**DOCKER (PAAC) CONTINUOUS INTEGRATION IN JENKINS**:

FLOW OF EXECUTION:

1. Install docker engine in Jenkins (Add Jenkins user to docker group & reboot)
2. Install AWS cli
3. Create IAM User
4. Create ECR repo
5. plugins

* Ecr, docker pipeline, aws sdk for credentials

1. Store aws credentials in Jenkins
2. Run the pipeline
3. **Install docker engine in Jenkins (Add Jenkins user to docker group & reboot)**

* Open git bash and SSH to the Jenkins instance (ssh –i Downloads/jenkinskey.pem ubuntu@IP)
* sudo –i
* You are free to use the documentation resource while you use the codes below: (docs.docker.com)
* sudo apt-get update -y
* sudo apt-get install \

ca-certificates \

curl \

gnupg \

lsb-release –y

* sudo mkdir –p /etc/apt/keyrings
* curl –fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo gpg --dearmor –o /etc/apt/keyrings/docker.gpg
* echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

* sudo apt-get update
* sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin –y
* id Jenkins
* usermod –a –G (group name which was docker, provide yours if you have) Jenkins

1. INSTALLING AWSCLI

* apt install awscli –y
* reboot

1. CREATING AN IAM USER

* Go to AWS console
* search and open IAM (Identity and Access Management)
* click on Add user and give the username = Jenkins,
* click to check on Access key-programmatic access.
* click on Next permissions,
* click on attach existing policies,
* click on AmazonEC2ContainerRegistryFullAccess and AmazonECS\_FullAccess
* Click on create user and make sure you save the .csv file which contains the access and secret keys

1. CREATING ECR REPOSITORY

* In your AWS console, search for ECR (elastic container registry)
* click on create repository,
* click to check private and then give your repository a name in the box=vprofileappimg
* scroll down to the end and click on create repository. Take note of your repository url.

1. INSTALLING PLUGINS (Ecr, docker pipeline, aws sdk for credentials

* Go to the Jenkins EC2 instance and copy the public IP
* Use the public IP in your browser to login to Jenkins using your username and password
* click on manage Jenkins and click on manage plugins, click on the available tab and search for the following:

Amazon Ecr,

Docker pipeline,

amazon web services SDK for credentials

CloudBees Docker Build and Publish

click on install without restart

1. Store AWS credentials in Jenkins (USING THE I AM USER)

* Go to dashboard and click on manage Jenkins.
* scroll down and click on manage credentials
* click on Jenkins below “stores scoped to Jenkins”
* click on global credential,
* click on add credentials on the left pane (kind = AWS Credentials, ID = awscreds, Description = awscreds)
* paste the access key and secret access key from the .csv
* copy the PAAC and paste it:
* Remember to edit the PAAC below with the right info:

ECR = your EC2 instance

APPREGISTRY = the ECR url you copied and saved above

VPROFILEREGISTRY = https:// ECR url you copied and saved above (remove the vprofileappimg at the end)

pipeline {

agent any

tools {

maven "MAVEN3"

jdk "OracleJDK8"

}

environment {

registryCredential = 'ecr:us-east-2:awscreds'

appRegistry = "951401132355.dkr.ecr.us-east-2.amazonaws.com/vprofileappimg"

vprofileRegistry = <https://951401132355.dkr.ecr.us-east-2.amazonaws.com> (ECR url)

}

stages {

stage('Fetch code'){

steps {

git branch: 'docker', url: 'https://github.com/devopshydclub/vprofile-project.git'

}

}

stage('Test'){

steps {

sh 'mvn test'

}

}

stage ('CODE ANALYSIS WITH CHECKSTYLE'){

steps {

sh 'mvn checkstyle:checkstyle'

}

post {

success {

echo 'Generated Analysis Result'

}

}

}

stage('build && SonarQube analysis') {

environment {

scannerHome = tool 'sonar4.7'

}

steps {

withSonarQubeEnv('sonar') {

sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \

-Dsonar.projectName=vprofile-repo \

-Dsonar.projectVersion=1.0 \

-Dsonar.sources=src/ \

-Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

-Dsonar.junit.reportsPath=target/surefire-reports/ \

-Dsonar.jacoco.reportsPath=target/jacoco.exec \

-Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml'''

