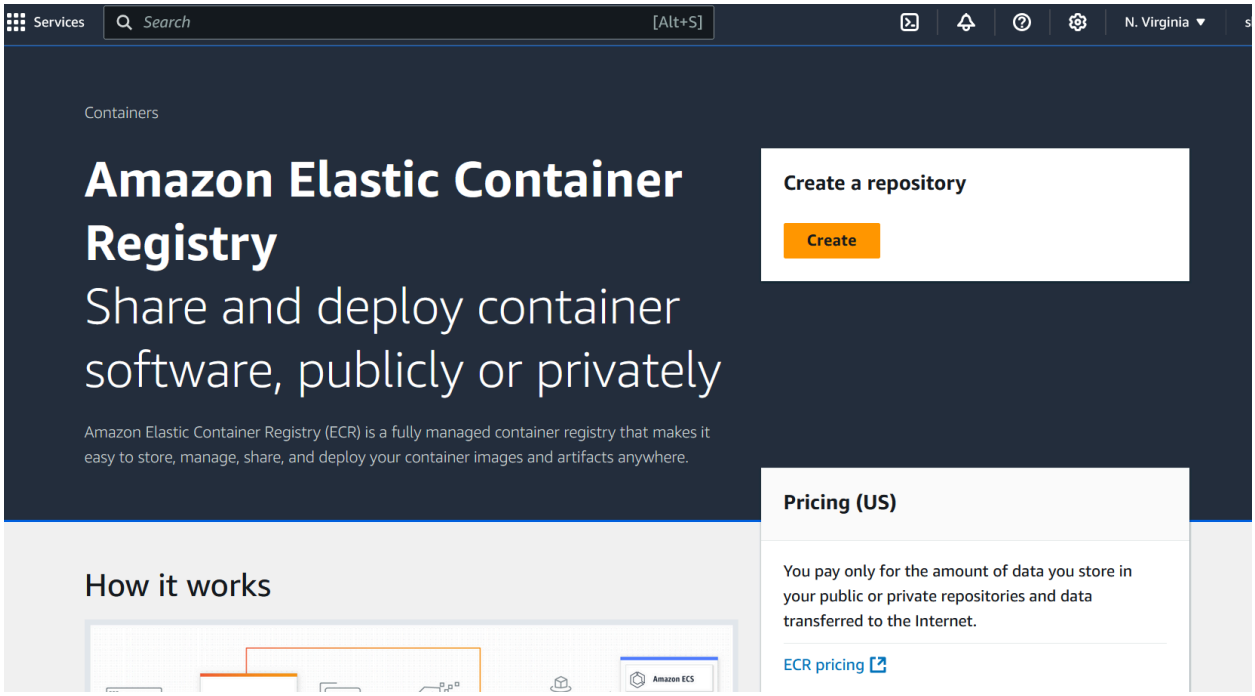
We are successfully worked with Snyk and Sonarcloud.

Building Docker Image using jenkins pipeline

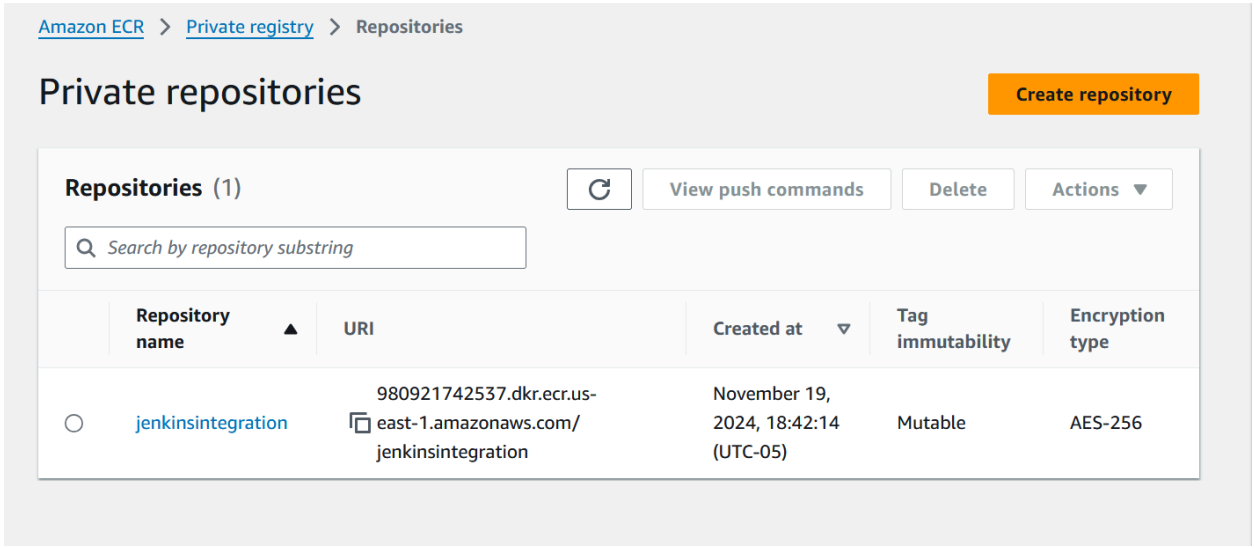


Based on the diagram we are going to build the docker file and deploy the image to AWS ECR (Elastic Container Registry) for better management.

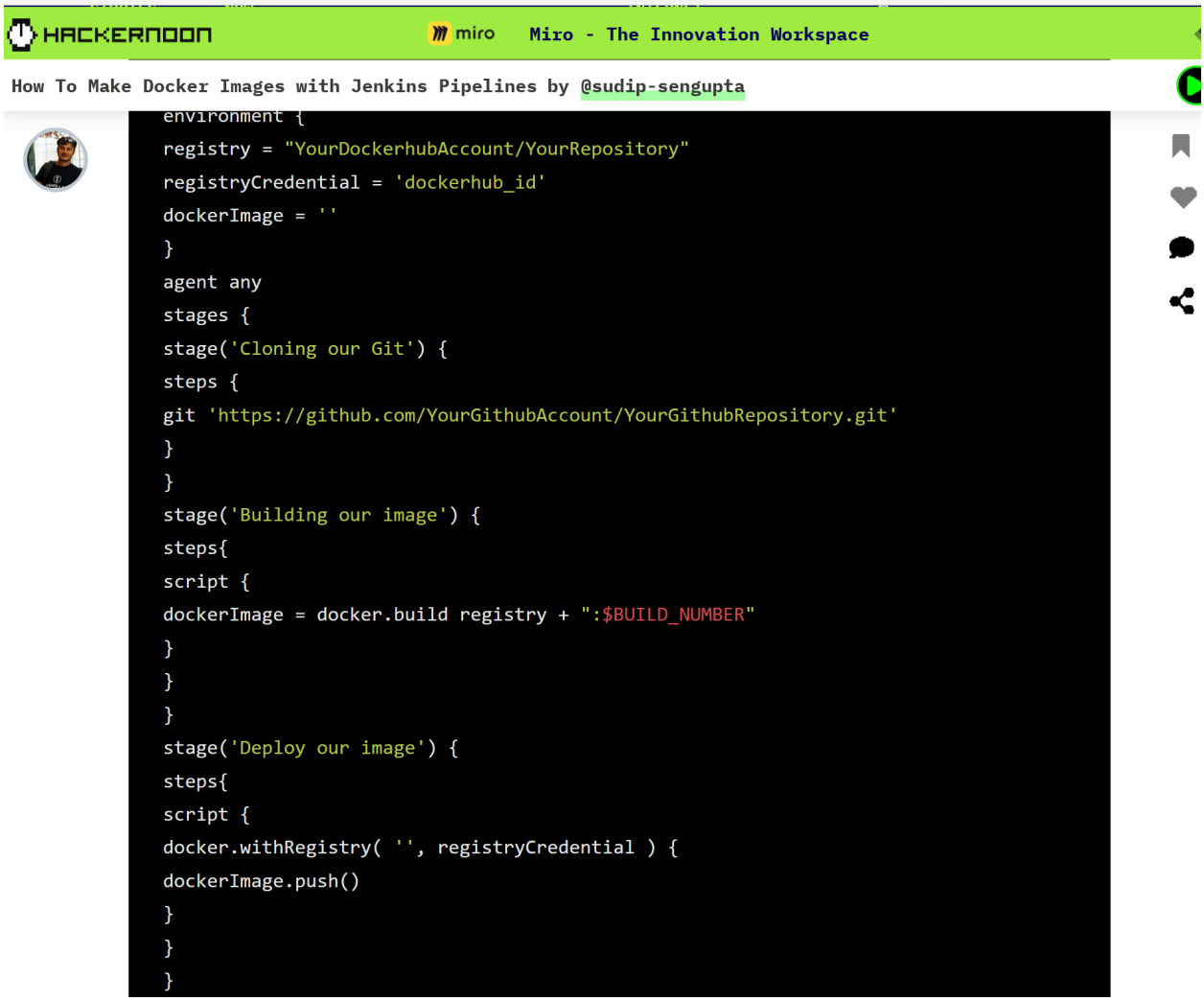
- Go to aws console
- Search for google, build the docker image using jenkins pipeline. Go to this website and copy the two blocks of stages for creation and deployment.



- Create a simple private repository with a name you want to use.



- Go to jenkins server create a new pipeline and copy the old pipeline from above projects and try to add two parts of the code for integrating docker.
  Yes, we are going to build and deploy docker images using jenkins server so copy that part of the code to include in the pipeline.



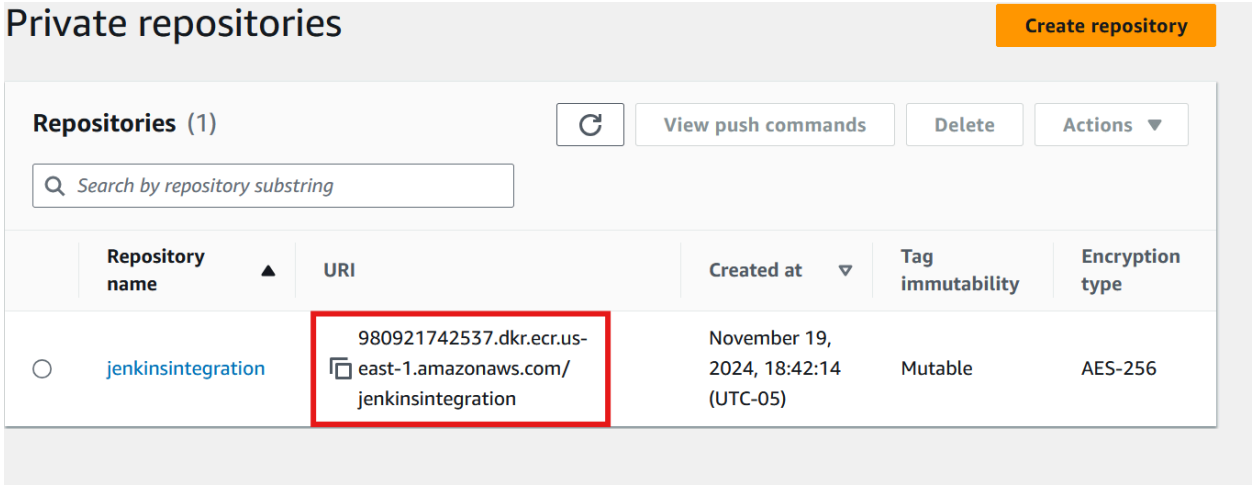
- Create a new pipeline at jenkins and change the github repository with something without any vulnerabilities.
- Change the github url to “https://github.com/Shikhar82/springboot-maven-micro.git” or any git project without any vulnerabilities.

```
}
stages {
    stage ("Checking out the git project") {
        steps {
            git branch: 'master', url: 'https://github.com/Shikhar82/springboot-maven-micro.git'
        }
    }
```

- Let’s add the docker scripts to our newly created pipeline.

```
stage('Building our image') {
steps{
script {
dockerImage = docker.build registry + ":$BUILD_NUMBER"
}
}
}
stage('Deploy our image') {
steps{
script {
docker.withRegistry( '', registryCredential ) {
dockerImage.push()
}
}
}
}
```

- In your pipeline add the registry value as a URI from the ECR repository for example below.

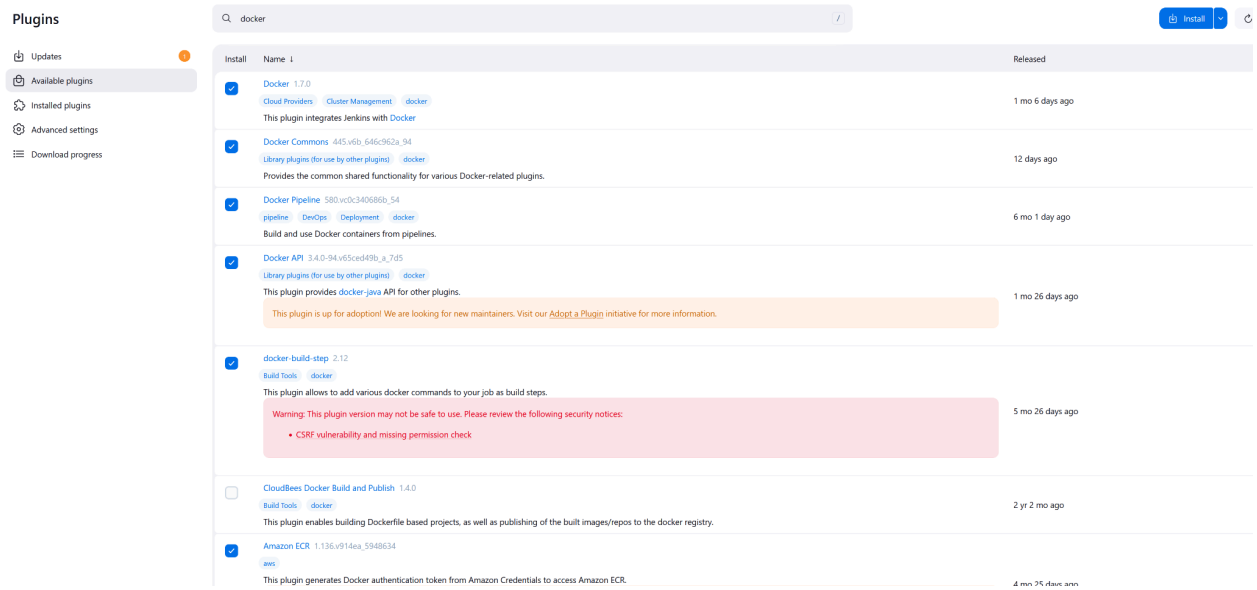



```
pipeline {
    agent any
    tools {
        maven 'maven-3.9.9'
    }
    environment {
        registry = "980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration"
    }
    stages {
        stage ("Checking out the git project") {
            steps {
                git branch: 'master', url: 'https://github.com/Shikhar82/springboot-maven-micro.git'
            }
        }
```

- We need to install the docker on our jenkins server (Virtual Machine), we can find commands online for installing docker on RHEL or you can simple CHATGPT it.

- Commands for installing docker on RHEL.
    1. sudo yum update -y
    2. yum install -y yum-utils device-mapper-persistent-data lvm2
    3. sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
    4. sudo yum install -y docker-ce docker-ce-cli containerd.io
    5. sudo systemctl start docker
    6. sudo systemctl enable docker
    7. docker --version
    8. sudo docker run hello-world
    9. sudo usermod -aG docker $USER (Change USER according to your username)

- After installing the docker on the jenkins server go to “manage jenkins” before that save your code and in manage jenkins>plugins>available plugins. Click on all the plugins that has Docker and Amazon ECR as well and install them.



- Once you enabled the required docker plugin simply run you pipeline. I will show how the pipeline looks below.

```
pipeline {
      agent any
      tools {
```

```
      maven 'maven-3.9.9'
      }
      environment {
      registry =
"980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration"
      }
      stages {
      stage ("Checking out the git project") {
            steps {
                  git branch: 'master', url:
'https://github.com/Shikhar82/springboot-maven-micro.git'
            }
      }

      stage ("Checking bugs on sonarcloud") {
            steps {
                  sh 'mvn clean verify sonar:sonar
-Dsonar.projectKey=devsecopskp -Dsonar.organization=devsecopskp
-Dsonar.host.url=https://sonarcloud.io
-Dsonar.token=d0d2af0eee7f1fdb21e937360084b8ece5a62cb4'
            }
      }
      // stage ("Build the package") {
      //    steps {
      //          sh 'mvn clean package'
      //    }
      // }

      stage ("Run SCA scan and analysis using snyk"){
            steps {
                  withCredentials([string(credentialsId: 'SNYK_TOKEN',
variable: 'SNYK_TOKEN')]) {
                  sh 'mvn snyk:test -fn'
                  }
            }
      }
      stage('Building our image') {
            steps{
                  script {
                  dockerImage = docker.build registry + ":$BUILD_NUMBER"
                  }
            }
      }
```

```
// stage('Deploy our image') {
// steps{
// script {
// docker.withRegistry( '', registryCredential ) {
// dockerImage.push()
// }
// }
// }
// }
      }

}
```

```
#4 sha256:899046e4a240e349763e42464f501b60a1bd429af9f38ccd927d9da2124b98de 62.68MB / 62.68MB 1.0s done
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 56.62MB / 151.61MB 1.1s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 65.01MB / 151.61MB 1.2s
#4 extracting sha256:899046e4a240e349763e42464f501b60a1bd429af9f38ccd927d9da2124b98de 0.1s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 73.40MB / 151.61MB 1.3s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 96.47MB / 151.61MB 1.6s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 105.91MB / 151.61MB 1.7s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 122.68MB / 151.61MB 1.9s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 131.07MB / 151.61MB 2.0s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 145.75MB / 151.61MB 2.2s
#4 sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 151.61MB / 151.61MB 2.8s done
#4 extracting sha256:899046e4a240e349763e42464f501b60a1bd429af9f38ccd927d9da2124b98de 4.2s done
#4 extracting sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d
#4 extracting sha256:5ce47bbf12c856f2c3827f5733bc33beb2ba86473a4817f4f8c1b87f1990cd6d 3.0s done
#4 DONE 8.6s

#6 [2/2] COPY target/springboot-maven-course-micro-svc-0.0.1-SNAPSHOT.jar app.jar
#6 DONE 0.9s

#7 exporting to image
#7 exporting layers 0.1s done
#7 writing image sha256:0bc277b3dacad71ba5415759812920918f6dafa6ecfb0f54d9c81590518bdc3f
#7 writing image sha256:0bc277b3dacad71ba5415759812920918f6dafa6ecfb0f54d9c81590518bdc3f done
#7 naming to 980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration:4 done
#7 DONE 0.1s
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Pipeline script

Script ?

```
                sh 'mvn snyk:test -fn'
            }
        }
    }
    stage('Building our image') {
        steps{
            script {
                dockerImage = docker.build registry + ":$BUILD_NUMBER"
            }
        }
    }
// stage('Deploy our image') {
// steps{
// script {
// docker.withRegistry( '', registryCredential ) {
// dockerImage.push()
// }
// }
```

- If you encounter any error while build especially building docker go to /var/run/docker.sock on you jenkins server VM and give the following command: “chmod 777 /var/run/docker.sock”

- Once everything runs successfully you can view your docker image build on the jenkins server VM through the command “docker image ls”.

```
[root@ip-172-31-84-17 docker_jenkins_maven]# docker image ls
REPOSITORY                                                      TAG       IMAGE ID       CREATED          SIZE
980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration   4         0bc277b3daca   28 minutes ago   483MB
```

Pushing our Docker image to AWS ECR:

- We need to establish an communication between AWS and jenkins server for them to communicate in the first place.
- For this purpose first we are going to install AWS CLI on our Jenkins server VM. Follow these three commands on your jenkins server VM.
    - curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
    - unzip awscliv2.zip
    - sudo ./aws/install

- Once entered the last command “./aws/install” add the path so that you can aws cli anywhere like shown in the screenshot.

```
[root@ip-172-31-84-17 ~]# ./aws/install
You can now run: /usr/local/bin/aws --version
[root@ip-172-31-84-17 ~]# echo $PATH
/root/.local/bin:/root/bin:/sbin:/bin:/usr/sbin:/usr/bin
[root@ip-172-31-84-17 ~]# ^C
[root@ip-172-31-84-17 ~]# /usr/local/bin/aws --version
aws-cli/2.22.1 Python/3.12.6 Linux/5.14.0-427.20.1.el9_4.x86_64 exe/x86_64.rhel.9
[root@ip-172-31-84-17 ~]# export PATH=$PATH:/usr/local/bin
[root@ip-172-31-84-17 ~]# aws --version
aws-cli/2.22.1 Python/3.12.6 Linux/5.14.0-427.20.1.el9_4.x86_64 exe/x86_64.rhel.9
[root@ip-172-31-84-17 ~]#
```

- After installing the aws-cli now our jenkins server is ready to talk with aws ECR. So go to the pipeline, now we will use the deploy stage in our pipeline which we commented out before.

Script ?

```
35          steps{
36              script {
37                  dockerImage = docker.build registry + ":$BUILD_NUMBER"
38              }
39          }
40      }
41  // stage('Deploy our image') {
42  // steps{
43  // script {
44  // docker.withRegistry( '', registryCredential ) {
45  // dockerImage.push()
46  // }
47  // }
48  // }
49  // }
50      }
51
52  }
```

- Once we enable the script in our code we need to have registryCredential, so we will create a user on AWS with certain policy which aligns with AWS ECR and we will derive the auth token for AWS and use it here at pipeline for the build.

Definition

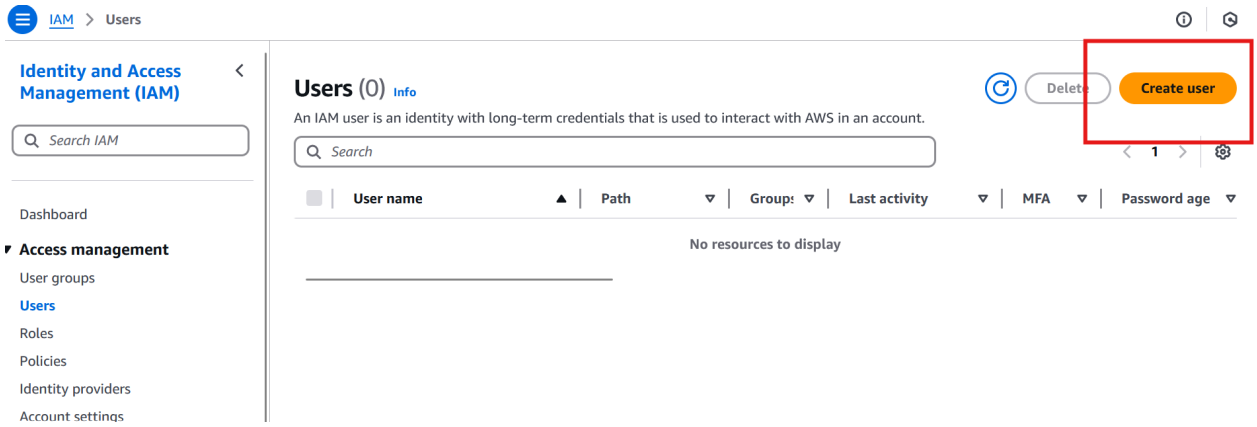
Pipeline script

Script ?

