**SCENARIO: (Current Situation)**

* MicroService Architecture of an Application
* Containerized Application
* Continuous Code Changes
* Continuous Build & Test
* Regular Build of Container Images
* Regular Deployment Requests to Ops Team

**PROBLEM**

* Operations team incharge of Managing containers Gets continuous deployment Requests.
* Manual Deployment creates dependency
* Time Consuming

**SOLUTION:**

* Automate Build & Release process
* Build Docker Images & Deploy Continuously as fast as the code commits.
* Continuous Deployment process

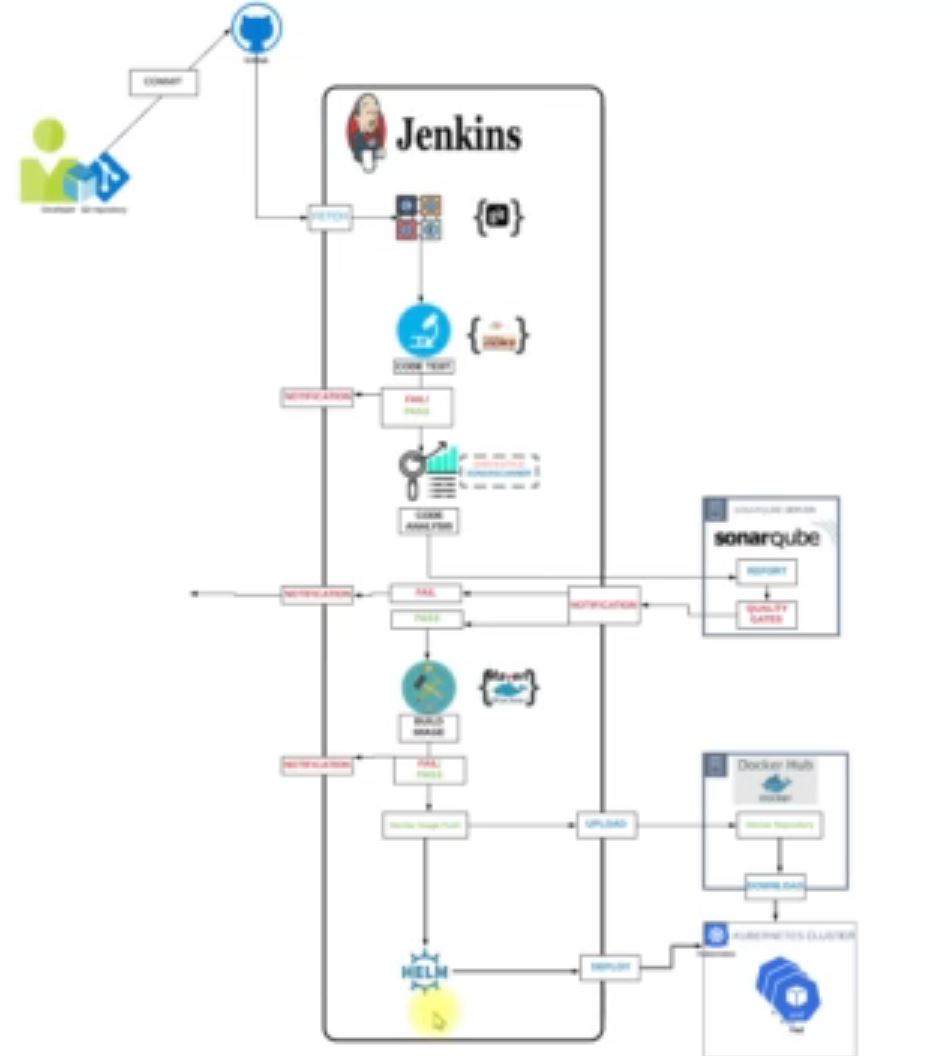
**TOOLS NEEDED:**

* Kubernetes
* Docker (Container Runtime)
* Jenkins (CICD Server)
* Docker Hub (Container Registry service)
* Helm (Packaging & Deploying on Kubernetes)
* Git (Version Control System)
* Maven Build tool
* Sonarqube for code Analysis server

