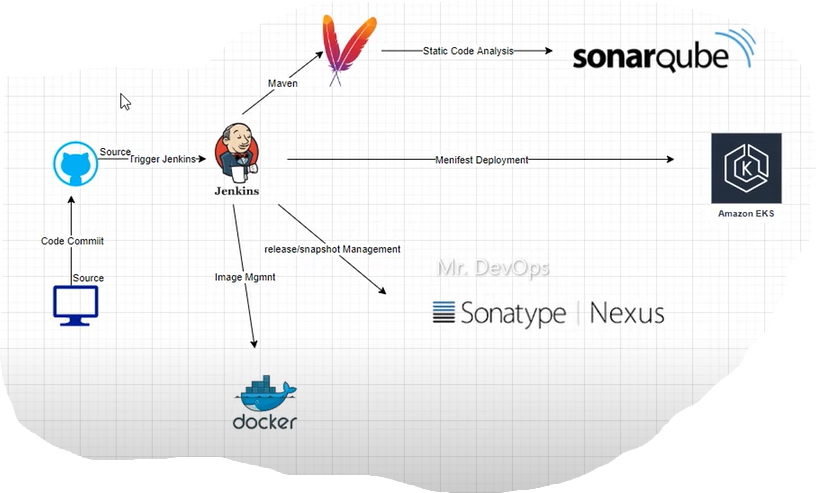
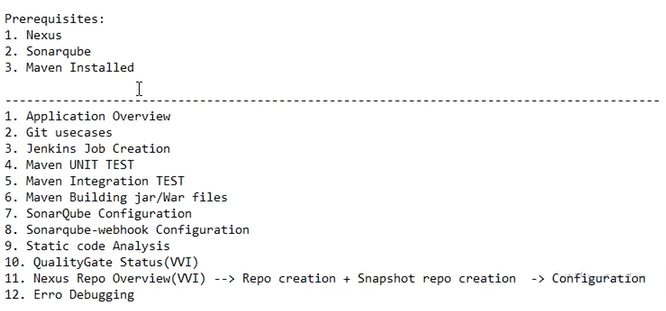
# End to End DevOps CI/CD Project | Git | Jenkins | Nexus | SonarQube | Docker | EKS





GitHub: https://github.com/sulbiraj06/counter-app-k8s.git

Installed and Configured

Jenkins

SonarQube

Nexus

Create Jenkins Job

stage('Static Code Analysis'){

             steps{

                script{

                  withSonarQubeEnv(credentialsId: 'sonar-api-key') {

                    sh 'mvn clean package sonar:sonar'

