

Implement production-grade CI/CD DevOps Principles

Name : Sadia Zaman Nishat

Employee ID : 00-30087

Date: 24.07.2023

Revision History

Ver.Doc	Date	Author	Reviewer
01	17.07.2023	Sadia Zaman Nishat	Nani San, Dewan San
02	21.07.2023	Sadia Zanan Nishat	Nani San, Dewan San

Contents

Subject	Page
1. Project Introduction	4
1.1 Project definition :	4
1.1.1 Purpose and goals of the project:	4
1.1.2 Project scopes:	4
1.1.2 Project Constraints:	4
1.2 Requirements Analysis:	5
2. Manual Deployment	6
2.1 Definition	6
2.2 Architecture Diagram	6
2.4 Configure the required installation	7
2.5 Configure the database settings	8
3. On Premise Server Provision and Deployment	8
3.1 Definition	8
3.3 Prepare three vm	9
3.4 Configure the required installation on single vm	10
3.4.1 Install ansible	10
3.4.2 Install docker & write Dockerfile	10
3.4.3 Install Jenkins	14
3.5 Kubernetes Set up in Master & Worker vm	15
3.5.1 Create frontend-deployment.yml file	23
3.5.2 Create mysql-deployment.yml file	25
3.5.3 Create mysql-pv.yml file	27
3.6 Create Jenkins pipeline	28
4. Aws Server Provision and Deployment	31
4.1 Definition	31
4.3 Create VPC network	32
4.4 Create RDS	33
4.5 Create ec2	33
4.5.1 Create security group	33
4.5.2 Create target group	35
4.5.3 Configure Application Load Balancer	35
4.5.3 Configure Auto Scaling group	36
4.5.4 Configure Route 53	37
4.6 Create Jenkins Pipeline	38
4.6.1 Deploy an application in aws	38
5. AWS EKS-Managed Service	40
5.1 Definition	40
5.2 Design architecture	40
5.3 Explain on deployment process	41
6. References	45

1. Project Introduction

1.1 Project definition :

This Project is to implement production-grade CICD devops Principles. It focuses on DevOps automation, aiming to deploy a basic application on an on-premises Kubernetes cluster and AWS Cloud utilizing a Jenkins pipeline. All configuration files, settings, and scripts are stored and managed in a Git repository for efficient version control and convenient accessibility. Need to deploy spring boot project with the proper CI/CD pipeline by manually, on premises server provision and deployment, aws server provision and deployment with the AWS EKS managed service.

1.1.1 Purpose and goals of the project:

To build the project and deploy successfully is the main purpose of this project and every user can access properly with the expected result is the main goal of this project.

1.1.2 Project scopes:

Application Features: Specify the features and functionality that a Spring Boot application must have. Outline the user needs and the essential business logic.

Deployment Environment: Identify the deployment's target environment. On-site servers, cloud services, or a combination of the two could be used. Determine the database technology and configuration required for the application to efficiently store and retrieve data.

Scalability: Take into account the anticipated user load and decide whether the system needs to be scalable to meet future traffic growth. Address security issues, such as those with data protection, access regulation, and authentication procedures.

1.1.2 Project Constraints:

Budget: The resources, infrastructure, and tool options that can be deployed may be influenced by the project's financial constraints.

Time: Depending on the project's urgency, there may be time restrictions that affect the deployment method or the complexity of the CI/CD pipeline.



Infrastructure: If the project requires the supply of on-premises servers, the hardware resources may be constrained, and scalability may be more difficult than with cloud-based solutions.

Technical Knowledge: The team's skill set and knowledge may be able to reduce the complexity of the deployment process and CI/CD pipeline that is ultimately chosen.

Regulatory Compliance: Compliance requirements may place restrictions on deployment strategies if the application handles sensitive data or is subject to particular regulations.

Network and Security Policies: The organization's current network and security policies may have an impact on the deployment strategy.

1.2 Requirements Analysis:

Defining the needs and expectations for the project is part of the requirements analysis process for a Spring Boot deployment project with a CI/CD pipeline using manual deployment and on-premises server provisioning with AWS ECS or EKS. An overview of the prerequisites is given below:

1. **Requirements for Applications:** The Spring Boot application's features, functionality, and user stories should be precisely defined. Indicate any APIs or integrations with external services that the application requires.

2. **Environment for Deployment:** Make a decision regarding the application's deployment—on-premises, via EKS (Elastic Kubernetes Service). Choose the region(s) and availability zone(s) where the application should be deployed if using AWS EKS.

3. **Requirements for CI/CD Pipelines:** To handle the source code for the application, use a version control system (like Git). Install a CI/CD server to automate the build, test, and deployment processes (such as Jenkins, Github CI/CD, or AWS CodePipeline). Specify the build procedure for packaging the Spring Boot application as a Docker image for deployment on Kubernetes or AWS EKS.

