1) Launch a Developer Amazon Linux2 AMI

* Default settings, default VPC and launch
* Login into server
* Install Git (sudo -i, yum update –y, yum install git –y)

--> git –version

--> git clone "repository endpoint"

--> git config –local user.name "username"

--> git config –local user.email "usereamil"

2) SETTING UP APACHE MAVEN

* Launch another EC2 to configure as a build server

--> Amazon Linux AMI, Default settings

--> Tage=Build-Server

--> Launch and SSH into server

--> sudo -I

* Install and configure JDK for apache maven to make use of when building the java source code

--> amazon-linux-extras list (search for JDK package)

--> amazon-linux-extras install **java-openjdk11** –y

--> java –version

* Download Apache Maven, search for "**Download apache maven"** and copy the **.zip** package (latest)

--> cd /opt

--> wget "paste apache maven .zip file link"

--> ls

--> unzip "**apache-maven-3.6.3-bin.zip"**

--> cd "**apache-maven-3.6.3"** (This is the home directory for Apache Maven, in here you see all apache maven related binaries and configuration)

--> ls –l

--> cd bin

--> ./mvn --version (to check the apache maven version)

* We need to submit our Java application code to Apache Maven for the project build

--> cd ~ (Go to home directory where we have the bitbucket project repository to do the build)

--> cd "pipeline-project"

--> /opt/apache-maven-3.6.3/bin/mvn package (This helps you package the project and execute for build to get the build **Artifact.** And make sure you run this **package** command from the project directory)

* We need to check the build artifacts that was generated by maven

--> ls –l (You should see an extra folder that was created during the Build process "**target**" folder)

--> cd "target"

--> ls –l (You should see a **.war** file that was created by Maven)

--> mv "iwayQApp-1.0-RELEASE.war" "iwayQApp.war" (Rename .war file before deploying in Application servers)

--> Done Setting up Build Tool.

3) SETTING UP SonarQube Static Code Quality Analysis reporter and Scanning of code Vulnerabilities, Code Smells, bugs, Code duplicates and other things. Then prepares a report about findings, reports that also on the dashboard.

After SonarQube must have scan the reports it pushes those reports to a database so you can access then at any time, whenever you want to carry out code reviews. And SonarQube can be integrated with PostgreSQL or Oracle as a backend database to store these analysis reports.

A) INSTALL AND CONFIGURE PostgreSQL Database for SonarQube

* Launch an Amazon Linux EC2 instance

--> t2.medium (SonarQube requires at least 4GB RAM and 2 Processor for it to perform well)

--> Default Network

--> Tage=Sonar-Server

--> Create new Security Group for SonarQube (Open port 22 (SSH), 80 (http), 5432 (PostgreSQL), 9000 (SonarQube listens on port 9000), Source=Anywhere.

--> Launch Instances

--> SSH into server

There are pre-requisites you must meet to install and configure SonarQube. Visit <https://docs.sonarqube.org/latest/requirements/requirements/> or search for "**sonarqube prerequisites**". This will explain you the hardware requirements for **SonarQube**

* Amazon-linux-exras install postgresql11 –y

--> yum install postgresql-server postgresql-devel postgresql –y (Installing postgresql server package)

--> /usr/bin/postgresql-setup --initdb (We need to initialize the postgresql database)

--> systemctl start postgresql (We need to start the database)

--> systemctl status postgresql (check database status if it's running)

* Whenever you create a database one USER gets created automatically and you can check by running

--> tail –1 /etc/passwd I

--> su – postgres (Switch to the postgres user)

--> -bash-4.2$ **psql** (Run this command to successfully connect to the PostgresSQL Database)

* Now we have successfully connected to the database but we need to configure the database by creating one additional user and to grant this user privileges to the user to access database which SonarQube is going to use to store the analysis reports.