}

}

}

stage("Quality Gate") {

steps {

timeout(time: 1, unit: 'HOURS') {

// Parameter indicates whether to set pipeline to UNSTABLE if Quality Gate fails

// true = set pipeline to UNSTABLE, false = don't

waitForQualityGate abortPipeline: true

}

}

}

stage('Build App Image') {

steps {

script {

dockerImage = docker.build( appRegistry + ":$BUILD\_NUMBER", "./Docker-files/app/multistage/")

}

}

}

stage('Upload App Image') {

steps{

script {

docker.withRegistry( vprofileRegistry, registryCredential ) {

dockerImage.push("$BUILD\_NUMBER")

dockerImage.push('latest')

}

}

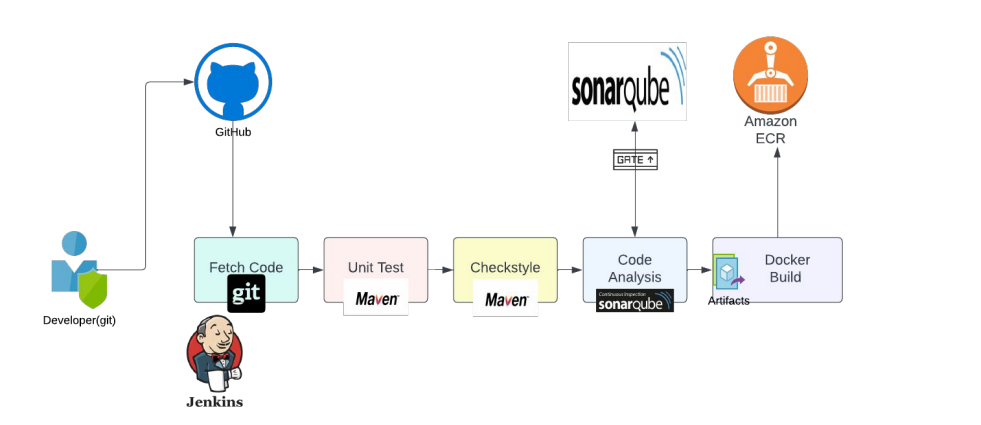
}

}

}

}

* copy the pipeline code, login to Jenkins through your browser, click on new item (item name = docker-ci-pipeline, then click on pipeline, and click ok)
* scroll down to the pipeline part and paste your code under script
* click on save and click on build now on the left pane
* when the build is completed, click on the blue circle below build history on the left pane



**Hosting the Image on ECS (DEPLOYMENT):**

We have two major types of container hosting platform:

1. Docker Engine: For local development environment
2. Kubernetes: For Production environment. Types of Kubernetes service providers are:

Standalone, EKS, AKS, GKE, Openshift etc.

1. Amazon ECS

STEPS:

* Login to Amazon console and search for Amazon ECS service
* Below the AWS logo on the left top pane, click to switch to ECS experience
* click on get started and click on create cluster
* Name = vprofile,
* keep the default vpc and subnets,
* make sure AWS fargate is checked under Monitoring (Here no need for EC2 instance, it will run behind the scene.)
* under Monitoring section, click to activate use container insights,
* click on create below
* Click on the menu button on the left pane
* Go to task definition under menu,
* click on create new task
* Task definitions family = vprofileapptask,
* under container -1 section – name = vproapp, image URL = put the url of the ECR created, port = 8080, protocol = TCP,
* scroll down and click on next
* Environment section
* app environ = fargate,
* 2GB ram
* 1vCPU,
* leave others as default and scroll down then click on next)
* Click on your created cluster name, scroll down and click on service tab, click on Deploy (application type=service, Task definition – family=select your task definition, service name = vprofileappsvc, keep others as default)
* scroll down and click on Load balancing
* type = application,
* check create a new load balancer,
* load balancer name = vprofileappelbecs,
* listener port = 80 (it was changed below becos it gave an unhealthy health check)
* protocol = HTTP,
* target group name = vproecsBg
* protocol = HTTP,
* health check = /login
* click on networking, create a new security group for the load balancer (name = vproappecselb-sg, inbound = HTTP TCP 80 from anywhere, public IP should be enabled, then click on Deploy)
* Go to EC2 service, scroll to load Balancers on the left pane,
* select your ELB and scroll down click on listeners tab
* click on default forwarding to link,
* click on your target group, if you see unhealthy,
* click on the Health checks tab and click on edit,
* click on the advanced health check, click on override and change to 8080, healthy threshold = 2, unhealthy threshold = 2
* click on save changes
* Go to the security groups,
* click on your security group vproappecselb-sg,
* edit the inbound rule, and add another inbound rule (customTCP 8080 from any IPv4 and 6, click on save)
* Go back to your ECS vprofileappsvc, click on networking, at DNS names click to open to view your app if successful