2. Manual Deployment

2.1 Definition

A Spring Boot project can be manually deployed in an Ubuntu Oracle VM by moving its executable JAR file there, installing Java Runtime Environment (JRE), configuring environment variables, and running the executable JAR file from the command line. This approach needs manual setup and configuration tasks, such as setting up the application's dependencies and settings. It is appropriate for developers with a preference for doing things yourself and experience with deployment setups.

2.2 Architecture Diagram

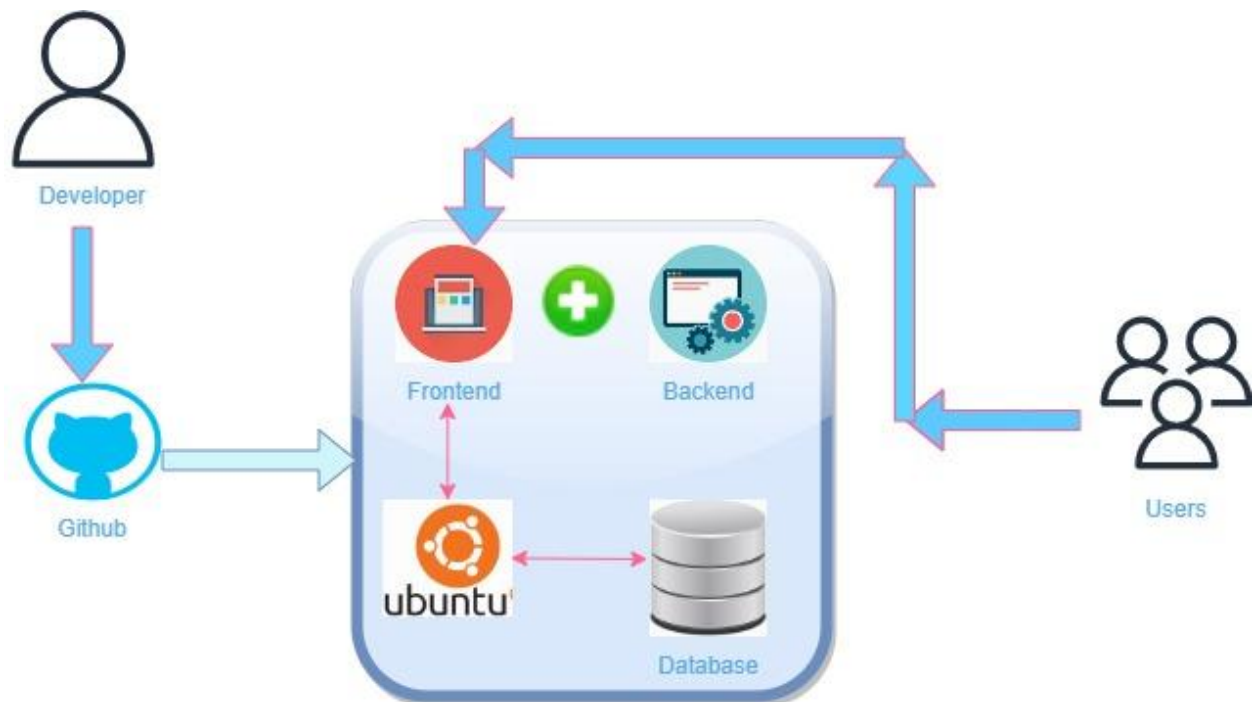


Fig 2.2 : Manual Deployment

2.3 Prepare One Virtual machine

To deploy a spring boot project manually, I'll use one ubuntu vm in oracle virtualbox. In order to install Ubuntu on an Oracle VM, do the following:

1. On the operating system serving as host machine, download and install Oracle VM VirtualBox.
2. Download an ISO image of Ubuntu from the official website.
3. To create a new virtual machine, launch VirtualBox and select "New".
4. Give the virtual machine a name, choose "Linux" as the operating system type, and "Ubuntu (64-bit)" as the version.
5. Set up a virtual hard drive and allot memory.
6. Adjust the network and storage settings for the virtual machine.
7. Launch your virtual machine and choose the Ubuntu ISO file as your installation medium.
8. To install Ubuntu, follow the installation wizard's instructions.
9. After installation, Ubuntu can be used in Oracle VM for a variety of functions.

2.4 Configure the required installation

2.4.1 Install jdk : `apt install jdk-1.8.0`

2.4.2 Install mysql : `apt install mysql`

2.4.3 Install maven : `apt install maven`

2.4.4 Install Git : `apt install git`

Now,

- ★ Clone the repo:

Git clone

https://github.com/Nishat792/YSD_Bo2_DevOps_Library_Management_System

- ★ Remove test repo under src repo: `rm -rf test`
- ★ Build application:
`mvn clean install`
- ★ Run this command under target repo:
`java -jar spark-lms-0.0.1-SNAPSHOT.jar`

2.5 Configure the database settings

Use following commands to configure database:

- ★ **mysql**
- ★ **show databases;**
- ★ **create database sparklmsdb;**

Use following command if there is any password issue:

- ★ **USE mysql;**
- ★ **UPDATE user SET plugin='mysql_native_password' WHERE User='root';**
- ★ **FLUSH PRIVILEGES;**

3. On Premise Server Provision and Deployment

3.1 Definition

To deploy a Spring Boot project on an on-premises server, I need to create three virtual machines (VMs), one running Ubuntu and the other two Alma Linux. To manage containerized apps, manually set up Kubernetes on two(master and worker) Alma Linux virtual machines. Ansible automation is used to set up Ansible, Docker, and Jenkins on the Ubuntu VM. Jenkins serves as the continuous integration and continuous deployment (CI/CD) tool, while Docker makes it possible to containerize apps. Ansible streamlines configuration and deployment processes. With this configuration, the Spring Boot project may be managed and deployed effectively in an on-premises setting.

3.2 Design architecture

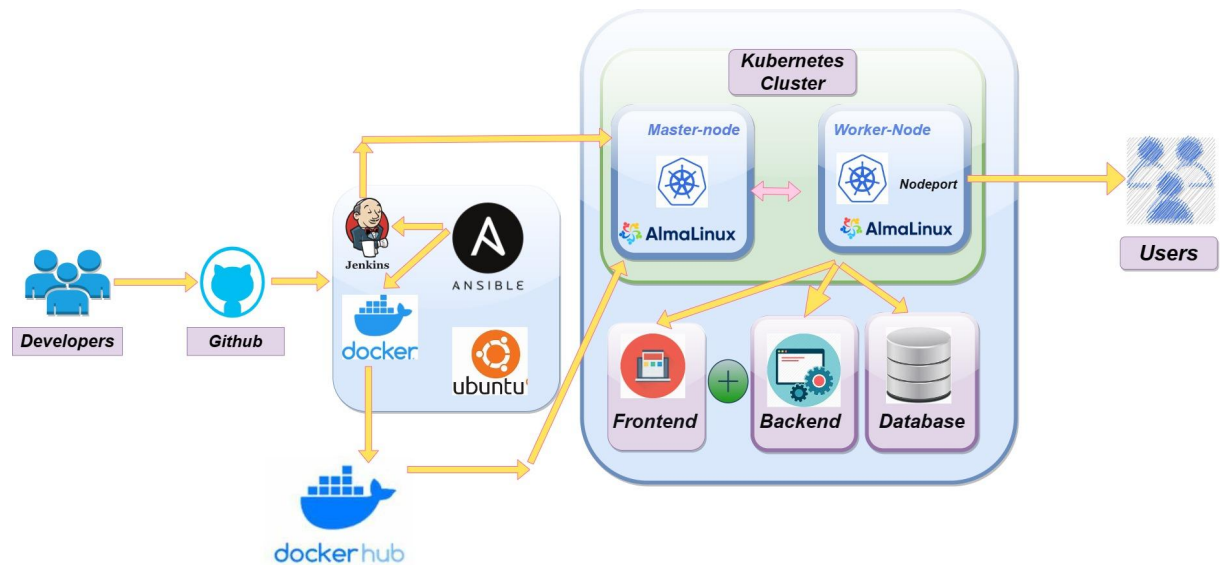


Fig 3.2 : On Premise Server Provision and Deployment

3.3 Prepare three vm

One is Ubuntu and Other two vms are Alma Linux.

Ubuntu - Install ansible, docker and jenkins.

Alma Linux(Master & Worker) - Kubernetes set up manually.

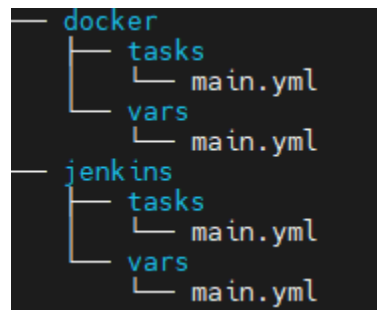
3.4 Configure the required installation on single vm

3.4.1 Install ansible

- ★ Configure the PPA Repository
sudo apt-add-repository ppa:ansible/ansible
- ★ Update the repository and install Ansible
sudo apt update
sudo apt install ansible -y

3.4.2 Install docker & write Dockerfile

Need to install docker using ansible playbook.



```
vi /etc/ansible/roles/docker/tasks/main.yml
```

```
---  
  
- name: Remove Docker-related packages  
  apt:  
    name:  
    - docker.io  
    - docker-doc
```

```

- docker-compose
- podman-docker
- containerd
- runc

state: absent

- name: Run apt-get update
  apt:
    update_cache: yes
- name: Install ca-certificates, curl, gnupg
  apt:
    name:
      - ca-certificates
      - curl
      - gnupg
    state: present
- name: Create /etc/apt/keyrings directory
  file:
    path: /etc/apt/keyrings
    state: directory
    mode: '0755'
- name: Download Docker GPG key
  get_url:
    url: https://download.docker.com/linux/ubuntu/gpg
    dest: /etc/apt/keyrings/docker.gpg
    mode: '0644'

- name: Import Docker GPG key
  shell: gpg --dearmor -o /etc/apt/keyrings/docker.gpg.gpg /etc/apt/keyrings/docker.gpg
  args:
    creates: /etc/apt/keyrings/docker.gpg.gpg

- name: Rename Docker GPG key
  command: mv /etc/apt/keyrings/docker.gpg.gpg /etc/apt/keyrings/docker.gpg
  args:
    creates: /etc/apt/keyrings/docker.gpg

- name: Set read permissions for Docker keyring
  file:

```

```

    path: /etc/apt/keyrings/
    mode: '0644'
- name: Get architecture
  command: dpkg --print-architecture
  register: architecture

- name: Install Docker CE dependencies
  apt:
    name:
      - apt-transport-https
      - ca-certificates
      - curl
      - gnupg
      - lsb-release
    state: present

- name: Add Docker GPG key
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present

- name: Add Docker repository
  apt_repository:
    repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu {{
ansible_distribution_release }} stable
    state: present

- name: Update apt cache
  apt:
    update_cache: yes

- name: Install Docker packages
  apt:
    name:
      - docker-ce
      - docker-ce-cli
      - containerd.io
      - docker-buildx-plugin
      - docker-compose-plugin

```

After installing docker, I created Dockerfile to containerize my application.

```
FROM openjdk:8-jre
ADD target/spark-lms-0.0.1-SNAPSHOT.jar spark-lms-0.0.1-SNAPSHOT.jar
EXPOSE 8086
ENTRYPOINT ["java", "-jar", "spark-lms-0.0.1-SNAPSHOT.jar"]
```

Using OpenJDK 8 as the base image, the following Dockerfile is used to construct a Docker image for a Spring Boot application, specifically for the "spark-lms" project. Let's analyze each line of the Dockerfile to determine what it does:

FROM :8-jre openjdk:

The base image to be used for the Docker image is specified on this line. In this instance, the Java Runtime Environment (JRE) image is the official OpenJDK 8 image. The Java runtime environment required to run the Spring Boot application is provided by the image.

Create a target for spark-lms-0.0.1-SNAPSHOT.jar.

The ADD command transfers the JAR file for the Spring Boot application (in this case, "spark-lms-0.0.1-SNAPSHOT.jar") from the host machine's target directory into the Docker image. The intended JAR file will be added to the image's root directory using the same name.

EXPOSE 8086:

The EXPOSE command tells Docker that the container is now listening on the given network ports. It means that the Spring Boot application running within the container will listen on port 8086 in this instance. However, this merely records that the container uses this port; it does not reveal the port to the host.

Entrance Point ["java", "-jar", "spark-lms-0.0.1-SNAPSHOT.jar"]:

The default command that will be executed when the container starts is specified by the ENTRYPOINT instruction. In this instance, it sets up the container to run the Java command with the "-jar" option, launching the "spark-lms-0.0.1-SNAPSHOT.jar" file as the Spring Boot application's entry point.

3.4.3 Install Jenkins

Need to install Jenkins using an ansible playbook.

```
vi /etc/ansible/roles/jenkins/tasks/main.yml
```

```
---
- name: Update apt cache
  apt:
    update_cache: yes

- name: Install OpenJDK 11
  apt:
    name: openjdk-11-jre
    state: present

- name: Check Java version
  command: java -version
  register: java_version_output
  changed_when: false

- name: Download Jenkins repository key
  get_url:
    url: https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
    dest: /usr/share/keyrings/jenkins-keyring.asc
    mode: '0644'

- name: Add Jenkins repository to sources list
  blockinfile:
    dest: /etc/apt/sources.list.d/jenkins.list
    block: |
      deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable
binary/

- name: Update apt cache
  apt:
    update_cache: yes

- name: Install Jenkins
```

```
apt:
  name: jenkins
  state: present

- name: Start Jenkins service
  service:
    name: jenkins
    state: started
    enabled: true
```

vi /etc/ansible/project.yml

```
- name: Setup Jenkins and Docker
  hosts: localhost
  become: true
  roles:
    - docker
    - jenkins
```

To run those playbooks use following command:

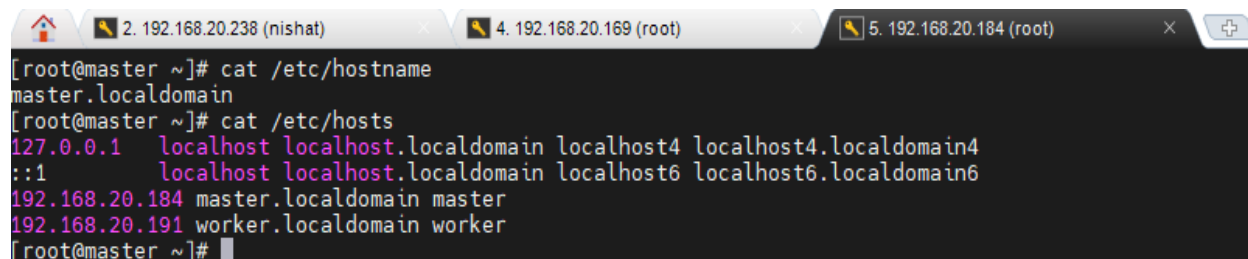
```
ansible-playbook project.yml
```



3.5 Kubernetes Set up in Master & Worker vm

Prerequisites	Setting up Systems
<ul style="list-style-type: none"> ★ At least two VMs(one for master and other for worker) ★ A non-root user with root/administrator privileges. 	<ul style="list-style-type: none"> ★ Setup correct /etc/hosts file ★ Setup Firewall ★ Setup SELinux ★ Enable kernel Modules ★ Disable SWAP

Setup hosts and hostname in master node:

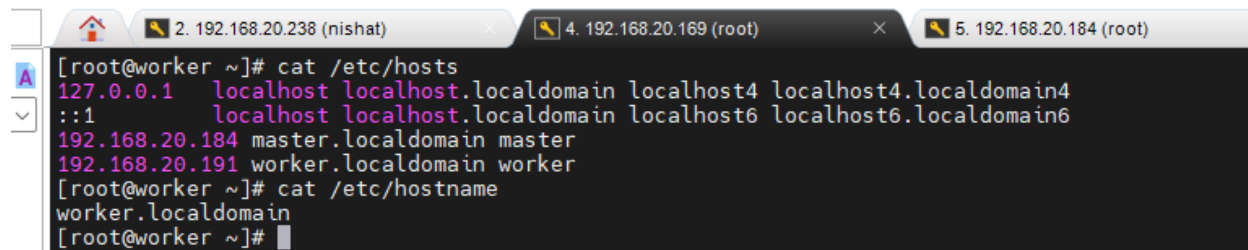


```

[2. 192.168.20.238 (nishat)] [4. 192.168.20.169 (root)] [5. 192.168.20.184 (root)]
[root@master ~]# cat /etc/hostname
master.localdomain
[root@master ~]# cat /etc/hosts
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.20.184 master.localdomain master
192.168.20.191 worker.localdomain worker
[root@master ~]#

```

Setup hosts and hostname in Worker node:



```

[2. 192.168.20.238 (nishat)] [4. 192.168.20.169 (root)] [5. 192.168.20.184 (root)]
[root@worker ~]# cat /etc/hosts
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.20.184 master.localdomain master
192.168.20.191 worker.localdomain worker
[root@worker ~]# cat /etc/hostname
worker.localdomain
[root@worker ~]#

```


Setup Firewall:

```

[root@master ~]# systemctl status firewalld
○ firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; preset: enabled)
   Active: inactive (dead)
     Docs: man:firewalld(1)
[root@master ~]#

```

Setup SELinux:

```
sudo setenforce 0
```

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

```
sestatus
```

```

#
# To revert back to SELinux enabled:
#
# grubby --update-kernel ALL --remove-args selinux
#
SELINUX=disabled
# SELINUXTYPE= can take one of these three values:
#   targeted - Targeted processes are protected,
#   minimum - Modification of targeted policy. Only selected processes are protected.
#   mls - Multi Level Security protection.
SELINUXTYPE=targeted

```

Enable kernel Modules:

Run the following command to enable the kernel modules "overlay" and "br_netfilter".

```
sudo modprobe overlay
```

```
sudo modprobe br_netfilter
```

To make it permanent, create the configuration file to "/etc/modules-load.d/k8s.conf". This will enable Linux systems to enable kernel modules during the system boot.

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
```

```
overlay
```

```
br_netfilter
```

EOF

Next, create the systemctl params required using the following command.

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF
```

To apply the new sysctl configuration without reboot, use the following command. We should get the list of default sysctl params on the system and be sure to get sysctl params that just added in the file "k8s.conf".

```
sudo sysctl --system
```

Disable SWAP:

To disable SWAP, we will need to comment on the SWAP configuration on the "/etc/fstab" file. This can be done by using the single command via sed (stream editor) or manually editing the /etc/fstab file.

```
sudo sed -i ' / swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Now turn off the SWAP on the current session using the below command. Then, verify the SWAP is off using the "free -m" command. we should see the SWAP has "0" values, which means it's now disabled.

```
sudo swapoff -a
```

```
free -m
```

Installing Container Runtime: Containerd:

Before adding the repository, run the DNF command below to install additional tools "dnf-utils".

```
sudo dnf install dnf-utils
```

Now run the following command to add the Docker repository for CentOS-based systems.

```
sudo yum-config-manager \  
--add-repo \  
https://download.docker.com/linux/centos/docker-ce.repo
```

Run the following command to check and verify repositories on Rocky/Alma Linux systems. Then, generate a new metadata cache. We will see the Docker repository is added to Rocky Linux.

```
sudo dnf repolist  
sudo dnf makecache
```

Next, install the containerd package using the DNF command below. Input **Y** to confirm the installation and press **ENTER**, and the installation will begin. When asked to import the Docker GPG key, input **y** to add.