```
        steps{
            script {
                dockerImage = docker.build registry + ":$BUILD_NUMBER"
            }
        }
    }
    stage('Deploy our image') {
        steps{
            script {
                docker.withRegistry( '', registryCredential ) {
                dockerImage.push()
                }
            }
        }
    }
}
```

Use Groovy Sandbox ?

- Go to AWS management console and search for IAM and create a user.



Specify user details

User details

User name

awsecr_jenkins_user

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

ⓘ If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more

Cancel Next

Give permission as Attach policies and the policy name would be **AmazonEC2ContainerRegistryFullAccess.**

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

○ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

◉ Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1280)

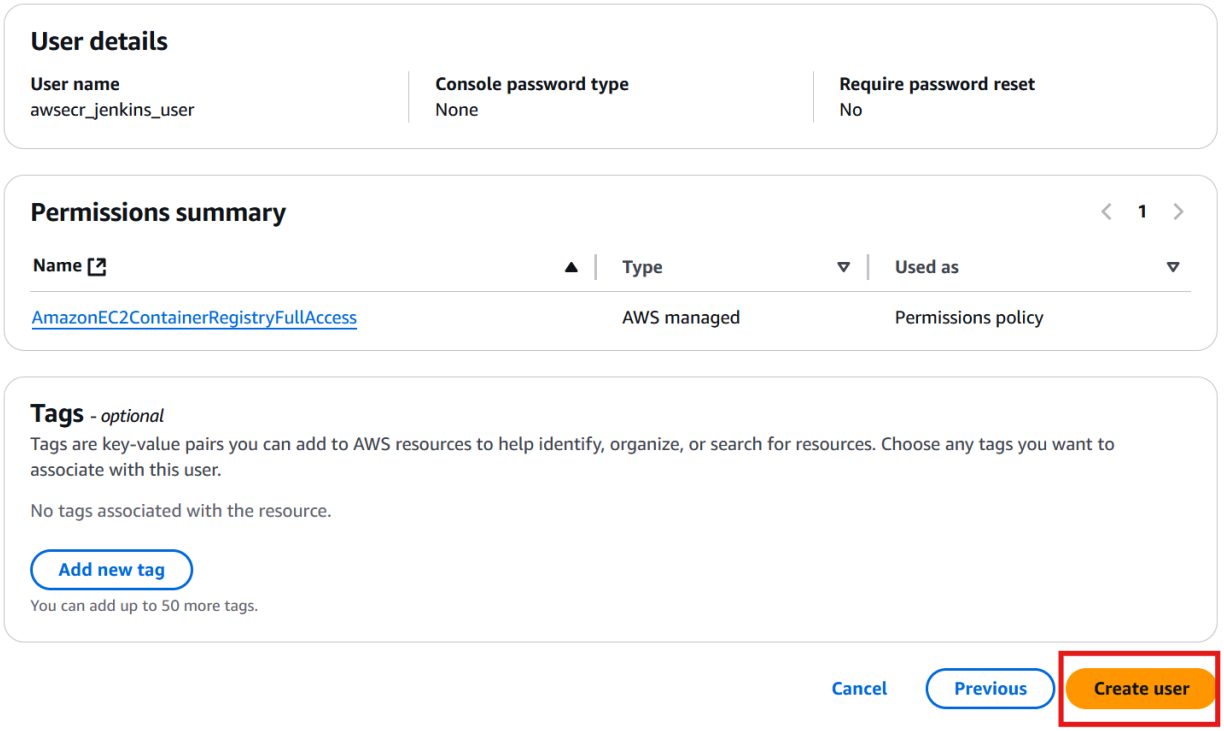
Choose one or more policies to attach to your new user.

Create policy

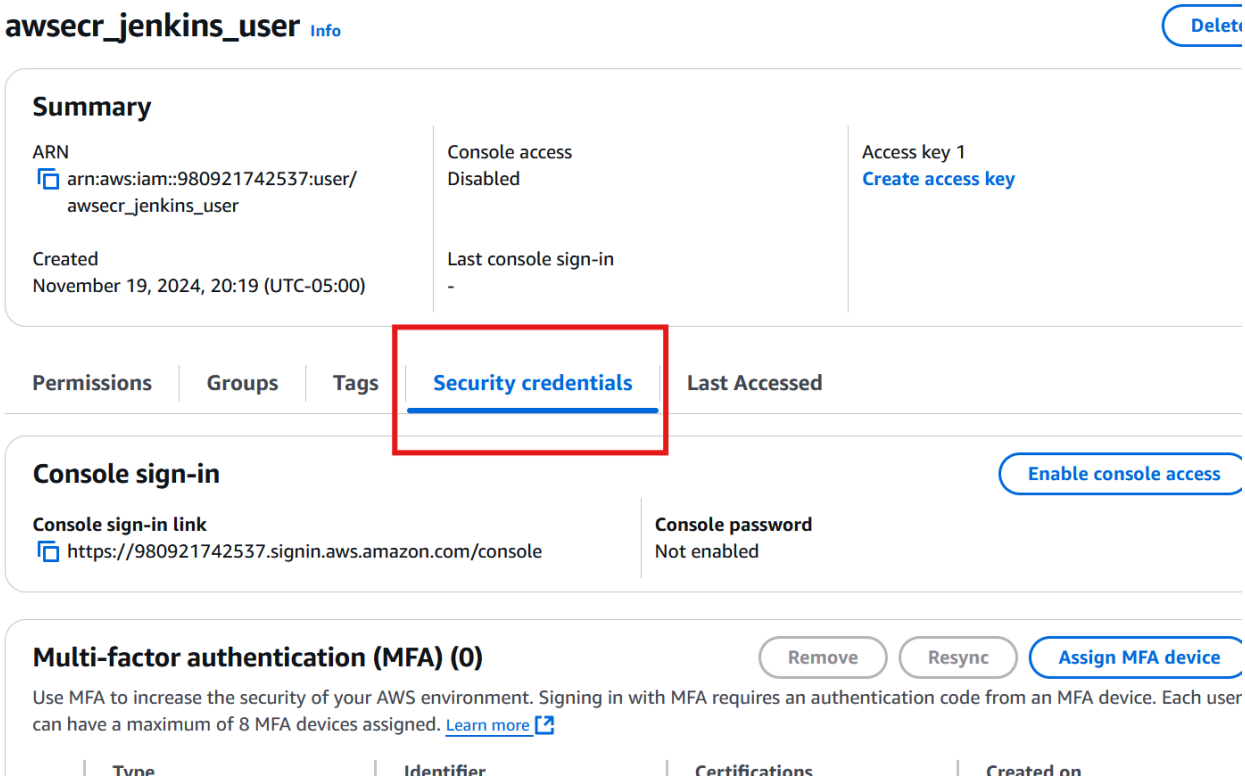
Filter by Type

container | All types | 15 matches | 1

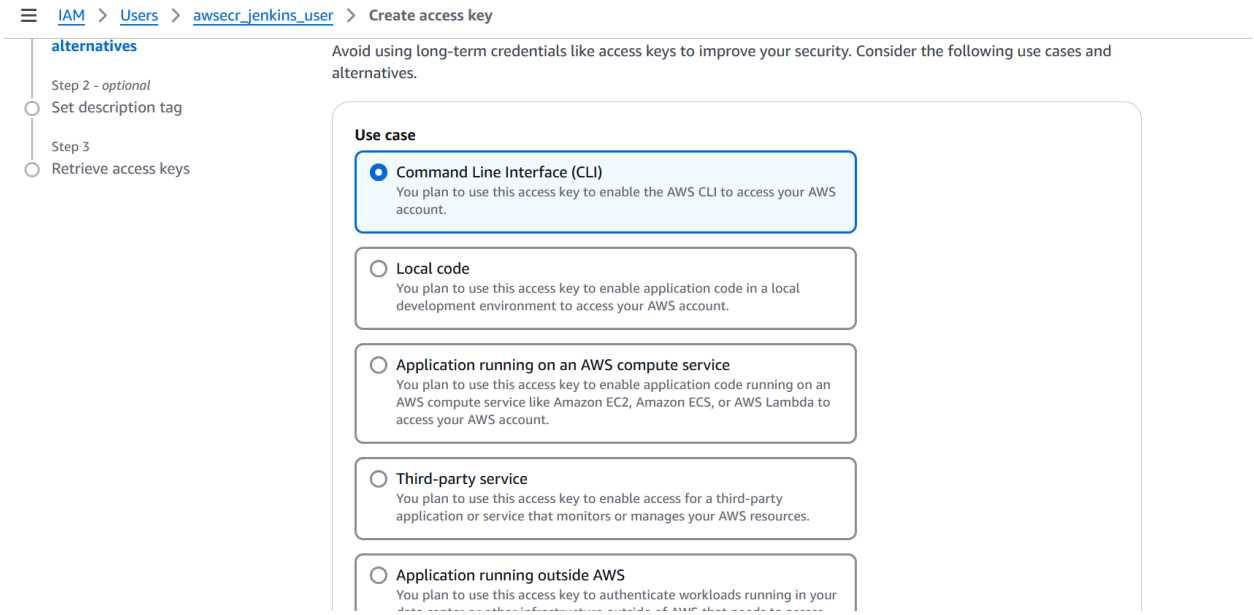
Policy name	Type	Attached entities
☑ AmazonEC2ContainerRegis...	AWS managed	0
☐ AmazonEC2ContainerRegis...	AWS managed	0
☐ AmazonEC2ContainerRegis...	AWS managed	0



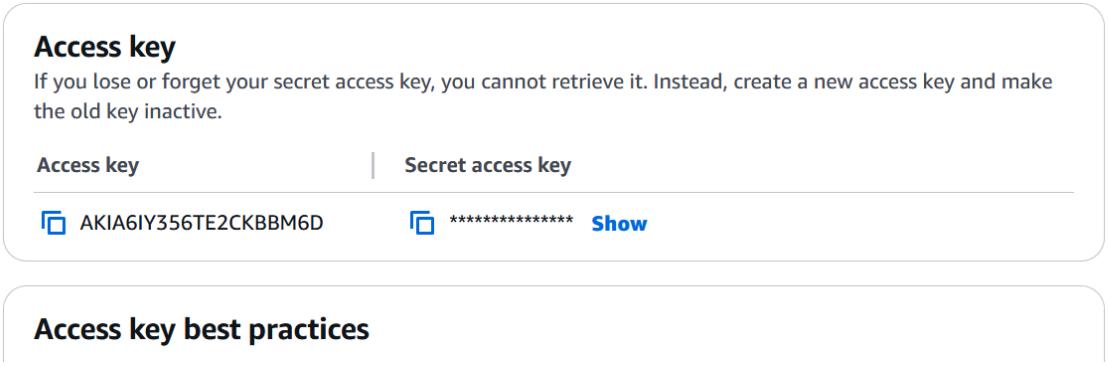
- Once the user is created go to security credentials and navigate to access keys and create a access key.



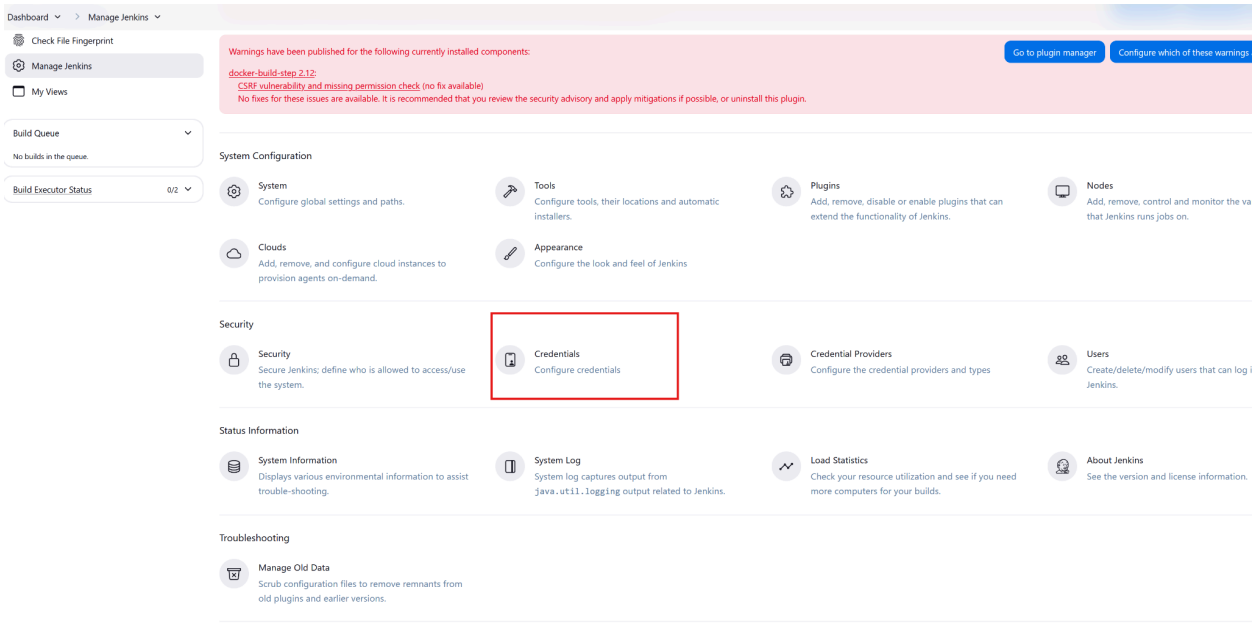
- On create access key, click on Command Line Interface (CLI)



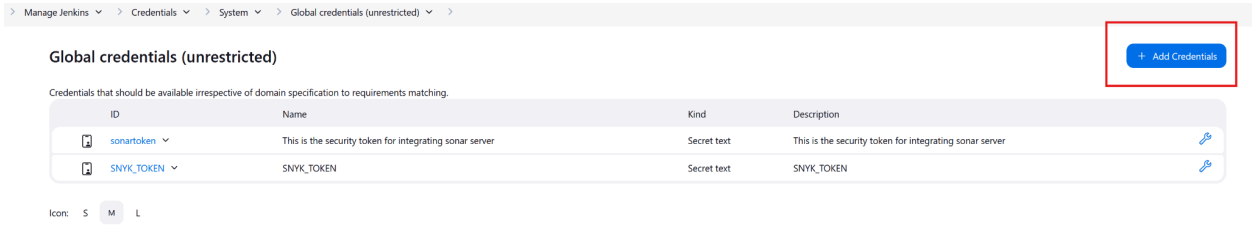
- Save both access keys on the notepad or notes for adding it in the jenkins server.



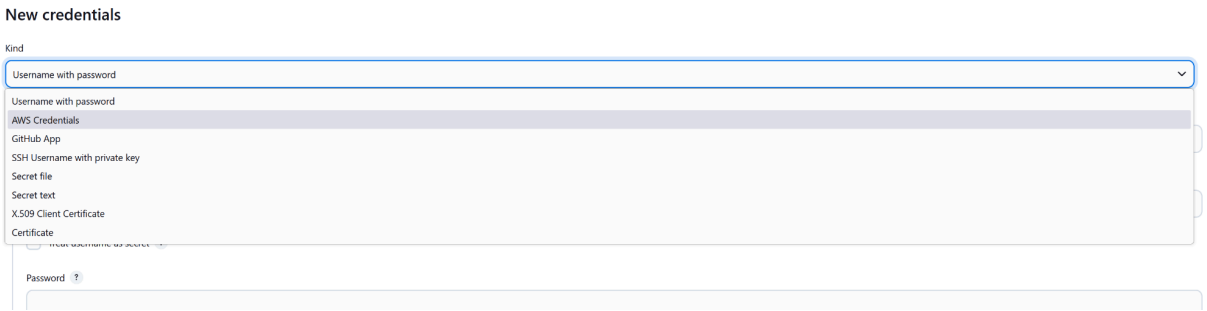
- Go to jenkins dashboard and manage jenkins>credentials.



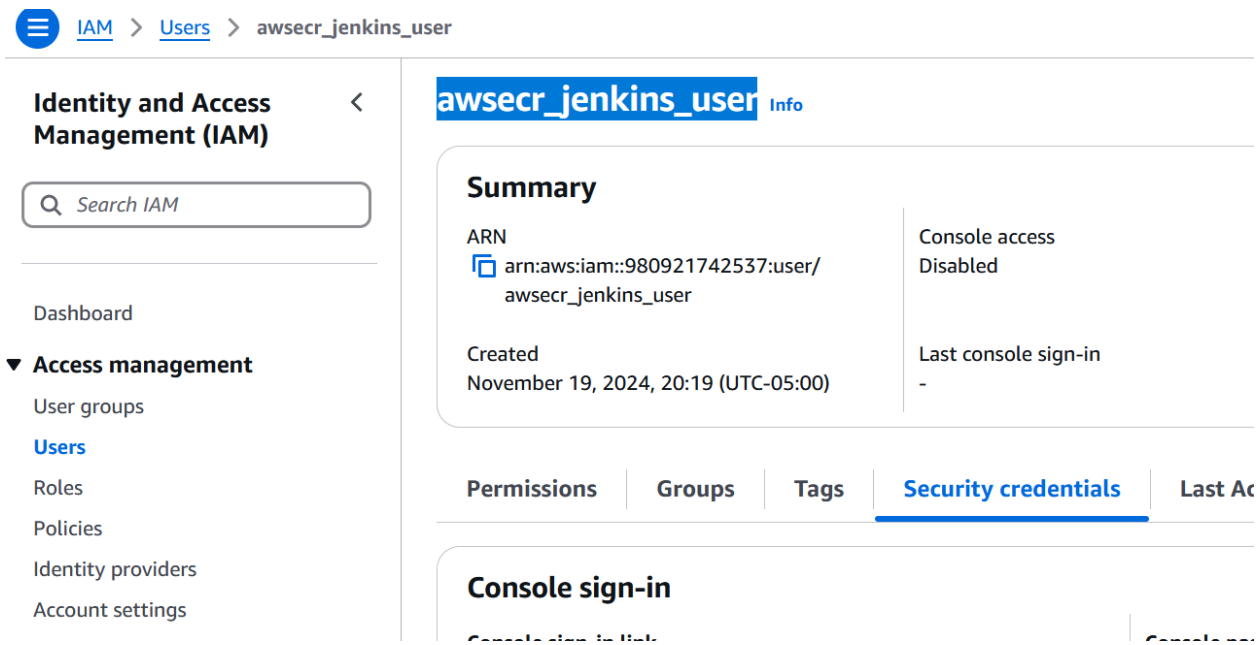
- Add credentials



Click on AWS credentials



For ID paste the name of user you created on the AWS



After inputting the access key and other values hit create.

- Now go back to your pipeline and add the registryCredential parameter to the environment code. Credential parameter is the ID of the credential or the username of the AWS user you have created.

Adding new parameter in the “docker.withRegistry parameter” as “http://” + registry (which is uri of ECR registry), “ecr:us-east-1:” + registryCredential) Note: change you zone accordingly.

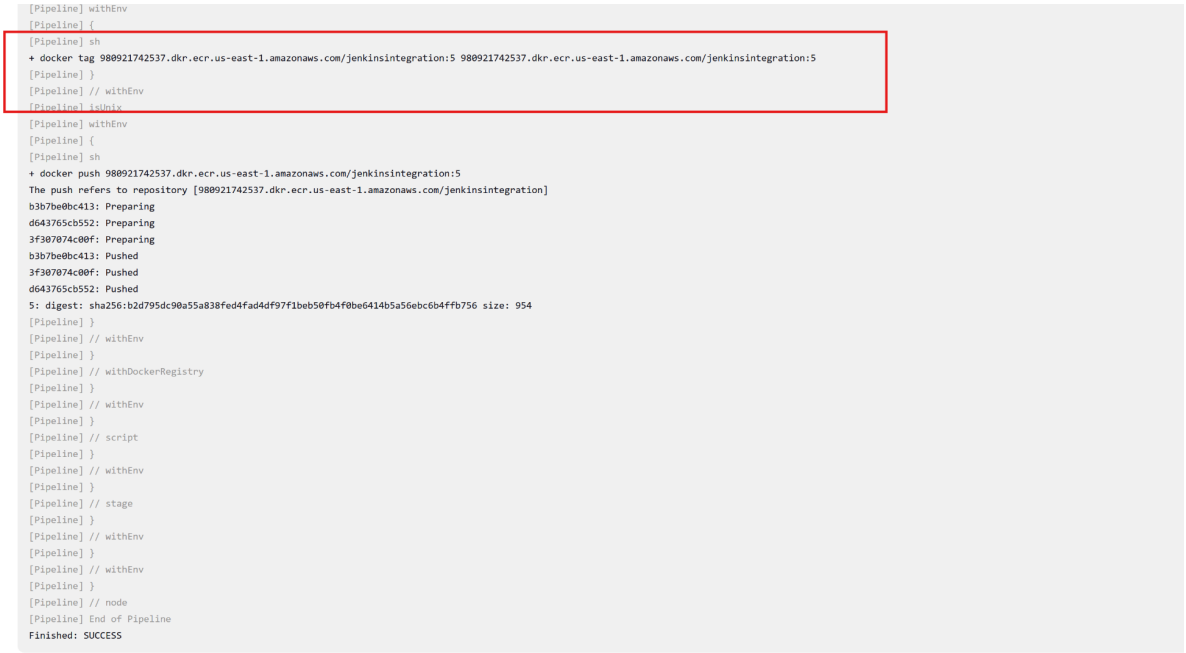
- Whole code below for the pipeline.

```
pipeline {
      agent any
      tools {
      maven 'maven-3.9.9'
      }
      environment {
      registry =
```

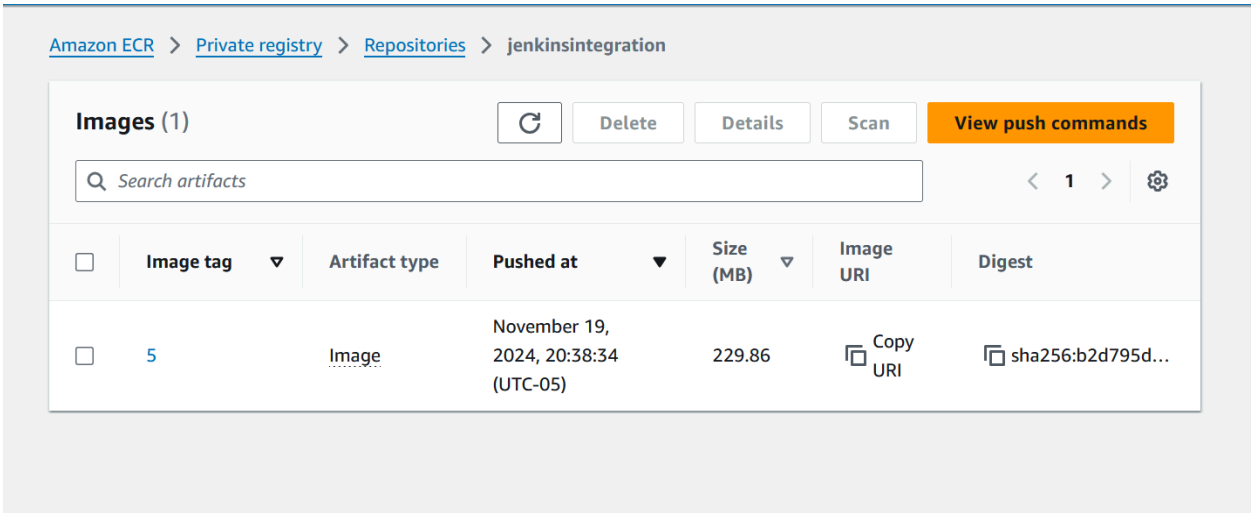
```
"980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration"
      registryCredential = 'awsecr_jenkins_user'
      }
      stages {
      stage ("Checking out the git project") {
            steps {
                  git branch: 'master', url:
'https://github.com/Shikhar82/springboot-maven-micro.git'
            }
      }

      stage ("Checking bugs on sonarcloud") {
            steps {
                  sh 'mvn clean verify sonar:sonar
-Dsonar.projectKey=devsecopskp -Dsonar.organization=devsecopskp
-Dsonar.host.url=https://sonarcloud.io
-Dsonar.token=d0d2af0eee7f1fdb21e937360084b8ece5a62cb4'
            }
      }
      // stage ("Build the package") {
      //    steps {
      //          sh 'mvn clean package'
      //    }
      // }

      stage ("Run SCA scan and analysis using snyk"){
            steps {
                  withCredentials([string(credentialsId: 'SNYK_TOKEN',
variable: 'SNYK_TOKEN')]) {
                  sh 'mvn snyk:test -fn'
                  }
            }
      }
      stage('Building our image') {
            steps{
                  script {
                  dockerImage = docker.build registry + ":$BUILD_NUMBER"
                  }
            }
      }
      stage('Deploy our image') {
            steps{
                  script {
```

```
                    docker.withRegistry("http://" + registry,
"ecr:us-east-1:" + registryCredential ) {
                    dockerImage.push()
                    }
                    }
               }
          }
          }

}
```

- Once everything set in your pipeline, save the code and click build now.



- Let’s check whether image is successfully pushed to AWS ECR.



We got the image pushed successfully.

- Let’s also check about it on jenkins server vm.

Successfull build and deploy of docker image.

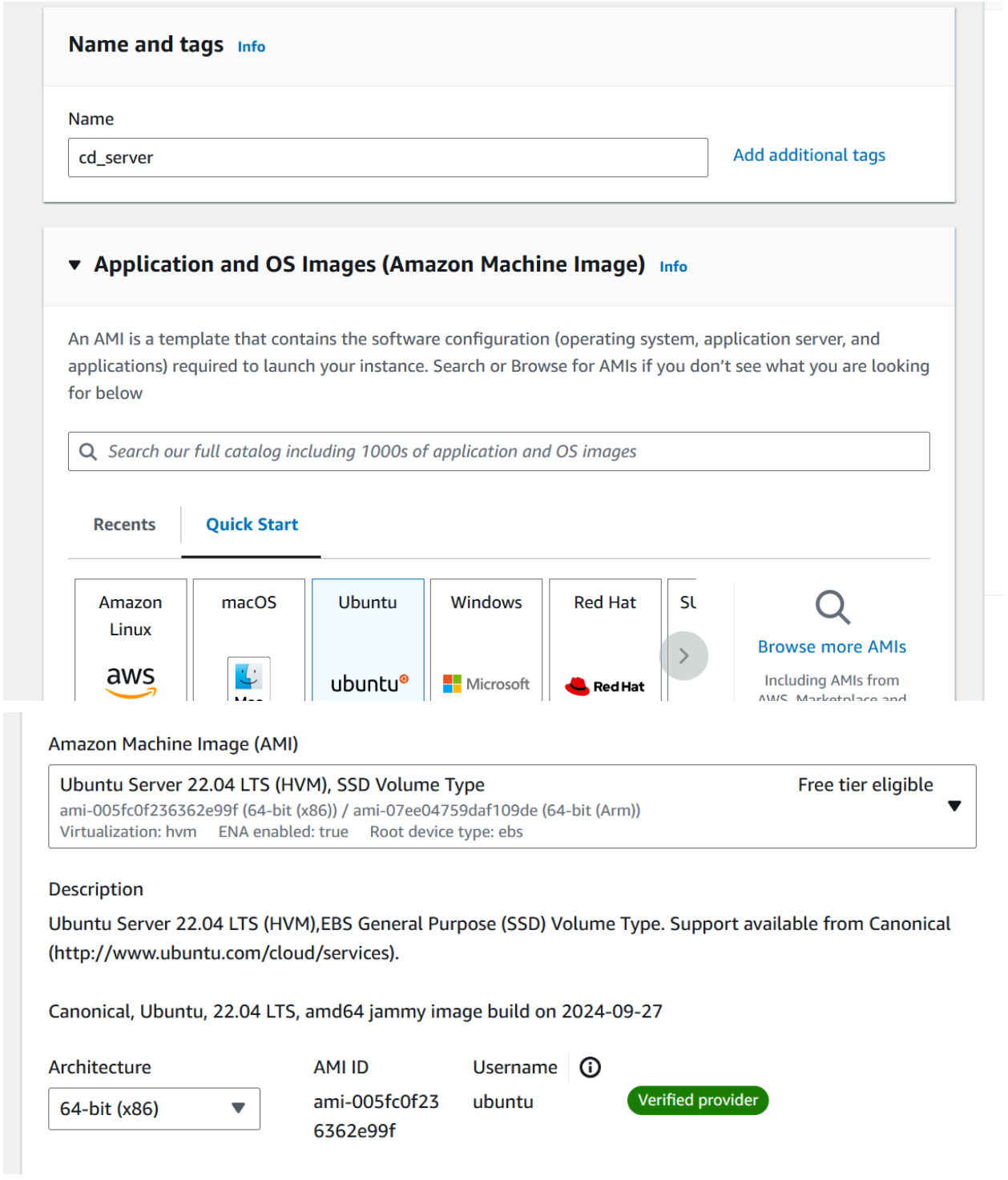
Great work. CI Continuous Integration is ready.

Time to do the Continuous Deployment.

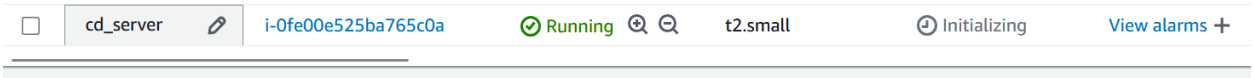
What is Continous Deployment?
Continuous Deployment (CD) is a software development practice where changes to the codebase are automatically deployed to production environments without manual intervention. It is the next step after **Continuous Integration (CI)** and **Continuous Delivery (also CD)**, focusing on delivering software updates directly to end-users as soon as they pass automated testing and quality checks.

CD server

- Go to AWS and create a ec2 instance and we will use UBUNTU as a CD server.



- Once the ubuntu is running we will access it through ssh client and install docker and aws cli.



- Update the system