OR:

* Go to your cluster vprofile,
* click on tasks tab,
* click on your container and copy and paste the public IP in your browser
* Now go to your code editor, in your pipeline code, add the clustername (vprofile) and service name (vprofileappsvc)
* Go to Jenkins server,
* click on manage Jenkins, and click on manage plugins,
* click on available tab and search for (Pipeline: AWS Steps)
* click on install without restarting
* click on dashboard, and click on new item (item name: cicd-pipeline-ecs),
* click on pipeline and click on ok,
* scroll down to script and paste the script below, make sure you edit the code with the right credentials, then click on save

pipeline {

    agent any

    tools {

        maven "MAVEN3"

        jdk "OracleJDK8"

    }

    environment {

        registryCredential = 'ecr:us-east-2:awscreds'   (awscreds is the credential name in Jenkins)

        appRegistry = "951401132355.dkr.ecr.us-east-2.amazonaws.com/vprofileappimg"  (This is the ECR url)

        vprofileRegistry = https://951401132355.dkr.ecr.us-east-2.amazonaws.com  (ECR url)

        cluster = "vprofile"

        service = "vprofileappsvc"

    }

  stages {

    stage('Fetch code'){

      steps {

        git branch: 'docker', url: 'https://github.com/devopshydclub/vprofile-project.git'

      }

    }

    stage('Test'){

      steps {

        sh 'mvn test'

      }

    }

    stage ('CODE ANALYSIS WITH CHECKSTYLE'){

            steps {

                sh 'mvn checkstyle:checkstyle'

            }

            post {

                success {

                    echo 'Generated Analysis Result'

                }

            }

        }

        stage('build && SonarQube analysis') {

            environment {

             scannerHome = tool 'sonar4.7'

          }

            steps {

                withSonarQubeEnv('sonar') {

                 sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \

                   -Dsonar.projectName=vprofile-repo \

                   -Dsonar.projectVersion=1.0 \

                   -Dsonar.sources=src/ \

                   -Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

                   -Dsonar.junit.reportsPath=target/surefire-reports/ \

                   -Dsonar.jacoco.reportsPath=target/jacoco.exec \

                   -Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml'''

                }

            }

        }

        stage("Quality Gate") {

            steps {

                timeout(time: 1, unit: 'HOURS') {

                    // Parameter indicates whether to set pipeline to UNSTABLE if Quality Gate fails

                    // true = set pipeline to UNSTABLE, false = don't

                    waitForQualityGate abortPipeline: true

                }

            }

        }

    stage('Build App Image') {

       steps {

         script {

                dockerImage = docker.build( appRegistry + ":$BUILD\_NUMBER", "./Docker-files/app/multistage/")

             }

     }

    }

    stage('Upload App Image') {

          steps{

            script {

              docker.withRegistry( vprofileRegistry, registryCredential ) {

                dockerImage.push("$BUILD\_NUMBER")

                dockerImage.push('latest')

              }

            }

          }

     }

     stage('Deploy to ecs') {

        steps {

            withAWS(credentials: 'awscreds', region: 'us-east-2') {

                sh 'aws ecs update-service --cluster ${cluster} --service ${service} --force-new-deployment'