```
sudo dnf install containerd.io
```

After installation is finished, run the following command to back up the default containerd configuration and generate a new containerd config file.

```
sudo mv /etc/containerd/config.toml /etc/containerd/config.toml.orig  
sudo containerd config default > /etc/containerd/config.toml
```

Now modify the containerd config file "/etc/containerd/config.toml" using the following command.

```
sudo nano /etc/containerd/config.toml
```



Change the value of cgroup driver "**SystemdCgroup = false**" to "**SystemdCgroup = true**". This will enable the systemd cgroup driver for the containerd container runtime.

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
```

```
...
```

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
```

```
SystemdCgroup = true
```

When it is finished, save and close the file.

Next, run the following systemctl command to start and enable the containerd service. This will add the containerd service to be run automatically at system boot.

```
sudo systemctl enable --now containerd
```

Lastly, check and verify the containerd service using the below command. We should see the containerd is enabled and will be run automatically at system boot. And the current status of containerd service is running.

```
sudo systemctl is-enabled containerd
```

```
sudo systemctl status containerd
```

Installing kubernetes Packages:

Run the following command to add the Kubernetes repository for RHEL/CentOS-based operating systems.

```
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
```



```

pgpkey=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

```

Now check the list of repositories and generate a new metadata cache using the `dnf` command below. And we should see the Kubernetes repository is added to Rocky/alma Linux systems.

```

sudo dnf repolist
sudo dnf makecache

```

Next, install Kubernetes packages using the following `dnf` command. Input Y to confirm the installation and press ENTER to continue, and the installation will begin.

```

sudo dnf install kubelet kubeadm kubectl --disableexcludes=kubernetes

```

This service is the main Kubernetes service that will wait for any events when we initialize the cluster or join the node to the cluster.

```

sudo systemctl enable --now kubelet

```

Initializing Kubernetes Control plane:

Before initializing the Control Plane node, run the following command to check the "br_netfilter" kernel modules are enabled. If we get an output from the command, it means the "br_netfilter" module is enabled.

```

lsmod | grep br_netfilter

```

Next, run the following command to download images that are required for the Kubernetes Cluster. This command will download all container images that are needed for creating Kubernetes Cluster such as coredns, kube-api server, etcd, kube-controller, kube-proxy, and the pause container image.

```

sudo kubeadm config images pull

```



After the download is finished, run the following "kubeadm init" command to initialize the Kubernetes Cluster on the "master" server. This node "master" will automatically be selected as the Kubernetes Control Plane because this is the first time initializing the cluster.

```
sudo kubeadm config kubeadm init --control-plane-endpoint=192.168.20.184
--pod-network-cidr=10.10.0.0/16
--cri-socket=unix:///run/containerd/containerd.sock
pull
```

Before you start using the Kubernetes Cluster, you will need to set up the Kubernetes credentials. Run the following command to set up the Kubernetes credentials.

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Now you can use the "kubectl" command to interact with your Kubernetes cluster. Run the following "kubectl" command to check the Kubernetes Cluster information. And you should see the Kubernetes control plane and the coredns running.

```
kubectl cluster-info
```

Installing CNI Plugin:Calico

```
wget
https://raw.githubusercontent.com/projectcalico/calico/master/manifests/calico.ya
ml
```

```
kubectl apply -f calico.yaml
```

Check the list of running pods on your Kubernetes using the following command. If your Kubernetes installation is successful, you should see all main pods for Kubernetes which include Flannel pods are running.

```
kubectl get pods --all-namespaces
```



Adding Worker Nodes to Kubernetes:

After initializing the Kubernetes Control Plane on the "k8s-master" server, you will add worker nodes "worker1"

```
kubectl get nodes -o kubeadm join 192.168.20.184:6443 --token
4b9ron.d8obxeokwthzomgk \
--discovery-token-ca-cert-hash
sha256:6024e5f28537a1464bf93334e7bd018e72906733e93e400f99639667a0d4bf9b
```

Lastly, check and verify all available nodes on the Kubernetes Cluster using the 'kubectl' command below. You should see the 'master' server is running as the Kubernetes Control Plane, and the "worker1" servers are running as the worker node.

```
kubectl get nodes -o wide
```

3.5.1 Create frontend-deployment.yml file

After setting up the kubernetes, I created a **deployment.yml** file for my frontend part.

```
apiVersion: v1
kind: Service
metadata:
  name: webapp-sql
spec:
  selector:
    app: webapp-sql
    tier: frontend
  ports:
    - protocol: "TCP"
      port: 80
      targetPort: 8080
      nodePort: 30090
      type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

```

name: webapp1
labels:
  app: webapp-sql
  tier: frontend
spec:
  selector:
    matchLabels:
      app: webapp-sql
      tier: frontend
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp-sql
        tier: frontend
    spec:
      containers:
        - name: webapp1
          image: nishat792/ysd-lib-manage
          imagePullPolicy: Always
          ports:
            - containerPort: 8080

```

This YAML file specifies a Kubernetes Service and Deployment for the "webapp-sql" tier's "webapp1" frontend application. The application is set to use a specific Docker image downloaded from Docker Hub, and it is exposed via a NodePort Service type.