**OBJECTIVE:**

* Continuous Delivery of the Containers

**PROJECT FLOW OF EXECUTION STRUCTURE:**



1. Continuous Integration Setup
2. Jenkins
3. Sonarqube & Nexus (Continuous Integration Project)
4. Dockerhub account (Containerization Project)
5. Store Dockerhub credentials in Jenkins
6. Setup Docker Engine in Jenkins
7. Install Plugins in Jenkins
8. Docker-pipeline
9. Docker
10. Pipeline utility
11. Create Kubernetes Cluster with Kops
12. Install Helm in Kops VM
13. Create Helm Charts
14. Test Charts (Entire Stack) in K8s Cluster in test namespace
15. Add Kops VM as Jenkins Slave
16. Create Pipeline code [Declarative]
17. Update Git Repository with:
18. Helm Charts
19. Dockerfile
20. Jenkins (Pipeline code)
21. Create Jenkins job for Pipeline
22. Run & Test the job

**JENKINS, SONAR & DOCKER INTEGRATION**

1. Setup Jenkins server & Sonar server. Nexus server will not be required

Check the Doc: (Steps to create a Job (Pipeline)) and Edit with the Pipeline in these folder task

1. In your setup, make sure you T2.small for Jenkins
2. Login to Jenkins first through the browser, setup Sonar integration and create K8s cluster from Kops
3. Login to Sonar through the browser and click on the account logo beside search bar at the top right hand
4. click on Administrator / my account and click on security
5. Generate Tokens = kube-jenkins and click on generate
6. copy the token and save in word file
7. Login to Jenkins in the browser and click on manage Jenkins
8. scroll down and click on configure system
9. Under SonarQube Servers section, Name = sonar-pro (the name given in your Jenkins file)
10. server URL = sonarqube private IP
11. server authentication = click add

kind = secret text

secret = paste the token from sonar in word file

ID = kube-sonar-token

Description = kube-sonar-token

click on add

server authentication = select the token add above

1. click on save
2. Go to Jenkins security group and ensure there’s connection from sonar

all traffic / sonar S.G

1. and sonar security group and ensure there’s connection from jenkin

all traffic / jenkins S.G (check it should exist already)

1. from the Jenkins file, there is Deploy Image section to deploy to docker hub, get the Docker hub credentials from your account for use
2. return to Jenkins tab in your browser, click on manage Jenkins and click on manage credentials
3. under stores scoped to Jenkins, click on the Jenkins
4. click on add credentials on the left pane

username = Dockerhub username

password = Dockerhub pass

ID = Dockerhub (the same name under environment section in the jenkins file)