            }

        }

     }

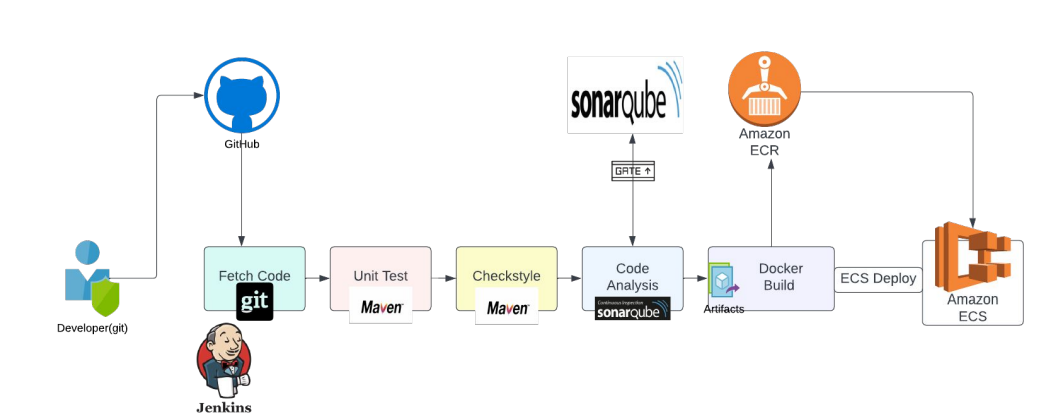
  }

}

* click on Build now to test it

**CLEANUP OF ALL SERVICES TO AVOID BILLING**

* Starting with the EC2 instances, you can stop them or delete them outrightly
* next, is the cluster, click on the service
* click on edit, reduce the Desired task to zero
* click on update below
* go to task section and click to stop the task
* return to the services section, click to check the service and delete it
* return to the clusters section, click on the cluster to select
* click delete



**BUILD TRIGGERS:**

These helps to Execute jobs automatically. Examples of famous triggers are:

* Git Webhook
* Poll SCM
* Scheduled jobs
* Remote triggers
* Build after other projects are built
* STEPS: (GIT WEBHOOKS)

1. Create git repository on github
2. SSH auth
3. Create a Jenkinsfile in git repo & commit
4. Create Jenkins job to access jenkinsfile from git repo
5. Test triggers
6. Create GIT repository on Github:

* Sign in to your Github account,
* click on New or create repository, give it a name and click on private (Let’s learn the private part) click on create
* copy and save the SSH link

1. SSH auth:

* Activate git ssh, open your git bash and type: ssh-keygen.exe
* if you are asked to overwrite, type No! This means you already have the key (check: ls ~/.ssh/
* ls ~/.ssh/ (shows you the location of the public key – id\_rsa.pub & private key – id\_rsa)
* cat ~/.ssh/id\_rsa.pub ……. copy the key
* Go to your github account setting and not repository settings,
* click on SSH and GPG keys on the left pane, click on New SSH Key (Title = myLaptop and paste the key, click on add SSH Keys, provide your github password)
* Go to your newly created repository and click on SSH beside HTTPS, copy the SSH link and head to git bash, mkdir –p /c/gitrepos, cd /c/gitrepos,
* git clone (paste the SSH link)
* ls
* cd into the repository in ls view
* git add . (to move your artifact to github)
* git commit –m “first commit”
* git push origin master
* Go to your Jenkins Server, click on Manage Jenkins and click on Configure Global Security,
* scroll down to Git Host key Verification and click on the drop down and select Accept first connection

1. Create a Jenkinsfile (script) in git repo & commit

* create the Jenkinsfile (script) in your code editor and save (name: jenkinsfile type: All files)……. when you see (.groovy extension, do the next step)
* mv Jenkinsfile.groovy Jenkinsfile
* ls
* git add .
* git commit –m “first commit”
* git push origin master

1. **CREATE JENKINS JOB TO ACCESS JENKINSFILE FROM GITHUB REPO (PIPELINE)**

* click on create job
* (Name = Build, select pipeline and click ok)
* scroll down to pipeline section,
* definition = Pipeline script from SCM,
* SCM = Git,
* Repository url = paste the Git SSH url here,
* click on the add button under credentials and select Jenkins,
* kind = SSH Username with private Key,
* ID = gitsshkey,
* Username = your Github account
* under private key section,
* click to check enter directly (return to git bash to copy the private key: cat ~/.ssh/id\_rsa) copy it together with the open and closing heading, paste it here

(paste the private key under key,

scroll down and click on Add,

Credentials = gitsshkey,

scroll down to Script Path = jenkinsfile (make sure it is on the main branch)

click save)

**Git Webhooks:**

* In the url of Jenkins server, copy only the IP address and not with path (e.g. http://18.221.221.216:8080/)
* Go to repository settings and not account settings,
* click on webhooks on the left pane, and click on Add webhook,
* under payload URL = (paste the IP in these format: <http://18.221.221.216:8080/github-webhook/>),
* in the content type = application/json,
* check on the just the push event,
* then click on Add webhook and refresh the page. You should see a green check beside the url,
* click on the Recent Deliveries tab to check.
* if it has a red cross check the Jenkins security group 8080 should be allowed from anywhere
* Go to your Jenkins job,
* click on Configure
* scroll down to Build Triggers and select Github hook trigger for GITScm polling
* Then click on save
* **STEPS: (Poll SCM)**
* If you want jenkins to be checking github for commits, the last step above – uncheck the Github hook trigger for GITScm polling
* check the Poll SCM option.
* Then type (\*\*\*\*\*) then click on save
* **Scheduled jobs**

Here, Jenkins will only run the job at the scheduled time and day saved.