The YAML file's sections are described in the following detail:

Service:

This specifies the Kubernetes API version used for this resource, which is apiVersion: v1.

kind: Service: Indicates that this resource belongs to a Kubernetes Service.

metadata: contains information about the Service, such as its name, in this case "webapp-sql".

spec: Specifies the requirements for the Service.

selector: The labels chosen by this Service to choose the pods it will target. The Service will in this instance focus on pods identified by the labels app: webapp-sql and tier: frontend.

Ports: specifies which ports the Service should listen on (ports).



protocol: The Service's chosen protocol, in this case TCP.

port: The Kubernetes cluster's default port (80) on which the Service will listen.

targetPort: Port 8080 on the pods is the port to which the Service directs traffic.

nodePort: The port (port 30090) on the Kubernetes nodes through which the Service is made accessible to the outside world.

This specifies that the Service should be of type "NodePort," making it reachable from the outside world via the node's IP address and the designated nodePort (port 30090).

Deployment:

ApiVersion: apps/v1: This resource's Kubernetes API version is indicated by the string.

Kind: Deployment: Indicates that this resource belongs to a Kubernetes Deployment. Contains information on the deployment, such as labels and its name, which is "webapp1" in this instance.

spec: Specifies the requirements for the Deployment.

selector: The labels that are used to choose which pods this deployment manages. In this instance, it chooses pods with the labels tier: frontend and app: webapp-sql.

replicas: 1: Specifies the desired number of application instances (replicas), in this example one.

template: Provides details on the replica pod template.

metadata: Pods made using this template's app: webapp-sql and tier: frontend labels are included here.

spec: Specifies the requirements for the pod.

Containers: explains the containers that will operate inside the pod.

url: webapp1 The container's name.

image:nishat792/ysd-lib-manage The "nishat792/ysd-lib-manage" Docker image was taken from the Docker Hub repository to be used for the container.

imagePullPolicy: Always: This states that the container should always start with the most recent image.

Ports: specifies the ports that the container has made available. Port 8080 is exposed in this situation.

In conclusion, the YAML file defines a Kubernetes Service called "webapp-sql," which externally exposes the frontend application "webapp1" on port 30090. In addition, it establishes a deployment with a single replica running the container

"webapp1" with the chosen Docker image. Inside the container, the application is running on port 8080, and the service routes external traffic to this

3.5.2 Create mysql-deployment.yml file

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  selector:
    app: webapp-sql
    tier: backend
  ports:
    - protocol: "TCP"
      port: 3306
      targetPort: 3306
  clusterIP: None
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: sqldb
  labels:
    app: webapp-sql
    tier: backend
spec:
  selector:
    matchLabels:
      app: webapp-sql
      tier: backend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: webapp-sql
        tier: backend
    spec:
```

```

containers:
- image: mysql
  name: mysql
  env:
  - name: MYSQL_ROOT_PASSWORD
    value: password
  - name: MYSQL_DATABASE
    value: sparklmsdb
  - name: MYSQL_USER
    value: db_user
  - name: MYSQL_PASSWORD
    value: password
  args: ["--default-authentication-plugin=mysql_native_password"]
  ports:
  - containerPort: 3306
  volumeMounts:
  - name: mysql-persistent-storage
    mountPath: /var/lib/mysql
  volumes:
  - name: mysql-persistent-storage
    persistentVolumeClaim:
      claimName: mysql-pv-claim

```

This YAML configuration file creates a Kubernetes Service and Deployment to run a MySQL database as a backend service inside the cluster. Through the "mysql" Service, the database will be accessible to other pods in the cluster. Environment variables for database setup and data storage on persistent storage volumes are pre-configured in the MySQL container. The backend tier can be chosen and handled by the associated Service and Deployment because it is identified with the labels app: webapp-sql and tier: backend.

3.5.3 Create mysql-pv.yml file

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi

```

In the Kubernetes cluster, this YAML file configures a persistent storage solution for a MySQL database. The physical storage existing on the host machine is represented by the PV ("mysql-pv-volume"). The MySQL pods use the PVC ("mysql-pv-claim") to request a piece of that storage and to store and persist data.

3.6 Create Jenkins pipeline

Follow these steps to create & build a Jenkins pipeline job:

1. Click "New Item" when the Jenkins dashboard is open.
2. Choose "Pipeline" as the job type, then enter a name for the position.
3. Select whether to define the pipeline script directly in the pipeline configuration or to obtain it via a version control system.
4. Write the pipeline code in the Script block if you're using a script. If not, be specific about the repository and branch.
5. Set up additional job parameters such as triggers, builds, and post-build actions.
6. Click "Build Now" to launch the pipeline after saving the job setup.