```
ubuntu@ip-172-31-83-200:~$ sudo su
root@ip-172-31-83-200:/home/ubuntu# apt update
```

- Install aws cli and add it to the path which we have done above already for jenkins server. Take it as a reference.

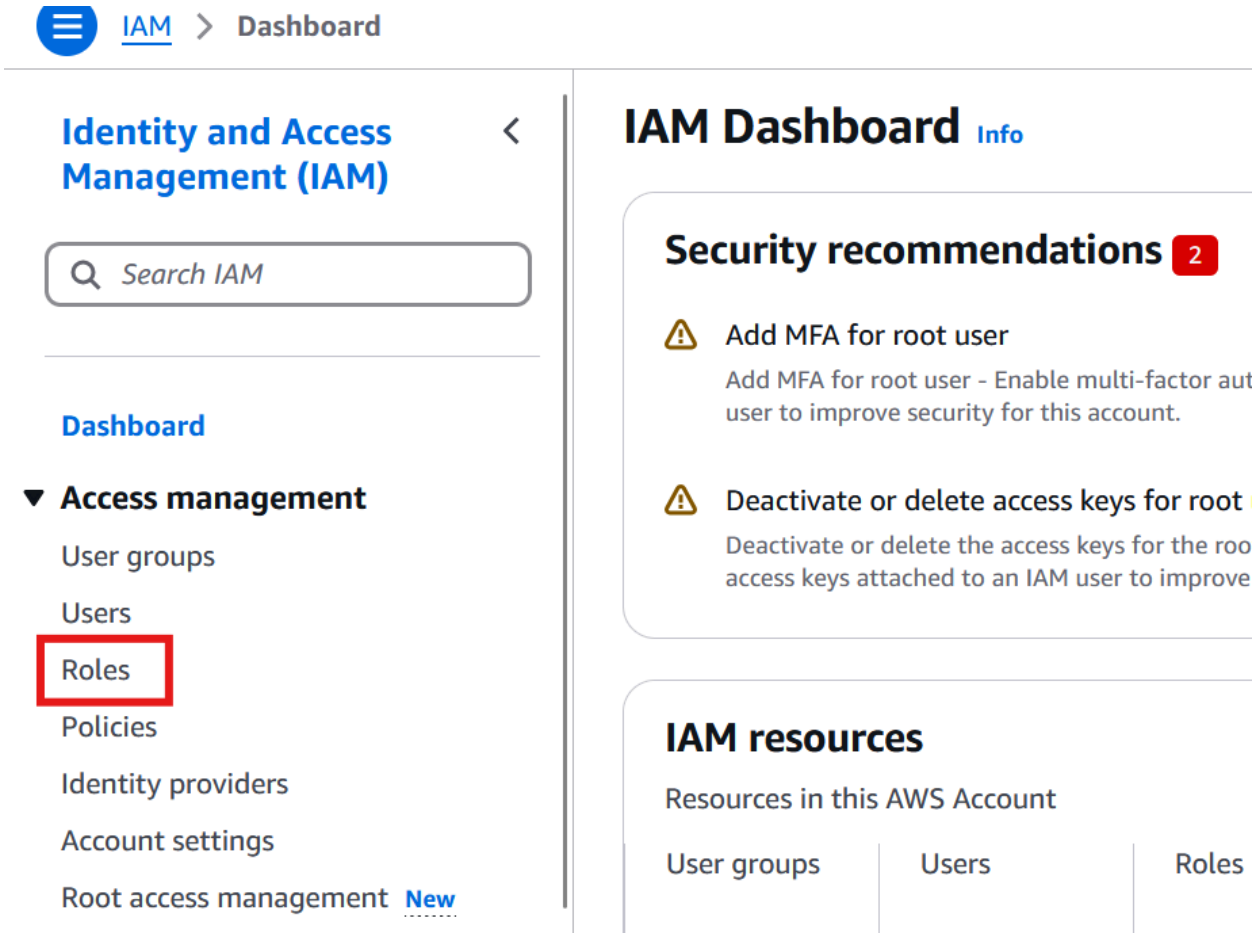
```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"
-o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

```
aws  awscliv2.zip
root@ip-172-31-83-200:/home/ubuntu# cd aws
root@ip-172-31-83-200:/home/ubuntu/aws# ls
README.md  THIRD_PARTY_LICENSES  dist  install
root@ip-172-31-83-200:/home/ubuntu/aws# ./install
You can now run: /usr/local/bin/aws --version
root@ip-172-31-83-200:/home/ubuntu/aws# export PATH:$PATH:/usr/local/bin
bash: export: `PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/loca
bin': not a valid identifier
root@ip-172-31-83-200:/home/ubuntu/aws# export PATH=$PATH:/usr/local/bin
root@ip-172-31-83-200:/home/ubuntu/aws# aws --version
aws-cli/2.22.1 Python/3.12.6 Linux/6.8.0-1015-aws exe/x86_64.ubuntu.22
root@ip-172-31-83-200:/home/ubuntu/aws#
```

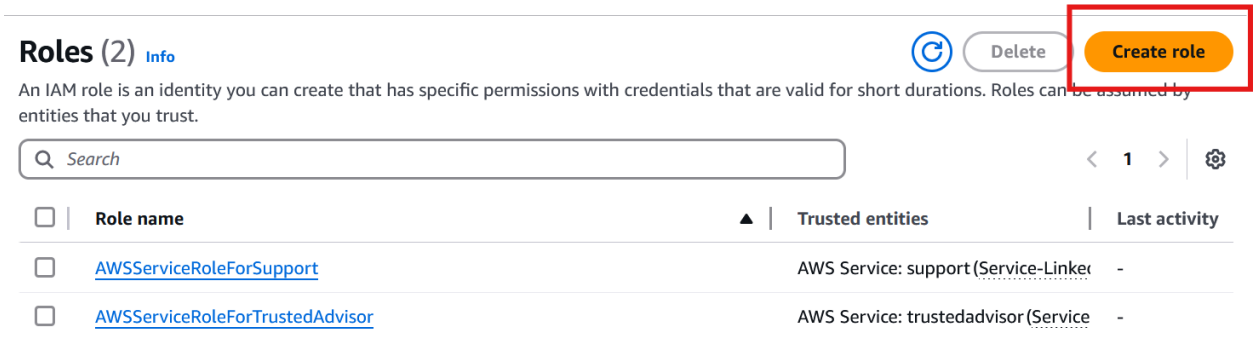
- Now let's install the docker on the Ubuntu which is cd server.
    1. apt install docker
    2. apt install podman-docker

```
root@ip-172-31-83-200:~# apt install podman-docker
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  buildah catatonit conmon containernetworking-plugins crun dconf-gsettings-backend dconf-service dns-root-data dnsmasq-base
  docker-compose fuse-overlayfs glib-networking glib-networking-common glib-networking-services
  golang-github-containernetworking-plugin-dnsname golang-github-containers-common golang-github-containers-image
  gsettings-desktop-schemas libavahi-client3 libavahi-common-data libavahi-common3 libavahi-glib1 libdconf1 libostree-1-1
  libproxy1v5 libslirp0 libsoup2.4-1 libsoup2.4-common libyajl2 podman python3-docker python3-dockerpty python3-docopt
  python3-dotenv python3-texttable python3-websocket session-migration slirp4netns uidmap
Suggested packages:
  containers-storage
Recommended packages:
  docker.io
The following NEW packages will be installed:
  buildah catatonit conmon containernetworking-plugins crun dconf-gsettings-backend dconf-service dns-root-data dnsmasq-base
  docker-compose fuse-overlayfs glib-networking glib-networking-common glib-networking-services
  golang-github-containernetworking-plugin-dnsname golang-github-containers-common golang-github-containers-image
  gsettings-desktop-schemas libavahi-client3 libavahi-common-data libavahi-common3 libavahi-glib1 libdconf1 libostree-1-1
  libproxy1v5 libslirp0 libsoup2.4-1 libsoup2.4-common libyajl2 podman podman-docker python3-docker python3-dockerpty
  python3-docopt python3-dotenv python3-texttable python3-websocket session-migration slirp4netns uidmap
```

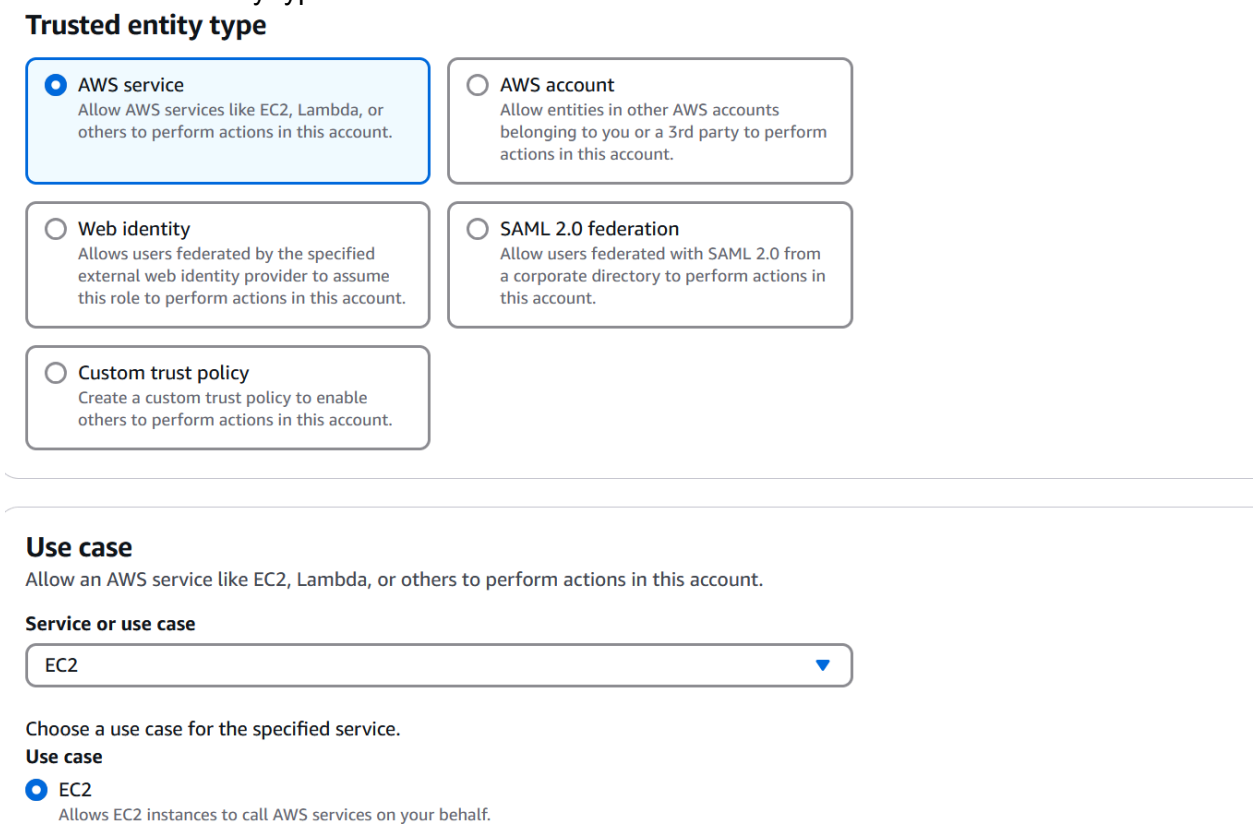
- We have to create a role now like we have created a policy for a user. Role is important for connecting the ec2 with the AWS ECR.
- Go to aws management console and navigate to IAM and click on roles.



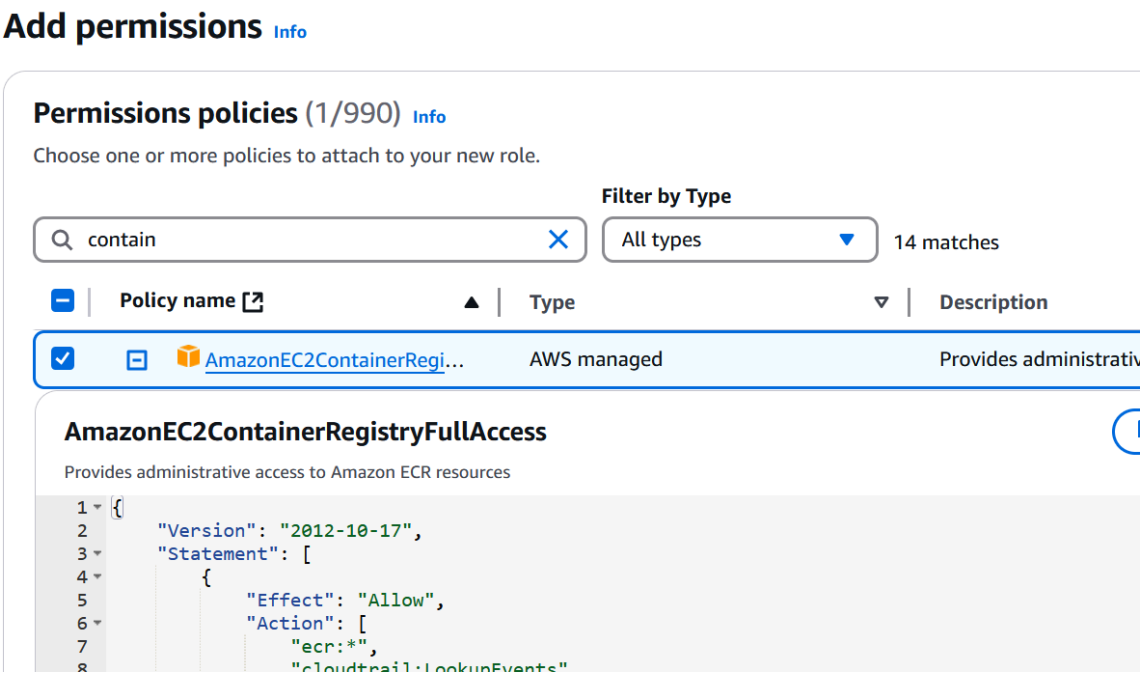
- Create role



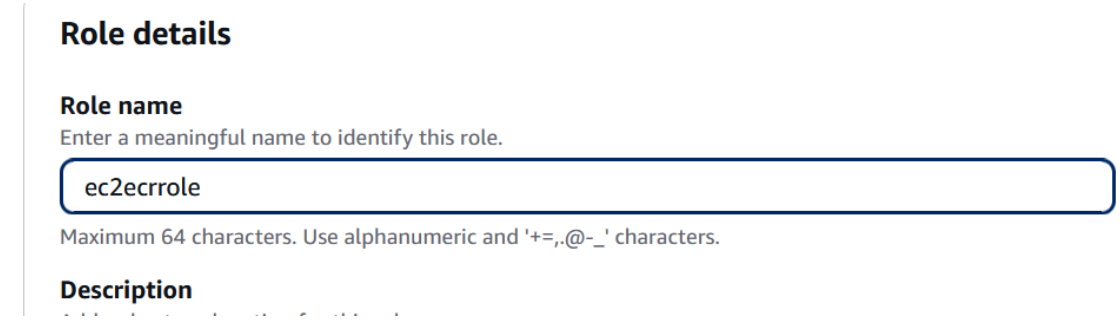
- Select Trusted entity type as AWS service and use case is EC2.



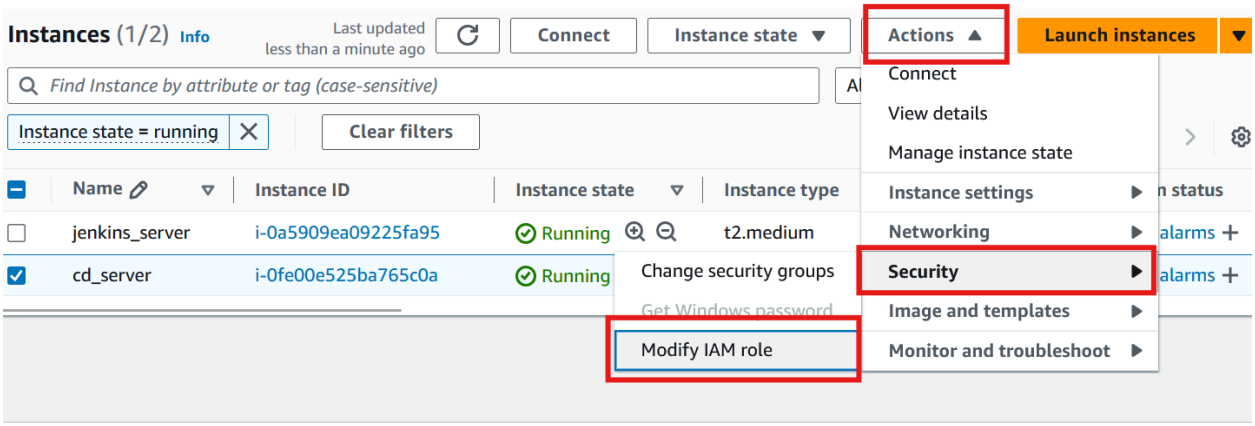
- Click on AmazonEC2ContainerRegistryFullAccess



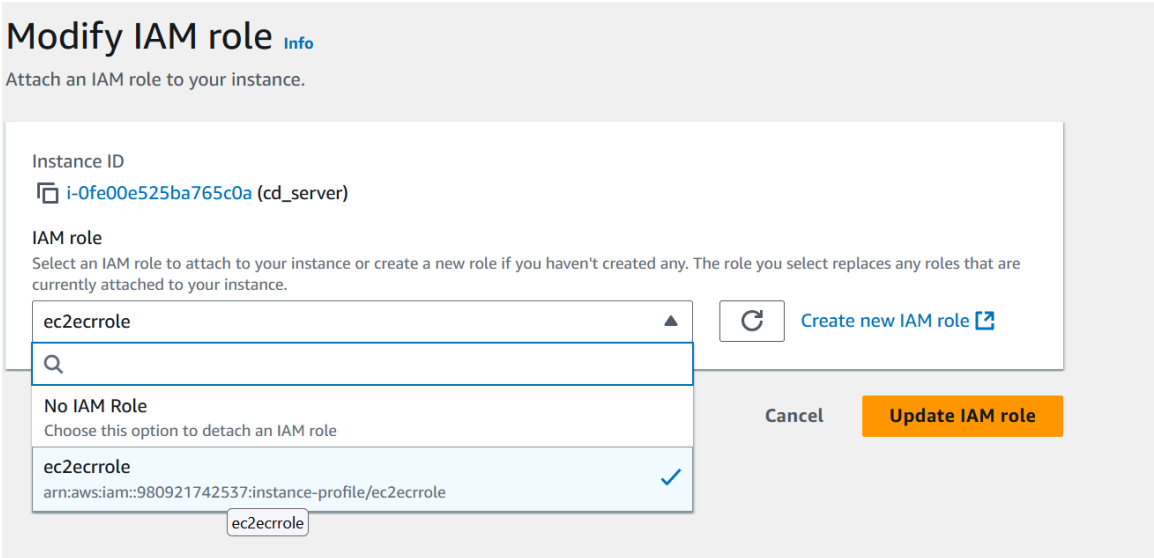
- Give a role name and make sure you remember this role name or note it somewhere and click on create role.



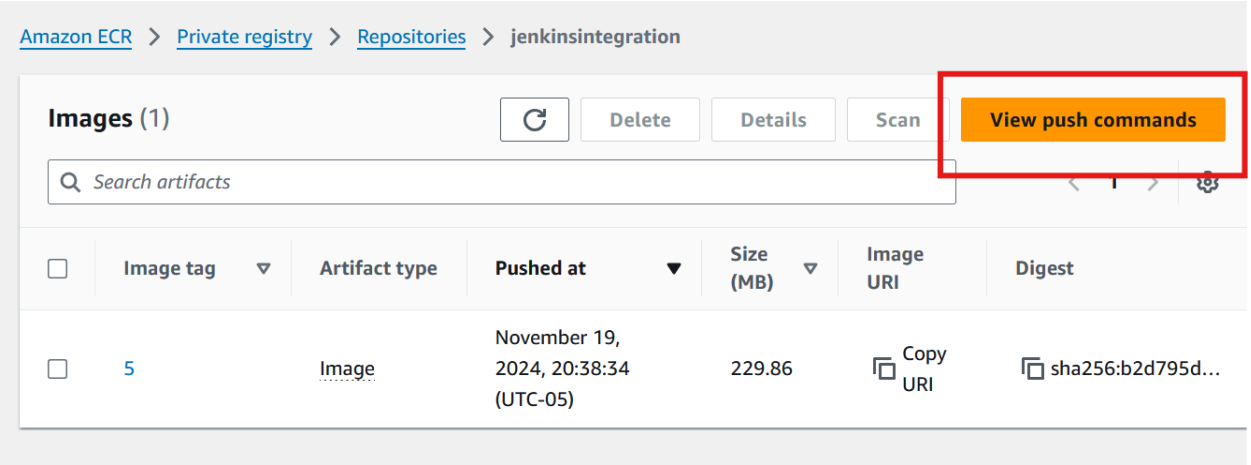
- Once the role is created go to ec2 instances and locate your cd_server instance and click on actions on the top and click on modify IAM role.



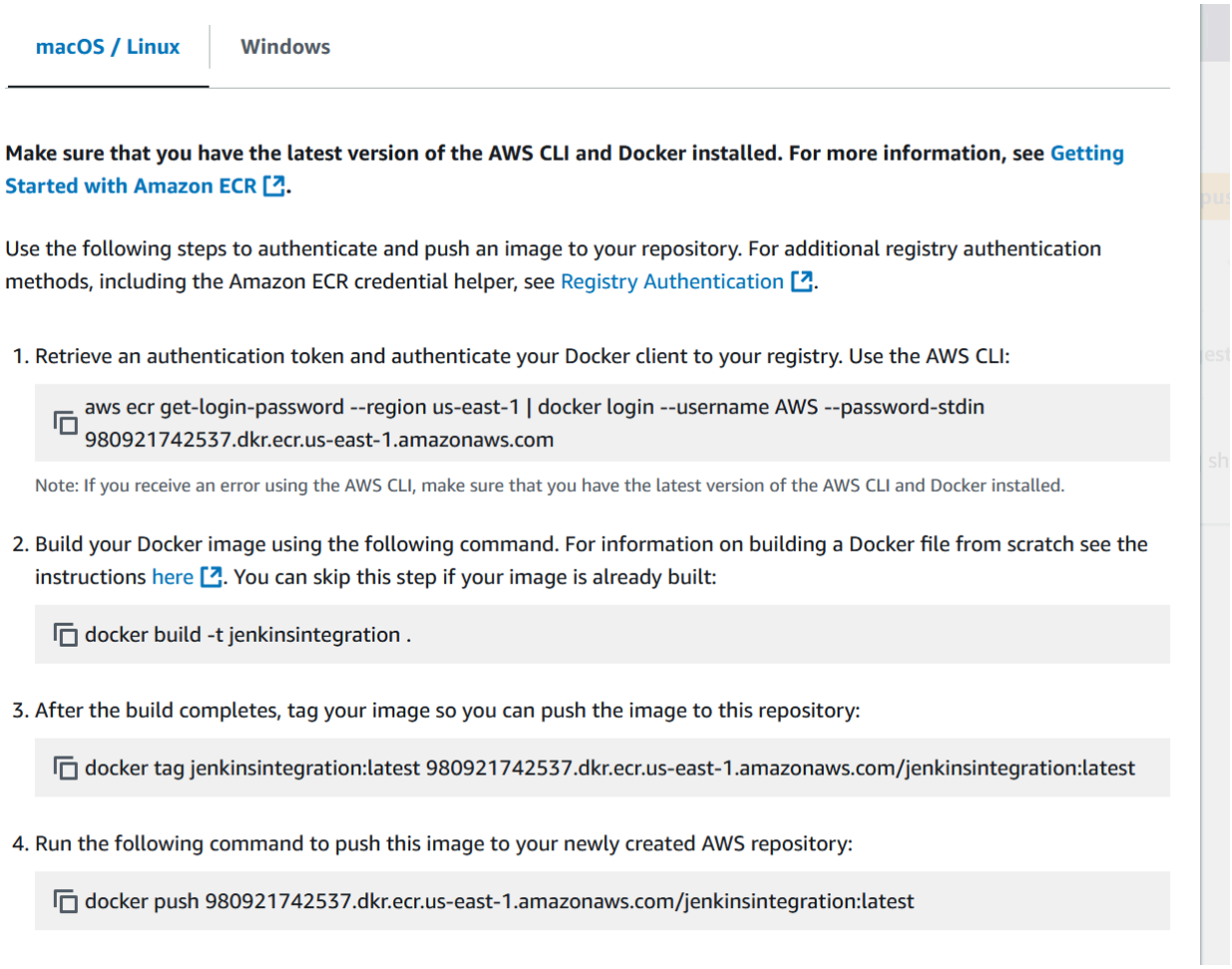
- Choose your newly create role and update IAM role.



- Now everything is set time to push ECR into the CD server which is ubuntu in our case.
- Navigate to ECR on aws management console and click on view push commands.



- We got certain commands for the integration, copy the first command and paste it on your ubuntu terminal which is cd server.