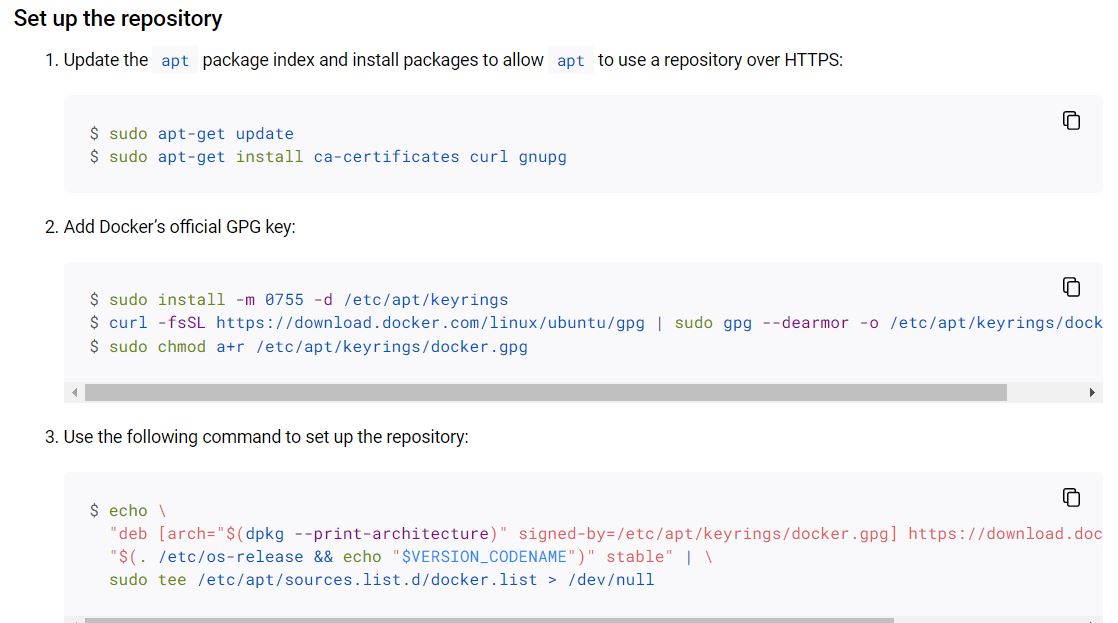
Description = Dockerhub

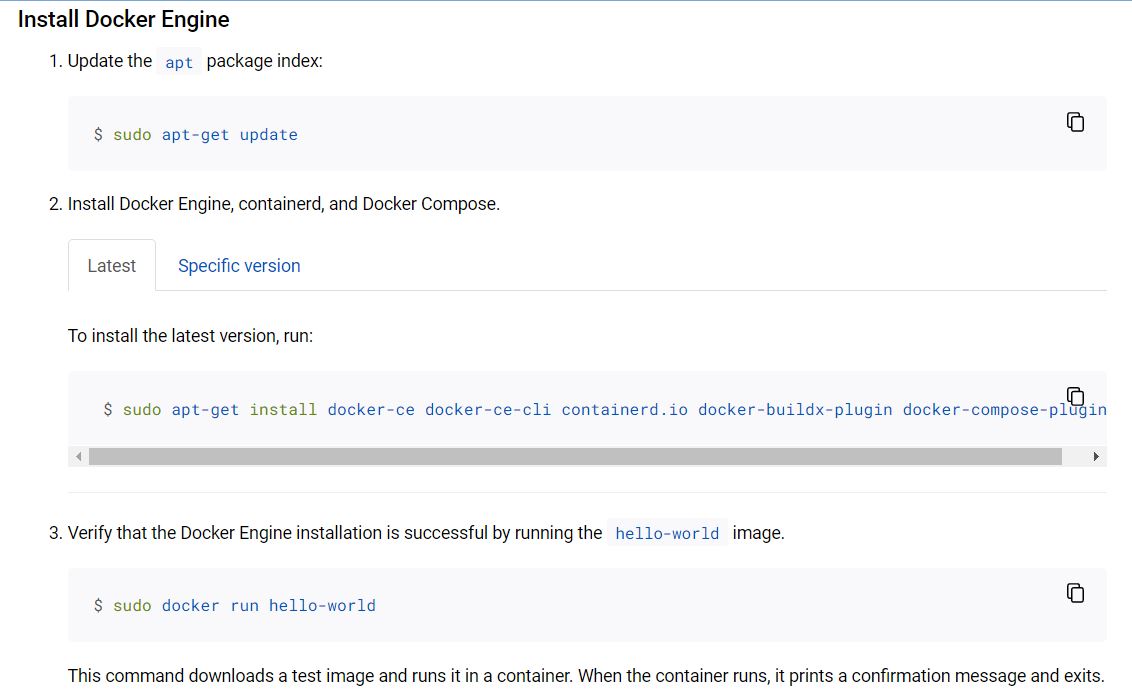
click on ok

1. let’s install docker engine in the Jenkins server
2. Open GitBash, and SSH to Jenkins

ssh –i Downloads/cicd-key.pem ubuntu@public IP

1. Use the Ubuntu user
2. Using the Docker document as a guide, I want to install Docker Engine





1. To check the Docker installation
2. sudo systemctl status docker
3. Let’s login as the Jenkins user
4. sudo –i
5. su – Jenkins
6. docker images (You will not be able to run docker comds as Jenkins user)
7. add Jenkins user in the Docker group
8. usermod -aG docker Jenkins
9. id jenkins
10. reboot (To make configurations stable, wait for few mins for Jenkins EC2 to startup)

**PLUGINS, KUBERNETES CLUSTER & HELM**

1. Once the Jenkins EC2 instance is up
2. Let’s install plugins
3. Go to your jenkins browser and refresh the page
4. Login with your Username and pass
5. click on manage Jenkins and scroll down to manage plugins, click on it
6. click on available tab and search for:

Docker Pipeline

Docker

Pipeline Utility steps

1. Click on Install without restart
2. While it is installing, Go to Gitbash and login to Kops EC2 to create the K8s cluster

ssh –i Downloads/or-kops-key.pem ubuntu@public IP

1. kops command to create the kubernetes cluster: (Follow the steps in the Doc: App Deployment on Kubernetes Cluster (Project))
2. kops create cluster - -name=vprokube.groophy.in \ (press Enter)