* If you want jenkins to be checking github for commits, the last step above – uncheck the Poll SCM option
* select the Build periodically then specify using the format below:

Min Hr everyday everymonth everyday of the week(Range)

(30 20 \* \* 1-5)

click on save when done

* **Remote triggers**

From anywhere you can trigger the Jenkins job

* If you want remote access to jenkins to perform the Build Job,
* the last step above – uncheck the Scheduled jobs option and select the trigger Builds remotely (authentication token = mybuildtoken – any name of your choice).
* copy the URL below the authentication token showing, to trigger build remotely and paste it in a code editor to edit
* click on save
* This is the format to edit the remote url (http://(Jenkins Ip : 8080)/job/Build/ build? token = mybuildtoken – the name of your choice given)
* Go to google and type wget for gitbash, scroll down to Wget and click on eternallybored to download (64bit Zip file), Once downloaded click on it to open, click on wget.exe, extract to /c/program files/git/maing64/bin, click on ok
* go to Jenkins server and click on the admin at the top right tab,
* then select configure,
* scroll down to API Token and click on Add new Token,
* click on generate, copy the token and paste in the editor in a line below

**HOW IT SHOULD BE DONE IN THE CODE EDITOR**

Job URL

(http://Jenkins Ip : 8080/job/Build/ build? token = mybuildtoken)

Token

Jenkins username : (paste the token here)

CRUMB (edit with your details)

wget -q --auth-no-challenge --user (Jenkins username) --password (Jenkins password) --output-document - 'http://(JENNKINS\_IP):8080/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)'

* Go to git bash
* Paste the link under CRUMB and hit enter,
* copy the Jenkins-Crumb and paste in the code editor in a line below

Job URL

(http://Jenkins Ip : 8080/job/Build/ build? token = mybuildtoken)

Token

Jenkins username : (paste the token here)

CRUMB (edit with your details)

wget -q --auth-no-challenge --user username --password password --output-document - 'http://JENNKINS\_IP:8080/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)'

Jenkins-Crumb

(paste it here)

(Edit the code below with information here above from the code Editor.

E.g: Token should replace (username:APItoken)

curl -I -X POST http://username:APItoken @Jenkins\_IP:8080/job/JOB\_NAME/build?token=TOKENNAME -H "Jenkins-Crumb:CRUMB"

E.g: **(After editing it should look like the code below)**

curl -I -X POST http://admin:110305ffb46e298491ae082236301bde8e@52.15.216.180:8080/job/ vprofile-Code-Analysis/build?token=testtoken -H "Jenkins-Crumb:8cb80f4f56d6d35c2121a1cf35b7b501"

* copy only the edited code above and return to git bash to paste it and run.
* You can use it as a script be such as bash script, Python script, Ansible
* the script can be run from anywhere to trigger jenkins
* you can now confirm success from Jenkins server
* **Build after other projects are built**

It is mostly used in Freestyle projects

**JENKINS MASTER AND SLAVE**

**Use Cases:**

* Load Distribution: - - - Jenkins Master Executes build Job on Node it selected
* Cross Platform Builds --- Executing Build of other platforms like .net(windows), iOS(Mac OS) from Jenkins Master(Linux)
* Software Testing – Executes Testers test Automation scripts from Node
* When you want to run your scripts from another machine

**PREREQUISITES FOR NODE SETUP**

1. Any OS
2. Network access from Master (Note: Check Firewall rules)
3. Jave, JRE, JDK
4. User
5. Directory with User ownership
6. Tools as required by the Jenkins job e.g. Maven, Ant, Git etc

**STEPS:**

1. Create an EC2 instance

* Ubuntu 18
* Tag section (key = Name, value = Slave-Jenkins)
* create new security group (SSH/TCP/22/My IP)
* You can use same Jenkins key or create a new key pair, click on i have access to the selected private key
* click on launch instance