-->postgres=# CREATE USER sonar WITH PASSWORD 'sonar';

--> postgres=# ALTER USER sonar WITH SUPERUSER (Grant SUPERUSER role to sonar user)

--> postgres-# \du (Run this to verify the user you just created and existing users)

--> postgres-# \? (This will give you postgres commands)

--> postgres-# \q (Quit from database)

--> -bash-4.2$ exit

* We need to configure postgresql in such a way to provide access through a another configuration file sitted in **var/lib/pgsql/data** and the file is **pg\_hba.conf**

**-->** vi /**var/lib/pgsql/data/pg\_hba.conf**

**--> /peer or /ident (**Edit and change all the methods that have both **peer** and **ident,** replace them with **trust) (**This will give the **sonar** user access to the database**)**

* Let's create a Database which we'll use to store the analysis reports generated by the SonarQube scanner.   
  --> su – postgres

--> -bash-4.2$ **psql**

--> postgres=# CREATE DATABASE sonar;

--> postgres=# \l (Run this to list all the Databases running in this PostgresSQL and Priviledges)

--> postgres=# GRANT ALL PRIVILEGES ON DATABASE sonar TO sonar; (Run this command to grant the **sonar user** complete access to the **sonar database**)

--> postgres=# \l (Run this to list all the Databases running in this PostgresSQL and priviledges)

B) INSTALL AND CONFIGURE SonarQube Scanner

--> visit **"sonarqube downloads"** on the browser and copy the download link for the community version, or use this <https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.6.0.39681.zip>

--> cd /opt

--> wget <https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.6.0.39681.zip>

--> unzip "sonarqube-8.4.1.zip"

--> ls –l

--> cd "sonarqube-8.4.1"

--> amazon-linux-extras install java-openjdk11 –y

--> java –version

--> ls –l

--> adduser sonar (Create an application user so sonarqube can make use of the user because sonarqube runs on a on root user)

--> cd ~

--> chown -R sonar /opt/sonarqube-8.4.1.35646 (Change ownership of both directories to sonar)

--> su – sonar

--> cd /opt/sonarqube-8.4.1.35646/

--> ls -l

--> cd conf (to see the configuration files)

--> ls –l

--> vi sonar.properties (we need to edit this file and describe the postgresql details so sonar can use this file to connect with postgres and publish the analysis reports)

--> copy and paste the bellow block of code in the **sonar.properties** file

**sonar.jdbc.username=sonar**

**sonar.jdbc.password=sonar**

**sonar.jdbc.url=jdbc:postgresql://localhost/sonar**

**sonar.path.data=/var/sonarqube/data**

**sonar.path.temp=/var/sonarqube/temp**

--> Save and Quit

--> exit. (back to root)

We need to set some twining parameters or variables and apply some changes

--> vi /etc/sysctl.conf (Find the variables or parameters bellow and Add parameters at the bottom of file)

**vm.max\_map\_count=524288**

**fs.file-max=131072**

--> Save and Quit

--> sysctl –p (to view these variables or parameters)

--> vi /etc/security/limits.conf (Edit this file to add the limits, find limits bellow)

**sonar hard nofile 65535**

**sonar soft nofile 65535**

--> Save and Quit

--> su - sonar

* Let's START the SonarQube service using the **sonar.sh**

--> cd /opt/sonarqube-8.4.1.35646/

--> cd bin

--> cd linux-x86-64/

--> ls –l

--> ./sonar.sh start

To verify if it was successfully started we need to check the conar logs in the home directory

--> cd ../../logs

--> cat sonar.logs

--> ls

--> cat sonar.log. (Sonarcube service didn't start because it didn’t have the right permissions to create the **data** directory under **/var/sonarqube**)

--> exit (switch back to root)

--> mkdir /var/sonarqube

--> chown –R sonar:sonar /var/sonarqube

--> su – sonar (switch back to sonar user since we need to start sonarqube as a non-root user)

--> cd /opt/sonarqube-8.4.1.35646/

--> cd bin

--> cd linux-x86-64/

--> ls –l

--> ./sonar.sh start

To veryfy again if it was successfully started we need to check the conar logs in the home directory

--> su - sonar

--> cd /opt/sonarqube-8.4.1.35646/bin/linux-x86-64/

--> ls –l

--> ./sonar.sh status (If the STATUS shows RUNNING then you're good)

* NOW LET'S CONNECT TO THE SonarQube Dashboard

--> Get the public IP of the Sonar-Server and put on the browser (IP:9000)

--> Click on Login (Default Username=admin, Default Password=admin)

--> Create New Project (To get started), specify project key and display name as "**iwayq**"

--> Provide a token, type "**iwayq**" in the box and click **Generate**

**-->** Copy the **Token** and Save, click Continue

--> Specify source code Programming Language "**Java**"

--> Select Build tool "**Maven**" (It will automatically generate the code we'll need to provide when running the Maven Build so it will publish the source code to SonaQube for Code analysis)

--> COPY the block of CODE and save as well.

* WE NEED TO DO ANOTHER MAVEN BUILD AND PROVIDE THE PIECE OF CODE WE COPIED TO MAVEN FOR PUBLISHING.