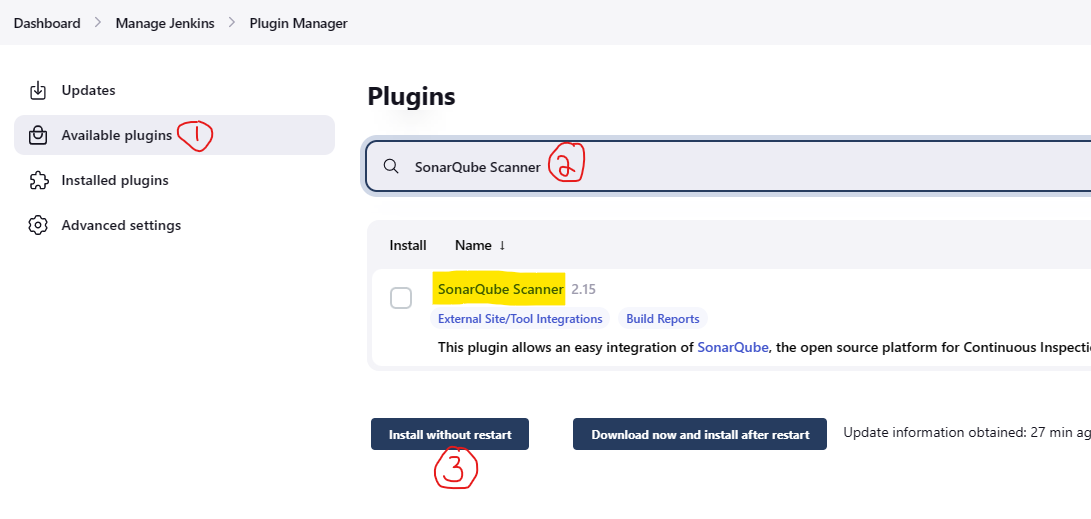
                    }

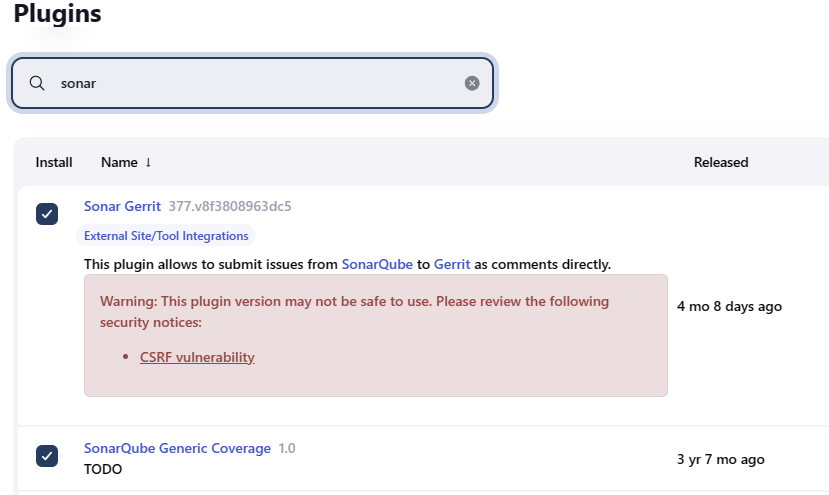
                }

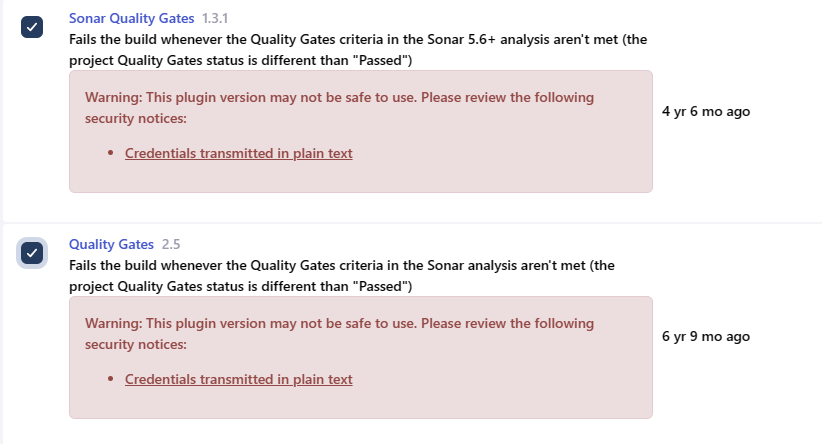
             }

        }

Make sure we have installed SonarQube plugins



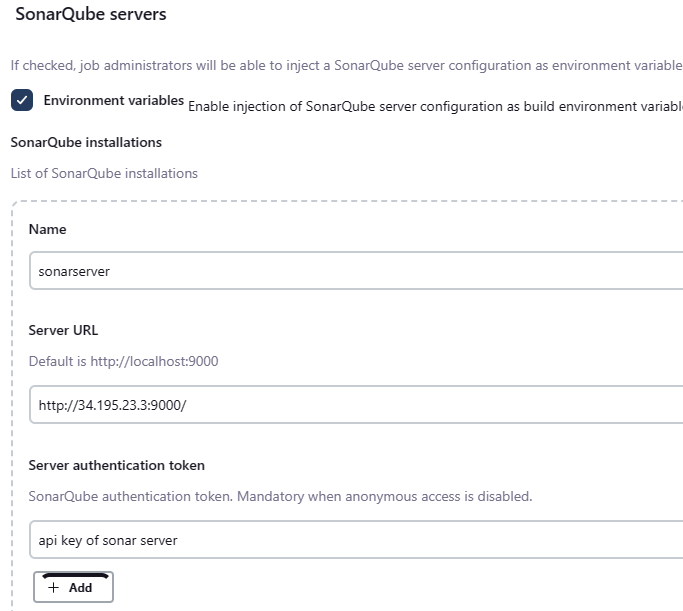




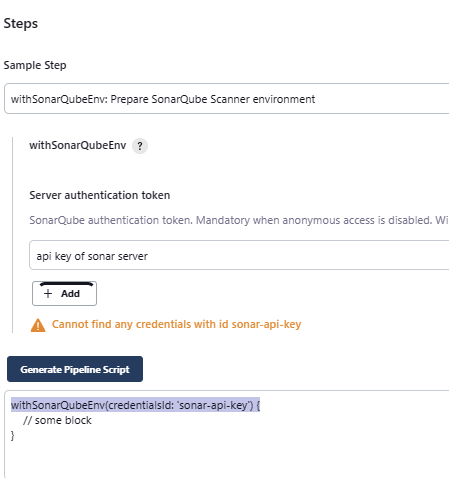
Also make sure Under Manage Jenkins--> Configure System --> SonarQube servers have required entries.

Create api token in SonarQube server and add it in Jenkins --> credentials (refer SonarQube installation document)

PS:- In screenshot below Server URL ends with '**/**' which is wrong. It should be just http://ip-addrs:port



