DOCKER 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, sudo –i
* You are free to use the documentation resource while you use the codes below: (docs.docker.com)
* sudo –i, 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, open IAM (Identity and Access Management, click on Add user and give the username = Jenkins, click to check on Access key-programmatic access. and then click on Next permissions, click on attach existing policies, then click on AmazonEC2ContainerRegistryFullAccess and AmazonECS\_FullAccess, Click on create user and make sure you save the .csv file

1. CREATING ECR REPOSITORY

* In your AWS console, search for ECR (elastic container registry) and click on create repository, 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, Loggin to Jenkins using your username and password in your browser
* click on manage Jenkins and click on manage plugins, click on the available tab and search for the following:

Amazon Ecr,

docker pipeline,

aws sdk for credentials

CloudBees Docker Build and Publish …….. click on install without restart

1. Store AWS credentials in Jenkins

* Go to dashboard and click on manage Jenkins. Then click on manage credentials
* click on Jenkins below stores scoped to Jenkins, click on global credential, then click on add credentials on the left pane (kind = AWS Credentials, ID = awscreds)
* paste the access key and secret access key from the .csv
* copy the PAAC and paste it:

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)

}

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')

}

}

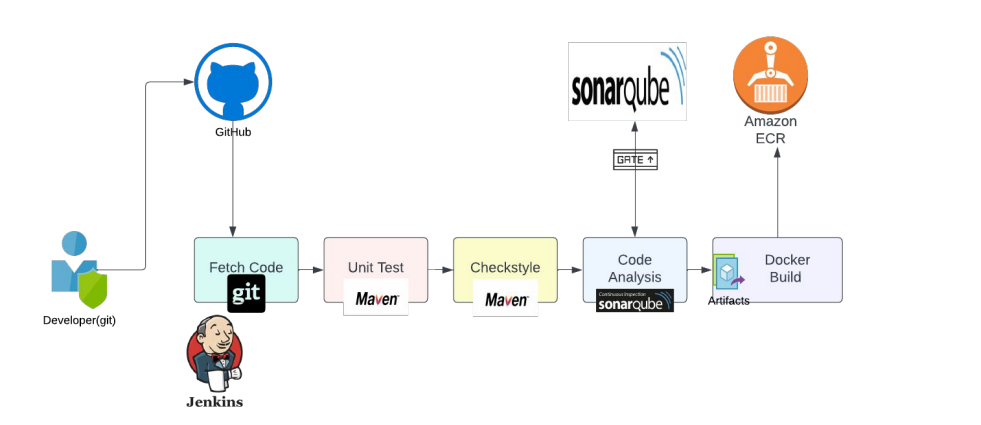
}

}

}

}

* 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, under Monitoring, click to use container insights, click on create below)
* Go to task definition on the left pane, click on create new task(Task definitions = vprofileapptask, under container – name=vproapp, put the url of the ECR created, port=8080, protocol=TCP, scroll down and click on next)
* Environment(app environ=fargate, 2GB ram and 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, port = 80 HTTP, target group name = vproecsBg 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 and 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, 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, click on manage plugins, click on available tab and search for Pipeline: AWS Steps, click on install without restarting
* click on dashboard, 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, 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'

            }

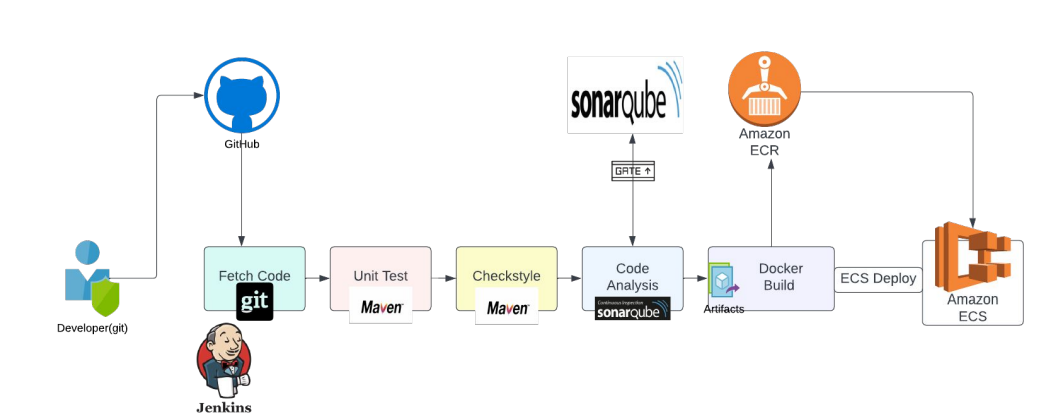
        }

     }

  }

}

* click on Build now to test it



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 create repository, give it a name and click on private (Let’s learn the private part) click on create

1. SSH auth:

* Activate git ssh, open your git bash and type: ssh-keygen.exe, /c/Users/kiwi/.ssh/id\_rsa
* if you are asked to overwrite, type No! This means you already have the key (check: ls ~/.ssh/
* 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
* 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 in git repo & commit

* create the Jenkinsfile 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 git repo (PIPELINE)

* click on create job (Name = Build, select pipeline and click ok)
* scroll down to pipeline, 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 = Github account
* under private key, return to git bash to copy the private key: cat ~/.ssh/id\_rsa
* 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
* In the url of Jenkins server, copy only the IP address and not with path
* Go to repository settings and not account settings, click on webhooks on the left pane, 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.
* Go to your Jenkins job, click on Configure and 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 and 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 and 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). copy the URL to trigger build remotely and paste it in a code editor to edit, …. click save
* This is the format to edit the remote url (http://Jenkins Ip : 8080/job/Build/ build? token = mybuildtoken)
* 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

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)'

* Go to git bash

Paste the CRUMB and hit enter, copy the Jenkins-Crumb and paste in the 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 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. 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"

* copy only the code above in yellow and return to git bash which can be runned from anywhere
* you can now confirm success from Jenkins server
* Build after other projects are builIt

It is mostly used in Freestyle projects