--> SSH into your Build-Server

--> cd "Into the bitbucket project directory"

--> Run the block of command you copied from SonarQube (BEFORE THAT LOOK AT THE BELLOW SCREENSHOT AND RECONCILE, ADD WHAT NEED TO BE ADDED)

[root@ip-172-31—22—58 devops—pipeline—project]# —U sonar: sonar 
—Dsonar . projectKey=iwayq -Dsonar. host . // 54 .193.20.162 : 9000 —Dsonar . login=6bcd7acb01aOb 
0572a1fda10f5a8158dce470246 
[INFO] Scanning for projects ... 
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/p1ugins/maven-r 
etadata. xm1 
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/maven—metadata 

--> Make sure you specify the project directory in the above command

--> Build Successful and Analysis reports Publish in SonarQube

--> Refresh SonarQube Dashboard to view results

--> Explore Every Detail of the Report from TOP to Bottom

--> Done Do Not Tier Down any of the Infrastructure

4) SET-UP **Jfrog** ARTIFACTORY WHERE ALL OUR ARTIFACTS WILL BE STORED. But for the PIPELINE we'll make use of **Amazon S3** instead.

After **Apache Maven** does the build and publish it to **SonarQube** for Quality Control, If it passes the the test and everything **Apache Maven** outputs the Artifacts that gets generated to our **Jfrog Artifactory or (S3)** then that gets deployed in the Application servers running in Dev, Test and Production.

Jfrog is making use of Database in the backend to store all these different artifacts and it also does version control on our artifacts so the newer artifacts that gets generated would be deployed instead and we can fall back to previous artifacts versions.

Just like SonarQube, Jfrog provides you with a nice Dashboard where you can view all of your artifacts and meta data information.

* Launch an Amazon Linux AMI instance

--> t2.medium, default network, Create New SG for Jfrog, port 22=anywhere, 8081=anywhere, 8082=anywhere (Jfrog listens on these two ports)

--> SSH into Artifactory-Server

--> sudo -I

--> amazon-linux-extras install java-openjdk11 –y

--> java –version

* We need to Install the Jfrog Artifactory either copy the bellow commands and run on the terminal or search for "Artifactory oss" on Google, Go all the way to the bottom of the page, Select Linux, Select npm on the drop down and COPY the Command you are provided with and RUN on your Terminal.

**wget https://releases.jfrog.io/artifactory/artifactory-rpms/artifactory-rpms.repo -O jfrog-artifactory-rpms.repo; sudo mv jfrog-artifactory-rpms.repo /etc/yum.repos.d/; sudo yum update && sudo yum install jfrog-artifactory-oss**

--> service artifactory start

--> service artifactory status

--> Copy Artifactory server public IP (**IP:8082/artifactory**) . This will get you to the JFrog Artifactory Dashboard

--> Default Username=**admin**, Default Password=**password, Click on LOGIN**

--> Click Get Started

--> Reset your Password and NEXT

--> [http://artifactory-server-publicip:8082](http://artifactory-server-publicip:8082/) (Specify base URL), NEXT

--> NEXT (No Proxy at the moment)

--> Select Maven, NEXT (Most Important step)

--> Click Finish (It will setup all the required repositoeries for the Maven Build)

--> Once the setup is DONE, Make sure you're on **Application** (Left Panel)

--> Click Artifactory (drop down)

--> Click Artifacts (The artifacts will be stored in either **libs-release** or **libs-snapshot** that's either **snapshot** version or a **release** version). (SNAPSHOT version means it's ready to be deployed in the NON-PRODUCTION environment while if it's in RELEASE version it means it's ready to be deployed in PRODUCTION)

Artifactory also provides a specific URL which you'll use to fetch the files in a folder highrachy.

* INTEGRATE JFrog Artifactory with APACHE MAVEN

--> Click on Set Me Up (On top right)

--> Specify your password and hit ENTER, Click on Generate Maven Settings, Click on Generate Settings, (Scroll down you'll see the auto Generated XML file which you'll provide Maven with, this settings file will be deployed in the apache Maven Configuration)

--> Download **Setting.XML** file

--> Edit the **Setting.XML** file and change both usernames to admin and both passwords to your JFrog new Password.

--> Use port number 8082 in the **Setting.XML** file

--> create a **Setting.XML** file in the **.m2** folder and paste the content (in the Home directory of ROOT)

--> ETC

5) SET-UP CONTAINER PLATFORM FOR OUR APACHE TOMCAT APPLICATION.