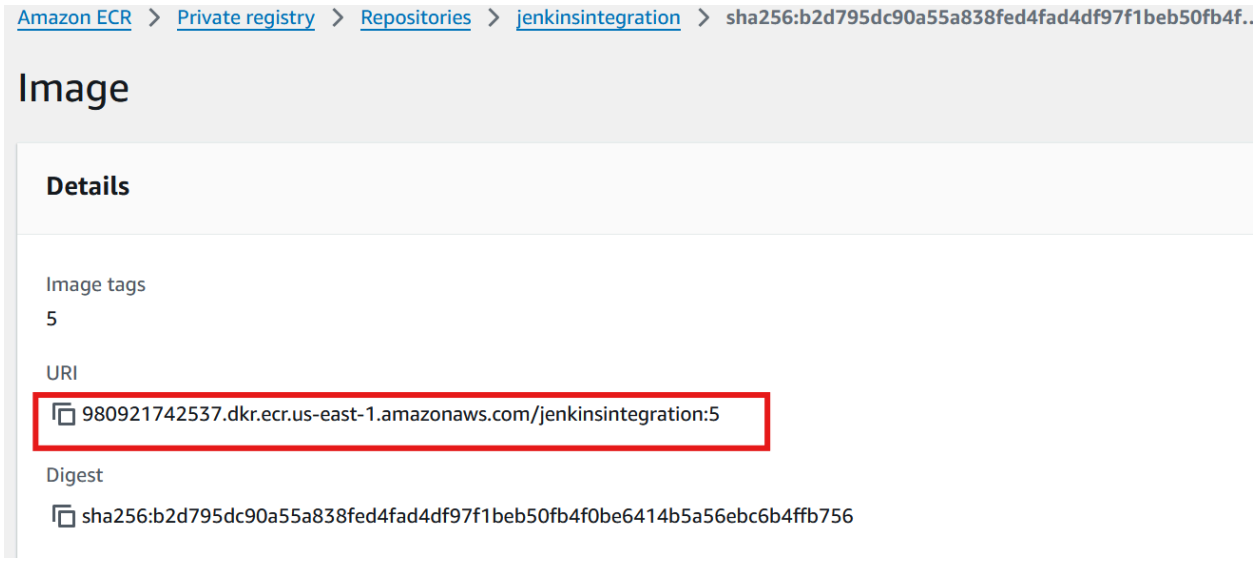
```
root@ip-172-31-83-200:~# aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 980921742537
.dkr.ecr.us-east-1.amazonaws.com
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
Login Succeeded!
```

Now I am able to access the aws ecr.

- Time to run the command to transfer the container and image to cd server.
  Use following command where you can modify the app name, port and URI.

```
docker container run --name=firstapp -p 8000:8000 --detach
980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration:5
```

I got the uri from here in the screenshot below.

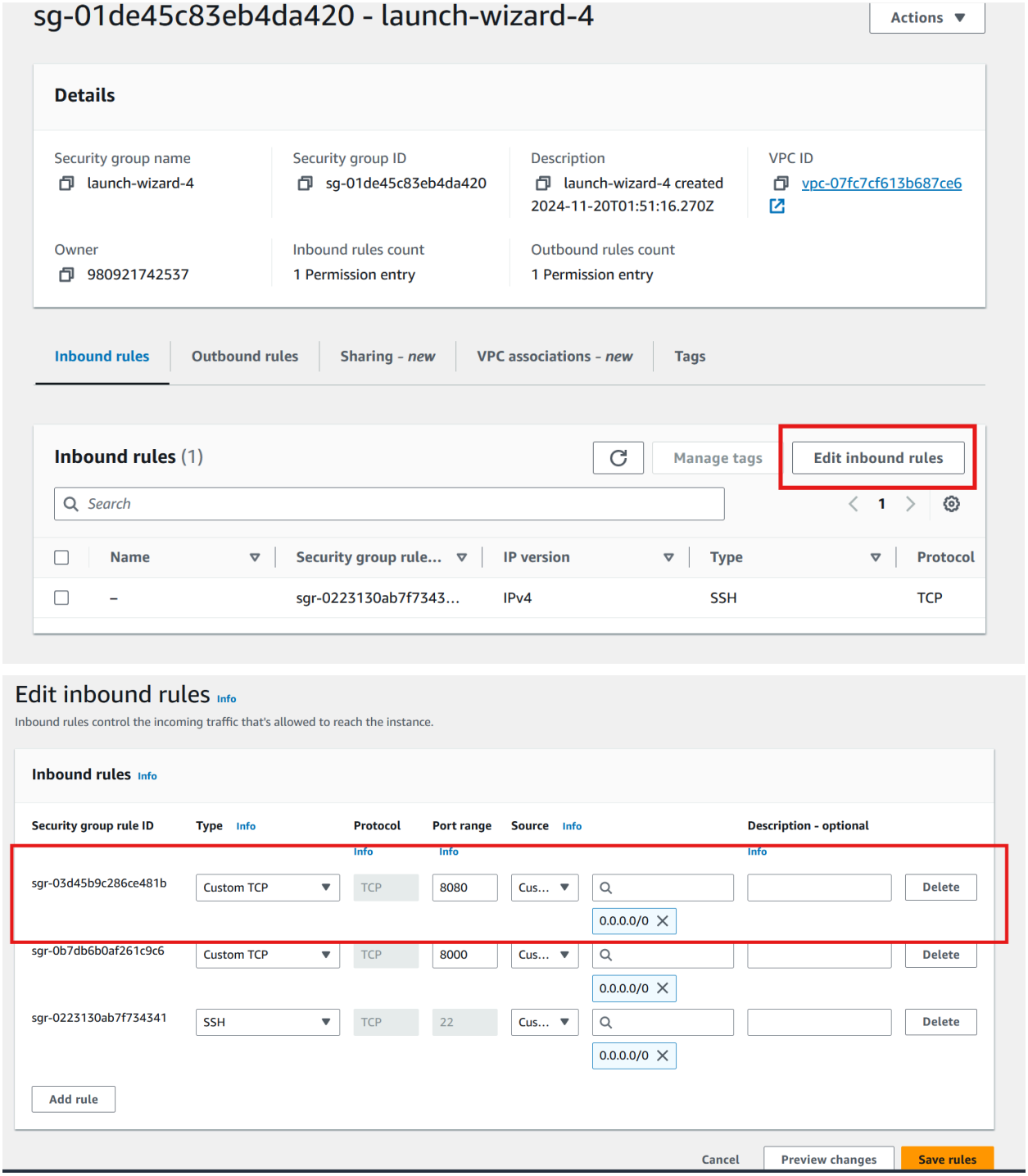


- Once the command run successfully try to look for container through the two commands.
  1. docker image ls
  2. docker container ls

```
root@ip-172-31-83-200:~# docker container run --name=firstapp -p 8080:8080 --detach 980921742537.dkr.ecr.us-east-1.amazonaws.com/j
enkinsintegration:5
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
Trying to pull 980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration:5...
Getting image source signatures
Copying blob 899046e4a240 done
Copying blob 849fae9bc522 done
Copying blob 5ce47bbf12c8 done
Copying config e90a3e7289 done
Writing manifest to image destination
Storing signatures
cf25e3cd91fae58f041f5893b93a44056d8233e4fcf2f3886a5cc4a35a55a601
root@ip-172-31-83-200:~# docker container ls
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
CONTAINER ID  IMAGE                                                                COMMAND     CREATED        STATUS            PORT
S                NAMES
cf25e3cd91fa  980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration:5             5 seconds ago  Up 6 seconds ago  0.0.
0.0:8080->8080/tcp  firstapp
root@ip-172-31-83-200:~#
```

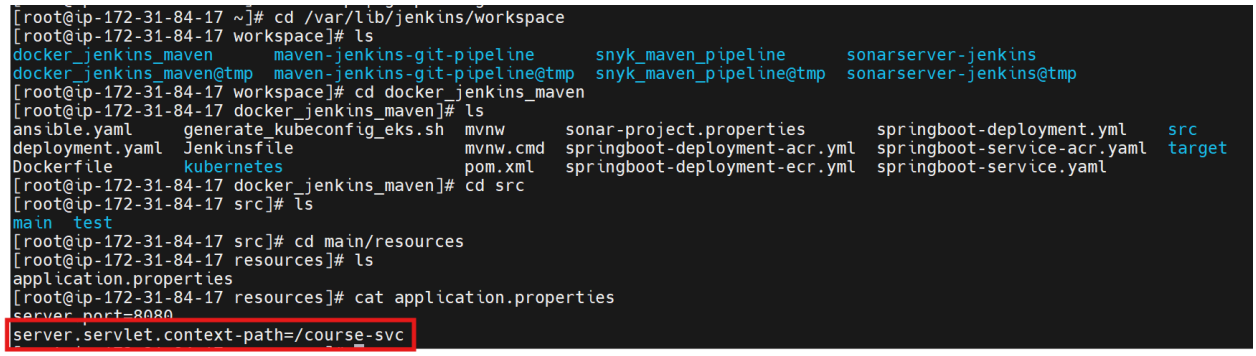
As we can see our container is running on port 8000 successfully.

- As we are using the port 8080(we are using port 8080 because docker container on aws ecr is using 8080 so we have to use the same port) we need to allow it on the inbound rules of security groups of our ubuntu instance on the aws management console.

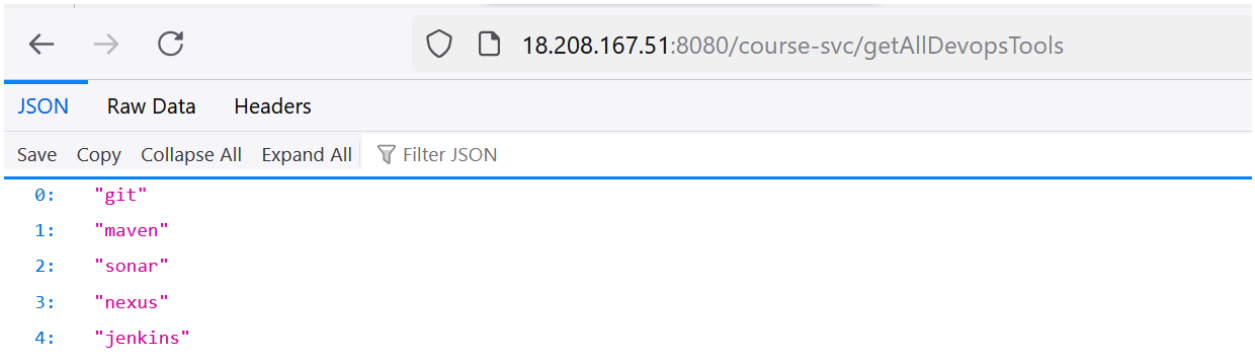


- As we enabled the port 8080 the application is running right now.
- We are going to access our application through the url which is IP address of the ubuntu vm and port 8080 but we need more parameters to it.

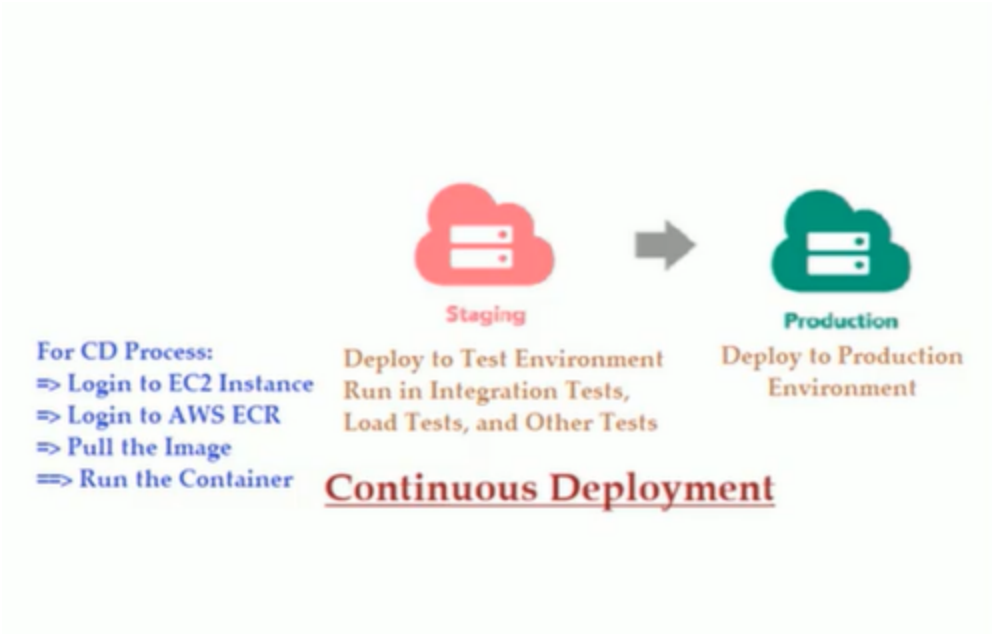
- How to look for it? Go to jenkins server vm go to “cd /var/lib/jenkins/workspace” and choose your pipeline name, in our case it is docker_jenkins_maven.



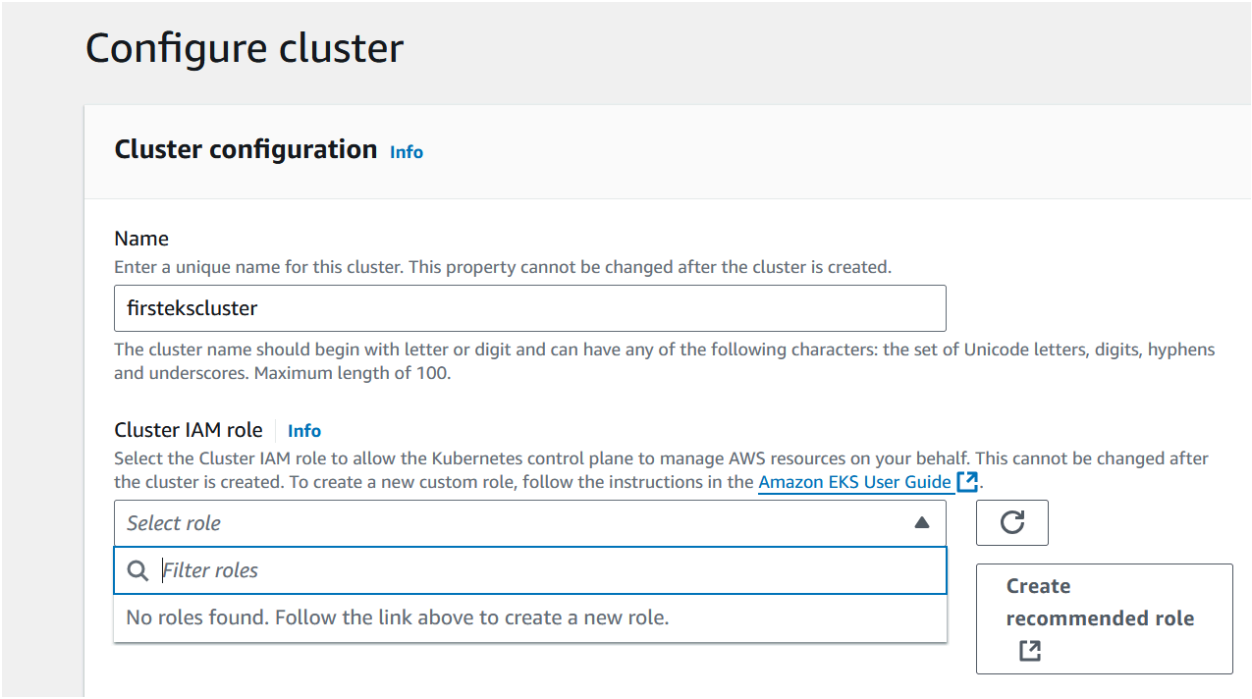
- Once you try to access the webpage through this url for this application you can access it successfully.



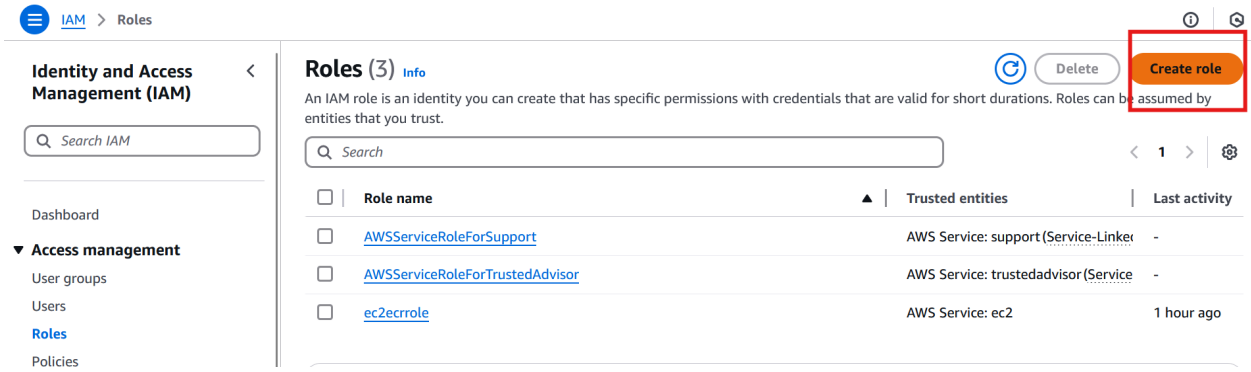
AWS EKS Cluster



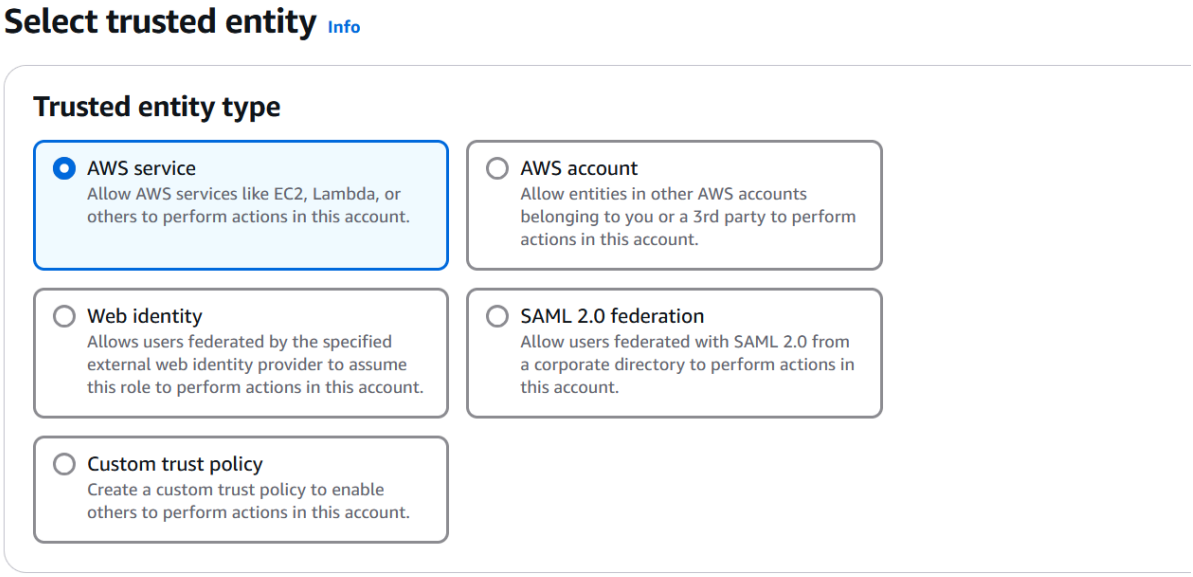
- EKS cluster Elastic Kubernetes Cluster has bunch of nodes running the docker images which will be controlled by the master node.
- Let's create a EKS cluster on AWS. Navigate to aws console and search for EKS.
- Click on create cluster for creating a new cluster, if you already created a cluster you can simply register it. But I am creating a new one.
- Give a name for your cluster and it is asking for IAM role which we need to create for AWS EKS.

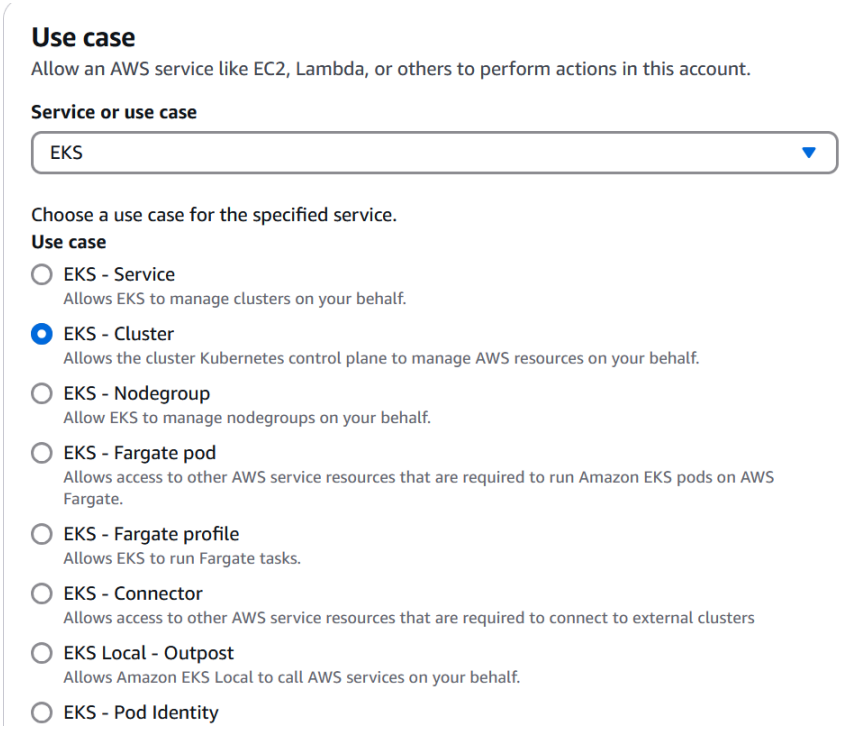


- Open a new windows and search for IAM and click on roles and create role.

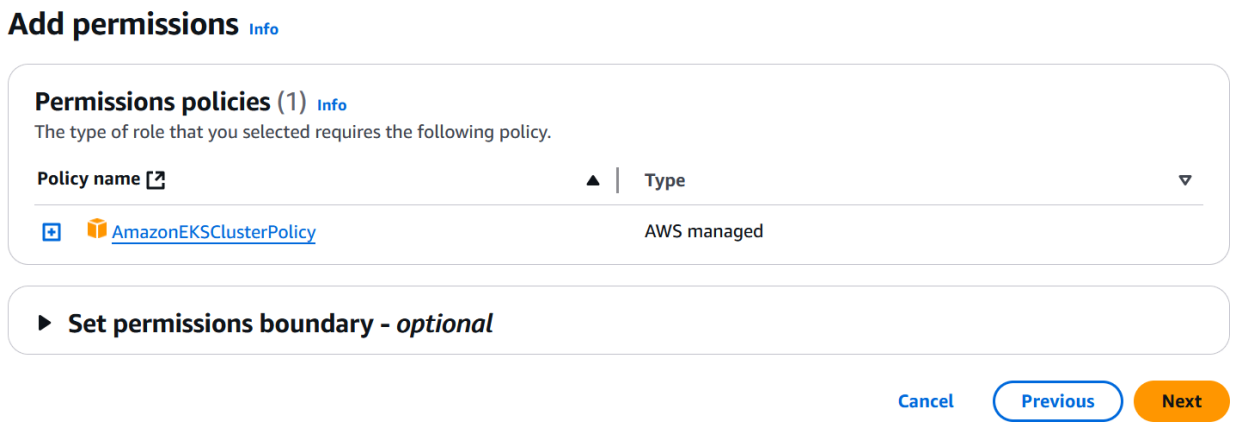


- Select AWS service and use case will be EKS> EKS-cluster like below.