Pipeline syntax for Jenkins to talk to SonarQube server



        stage('Quality Gate status Check'){

             steps{

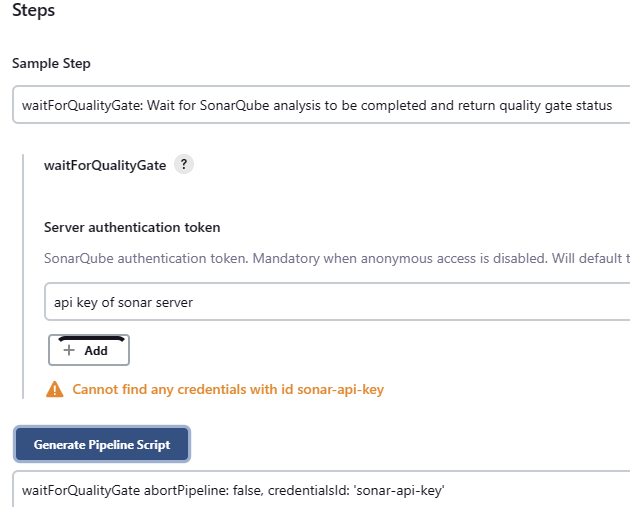
                script{

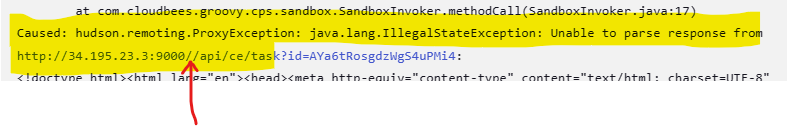
                   waitForQualityGate abortPipeline: false, credentialsId: 'sonar-api-key'

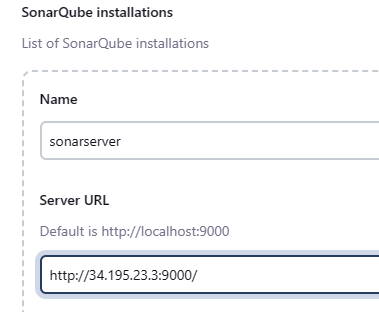
                }

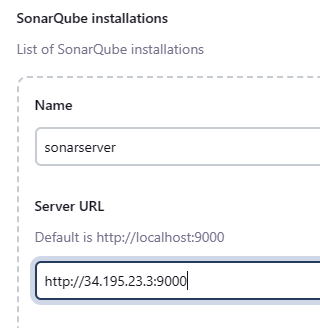
             }

        }









stage('Quality Gate status Check'){

             steps{

                script{

                   waitForQualityGate abortPipeline: false, credentialsId: 'sonar-api-key'

                }

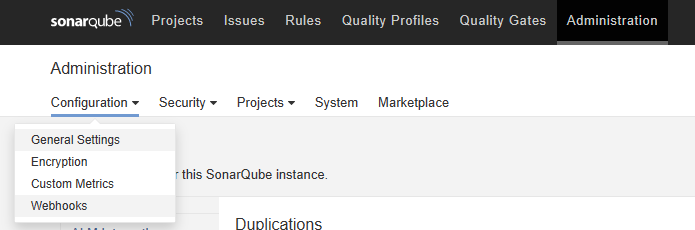
             }

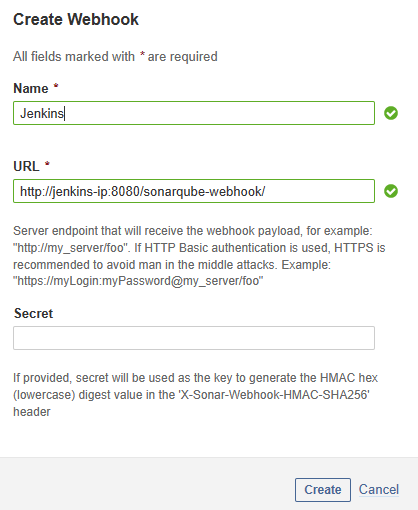
        }

Above stage will hang in PENDING status because of handshake issue

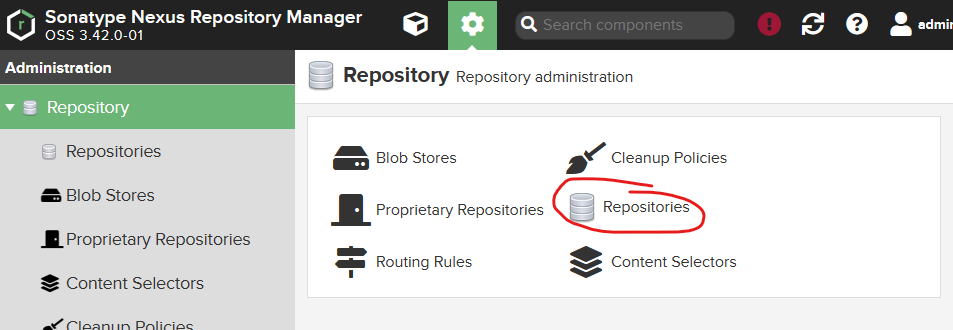


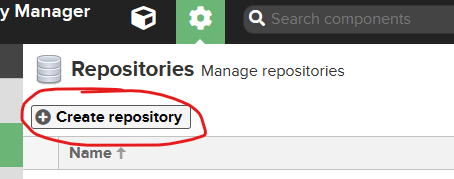
Jenkins can talk to SonarQube server using the API token we generated and added in credentials. But SonarQube can't talk back to Jenkins to confirm whether the 'Quality Gate status Check' is success or not. For that we need to generate a webhook in SonarQube server.