* Launch an Amazon Linux EC2 instance

--> Default settings, Tag=App-Server, Create Security Group port 22=anywhere and 8080=anywhere

--> SSH into server

--> Sudo -I

--> yum install docker –y

--> docker –version

--> service docker start

--> service docker status

* To create application containers we need to have the applications image

--> Go to DockerHub search for "tomcat"

--> Copy the "docker pull tomcat" command

--> docker pull tomcat (make image available locally)

--> docker images

--> wget "artifactory artifact URL", under RELEASE

--> Remame artifact and give it "**iwayq.war**"

* Now that we have the artifact available we need to create a custom image where we'll package the entire application with other dependencies

--> vi Dockerfile (Paste the bellow block of code)

# Create Custom Docker Image

# Pull tomcat latest image from dockerhub

From tomcat:latest

# Maintainer

MAINTAINER "PR Reddy - iwayQ"

# copy war file on to container

COPY ./iwayq.war /usr/local/tomcat/webapps

--> Save and Quit

--> docker build . --tag iwayq-tomcat

--> docker images

--> docker run --name iwayq-con –p 8080:8080 –d iwayq-tomcat

--> docker ps (see container running, port mapped to the host and the container status)

--> copy puplic ip address of host machine (IP:8080/iwayq)

--> Play around with the UI a little (Sign Up and LOGIN)

6) SETTING UP ANSIBLE FOR AUTOMATIC CREATION OF CUSTOM DOCKER IMAGES AND PUSHING THOSE IMAGES TO OUR DOCKERHUB REGISTRY.

* Ansible has three man files Inventory file (HOLDS number of ansible clients), Modules (Tells what needs to be deployed) and Control Engine (What does the Orchestration)

--> Launch an Amazon EC2 instance

--> Default settings all through and Launch. Tag=Ansible

--> SSH into Ansible-Server

--> amazon-linux-extras install ansible2 –y

--> ansible --version

* We need to install some dependencies like python modules because ansible needs it to create docker images automatically using ansible playbooks.

--> yum install python-pip -y

--> pip install docker-py –y

--> Open the YAML file (Ansible Playbook) in a Text Editor

--> Edit the Artifactory details, DockerHub Registry details, Push to docker hub details, and provide your Own Details

* Go back to your Ansible Server

--> mkdir /dockercode

--> vi /dockercode/Dockerfile (Paste the bellow block of Code)

# Create Custom Docker Image

# Pull tomcat latest image from dockerhub

From tomcat:latest

# Maintainer

MAINTAINER "PR Reddy - iwayQ"

# copy war file on to container

COPY /artifactory/iwayq.war /usr/local/tomcat/webapps

--> Save and Quit

--> Copy the Ansible Playbook you Edited. (Go back to the server)

--> vi play.yaml

--> mkdir /artifacts

* Execute the ansible playbook

--> ansible-playbook play.yaml (Check execution steps)

--> Go to DockerHub and verify the image was pushed

--> Done

7) SETTING UP A 3 NODE KUBERNETES CLUSTER TO RUN OUR JAVA APPLICATION

* Since we're going to set up a Kubernetes cluster for our application, we don't need the App-Server we set-up earlier so

--> Terminate App-Server

* We'll launch 4 EC2 instances. 1 will be our Master Node, 3 will serve as worker Nodes

--> Launch 4 Amazon Linux AMI instances, t2.medium, New Security Group port

--> Port 22=anywhere, 6443=anywhere, 2379-2380=anywhere, 10250=anywhere, 10251=anywhere, 10252=anywhere, 30000-32767=anywhere. Total 7-SG rules.

--> Review and Launch

--> Name your Instances (1 k8sMaster and 3 Workers)

A) SSH into Kubernetes Master Node and Configure it

* We need to disable SELinux

--> SSH into Master Node

--> sudo –I

--> sentenceforce 0

--> sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

--> service iptables stop

--> modprobe br\_netfilter

* Run the bellow block of command to configure the k8s network twinables

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

--> sudo sysctl --system

* Run below block of command to add Kubernetes Repo to the yum repo so we'll be able to install those packages by calling them.