> - -state=s3://vprofile-kop-states - -zones= provide the zones where your service is running, check zones status \ (press Enter)

> - -node-count=2 - -node-size=t3.small - -master-size=t2.micro - -dns-zone=kubevpro.groophy.in \ (press Enter)

> - -node-volume-size=8 - -master-volume-size=8 (press Enter to start creating)

configure the cluster: The command will show immediately after last command

1. kops update cluster - -name vprokube.groophy.in - -state=s3://vprofile-kop-states - -yes - -admin (remember to include your s3 bucket)
2. Let’s install Helm in the Kops EC2 instance
3. follow Helm documentation as a guide (Copy the link address in the path: https://github.com/helm/helm/releases)
4. As Ubuntu user
5. cd /tmp/
6. wget https://get.helm.sh/helm-v3.12.0-rc.1-linux-amd64.tar.gz (This is to download)
7. tar xzvf helm-v3.12.0-rc.1-linux-amd64.tar.gz
8. cd linux-amd64/
9. ls
10. sudo mv helm /usr/local/bin/helm
11. cd
12. helm - -help
13. kubectl get nodes (To check if the K8s cluster is up and running)

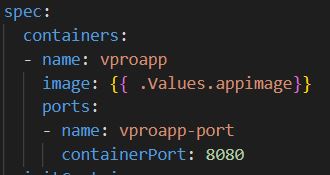
**HELM CHARTS & GIT REPO SETUP**

1. Login to your Github account and click on create a Repository

name = cicd-kube-docker

click on create a Repository and copy the HTTPS URL

1. git clone (paste the repo url)
2. cd cicd-kube-docker/
3. ls
4. cd
5. Clone your source code repo we need a lot of data from there
6. git clone https://github.com/devopshydclub/vprofile-project.git
7. ls
8. cd vprofile-project/
9. git checkout vp-docker
10. ls
11. git status
12. cp –r \* ../cicd-kube-docker/
13. cd ../cicd-kube-docker/
14. rm –rf Docker-db Docker-web compose ansible
15. ls
16. mv Docker-app/Dockerfile .
17. ls
18. rm –rf Docker-app/
19. ls
20. rm –rf helm (we would prepare our own)
21. ls
22. cat Dockerfile (To check if everything is intact)
23. cd kubernetes/
24. ls
25. cd vpro-app/
26. ls
27. cd ..
28. cd ..
29. ls (Check to make sure you are inside cicd-kube-docker)
30. mkdir helm
31. cd helm/
32. helm create vprofilecharts (the name is specified in the Jenkins file)
33. ls
34. cd vprofilecharts/
35. ls
36. cd templates/
37. ls (you will sample templates)
38. rm –rf \*
39. cd ../..
40. cd .. (Make sure you are in cicd-kube-docker dir)
41. ls
42. cp kubernetes/vpro-app/\* helm/vprofilecharts/templates/
43. ls helm/vprofilecharts/templates/
44. cd helm/vprofilecharts/templates/
45. ls
46. vim vproappdep.yml (You have to place the image in a variable using {{ }} )



1. When we run Helm command we would pass the name of the Image and Tag to the variable, so it will always have the latest image and Tag
2. Other places where you can give variables are: Labels, App name, Port no, Port name, container name
3. cd ..
4. ls (vprofilecharts dir)
5. cd ..
6. cd .. (cicd-kube-docker dir)
7. **It is time to work with the HelmCharts, to start we need to create a Namespace and do it in the Test environment**
8. kubectl create namespace test
9. helm install - -namespace test vprofile-stack helm/vprofilecharts - -set appimage=imranvisualpath/vproappdock : 9 (Set cmd is the image being passed to the variable in step 103)
10. helm list - -namespace test
11. kubectl get all - -namespace test (Everything should be created now: PODS, SERVICE, DEPLOYMENT and REPLICA SETS)
12. Copy the LoadBalancer link and test it in the browser
13. helm list - -namespace test
14. **To cleanup Helm in Test environment**
15. helm delete vprofile-stack - -namespace test
16. **PRODUCTION ENVIRONMENT**
17. make sure you are in the cicd-kube-docker dir
18. kubectl create namespace prod
19. git add .
20. git commit –m “helm charts”
21. ls
22. git status
23. git push origin master
24. **Writing the Pipeline as a Code: (JENKINS FILE)**
25. Open Vs Code and click on new under files
26. click on clone from version control