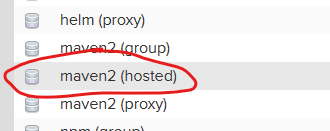




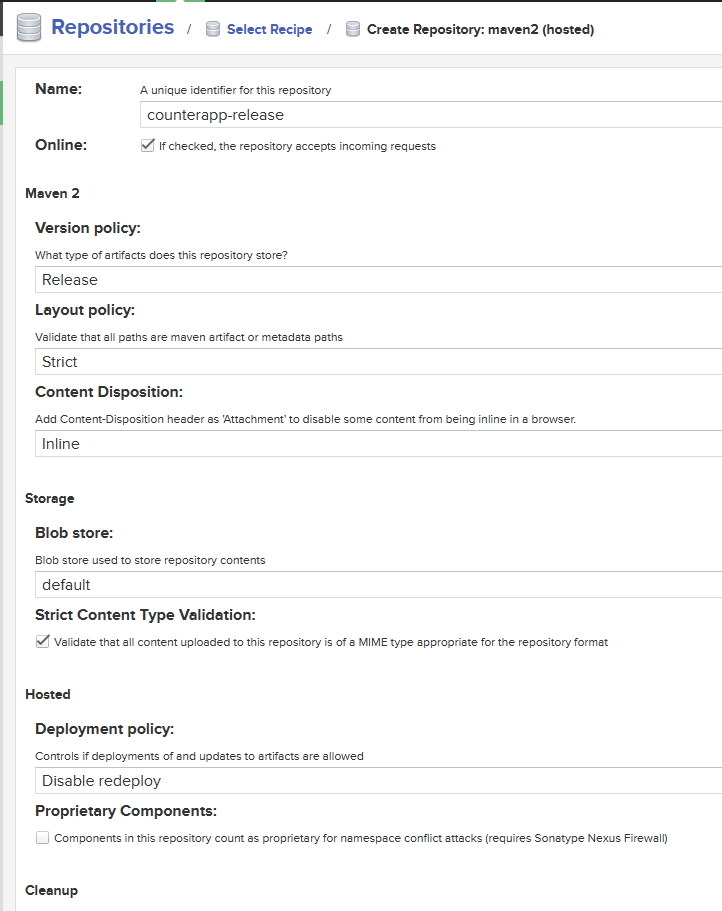
Create Nexus repository

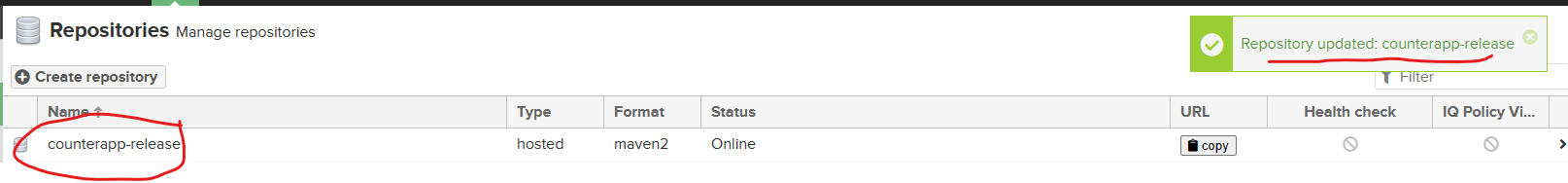






Enter name and leave everything AS-IS and create repository





        stage('Upload JAR to Nexus'){

            steps{

                script{

                    nexusArtifactUploader artifacts: [[artifactId: 'springboot', classifier: '', file: 'target/Uber.jar', type: 'jar']],

                    credentialsId: 'nexus-creds',

                    groupId: 'com.example',

                    nexusUrl: '3.213.160.126:8081',

                    nexusVersion: 'nexus3',

                    protocol: 'http',

                    repository: 'counterapp-release',

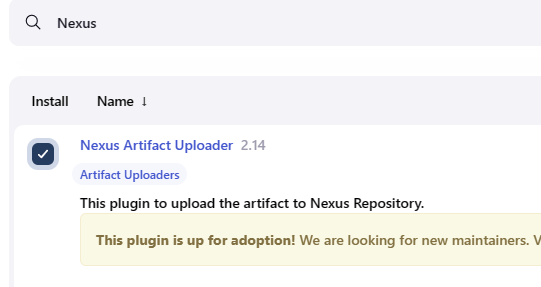
                    version: '1.0.0'