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

exclude=kubelet kubeadm kubectl

EOF

* Run below command to install the required Packages

--> yum install docker kubeadm kubectl kubelet --disableexcludes=kubernetes

* The packages have been installed, let's go ahead and start the docker engine

--> service docker start

--> chkconfig docker on (make it persistent, can withstand reboot)

--> service kubelet start

--> chkconfig kubelet on

* Run below block of command to configure CRI driver for Docker Run time

cat > /etc/docker/daemon.json <<EOF

{

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2"

}

EOF

* Run below command on Master node to initialize the kubernetes cluster

--> kubeadm init --pod-network-cidr=172.31.0.0/16 (Make Sure You're using Default VPC or else change the CIDR-BLOCK there)

--> COPY and Save the command underneath the last statement Output of this command which says **"Then you can join any number of worker nodes by running the following on each as root:"** It looks like the bellow picture

kubeadm join 172.31.77.56:6443 —token r5k7h2.rzlkqshp8flvwuvs \ 
—discovery-token-ca-cert-hash 
sha256:5c17 ac5e4649ce9d9314c4591430ef27b620a6e72f7066b8279b8b4dec891773 

--> You see the success message that says "Your Kubernetes control-plane has initialized successfully!"

* To start using your cluster, you need to run the following block of command as a regular user: (Find the command also on the Output section of the above command)

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

* We should now deploy a pod network to the cluster.

--> kubectl apply –f <https://docs.projectcalico.org/v3.14/manifests/calico.yaml>

--> kubectl config view

--> Master Node configuration is DONE

--> kubectl get nodes (currently there are no workers added to the master)

B) JOIN THE WORKER NODES TO THE CLUSTER

* We need to install the required packages as well in all the wrokers before we can join ADD them to the cluster (The installation is pretty much the same as the Master Node, the only difference in this case is, for nodes we won't run the initialize command)

--> SSH into Worker1, Worker2, Worker3

--> sudo –I

--> sentenceforce 0

--> sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

--> service iptables stop

--> modprobe br\_netfilter

* Run the bellow block of command to configure the k8s network twinables

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

--> sudo sysctl --system

* Run below block of command to add Kubernetes Repo to the yum repo so we'll be able to install those packages by calling them.

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

exclude=kubelet kubeadm kubectl

EOF

* Run below command to install the required Packages

--> yum install docker kubeadm kubectl kubelet --disableexcludes=kubernetes

* The packages have been installed, let's go ahead and start the docker engine

--> service docker start

--> chkconfig docker on (make it persistent, can withstand reboot)

--> service kubelet start

--> chkconfig kubelet on

* Use the **kubeadm join** Command to join this node to the cluster

--> COPY and RUN the **kubeadm join** you SAVED ABOVE (It looks like this: kubeadm join 172.31.77.56:6443 --token r5k7h2.rzlkqshp8flvwuvs --discovery-token-ca-cert-hash sha256:5c17ac5e4649ce9d9314c4591430ef27b620a6e72f7066b8279b8b4dec891773)

--> Check the Last Output Message to CONFIRM the node did join to the CLUSTER

* PERFORM THESAME STEPS ON THE OTHER WORKERS to ADD them to the CLUSTER

* GO BACK TO THE MASTER NODE

--> kubectl get nodes (You should see 4 nodes, 1 master 2 workers)

--> Our cluster is up, now let's deploy our application

But Before we deploy the application we need to get our YAML Manifest definition file and we need two manifest definition files. One to deploy the PODS and the other to deploy the NODE Port service. Open the **create-k8s-deployment.yaml**

--> Open the **create-k8s-deployment.yaml** in a text Editor

--> Edit the container image value and provide your **dockerhub-username/image:tag**

--> Copy the deployment file content and go back to the master node

--> cd ~

--> vi deployment.yaml (PASTE The content and SAVE)

--> kubectl apply –f deployment.yaml (deploy the deployment definition manifest file)

--> kubectl get deployments

--> kubectl get pods (Our application is running in three worker nodes and three pods)

LETS DEPLOY THE NODE-PORT SERVICE

--> Open the **nodePort.yaml** file in a text Editor

--> COPY the nodePort.yaml service content

--> Go back to the Master Node

--> vi nodeport.yaml (PASTE the content you copied above)

--> kubectl apply –f nodeport.yaml (Deploy the nodeport service)

--> kubectl get service

--> DONE (We've successfully deployed our application on a Kubernetes cluster)

* ACCESS APPLICATION

--> COPY the public IP of any of the worker nodes (IP:31000/iwayq)

--> Click on Login Here and Register a New Account

--> We can as well Introduce an ALB inthis architecture and also Rout53