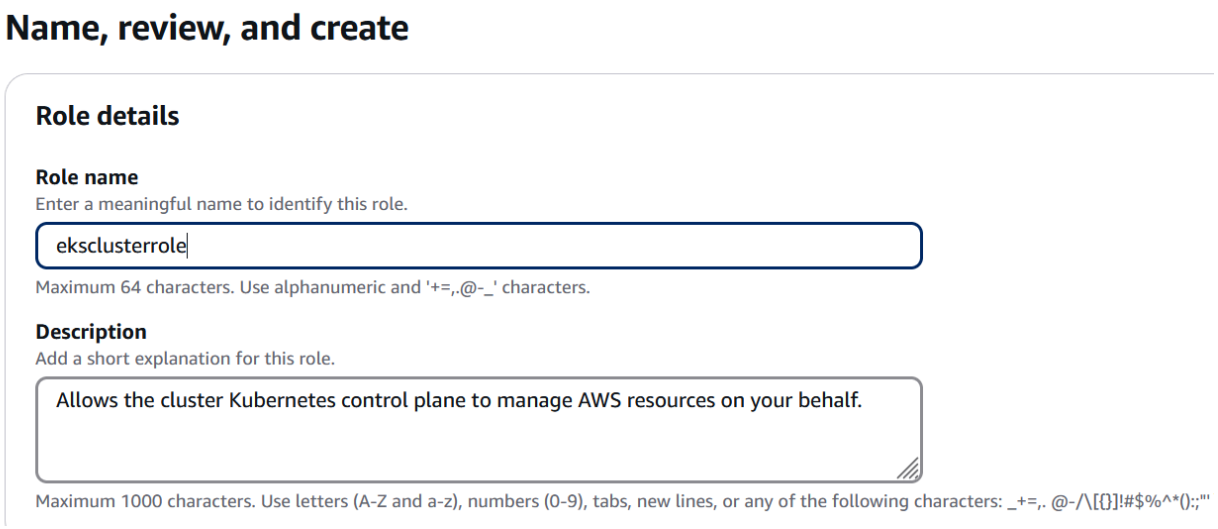




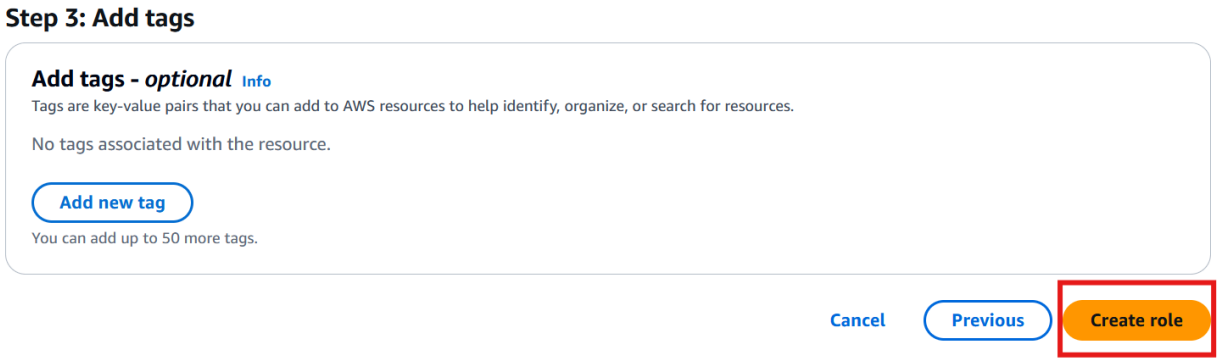
Keep everything default on add permissions



Give role name

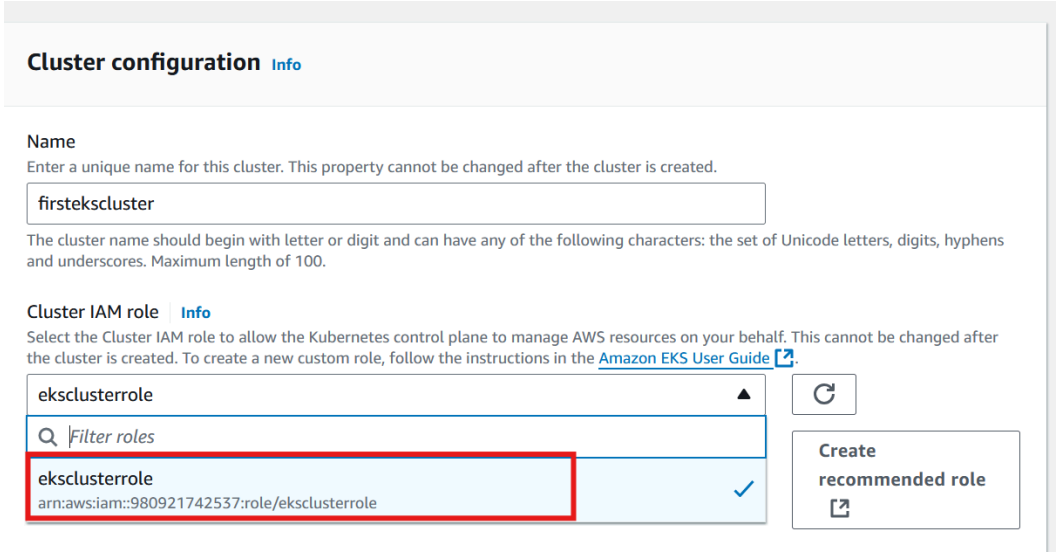


Click on create role

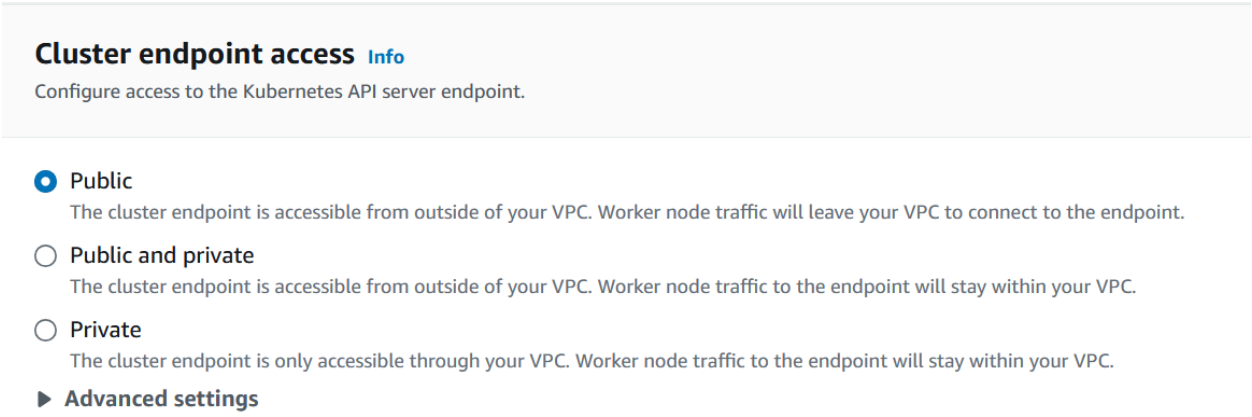


Once the role is create head back to EKS cluster creation process.

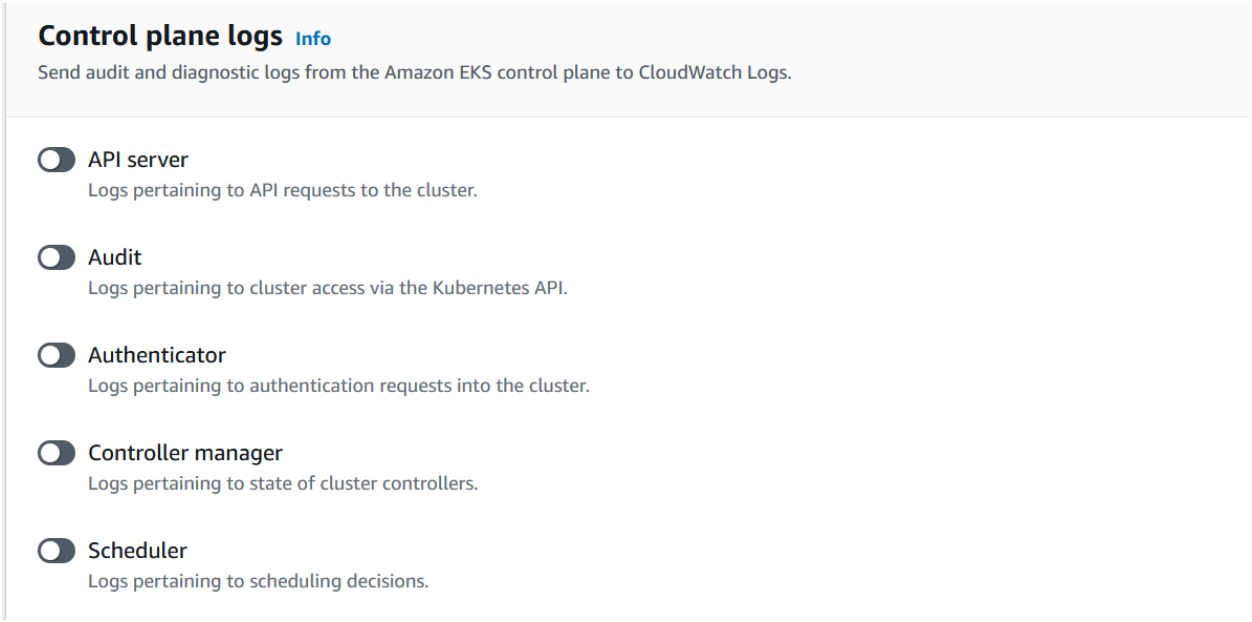
- Now select the role you have created for EKS cluster.



- Keep cluster endpoint access as public and leave everything default for this project, you can modify networking and other policies for your use case in the future.

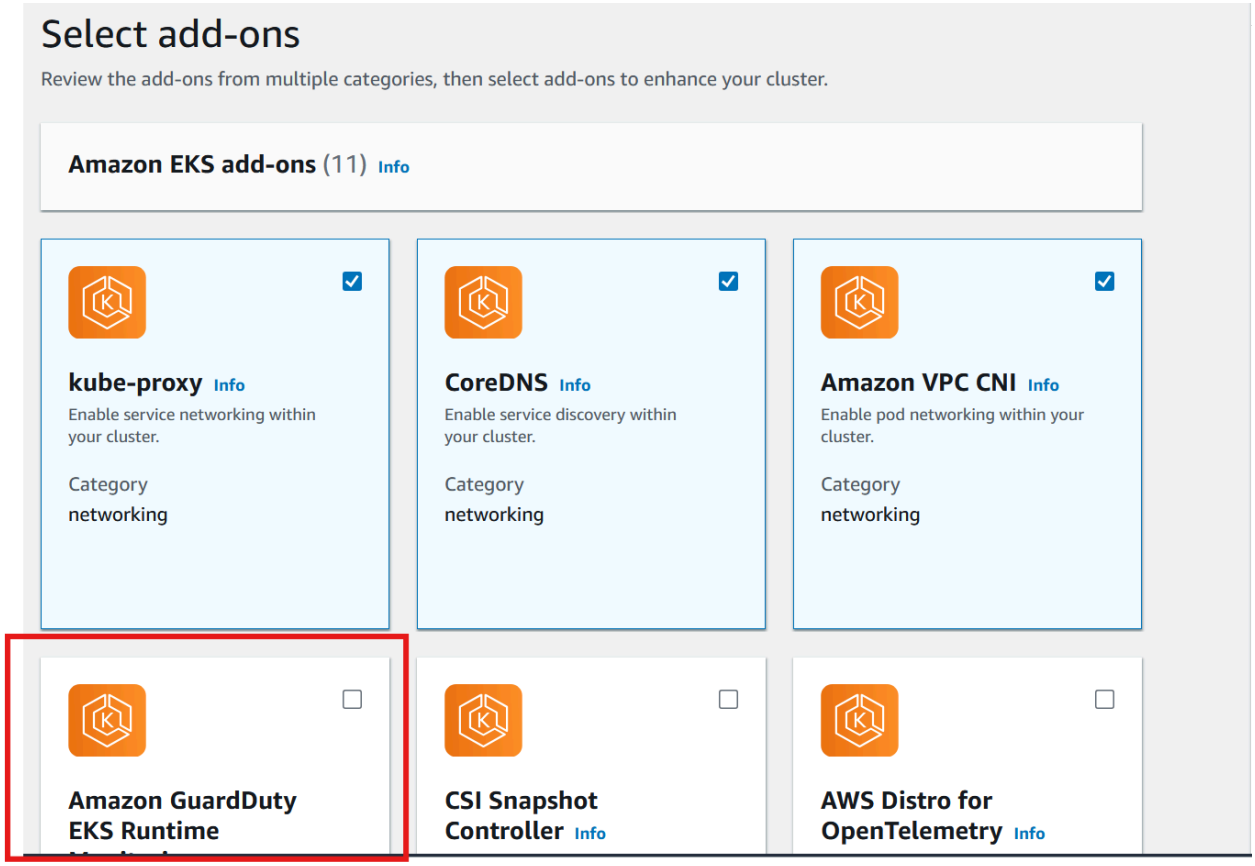


- You can play around here on control plane logs if you want those pertaining logs.

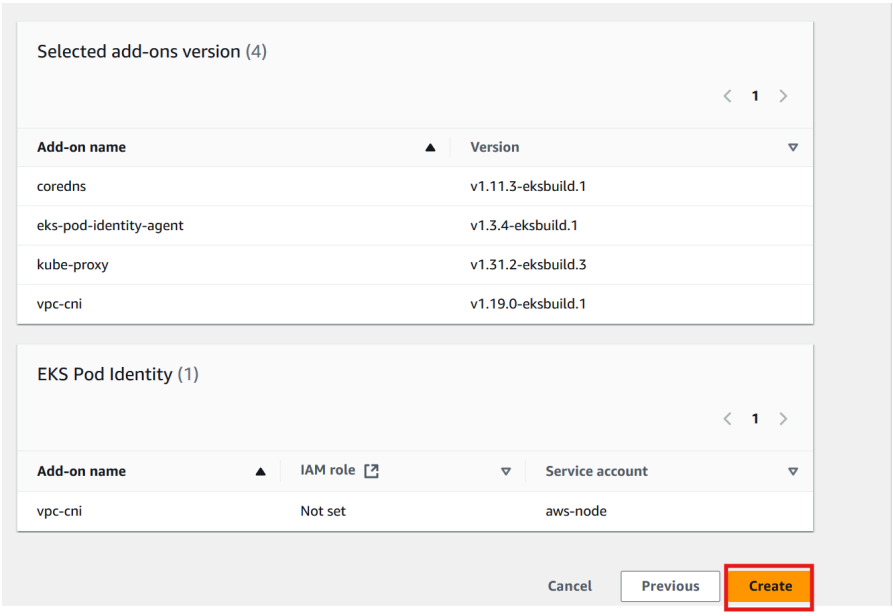


I am leaving everything default here as I don’t need those.

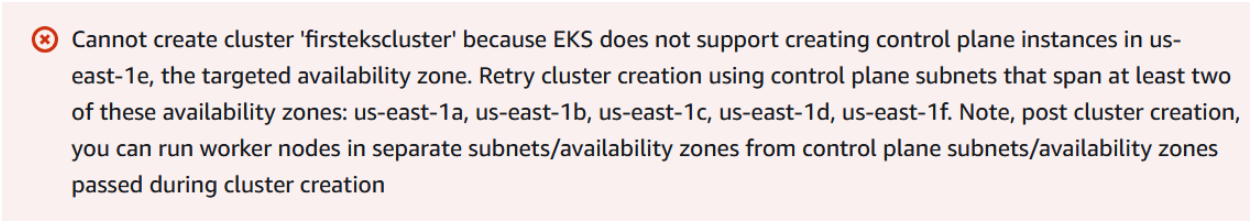
- Add-on if you need anything example guardDuty for protection you can use, I am going to destroy this application later so I don't need these.

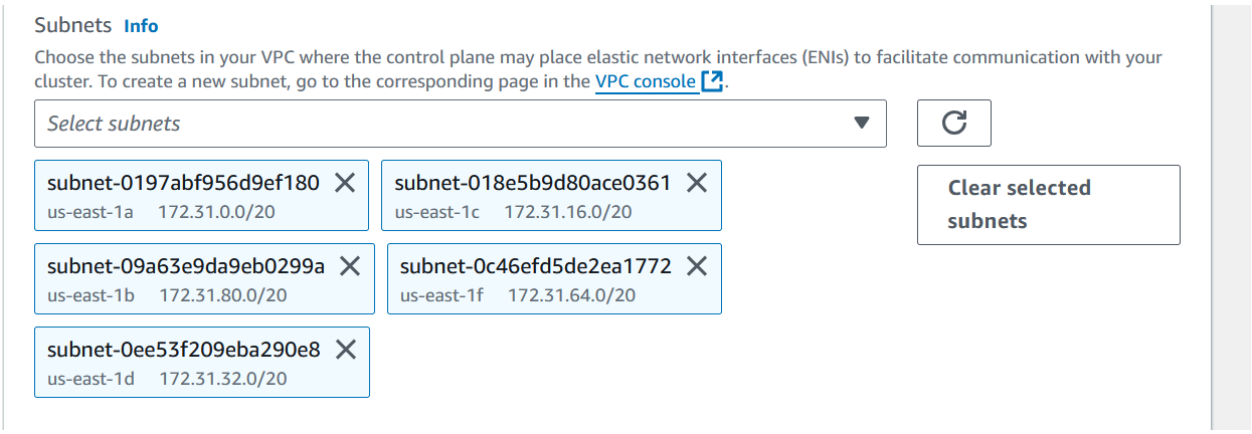


- Keep everything default and hit create cluster, Note: we can modify certain things if we need to.



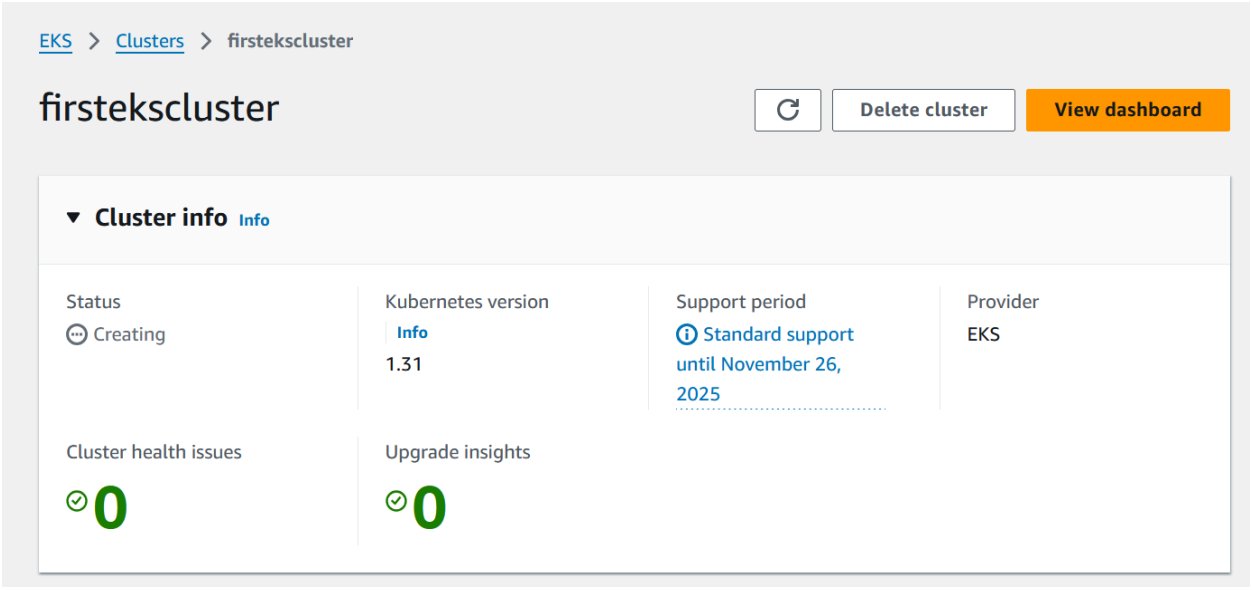
- I got the error while creating saying that one of subnet don't support creation control plane on subnet us-east-1c so I wlll remove that from my subnet list and click on create cluster again.





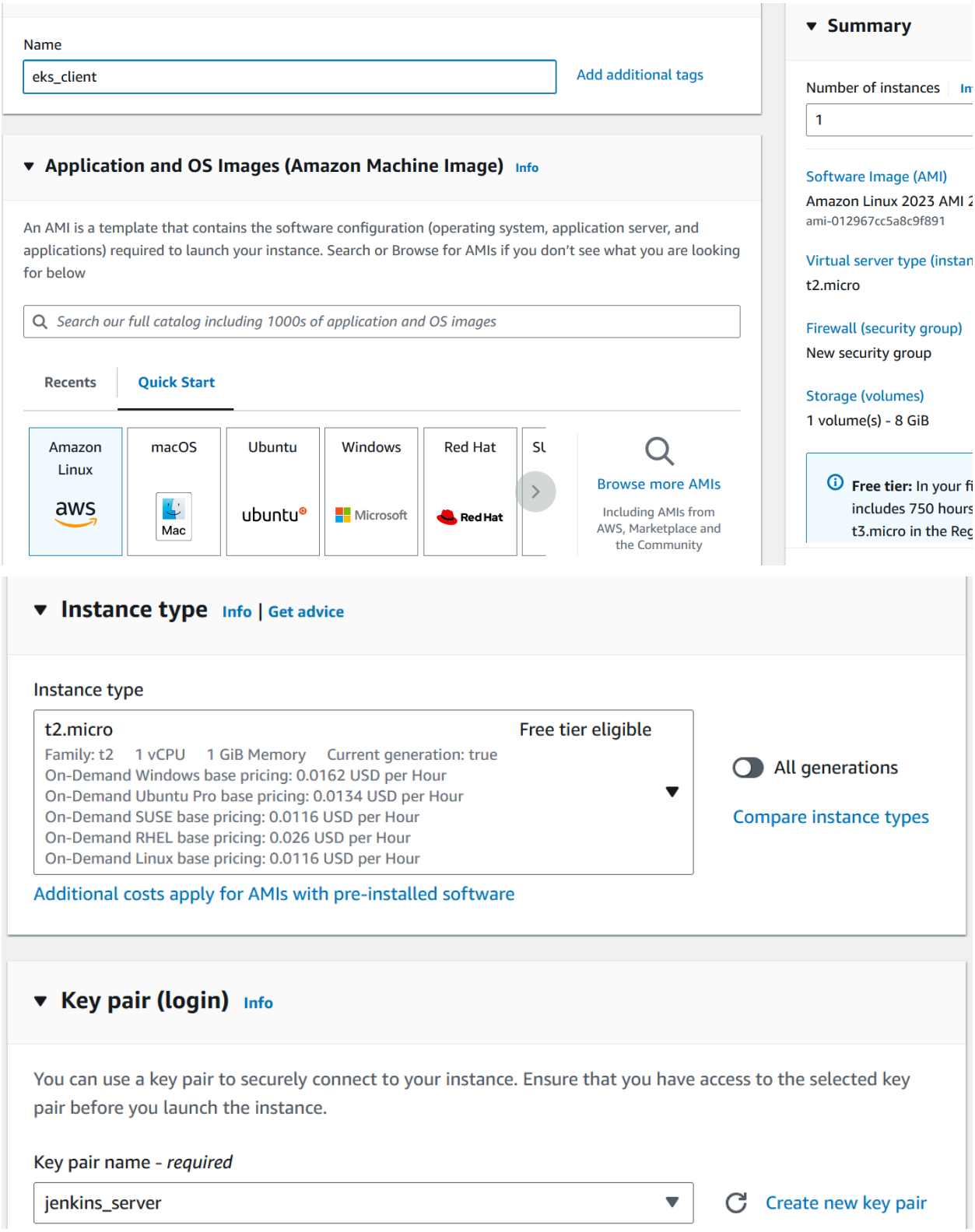
Now I only have 5 subnets, which is enough and fine.

- My first cluster is starting and it is in creation process.



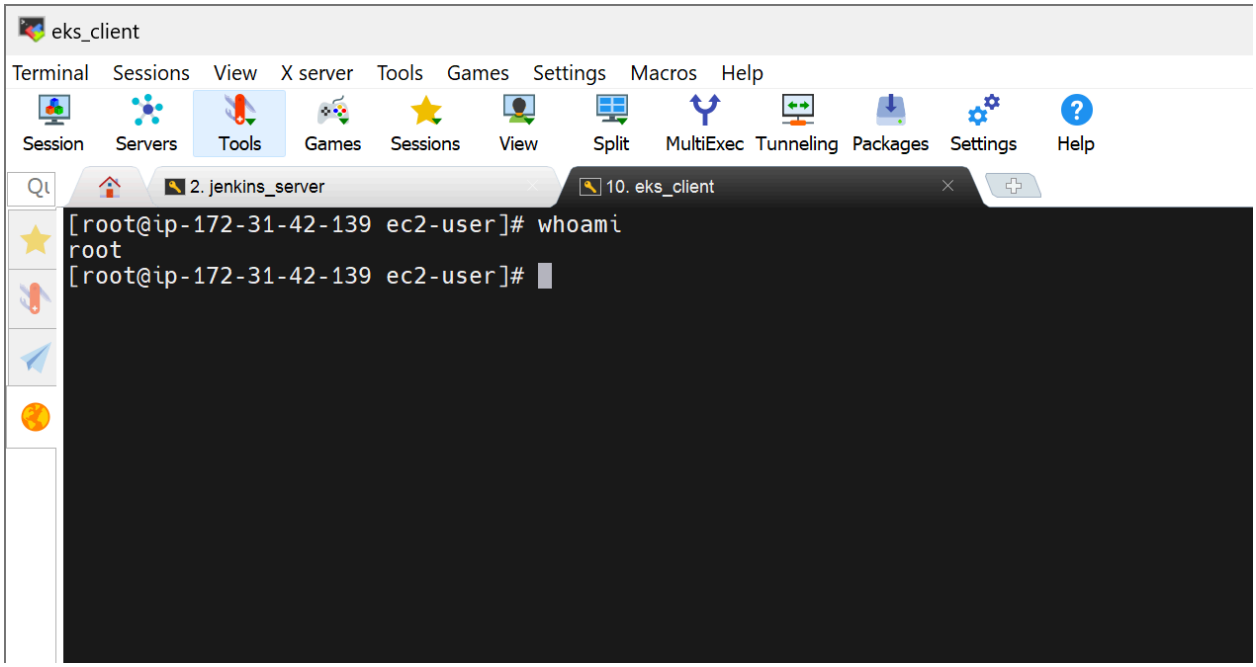
Wait until it creates it completely.

- We need to have a new instance or EKS client which can interact with the cluster so we will create one client machine which is ec2 instance.

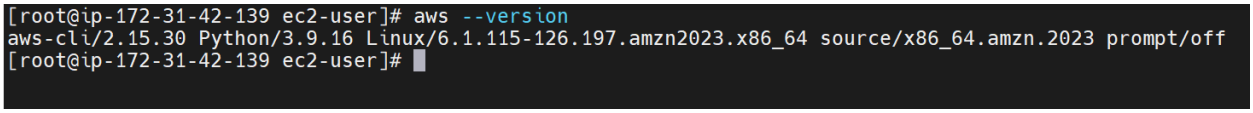


Keep everything default and lauch the instance.

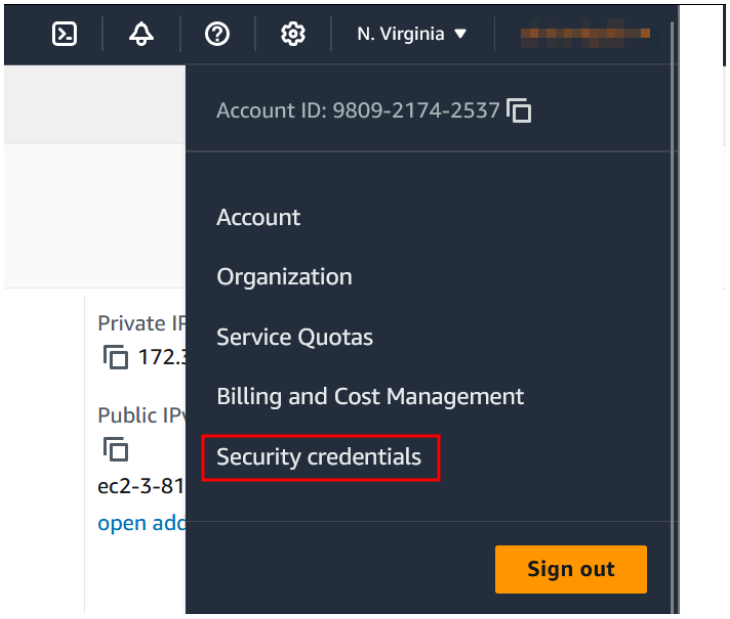
- Once the instance is running we will access it using mobaxterm which is ssh client to access the machine.



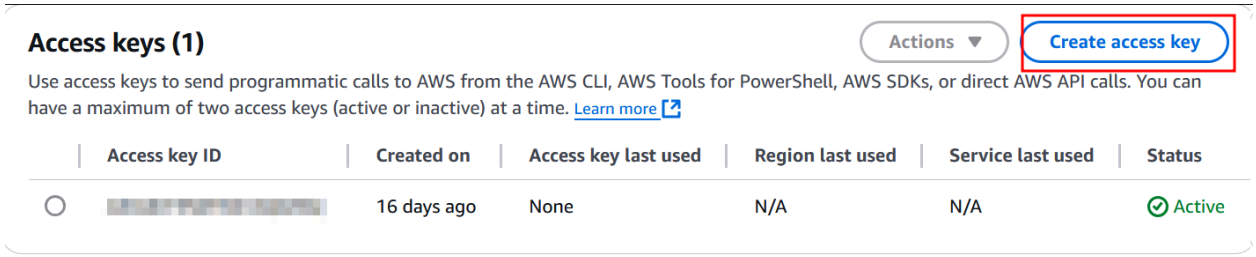
- In Amazon linux aws cli is configured by default for communicating it with aws.