URL = the repo https link

c:/cicd-kube-docker

click on clone

1. On the left pane, inside cicd-kube-docker create a new file

Name = Jenkinsfile

press Enter

1. study the Jenkins file here (C:\gitrepos\cloudcodesandsecurity\DevOps\Kubernetes\Kubernetes CI - CD\cicd-kube-docker\cicd-kube-docker) and edit to your project. Remember to save, commit and push it
2. We would run the Helm cmd from Kops so we need to add the Kops cmd as a slave to Jenkins b4 the Deploy stage in the Jenkins file will work
3. Login to the Kops EC2 instance

ssh –i Downloads/or-kops-key.pem ubuntu@public IP

1. Using the Ubuntu user
2. mkdir Jenkins-slave
3. sudo apt install openjdk-8-jdk -y
4. mkdir /opt/ Jenkins-slave
5. sudo chown ubuntu ubuntu /opt/ Jenkins-slave -R (Jenkins should be able to ssh to the Kops EC2 instance with Ubuntu user)
6. java –version (Required)
7. Update the Kops EC2 security group

custom TCP / 22 / Jenkins-SG (Allow Jenkins to do SSH)

1. Go to Jenkins opened in your browser, click on manage Jenkins
2. scroll down and click on manage nodes and clouds
3. on the left pane, click on NewNode

Node name = kops (same name as in your jenkinsfile)

check permanent Agent

click on ok

Remote root directory = /opt/Jenkins-slave

Label = KOPS

Usage = only build jobs with Label expressions matching this node

Launch method = Launch agents via SSH

Host = private IP of the Kops EC2

Credentials = click on add (kind = SSH username with private key,

ID = kops-login

username = ubuntu,

private key = (Go to gitbash, logout from kops and: cat Downloads/or-kops-key.pem, copy the key and paste here!)

click on Add

Make sure you select the created credential

1. Scroll down and click on Save
2. click on it (kops), on the left pane, click on configure
3. Host key verification strategy = Non verification strategy
4. click on save
5. click on Relaunch Agent
6. check to see “The agent is connected”
7. click on Nodes Tab up on the left pane
8. You can now set it in the Jenkins file under deploy, study the file as instructed in step 131
9. Go to Jenkins opened in your browser, click on Manage Jenkins
10. click on Global Tool Configuration and scroll down to SonarScanner section

Name = mysonarscanner4 (from the Jenkins file)

click on save

1. **EXECUTION, TROUBLESHOOTING & SUMMARIZING**
2. Go to sonar opened in your browser, click on the project name
3. click on Quality Gates tab and create Quality Gates or select if you have one created
4. click on project settings (to attach to the project)
5. click on quality gates and select your quality gates
6. Return back to project settings and click on webhooks
7. click on create:

Name = jenkins-webhook

URL = http://private IP of Jenkins:8080/sonarqube-webhook

click on create

1. Go to Jenkins on your browser, click on New item (To create a New job)
2. Item name = kube-cicd
3. click on Pipeline and click on ok
4. scroll down to the Pipeline section:

Definition = Pipeline script from SCM

SCM = Git

Repository Url = The Repository hosting the details (HTTPS)

scroll down, script path = Jenkinsfile

click on Save

1. click on your pipeline and click on configure
2. scroll down to build triggers and check on Poll SCM

schedule = \*\*\*\*\* (These pulls the code every day,min,hr a developer commits a code change)

click on save

1. On the Left pane, click on Build Now
2. If successful, login to your Kops instance to verify the Helm

ssh –i Downloads/or-kops-key.pem ubuntu@public IP

1. using the ubuntu user
2. helm list - -namespace prod
3. kubectl get pods - -namespace prod
4. kubectl get svc - -namespace prod (copy the LoadBalancer link and check it in your browser for functionality)
5. **For clean ups**
6. kops delete cluster - -name vprokube.groophy.in - -state-s3://vprofile-kops-state - -yes
7. shutdown the EC2 instances: kops, sonar-server and jenkins