                }

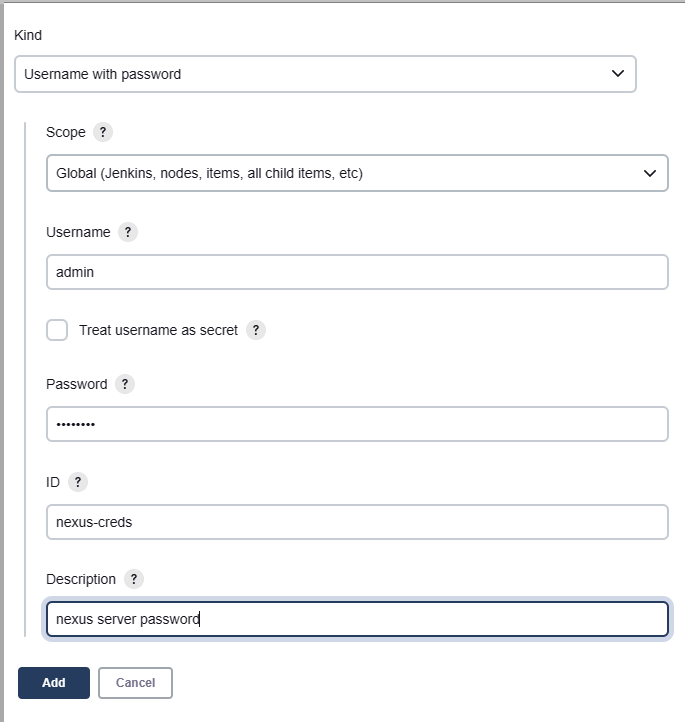
            }

        }

Install plugins which are required for Nexus



Add Nexus credentials to Jenkins



Pipeline Syntax generator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

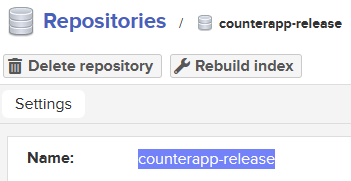
Use this values for pipeline generator

<groupId>com.example</groupId>

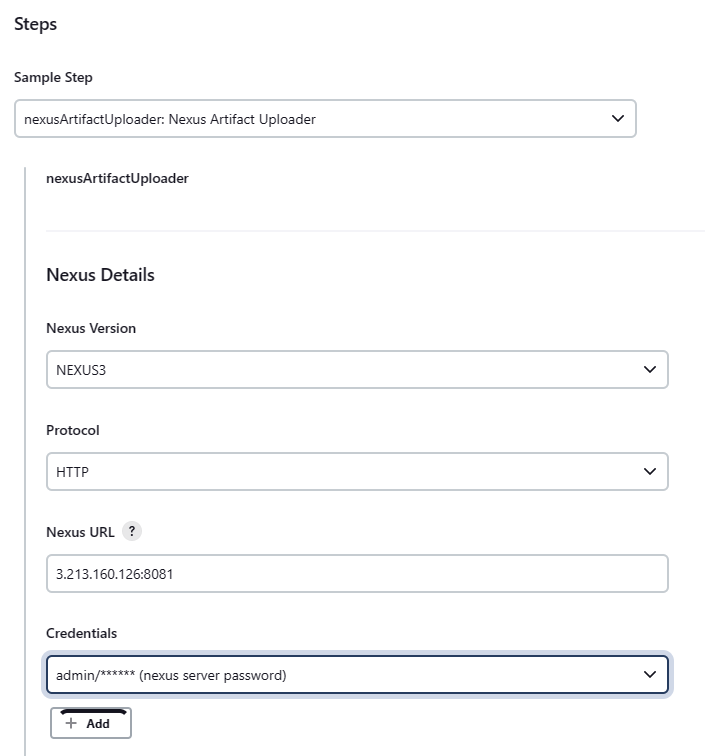
<artifactId>springboot</artifactId>

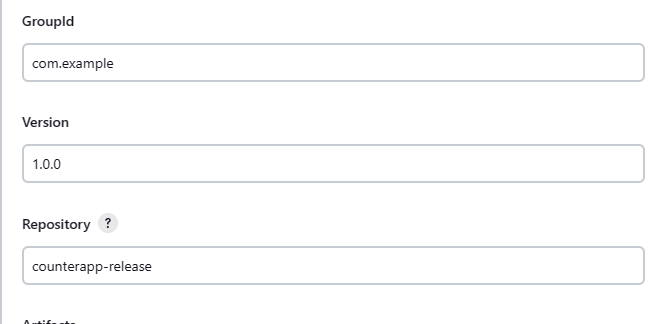
<version>1.0.0</version>

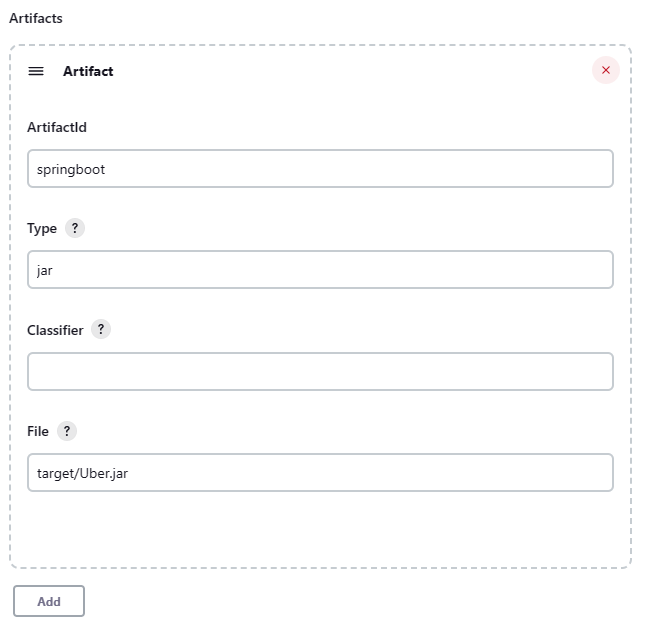
<finalName>Uber</finalName>



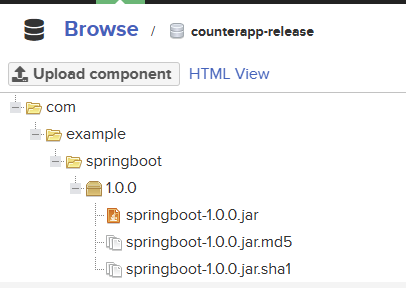
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*