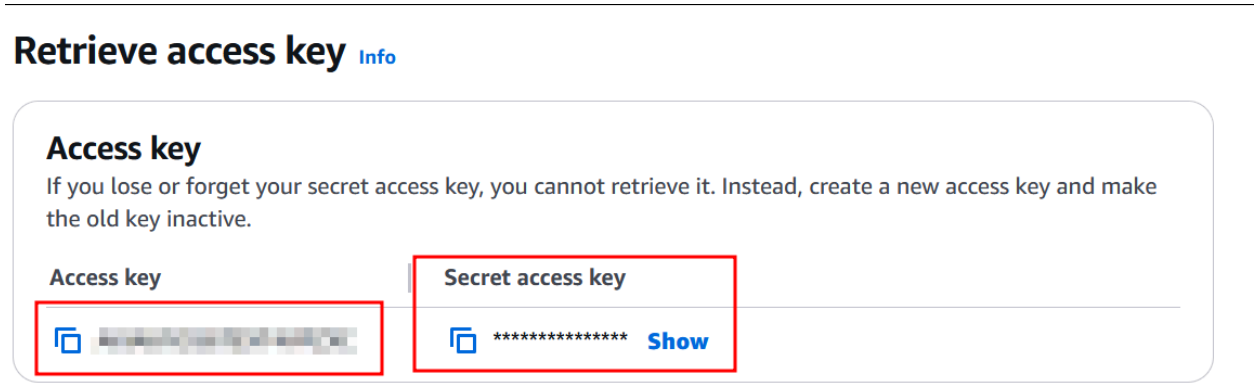
- Lets create a access key for our aws and allow this ec2 instance to communicate with our AWS account.
- Navigate to aws management console, and click on your account name and select security credentials.



- Click on create access key.



- Copying the root access key is not recommended but I am going to terminate this instanc e, but if you are using for production and important work, make sure your create a user at IAM and configure the policies and create access key for the user rather than using with the root account.
- Save the access key and secret access key for using it at ec2.



- Use AWS configure and add the access and secret access key to enable the communication between ec2 and aws account.

```
[root@ip-172-31-42-139 ec2-user]# aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-1
Default output format [None]: json
[root@ip-172-31-42-139 ec2-user]#
```

- We can check eks cluster status from ec2 instance for checking whether the ec2 is properly connected with ec2.
  aws eks --region us-east-1 describe-cluster --name firsteksclusterr --query cluster.status

```
[root@ip-172-31-42-139 ec2-user]# aws eks --region us-east-1 describe-cluster --name firsteksclusterr --query cluster.status
"ACTIVE"
[root@ip-172-31-42-139 ec2-user]#
```

  Change the cluster name parameter with your own cluster and region as well.

- We will add a config file for kubernetes here using the command.

```
[root@ip-172-31-42-139 ec2-user]# aws eks --region us-east-1 update-kubeconfig --name firsteksclusterr
Added new context arn:aws:eks:us-east-1:980921742537:cluster/firsteksclusterr to /root/.kube/config
[root@ip-172-31-42-139 ec2-user]# ls -ltra
total 12
-rw-r--r--. 1 ec2-user ec2-user 492 Jan 28  2023 .bashrc
-rw-r--r--. 1 ec2-user ec2-user 141 Jan 28  2023 .bash_profile
-rw-r--r--. 1 ec2-user ec2-user  18 Jan 28  2023 .bash_logout
drwxr-xr-x. 3 root     root      22 Nov 20 23:49 ..
drwx------. 2 ec2-user ec2-user  29 Nov 20 23:49 .ssh
drwx------. 3 ec2-user ec2-user  74 Nov 20 23:49 .
[root@ip-172-31-42-139 ec2-user]# cd ~
[root@ip-172-31-42-139 ~]# ls -ltra
total 24
-rw-r--r--.  1 root root 129 Feb  2  2023 .tcshrc
-rw-r--r--.  1 root root 100 Feb  2  2023 .cshrc
-rw-r--r--.  1 root root 429 Feb  2  2023 .bashrc
-rw-r--r--.  1 root root 141 Feb  2  2023 .bash_profile
-rw-r--r--.  1 root root  18 Feb  2  2023 .bash_logout
dr-xr-xr-x. 18 root root 237 Nov 13 18:25 ..
drwx------.  2 root root  29 Nov 20 23:49 .ssh
drwxr-xr-x.  2 root root  39 Nov 21 00:02 .aws
-rw-------.  1 root root  20 Nov 21 00:12 .lesshst
drwxr-xr-x.  2 root root  20 Nov 21 00:13 .kube
dr-xr-x---.  5 root root 144 Nov 21 00:13 .
[root@ip-172-31-42-139 ~]#
```

- We need to install kubectl, we need kubectl to interact with kubernetes API. So navigate to this website and copy the curl command based on your version of EKS cluster you have chosen while creating it.
  “https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html#linux_amd64_kubectl”

Linux (amd64)

1. Download the `kubectl` binary for your cluster's Kubernetes version from Amazon S3.
   - Kubernetes `1.31`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.2/2024-11-15/bin/linux/amd64/ku
     ```

   - Kubernetes `1.30`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.6/2024-11-15/bin/linux/amd64/ku
     ```

   - Kubernetes `1.29`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.10/2024-11-15/bin/linux/amd64/k
     ```

   - Kubernetes `1.28`

- Follow these two steps as well for adding it to path and including it in home directory.

3. Apply execute permissions to the binary.

   ```
   chmod +x ./kubectl
   ```

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of `kubectl`, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

   ```
   mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
   ```

```
[root@ip-172-31-42-139 ~]# curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.2/2024-11-15/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53.7M  100 53.7M    0     0  8413k      0  0:00:06  0:00:06 --:--:-- 9517k
[root@ip-172-31-42-139 ~]# ls
kubectl
[root@ip-172-31-42-139 ~]# chmod +x kubectl
[root@ip-172-31-42-139 ~]# mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
[root@ip-172-31-42-139 ~]#
```

```
[root@ip-172-31-42-139 ~]# kubectl version --client
Client Version: v1.31.2-eks-94953ac
Kustomize Version: v5.4.2
```

- EKSctl how to install it? Let's look for the amazon website and do the following steps.
  # Download the eksctl binary
  curl -s --location
  "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

  # Move the binary to a directory in your PATH
  sudo mv /tmp/eksctl /usr/local/bin

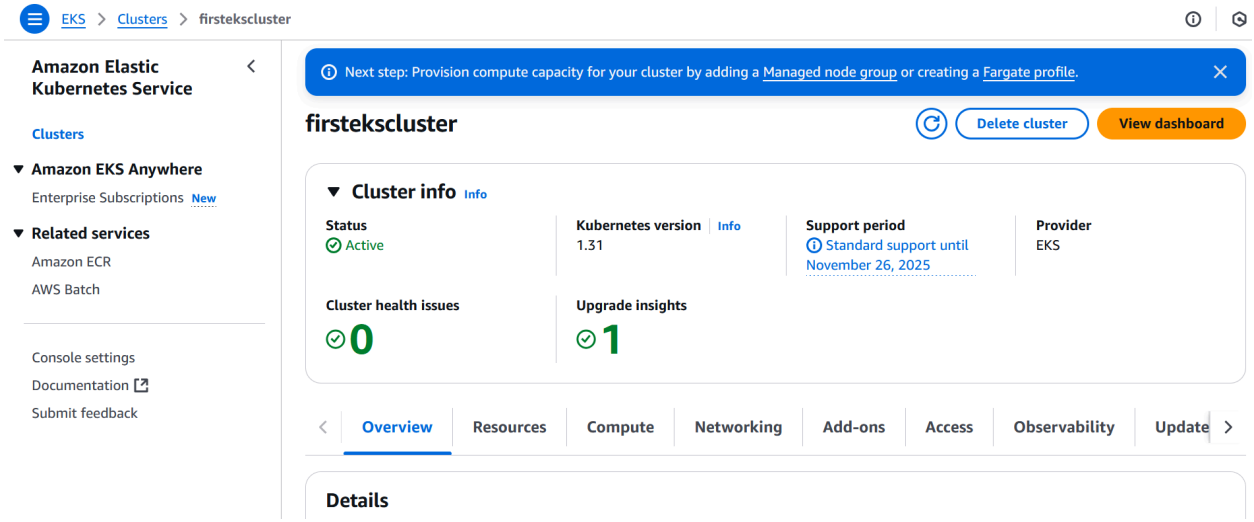
eksctl version

```
[root@ip-172-31-42-139 ~]# curl -s --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_am
d64.tar.gz" | tar xz -C /tmp
[root@ip-172-31-42-139 ~]# sudo mv /tmp/eksctl /usr/local/bin
[root@ip-172-31-42-139 ~]# eksctl version
0.194.0
```

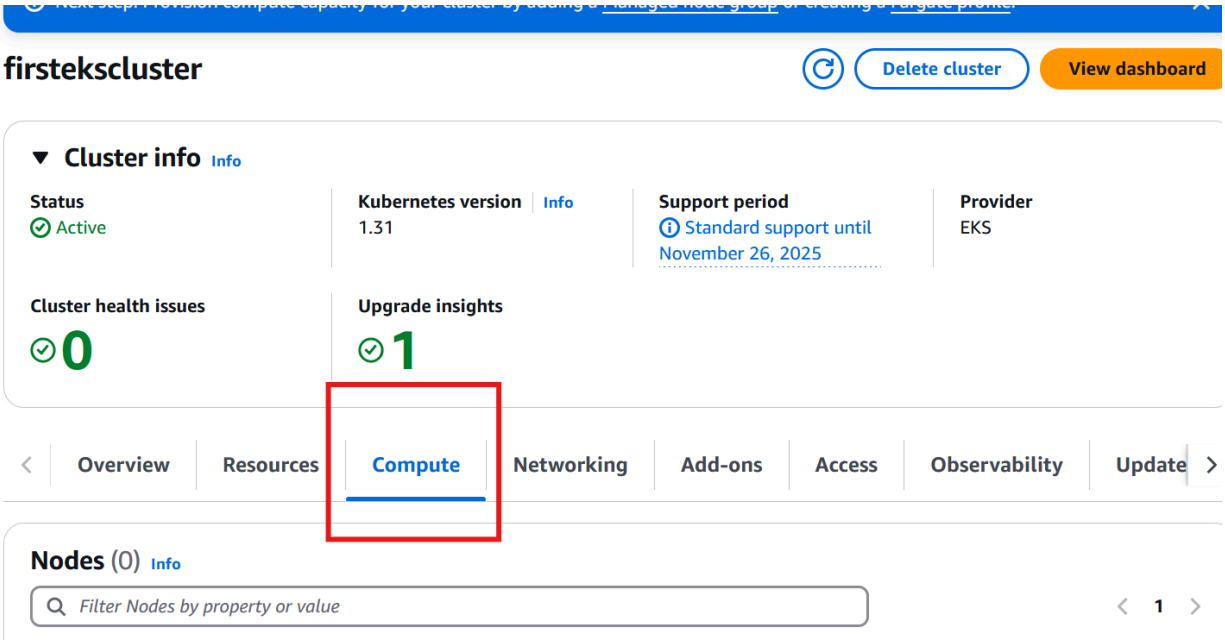
- Check whether the kubectl able to communicate with kubernetes.

```
[root@ip-172-31-42-139 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   6h4m
[root@ip-172-31-42-139 ~]#
```

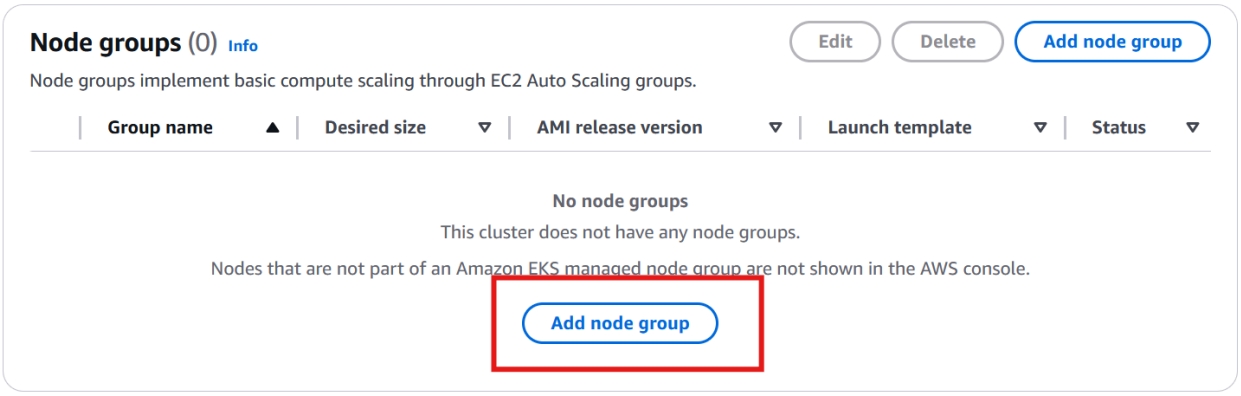
- We successfully deployed the master node, now we need to add the worker nodes and node group.
- Go to aws console and open the EKS service.



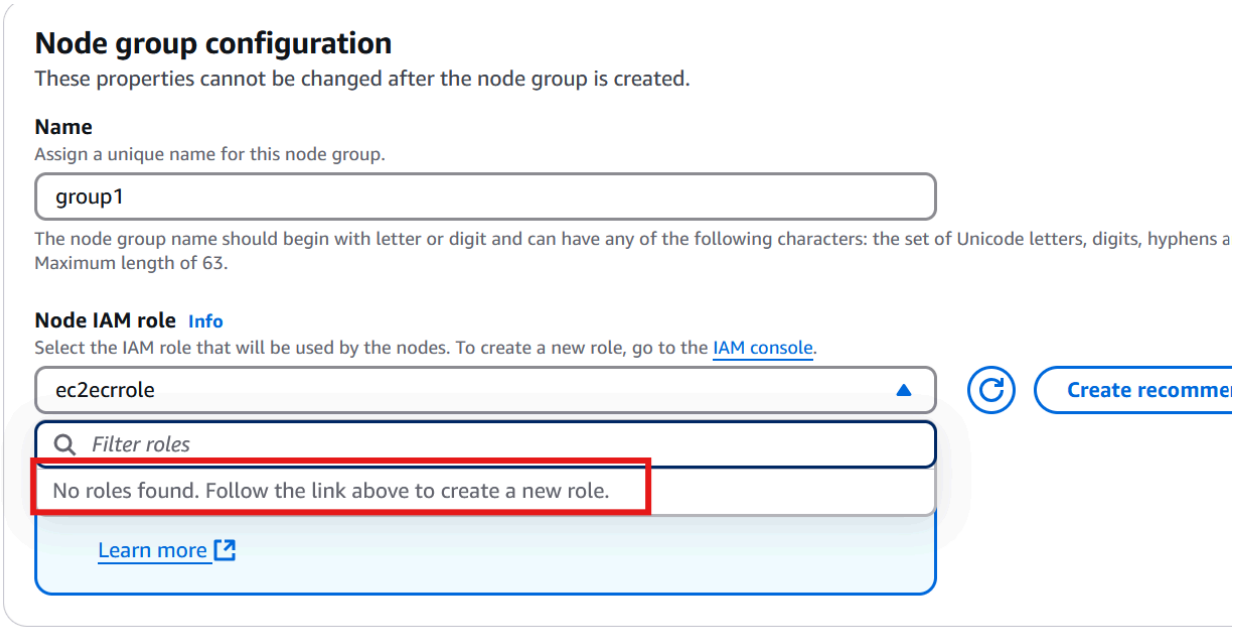
Navigate to compute



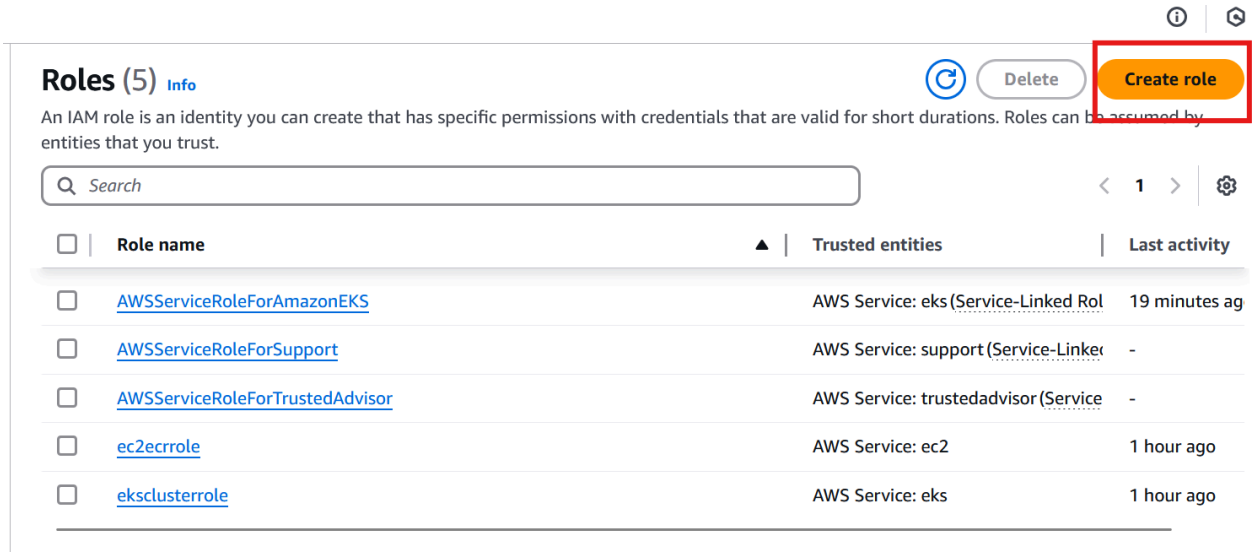
Scroll down and click on add node group for a node group where we will run the worker nodes which is ec2 instances.



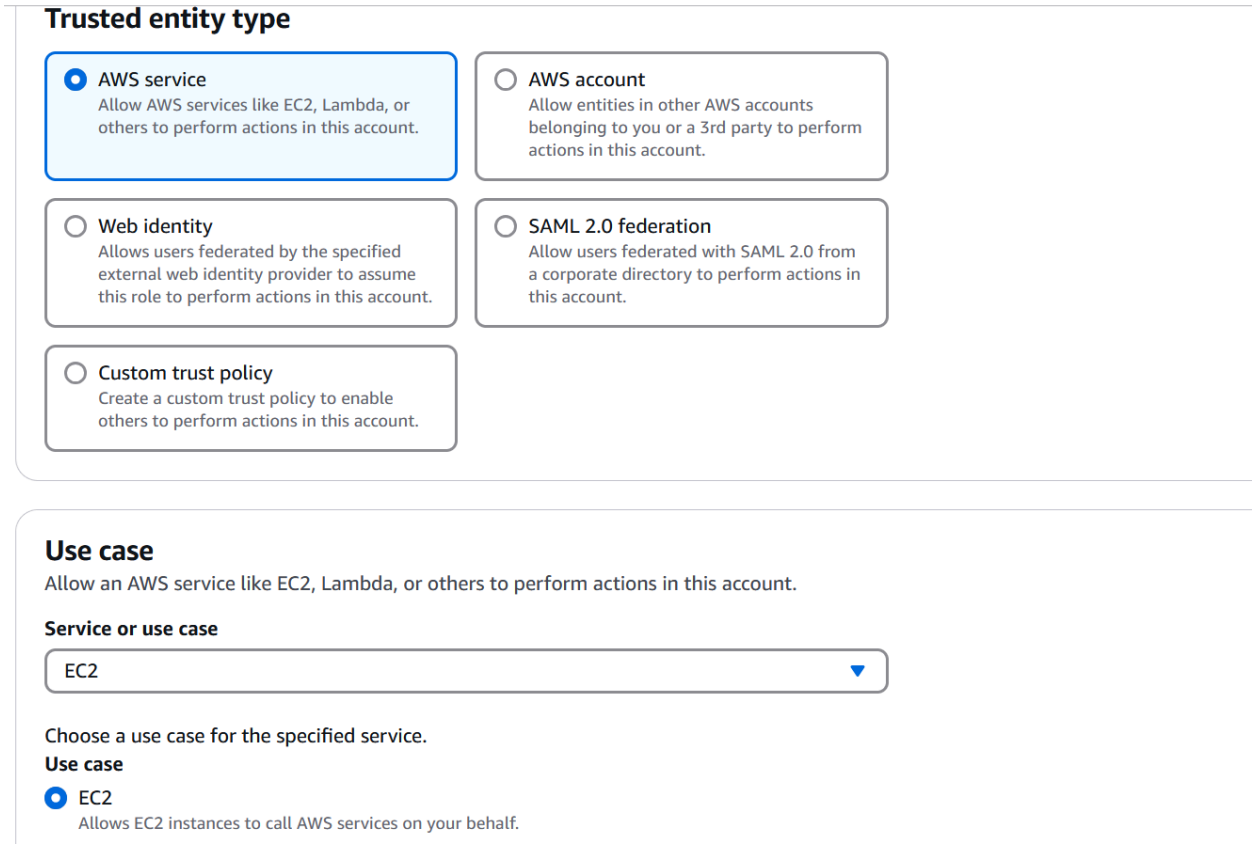
Add name to your node group and now we will need to create a IAM role for this node group.



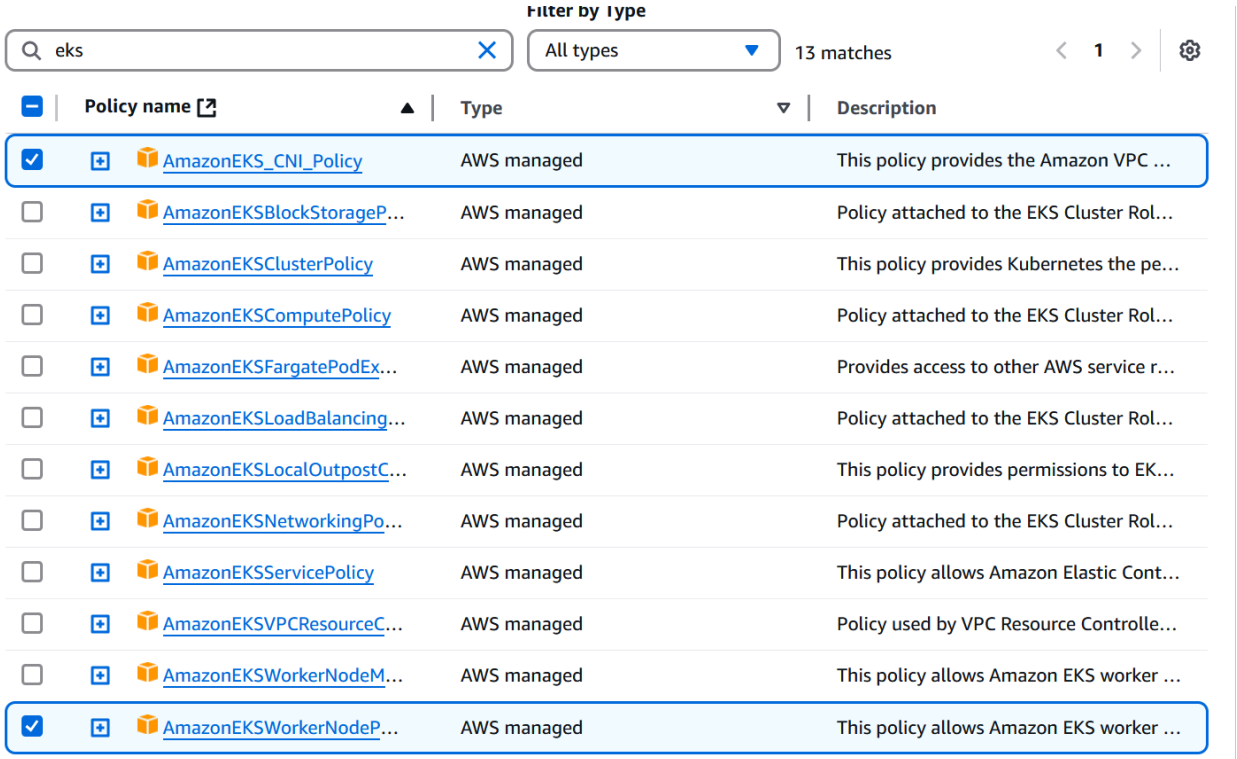
- Navigate to AWS IAM and hit create role



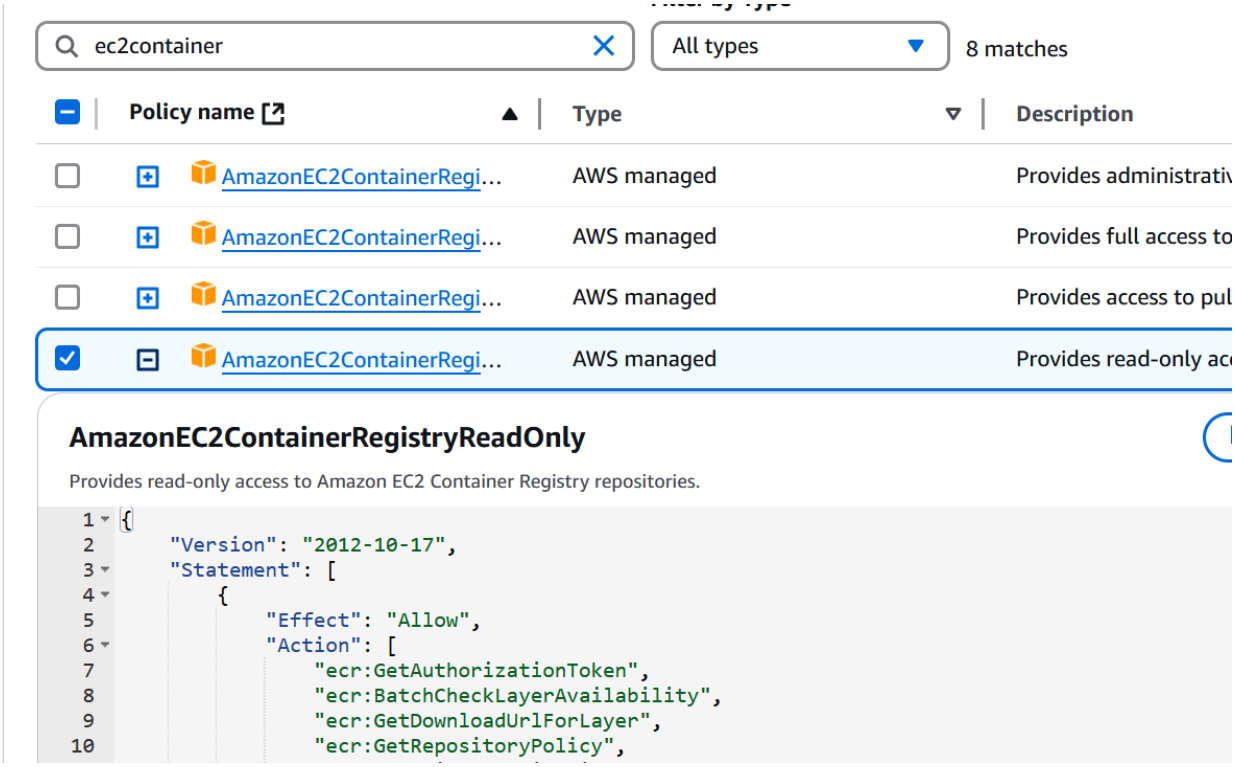
- Click on AWS service as trusted entity type and use case as EC2 as we are going to deploy EC2 as a worker nodes.