1. Open Git Bash and SSH (ssh –i Downloads/Jenkins-key.pem ubuntu@IP
2. sudo –i
3. apt update && apt install openjdk-11-jdk –y
4. useradd devops (to create a user)
5. passwd devops (to set a password)
6. userdel –r devops (Ubuntu doesn’t create home directory, the code is used to check for home directory)
7. adduser devops
8. insert your password
9. mkdir /opt/Jenkins-slave
10. chown devops.devops /opt/Jenkins-slave –R
11. vim /etc/ssh/sshd\_config
12. find the entry in the vim editor that says: PasswordAuthentication (Enable it)
13. :wq (to save and quit)
14. systemctl restart ssh (Ubuntu, for REDHAT = sshd)
15. Jenkins => SSH => Node^C
16. Go to Jenkins in the browser and click on Manage Jenkins
17. click on manage Nodes and Clouds
18. click on create New Node on the left pane

* Node Name = silver-node
* click on permanent agent
* click on Ok
* No of Execution = You decide (E.g. 5)
* Remote root directory = /opt/Jenkins-slave
* label = SILVER
* Usage =

Use the node as much as possible (This is used for load distribution),

Only build jobs with label expressions matching the node (This is used for a particular PC OS with same job Build with same OS)

* select only the coloured above
* Launch method = Launch agents via SSH
* Host = EC2 instance (Slave-Jenkins) private IP
* Edit security group (custom TCP/22/custom/Jenkins security group/Description = Allows Jenkins to SSH) ….. click on save below
* Return to the Jenkins in the browser
* credentials = click on Add again and select Jenkins to include password secured login details
* Kind = Username with password
* Username = devops
* Password = provide the passwd
* ID/Description = silver-login
* click on Add
* credentials = click on Add again and select Jenkins to include SSH secured login details
* click on the credential box to select password based login
* Host key verification strategy = Non verifying verification strategy
* scroll down and click on save
* click on Build to test
* under Build History section on the left pane, click on console output to check
* open gitbash and SSH to the slave EC2 instance
* sudo –i
* cd /opt/Jenkins-slave/
* ls
* you should see a coloured written file (remoting.jar)
* ls workspace/test-slave/

**To specify a path where the project will be run:**

* Go to Jenkins in the browser and click on test-slave
* click on configure
* click to check the box (Restrict where this project can be run)
* label expansion = SILVER
* scroll down and click on save
* To deactivate these settings and activate the selection mode through scripts
* Go to Jenkins in the browser and click on Manage Jenkins
* click on manage Nodes and Clouds
* click on create New Node on the left pane
* Node Name = silver-node
* click on permanent agent
* click on Ok
* No of Execution = You decide (E.g. 5)
* Remote root directory = /opt/Jenkins-slave
* label = SILVER
* Usage =

Use the node as much as possible (This is used for load distribution),

Only build jobs with label expressions matching the node (This is used for a particular PC OS with same job Build with same OS)

* select only the coloured above
* Make sure Maven is installed

**AUTHENTICATION AND AUTHORIZATION (user, permissions, roles, jobs permissions)**

These explains the rights or privileges you want to grant to the Ops team. When you have multiple CI/CD running you need to grant limited rights according to their job description.

AUTHENTICATION means the login rights while AUTHORIZATION means the privileges

(Don’t share login details and Don’t give full access to people)

**Securing Jenkins:**

* User Login
* Jenkins own database (signup)
* LDAP Integration

Permissions on Jenkins

* Admin
* Read
* Jobs
* Credentials
* Plugins etc

Permissions on Jobs:

* View
* Build
* Delete
* Configure
* etc

STEPS:

* Login to Jenkins in the browser
* click on manage Jenkins and scroll down to Security section
* click on configure global security
* under Authentication section,
* click to check (Allow users to sign up)
* please note LDAP is for a large organization with many Ops teams, you provide the details of each Op member and approved IP
* under Authorization section, apply your professional skills
* use the Project-based matrix Authorization strategy
* click to check the policies and click on save
* Use two different browsers and user accounts to confirm the permission set
* click on the dashboard and click on the project you want to set policies on
* click on configure and click on Enable project-based security
* Decide the users access

To create a role with the rights and policies which you will assign to a group of Ops team

* click on manage Jenkins and click on manage plugins
* search for role based Authorization Strategy
* click on install without restart
* click on dashboard and click on manage Jenkins
* click on configure global security
* scroll down and click to check Role-based Strategy
* click on dashboard and click on manage Jenkins
* scroll down to security section and click on Manage and Assign Roles
* click on manage roles
* under Role to Add, type DevOps and click on Add
* click on the roles and privileges you want to assign
* scroll down and click on Save
* To add users into the role:
* under manage and assign roles, click on Assign Roles
* assign users to roles created
* under security section in manage Jenkins
* click on manage users
* click on create user on the left pane