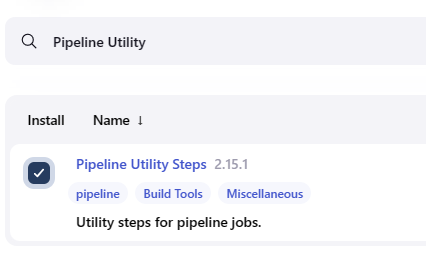


**nexusArtifactUploader artifacts: [[artifactId: 'springboot', classifier: '', file: 'target/Uber.jar', type: 'jar']], credentialsId: 'nexus-creds', groupId: 'com.example', nexusUrl: '3.213.160.126:8081', nexusVersion: 'nexus3', protocol: 'http', repository: 'counterapp-release', version: '1.0.0'**



But hardcoding version in Jenkinsfile is not a best practise

Install a plugin for this



Modify the stage like below

stage('Upload JAR to Nexus'){

            steps{

                script{

                    def readPomVersion = readMavenPom file : 'pom.xml'

                    def nexusRepo = readPomVersion.version.endsWith("SNAPSHOT") ? "counterapp-snapshot" : "counterapp-release"

                    nexusArtifactUploader artifacts: [[artifactId: 'springboot', classifier: '', file: 'target/Uber.jar', type: 'jar']],

                    credentialsId: 'nexus-creds',

                    groupId: 'com.example',

                    nexusUrl: '3.213.160.126:8081',

                    nexusVersion: 'nexus3',

                    protocol: 'http',

                    repository: nexusRepo,

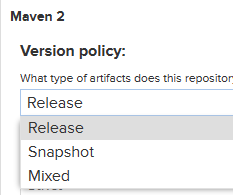
                    version: "${readPomVersion.version}"

                }

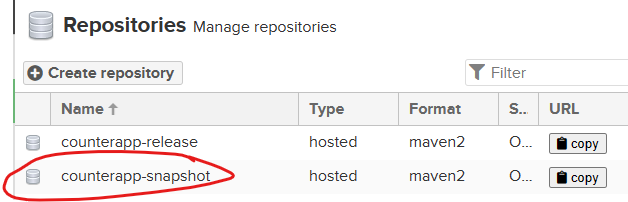
            }

        }

There are 2 types of Version policy in Nexus repo. SNAPSHOT & RELEASE



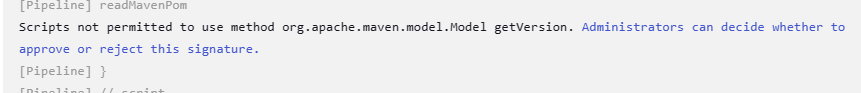
Lets create one more repo to store SNAPSHOT of this project



So, if Developer makes any changes in pom.xml like below we don’t need to update the Jenkinsfile as we made repository and version as variable. Similarly, we can make other things also as variable.

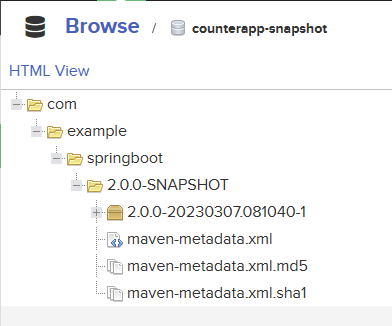
Run the Jenkins JOB.

First time we need to approve this JOB



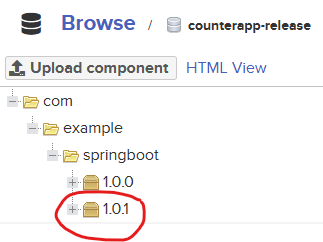


Job succeeded



Let's rebuild again by changing just version

<version>1.0.1</version>



Build Docker image and tag it

stage('Docker image Building'){

     steps{

         script{

            sh 'docker image build -t $JOB\_NAME:v1.$BUILD\_ID .'

            sh 'docker image tag $JOB\_NAME:v1.$BUILD\_ID sulbiraj/$JOB\_NAME:v1.$BUILD\_ID'

            sh 'docker image tag $JOB\_NAME:v1.$BUILD\_ID sulbiraj/$JOB\_NAME:latest'

                }

             }

        }

Push docker image to DockerHub

stage('Docker image push'){

  steps{

  script{

withCredentials([string(credentialsId: 'dockerhubpass', variable: 'dockerhubpasswd')]) {

                     sh 'docker login -u sulbiraj -p ${dockerhubpasswd}'

                     sh 'docker image push sulbiraj/$JOB\_NAME:v1.$BUILD\_ID'