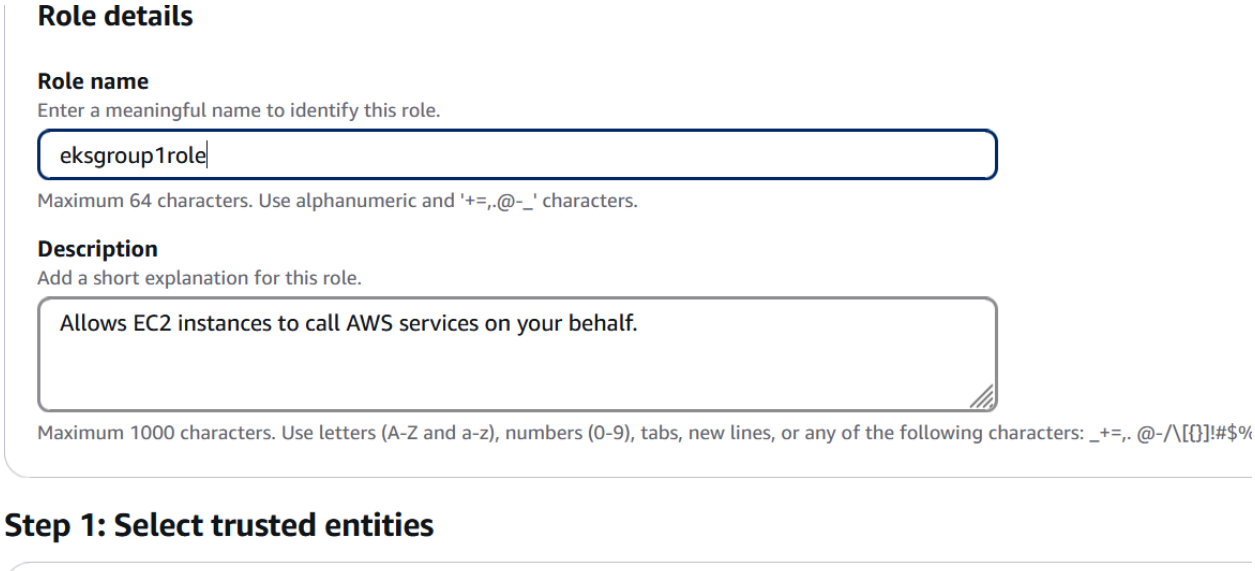
- Type ‘eks’ and add these two roles.



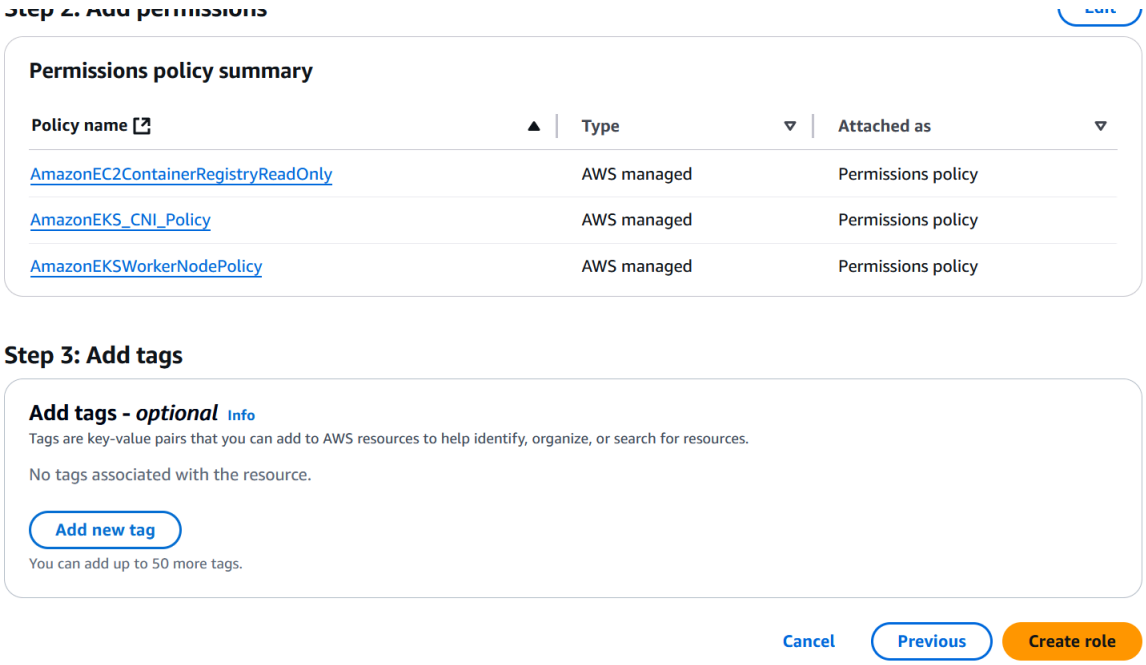
- Add this role as well for ec2 container registry read only.



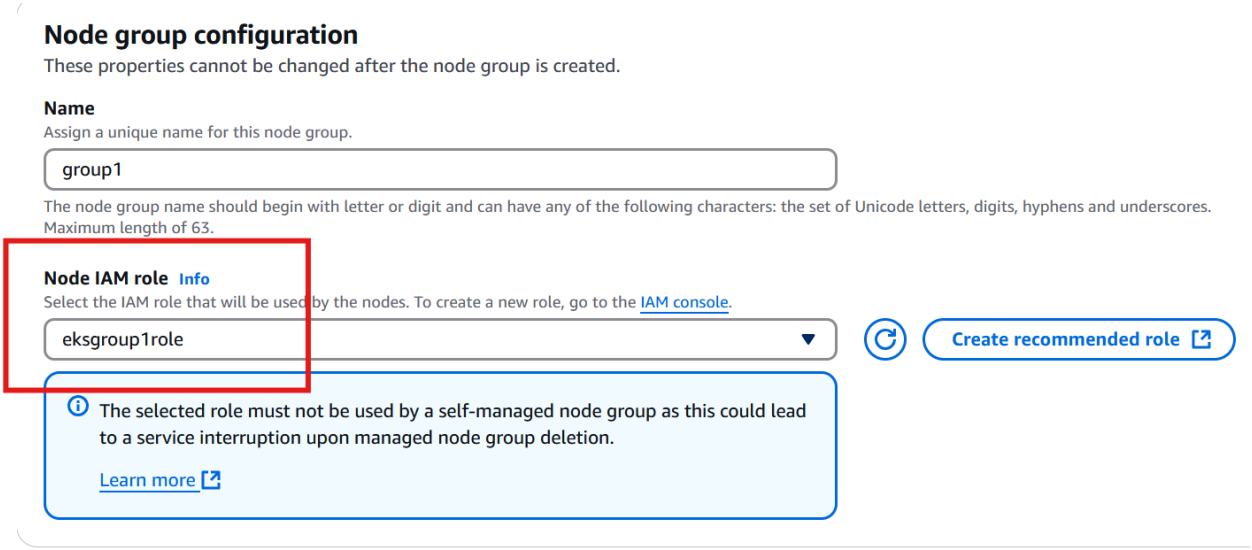
- Give a role name and create role.



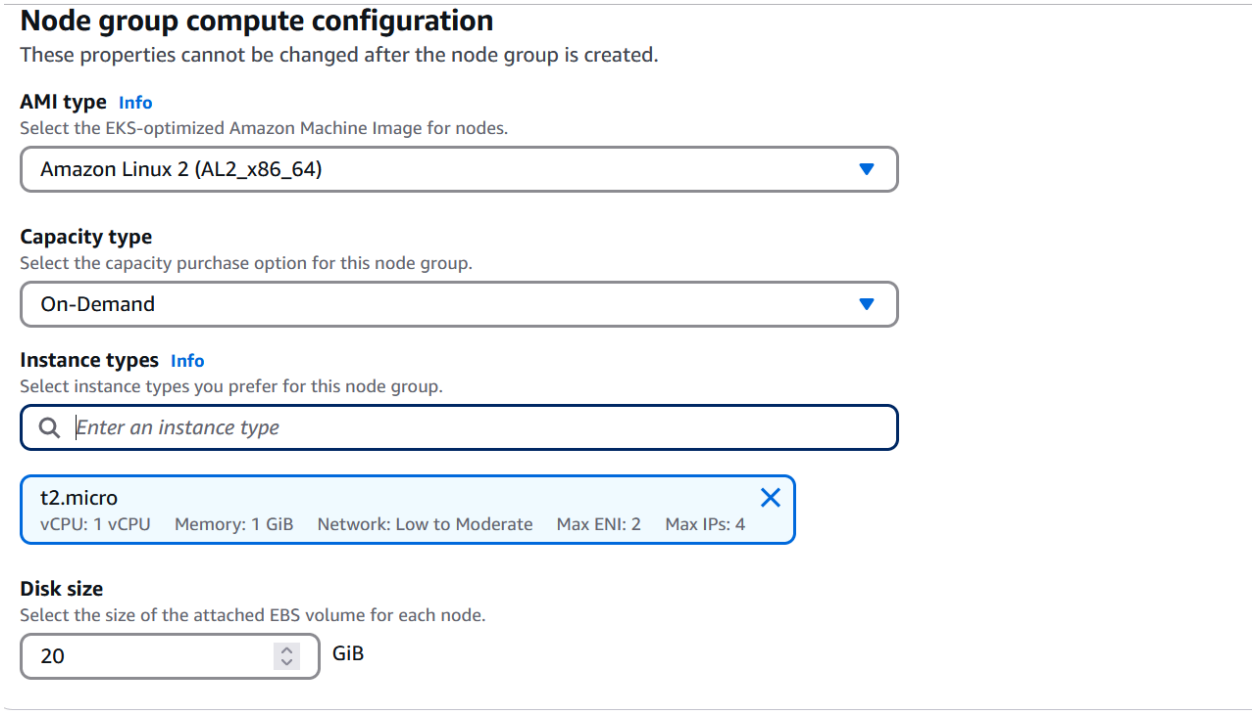
Step 1: Select trusted entities



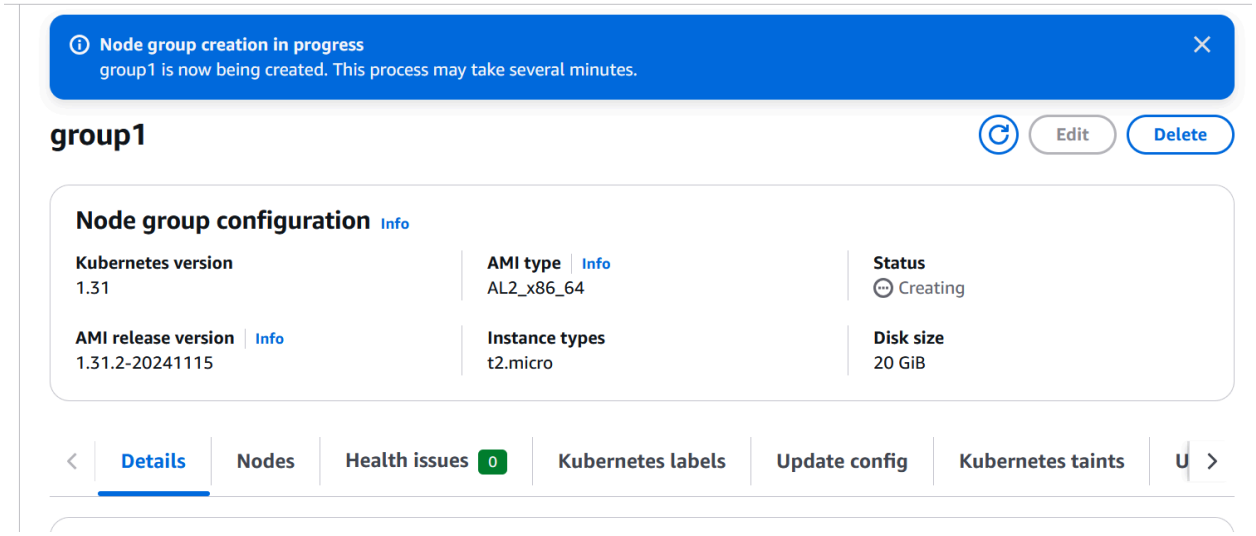
- Once the role is created add it here.



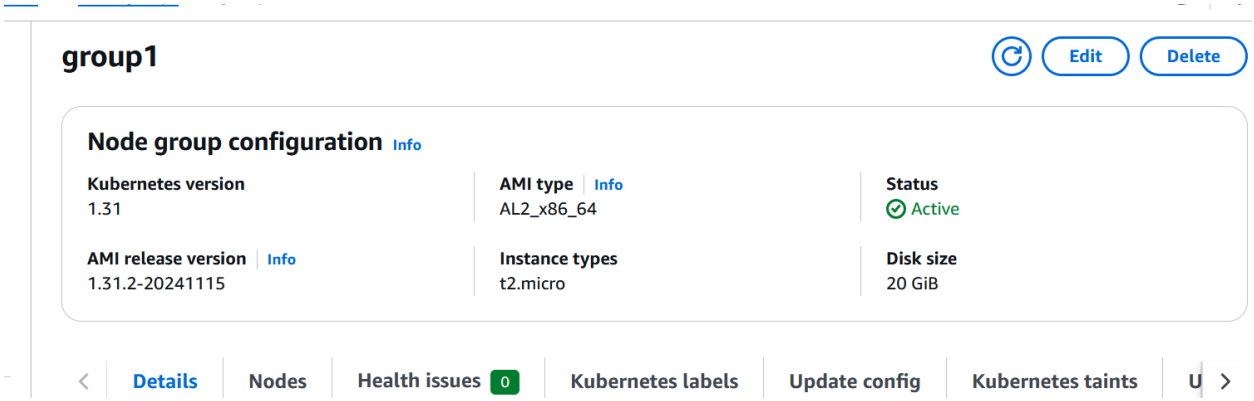
- Choose the image based on your needs and keep everything default and create the node group.



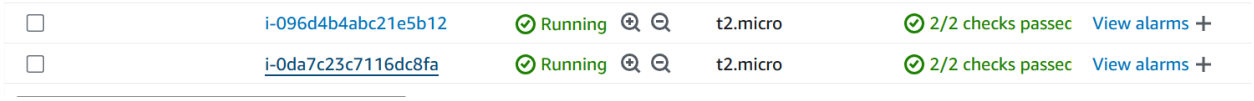
Update here, don't use t2.micro it will produce error so use t3.medium



- Once the node group is created successfully, we can see 2 more ec2 instances are running on EC2 which are worker nodes.



We can see two ec2 worker nodes are running successfully.

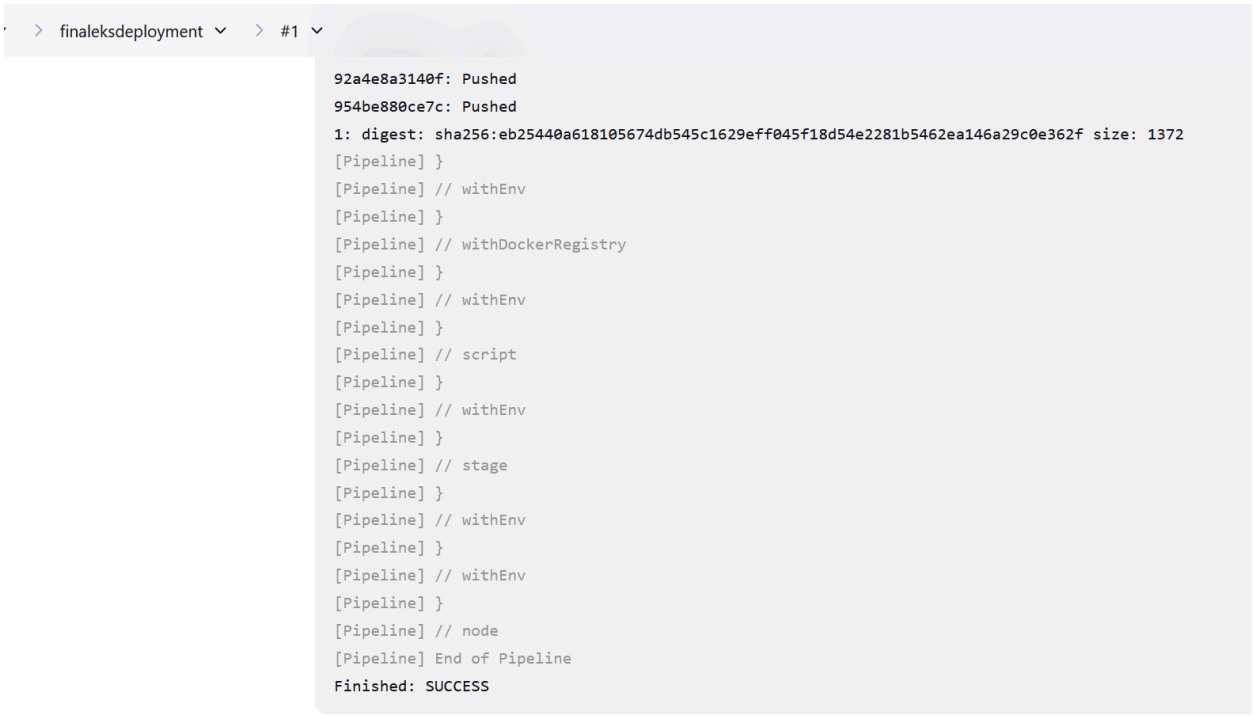



```
[root@ip-172-31-42-139 ~]# kubectl get nodes
NAME                            STATUS   ROLES    AGE   VERSION
ip-172-31-24-180.ec2.internal   Ready    <none>   26m   v1.31.2-eks-94953ac
ip-172-31-34-228.ec2.internal   Ready    <none>   26m   v1.31.2-eks-94953ac
```

- Now we will do the complete process of building the project, creating the image and adding it to our EKS cluster.
- Go the jenkins server and create a new pipeline and modify the github repo with this one, “https://github.com/Shikhar82/techstart-jenkins-deploy-to-k8-repo.git”.

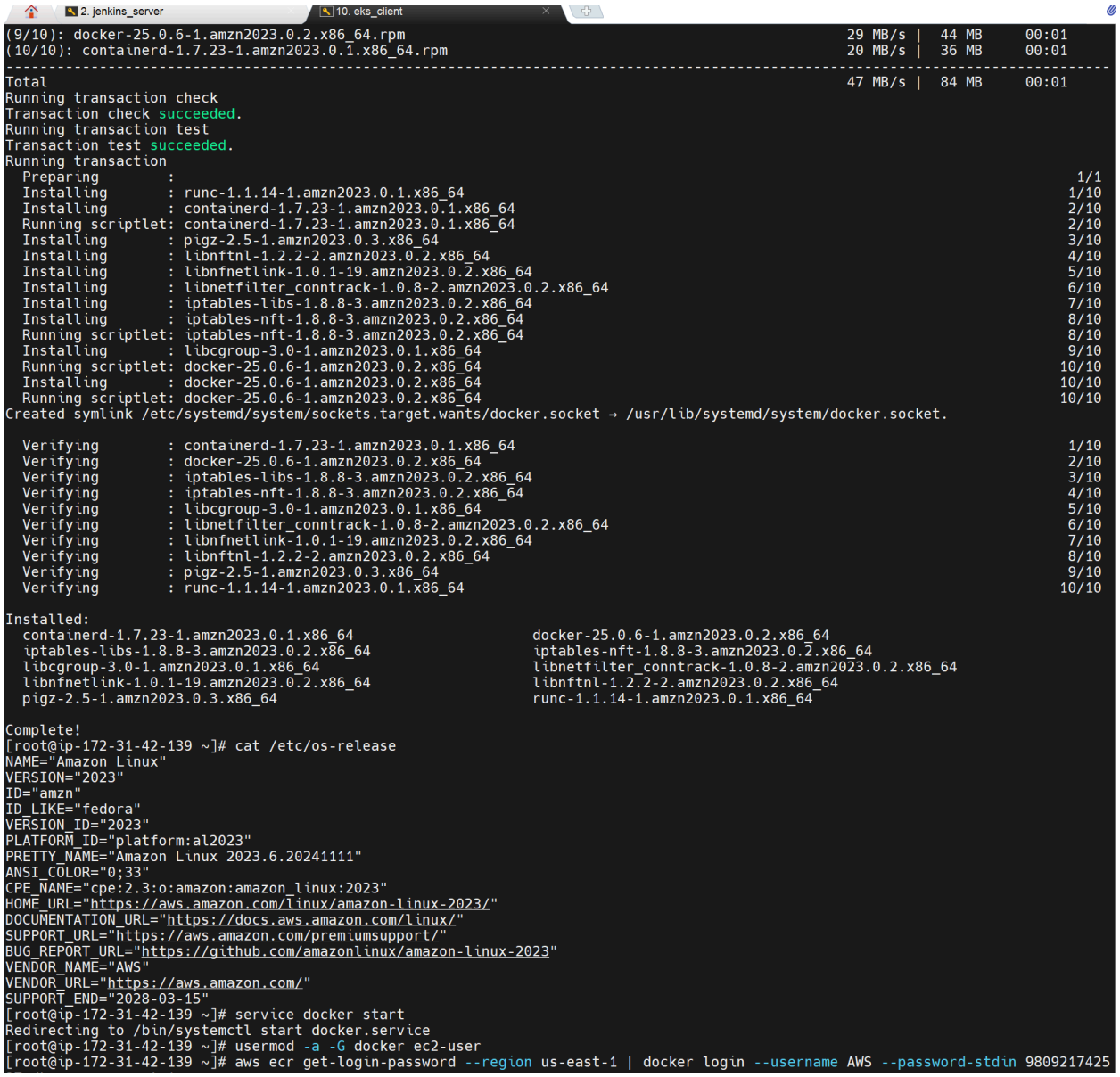
Change it to main branch as well as that github repository belongs to main branch.

Click build now and go to next step once the build is finished, It will take some time 😕



- Great, we have to focus on the .git repository, depends on your own project, where we need to have two files, one is deployment.yaml and service.yaml files. The deployment.yaml file has the URI and app information

- After this I installed the docker at eks_client before that I changed the URI on the deployment.yaml file. You will find this file at the github repository "https://github.com/Shikhar82/techstart-jenkins-deploy-to-k8-repo.git"
  For editing the deployment file, you can fork the above repository and modify the deployment file based on your URI that you find in you images at AWS ECR.