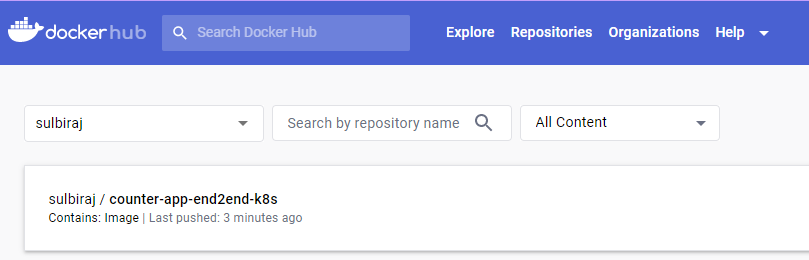
                     sh 'docker image push sulbiraj/$JOB\_NAME:latest'

                  }

                }

             }

        }



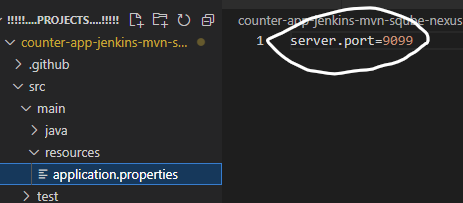
Deploy to Kubernetes

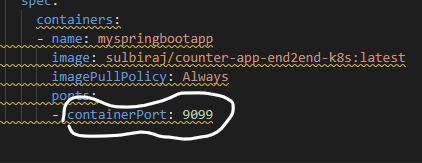
====================

* Make sure aws cli, eksctl and kubectl are installed in Jenkins server

eksctl create cluster --name **demo-eks** --region us-east-1 --nodegroup-name **my-nodes** --node-type **t2.micro** --managed --nodes 2

* Identify the application port from java source code => 9099 and use that as container port in deployments.yaml file





* Create deployments.yaml and services.yaml files for the deployment
* Update **Jenkinsfile**

parameters {

        choice(name : 'action', choices: 'create\ndestroy', description : 'Create or Destroy the eks cluster')

        string(name : 'cluster', defaultValue : 'demo-eks', description : 'EKS cluster name')

        string(name : 'region', defaultValue : 'us-east-1', description : 'EKS cluster region')

    }

stage('EKS connect') {

            steps {

                sh """

                    aws eks --region ${params.region} update-kubeconfig --name ${params.cluster}

                """;

            }

        }

stage('EKS deployments') {

            when { expression { params.action == 'create'}}

            steps {

                script {

                    def apply = false

                    try{

                        input message : 'please confirm to apply the deploymnts', ok : 'Ready to apply'

                        apply = true

                    }

                    catch(err) {

                        apply = false

                        CurrentBuild.result= 'UNSTABLE'

                    }

                    if(apply) {

                        sh """

                            kubectl apply -f .

                        """;

                    }

                }

            }

        }

stage('EKS Destroy') {

            when { expression { params.action == 'destroy'}}

            steps {

                script {

                    def destroy = false

                    try{

                        input message : 'please confirm to destroy the deploymnts', ok : 'Ready to destroy'

                        destroy = true

                    }

                    catch(err) {

                        destroy = false

                        CurrentBuild.result= 'UNSTABLE'

                    }

                    if(destroy) {

                        sh """

                            kubectl delete -f .

                        """;