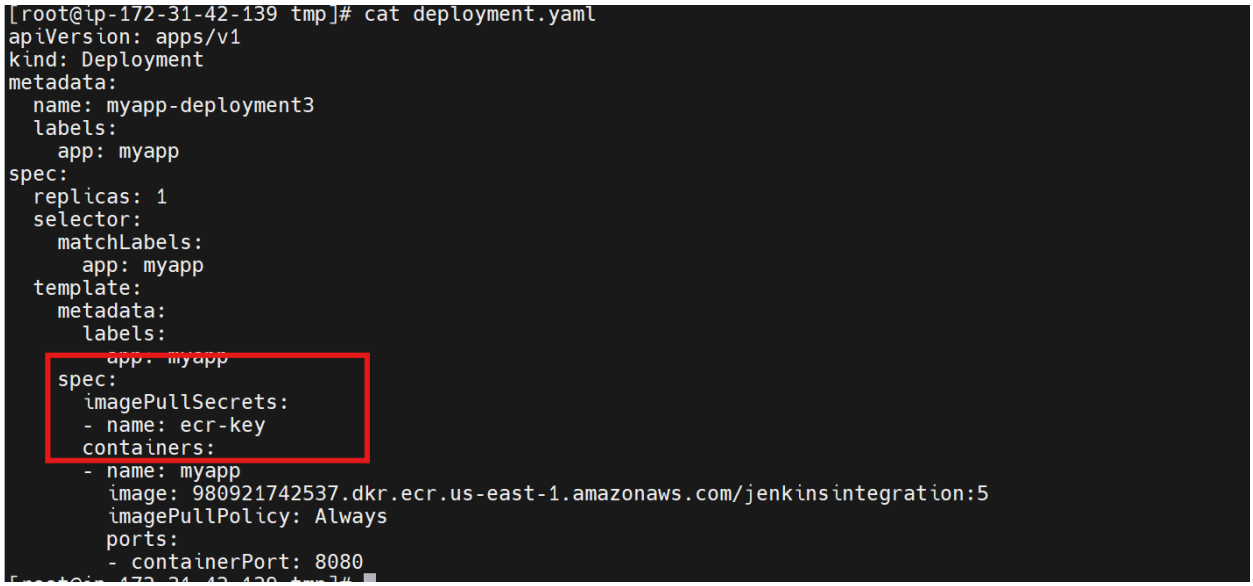


- Add ssh plugin at jenkins server dashboard after that write the pipeline using ssh where you can transfer the deployment and service.yaml files.
- Go to create pipeline syntax and give ssh values and use that for pipeline.
- I added this stage at my pipeline for transferring the files.

stage("Transferring deployment.yaml and service.yaml to EKS_client Machine"){
steps{
sshagent(['eksclientssh']) {
sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.84.17'
sh 'scp -o StrictHostKeyChecking=no /var/lib/jenkins/workspace/finaleksdeployment/deployment.yaml ec2-user@172.31.42.139:/tmp'
sh 'scp -o StrictHostKeyChecking=no

/var/lib/jenkins/workspace/finaleksdeployment/service.yaml
ec2-user@172.31.42.139:/tmp'
}
}
}
}

- We successfully transferred both files.
- Now time to create a secret key at client EKS for verification to pull the image from ECR.



```
[root@ip-172-31-42-139 tmp]# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment3
  labels:
    app: myapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      imagePullSecrets:
      - name: ecr-key
      containers:
      - name: myapp
        image: 980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration:5
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
```

- To create the secret key use the command

- Create a namepsace and after creating that execute the command for creating the secret key.
  kubectl create secret docker-registry ecr-key
  --docker-server=**980921742537.dkr.ecr.us-east-1.amazonaws.com**
  --docker-username=AWS --docker-password=$(aws ecr get-login-password)
  --namespace=securityapp

  Note: The bold text is the part of the URI of the image at ECR till amazonaws.com

- Try to run these commands just for verification whether everything is working fine.

```
[root@ip-172-31-42-139 tmp]# kubectl get nodes
NAME                          STATUS   ROLES    AGE   VERSION
ip-172-31-70-1.ec2.internal   Ready    <none>   34m   v1.31.2-eks-94953ac
ip-172-31-83-251.ec2.internal   Ready    <none>   34m   v1.31.2-eks-94953ac
[root@ip-172-31-42-139 tmp]# kubectl get namespace
NAME              STATUS   AGE
default           Active   9h
kube-node-lease   Active   9h
kube-public       Active   9h
kube-system       Active   9h
securityapp       Active   6m5s
[root@ip-172-31-42-139 tmp]# kubectl get deploy
No resources found in default namespace.
[root@ip-172-31-42-139 tmp]# kubectl create -f deployment.yaml -n securityapp
deployment.apps/myapp-deployment3 created
[root@ip-172-31-42-139 tmp]# kubectl get deploy
No resources found in default namespace.
[root@ip-172-31-42-139 tmp]# kubectl get deploy -n securityapp
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
myapp-deployment3   1/1     1            1           16s
[root@ip-172-31-42-139 tmp]# kubectl delete deploy -n securityapp
error: resource(s) were provided, but no name was specified
[root@ip-172-31-42-139 tmp]# kubectl delete deploy myapp-deployment3 -n securityapp
deployment.apps "myapp-deployment3" deleted
[root@ip-172-31-42-139 tmp]#
```

- We are going to add the stage in our pipeline to deploy the deployment.yaml and service.yaml automatically.
- Navigate to jenkins pipeline and the stages.

```
[root@ip-172-31-42-139 tmp]# whereis kubectl
kubectl: /root/bin/kubectl
```

```
stage("Executing the application"){
    steps{
        sshagent(['eksclientssh']) {
            sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.84.17 cd /tmp'
            sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.84.17 /root/bin/kubectl delete all --all -n securityapp'

        }
}
```

stage("Executing the application"){
steps{
sshagent(['eksclientssh']) {
sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.84.17 cd /tmp'
sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.84.17 /root/bin/kubectl delete all --all -n securityapp'
}

}
}

- Click on build now and run it for checking whether it is working fine. Then we will add two more lines.
- If kubectl is on the root directory you will get error as the permission is denied, so make sure you transfer the kubectl to ec2-user through the following code step.

sudo su - ec2-user
mkdir -p ~/bin
exit
sudo mv /root/bin/kubectl /home/ec2-user/bin/
sudo chown ec2-user:ec2-user /home/ec2-user/bin/kubectl
sudo chmod +x /home/ec2-user/bin/kubectl
echo 'export PATH=$PATH:/home/ec2-user/bin' >> ~/.bashrc
source ~/.bashrc
kubectl version --client

Once you run these commands you have successfully transferred the kubectl.

Add this pipeline code to your overall pipeline:

```
pipeline {
      agent any
      tools {
      maven 'maven-3.9.9'
      }
      environment {
      registry =
"980921742537.dkr.ecr.us-east-1.amazonaws.com/jenkinsintegration"
      registryCredential = 'awsecr_jenkins_user'
      }
      stages {
      stage ("Checking out the git project") {
            steps {
                  git branch: 'main', url:
'https://github.com/kp18-cpu/techstart-jenkins-deploy-to-k8-repo.git'
            }
      }

      stage ("Checking bugs on sonarcloud") {
            steps {
                  sh 'mvn clean verify sonar:sonar
-Dsonar.projectKey=devsecopskp -Dsonar.organization=devsecopskp
-Dsonar.host.url=https://sonarcloud.io
-Dsonar.token=d0d2af0eee7f1fdb21e937360084b8ece5a62cb4'
            }
      }
      // stage ("Build the package") {
      //    steps {
      //          sh 'mvn clean package'
      //    }
```

```
      // }

      stage ("Run SCA scan and analysis using snyk"){
            steps {
                  withCredentials([string(credentialsId: 'SNYK_TOKEN',
variable: 'SNYK_TOKEN')]) {
                  sh 'mvn snyk:test -fn'
                  }
            }
      }
      stage('Building our image') {
            steps{
                  script {
                  dockerImage = docker.build registry + ":$BUILD_NUMBER"
                  }
            }
      }
      stage('Deploy our image') {
            steps{
                  script {
                  docker.withRegistry("http://" + registry,
"ecr:us-east-1:" + registryCredential ) {
                  dockerImage.push()
                  }
                  }
            }
      }
      stage("Transferring deployment.yaml and service.yaml to EKS_client
Machine"){
            steps{
                  sshagent(['eksclientssh']) {
                  sh 'ssh -o StrictHostKeyChecking=no
ec2-user@172.31.84.17'
                  sh 'scp -o StrictHostKeyChecking=no
/var/lib/jenkins/workspace/finaleksdeployment/deployment.yaml
ec2-user@172.31.37.81:/tmp'
                  sh 'scp -o StrictHostKeyChecking=no
/var/lib/jenkins/workspace/finaleksdeployment/service.yaml
ec2-user@172.31.37.81:/tmp'
                  }
            }
      }
      stage("Executing the application"){
```

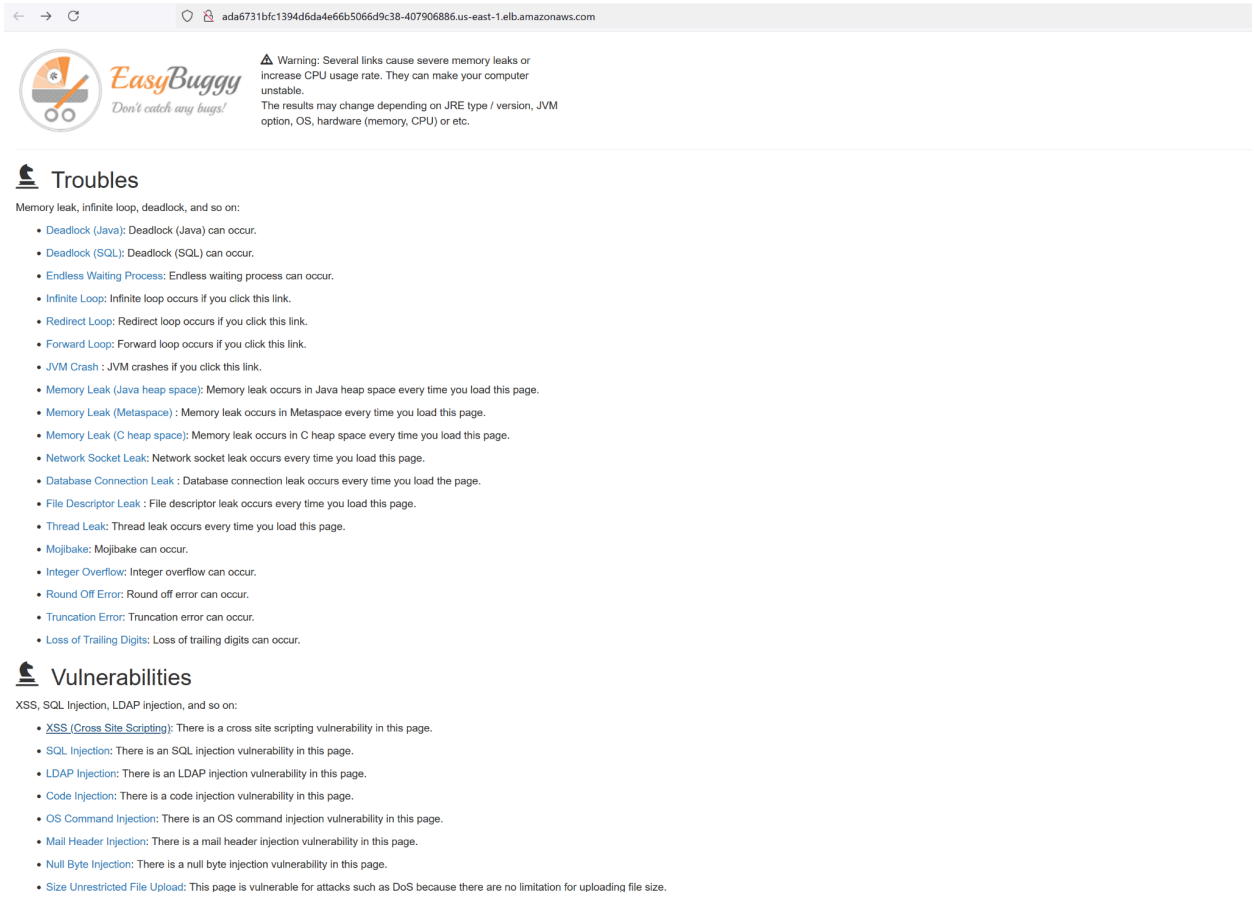
```
            steps{
                  sshagent(['eksclientssh']) {
                  sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.37.81
cd /tmp'
                  sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.37.81
/home/ec2-user/bin/kubectl delete all --all -n securityapp'
                  sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.37.81
/home/ec2-user/bin/kubectl create -f /tmp/deployment.yaml -n securityapp'
                  sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.37.81
/home/ec2-user/bin/kubectl create -f /tmp/service.yaml -n securityapp'
                  }

            }
      }
      }

}
```

172.21.37.81 is the private IP eks_client machine.
Where we try to deploy the deployment.yaml and service.yaml files to run on the cluster.

- We can verify that our hosted application is running on our aws cluster.

Copy the external ip and paste it on the browser to check whether it is working properly.

- We successfully created and hosted our application on the AWS Kubernetes service.



I am excited it is time to integrate the Zap tool which is a dynamic testing tool used for testing the deployed application. We are going to analyze the above-listed application and look for vulnerabilities.

What is Dynamic Testing?
Dynamic Testing is a software testing methodology that involves executing a program or application to identify defects. It focuses on the behavior of the software during execution, ensuring that it works as intended in a live environment. Unlike static testing, which reviews code without running it, dynamic testing involves actual execution.

What is Zap Tool ?
OWASP ZAP (Zed Attack Proxy) is an open-source web application security testing tool developed by the Open Web Application Security Project (OWASP). It is widely used for dynamic application security testing (DAST) to identify vulnerabilities in web applications.

Key Features of ZAP

1. Intercepts and inspects HTTP(S) traffic between the client and server.
2. Performs automated scans for security vulnerabilities.
3. Includes tools for manual security testing (e.g., fuzzing, forced browsing).
4. Integrates with CI/CD pipelines for automated security testing.
5. Provides an easy-to-use GUI, API, and command-line interface.

Now let's start integrating the zap with our application for the dynamic testing.

- Navigate to eks_client VM and install zap.
- GO to /tmp directory and create a zap directory through mkdir zap.
- Before installing make sure you have java installed on your eks_client.
- To install java:
  wget https://corretto.aws/downloads/latest/amazon-corretto-11-x64-linux-jdk.rpm
  sudo yum localinstall -y amazon-corretto-11-x64-linux-jdk.rpm
  java -version
  Once installed:
  readlink -f $(which java)
  export JAVA_HOME=/usr/lib/jvm/java-11-amazon-corretto export PATH=$JAVA_HOME/bin:$PATH
  echo 'export JAVA_HOME=/usr/lib/jvm/java-11-amazon-corretto' >> ~/.bashrc echo 'export PATH=$JAVA_HOME/bin:$PATH' >> ~/.bashrc
  source ~/.bashrc

```
[ec2-user@ip-172-31-37-81 zap]$ java --version
openjdk 11.0.25 2024-10-15 LTS
OpenJDK Runtime Environment Corretto-11.0.25.9.1 (build 11.0.25+9-LTS)
OpenJDK 64-Bit Server VM Corretto-11.0.25.9.1 (build 11.0.25+9-LTS, mixed mode)
[ec2-user@ip-172-31-37-81 zap]$ readlink -f $(which java)
/usr/lib/jvm/java-11-amazon-corretto/bin/java
[ec2-user@ip-172-31-37-81 zap]$ export JAVA_HOME=/usr/lib/jvm/java-11-amazon-corretto
[ec2-user@ip-172-31-37-81 zap]$ export PATH=$JAVA_HOME/bin:$PATH
[ec2-user@ip-172-31-37-81 zap]$ echo 'export JAVA_HOME=/usr/lib/jvm/java-11-amazon-corretto' >> ~/.bashrc
[ec2-user@ip-172-31-37-81 zap]$ echo 'export PATH=$JAVA_HOME/bin:$PATH' >> ~/.bashrc
[ec2-user@ip-172-31-37-81 zap]$ 'export PATH=$JAVA_HOME/bin:$PATH' >> ~/.bashrc
[ec2-user@ip-172-31-37-81 zap]$
-bash: export PATH=$JAVA_HOME/bin:$PATH: No such file or directory
-bash: [ec2-user@ip-172-31-37-81: command not found
[ec2-user@ip-172-31-37-81 zap]$ source ~/.bashrc
```

- After that install zap using wget command:
  wget
  https://github.com/zaproxy/zaproxy/releases/download/v2.15.0/ZAP_2_15_0_unix.sh
  chmod +x ZAP.sh (change name accordingly)
  ./zap.sh -y (change name accordingly)

```
[ec2-user@ip-172-31-37-81 zap]$ sudo ./ZAP_2_15_0_unix.sh
Starting Installer ...
This will install Zed Attack Proxy on your computer.
OK [o, Enter], Cancel [c]
o
Click Next to continue, or Cancel to exit Setup.
Please read the following License Agreement. You must accept the terms of
this agreement before continuing with the installation.


Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

[Enter]

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.
```

Keep everything default and complete the installation.

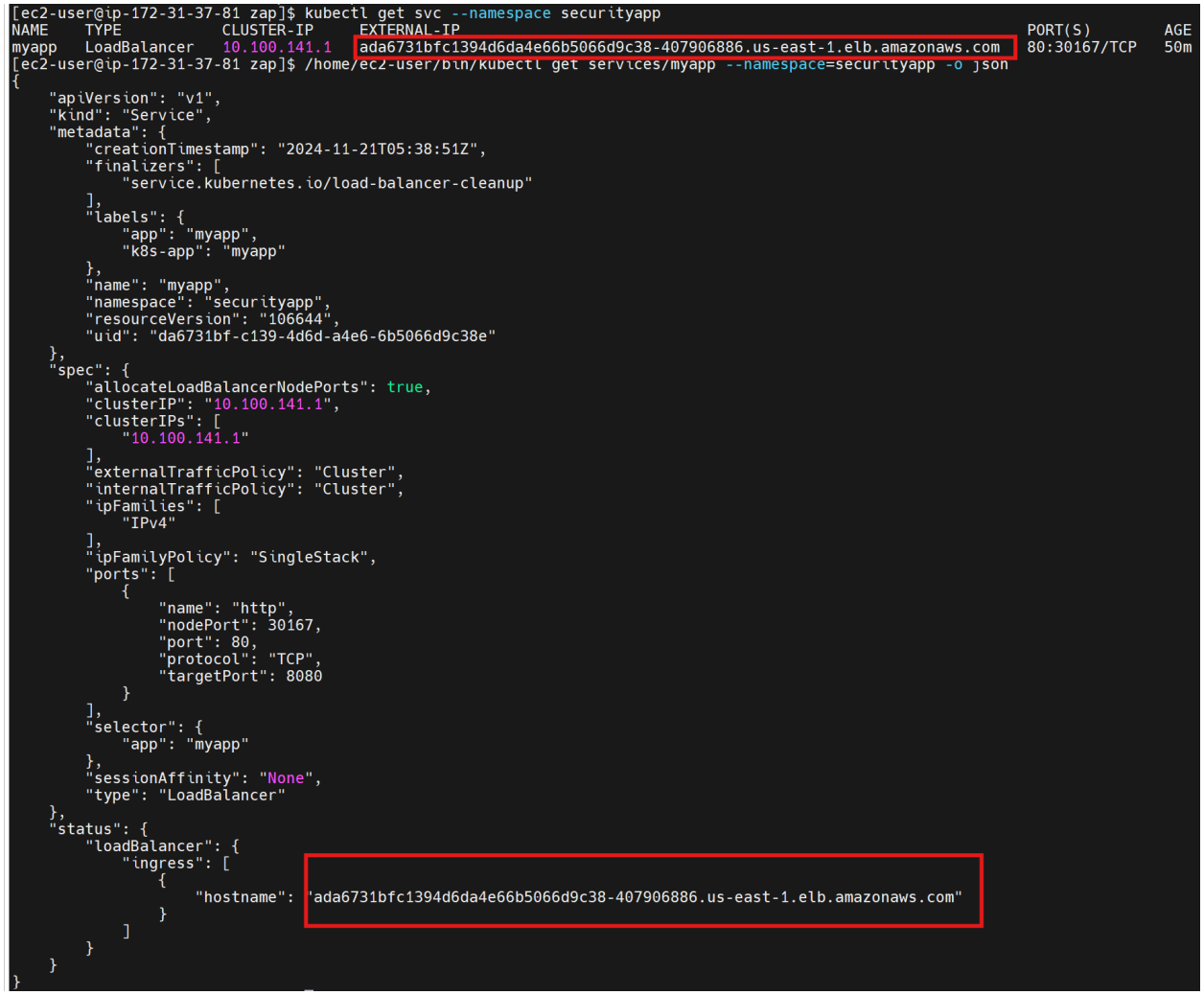
```
[ec2-user@ip-172-31-37-81 zap]$ zap.sh
Found Java version 11.0.25
Available memory: 949 MB
Using JVM args: -Xmx237m
1654 [main] INFO  org.parosproxy.paros.Constant - Copying default configuration to /home/ec2-user/.ZAP/config.xml
1879 [main] INFO  org.parosproxy.paros.Constant - Creating directory /home/ec2-user/.ZAP/session
1880 [main] INFO  org.parosproxy.paros.Constant - Creating directory /home/ec2-user/.ZAP/dirbuster
1880 [main] INFO  org.parosproxy.paros.Constant - Creating directory /home/ec2-user/.ZAP/fuzzers
1880 [main] INFO  org.parosproxy.paros.Constant - Creating directory /home/ec2-user/.ZAP/plugin
2082 [main] INFO  org.zaproxy.zap.GuiBootstrap - ZAP 2.15.0 started 21/11/2024, 06:26:21 with home: /home/ec2-user/.ZAP/ cores: 1
maxMemory: 230 MB
2084 [main] FATAL org.zaproxy.zap.GuiBootstrap - ZAP GUI is not supported on a headless environment.
Run ZAP inline or in daemon mode, use -help command line argument for more details.
ZAP GUI is not supported on a headless environment.
Run ZAP inline or in daemon mode, use -help command line argument for more details.
[ec2-user@ip-172-31-37-81 zap]$
```

Zap is successfully installed.

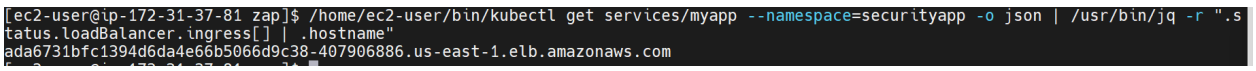
- We can see our url running on the container like this.

```
[ec2-user@ip-172-31-37-81 zap]$ kubectl get svc --namespace securityapp
NAME    TYPE           CLUSTER-IP     EXTERNAL-IP                                                               PORT(S)        AGE
myapp   LoadBalancer   10.100.141.1   ada6731bfc1394d6da4e66b5066d9c38-407906886.us-east-1.elb.amazonaws.com   80:30167/TCP   50m
[ec2-user@ip-172-31-37-81 zap]$
```

- We need to retrieve that url in which our app is running deployed on the cluster. Time to think how to retrieve this url



- Using this command we can retrieve the url dynamically, Make sure you change the values of –namespace, in our case its “securityapp”.
  /home/ec2-user/bin/kubectl get services/myapp --namespace=securityapp -o json | /usr/bin/jq -r ".status.loadBalancer.ingress[] | .hostname



If you have any error because of jq (yum install jq).

- The url is dynamic so we are try to add dynamic ways to bind it to the url and get the analysis through zap.
  The script of complete zap is here:
  /usr/local/bin/zap.sh -cmd -quickurl http://$(/home/ec2-user/bin/kubectl get services/myapp --namespace=securityapp -o json | /usr/bin/jq -r ".status.loadBalancer.ingress[] | .hostname") -quickprogress -quickout /tmp/zap_report.html

Note: Very Important code above for integrating zap.

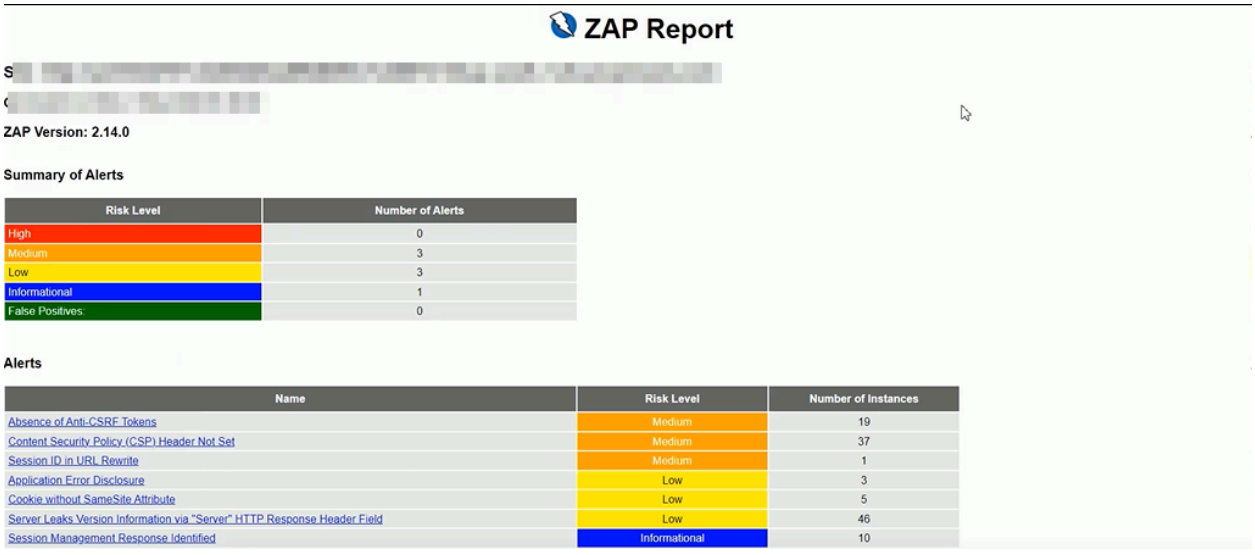
- We can integrate this script into the pipeline for automation, but I just run it on the terminal and we will get the report in the form of an HTML file.
- The testing will run for a while so have patience and we will get the file successfully.

```
[ec2-user@ip-172-31-37-81 zap]$ /usr/local/bin/zap.sh -cmd -quickurl http://$(/home/ec2-user/bin/kubectl get services/myapp --nam
space=securityapp -o json | /usr/bin/jq -r ".status.loadBalancer.ingress[] | .hostname") -quickprogress -quickout /tmp/zap_report
html
Found Java version 11.0.25
Available memory: 949 MB
Using JVM args: -Xmx237m
Accessing URL
Using traditional spider
[=======             ] 36% -
```

It's going to be 50 min, and the progress is still at 73 percent.

```
[ec2-user@ip-172-31-37-81 zap]$ /usr/local/bin/zap.sh -cmd -quickurl http://$(/home/ec2-user/bin/kubectl get services/myapp --
space=securityapp -o json | /usr/bin/jq -r ".status.loadBalancer.ingress[] | .hostname") -quickprogress -quickout /tmp/zap_rep
html
Found Java version 11.0.25
Available memory: 949 MB
Using JVM args: -Xmx237m
Accessing URL
Using traditional spider
[==============      ] 73% /
```

- Once the .html file is saved in the /tmp folder look for it.
- Time to transfer the file.
- cd to the /tmp folder where our file will be stored.
- Start the Python server, before that make sure you have Python installed.
- python3 -m http.server 8080
- Go to the web browser and access the server using the IP address and port number and you will see the file. You can download it into your local system.
- Click on zap_report.html and view the report of vulnerabilities and in-depth report.



ZAP Report

S[redacted]

ZAP Version: 2.14.0

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	3
Informational	1
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Absence of Anti-CSRF Tokens	Medium	19
Content Security Policy (CSP) Header Not Set	Medium	37
Session ID in URL Rewrite	Medium	1
Application Error Disclosure	Low	3
Cookie without SameSite Attribute	Low	5
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	46
Session Management Response Identified	Informational	10

This is just an overview of the report shared here, we can explore more of it and use it for further testing and reducing the bugs and code smell.

DEVSECOPS has so much and lot to learn and explore, especially playing around with different options on the AWS.

I want to sincerely thank Mr. Verma for his great guidance in learning and being on the path of DEVSECOPS.

Glad you guys learnt through this document.