                    }

                }

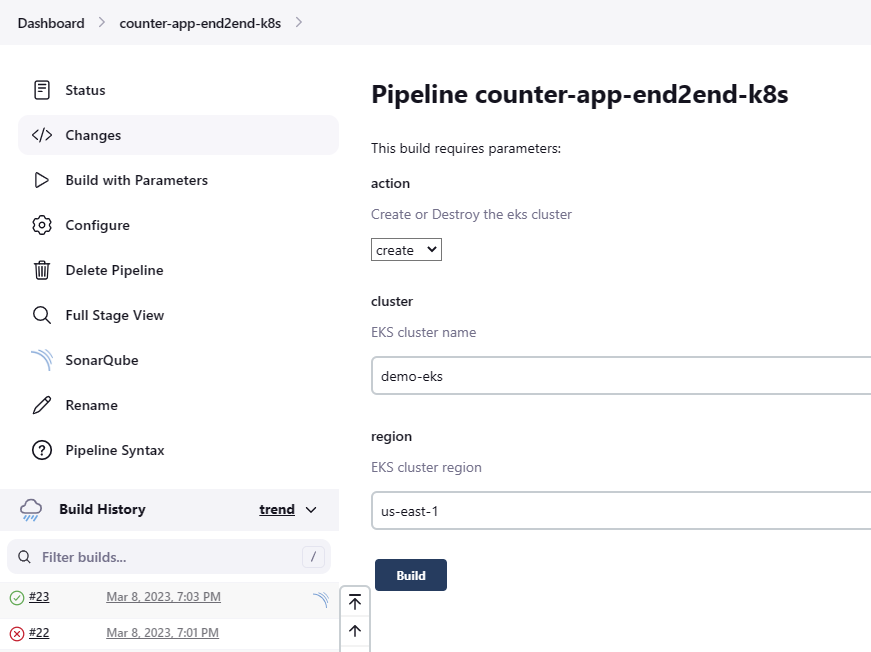
            }

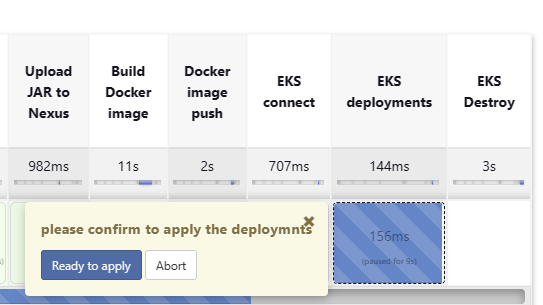
        }

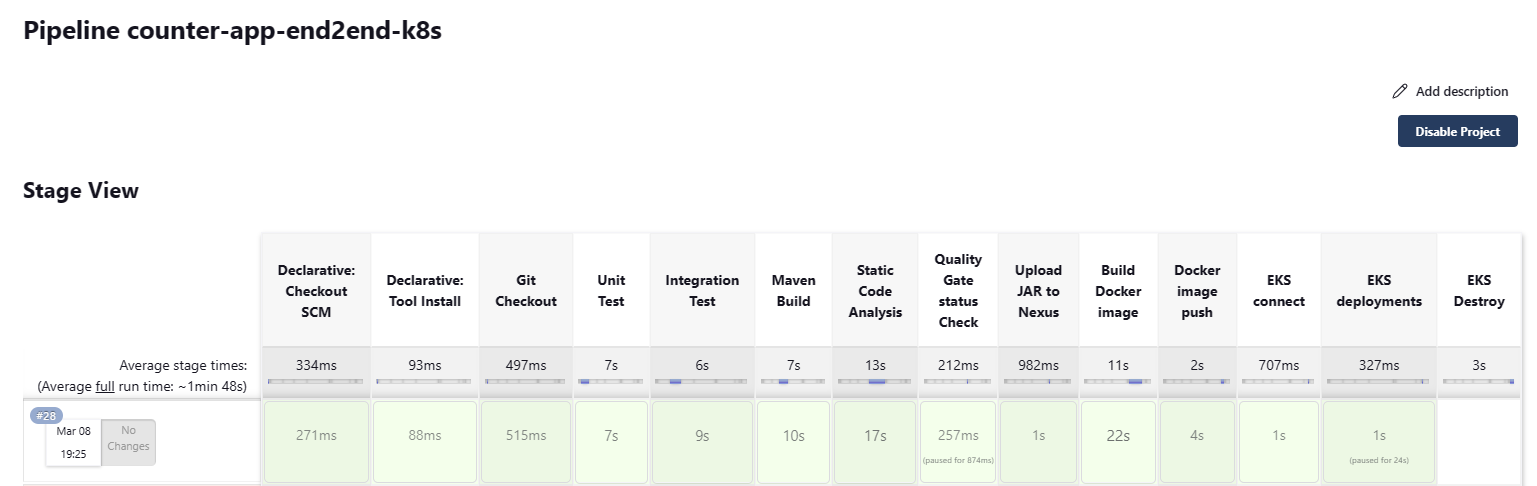
* Create a new role with admin privilege and attach the role to Jenkins Server so that 'EKS connect' stage works without any CLI authentication

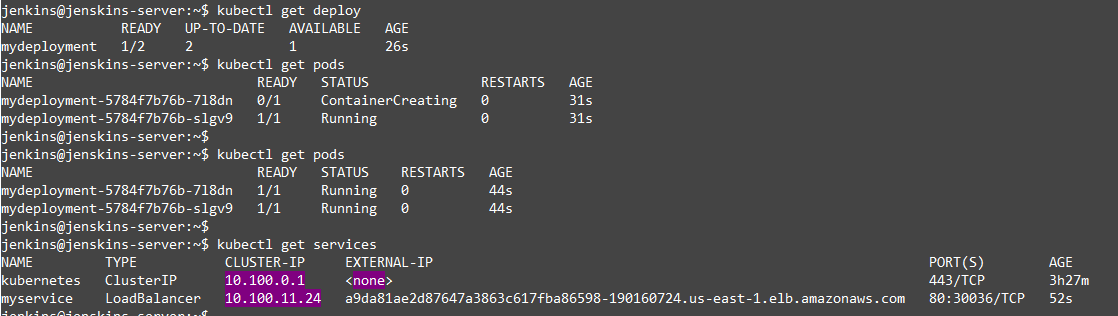
Build the Jenkins JOB with Parameters

=====================================



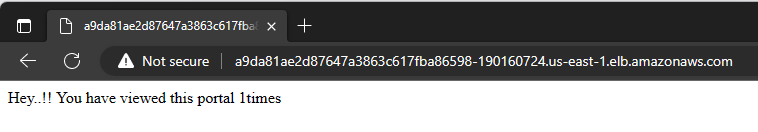




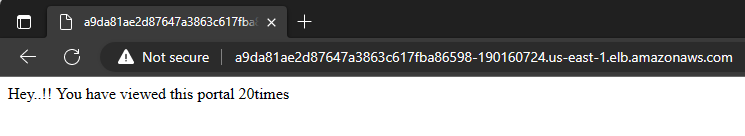


Browse the ELB url to access the application

<http://a9da81ae2d87647a3863c617fba86598-190160724.us-east-1.elb.amazonaws.com>



Keep refreshing and check again



Hurray..!!! Our application has been successfully deployed to EKS cluster

Important files used in this project