3.6.1 Deploy an application:

This is my pipeline job for this project to deploy on-premises:

```
pipeline {
  agent any

  triggers { pollSCM 'H/2 * * * *' }
  stages {
    stage('Git clone') {
      steps {
        git branch: 'main', credentialsId: 'git-clone', url:
'https://github.com/Nishat792/YSD_Bo2_DevOps_Library_Management_System.git'
      }
    }
    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Build Image') {
      steps{
        script{
          new_image=docker.build("nishat792/ysd-lib-manage")
        }
      }
    }
  }
}
```

```

}
stage('Push Image to DockerHub') {
  steps {
    script{

      withDockerRegistry(credentialsId: 'dockerhub-push',url: '') {
        new_image.push()
      }
    }
  }
}
stage('Deploy image'){
  steps{
    script{
      kubeconfig(caCertificate: "", credentialsId: 'k-deploy', serverUrl: "") {

        try {
          sh 'kubectl delete -f frontend-deployment.yaml'
          sh 'kubectl delete -f mysql-deployment.yaml'
          sh 'kubectl delete -f mysql-pv.yaml'
        } catch (Exception e) {
          echo "An error occurred while executing kubectl commands:
${e.message}"
        }

        sh 'kubectl apply -f mysql-pv.yaml'
        sh 'kubectl apply -f frontend-deployment.yaml'
        sh 'kubectl apply -f mysql-deployment.yaml'

      }
    }
  }
}
}

```

Description of on-premises Jenkins pipeline:

1. In the first stage clone the repo from github
2. In the next stage I used maven build for my project.
3. Then, I did image build using docker.build().
4. After that, I push the image in the docker registry which is docker-hub by creating credentials. Here, to create credentials I used docker-hub username.
5. Finally, the last stage is about deploying an image where I use try catch to handle exceptions.

After building jenkins pipeline properly, the pods and services are being runned like this:

```

Last login: Thu Jul 13 12:01:47 2023 from 192.168.20.208
[root@master ~]# kubectl get pods -o wide
NAME                                READY    STATUS    RESTARTS   AGE    IP              NODE                NOMINATED NODE    READINESS GATES
nginx-deployment-57d84f57dc-ds64j   1/1      Running   9 (69m ago)  20d    10.10.209.207   master.localdomain   <none>             <none>
nginx-deployment-57d84f57dc-fxgdl   1/1      Running   9 (69m ago)  20d    10.10.209.210   master.localdomain   <none>             <none>
nginx-deployment-57d84f57dc-r8jdr   1/1      Running   9 (69m ago)  20d    10.10.209.209   master.localdomain   <none>             <none>
sqldb-5c6667d64c-4h8l4             1/1      Running   0           4h35m  10.10.84.58     worker.localdomain   <none>             <none>
webapp1-68fcc7dd7b-5hxjx           1/1      Running   0           4h35m  10.10.84.60     worker.localdomain   <none>             <none>
[root@master ~]# kubectl get svc
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes ClusterIP   10.96.0.1     <none>         443/TCP    27d
mysql     ClusterIP   None          <none>         3306/TCP   20h
nginx-service NodePort    10.100.185.216 <none>        80:30080/TCP 26d
webapp-sql NodePort    10.103.191.248 <none>        80:30090/TCP 20h
[root@master ~]# kubectl get deployments
NAME          READY    UP-TO-DATE    AVAILABLE    AGE
nginx-deployment 3/3      3             3            26d
sqldb          1/1      1             1            20h
webapp1        1/1      1             1            20h
[root@master ~]# kubectl get pvc
NAME      STATUS    VOLUME    CAPACITY    ACCESS MODES    STORAGECLASS    AGE
my-pvc    Bound     my-pv     1Gi         RWO             manual          23d
mysql-pv-claim Bound     mysql-pv-volume 2Gi         RWO             manual          20h
[root@master ~]# kubectl get pv
NAME      CAPACITY    ACCESS MODES    RECLAIM POLICY    STATUS    CLAIM                STORAGECLASS    REASON    AGE
my-pv     1Gi         RWO             Retain            Bound     default/my-pvc       manual          23d
mysql-pv-volume 2Gi         RWO             Retain            Bound     default/mysql-pv-claim manual          20h
[root@master ~]#

```

4. Aws Server Provision and Deployment

4.1 Definition

There are various processes involved in deploying a Spring Boot project on AWS. To isolate resources, first construct a Virtual Private Cloud (VPC). After that, run an EC2 instance for the application and configure a relational database (RDS) instance to store data. Make a Jenkins pipeline for continuous integration and deployment after that. For high availability and scalability, set up an application load balancer (ALB) to spread traffic among various instances. To dynamically change the number of instances based on traffic, use auto scaling. Last but not least, set up Route 53 to manage DNS so that users may access the application using a unique domain name. With this extensive AWS configuration, the Spring Boot project is guaranteed a dependable, scalable, and highly available environment.

4.2 Design architecture

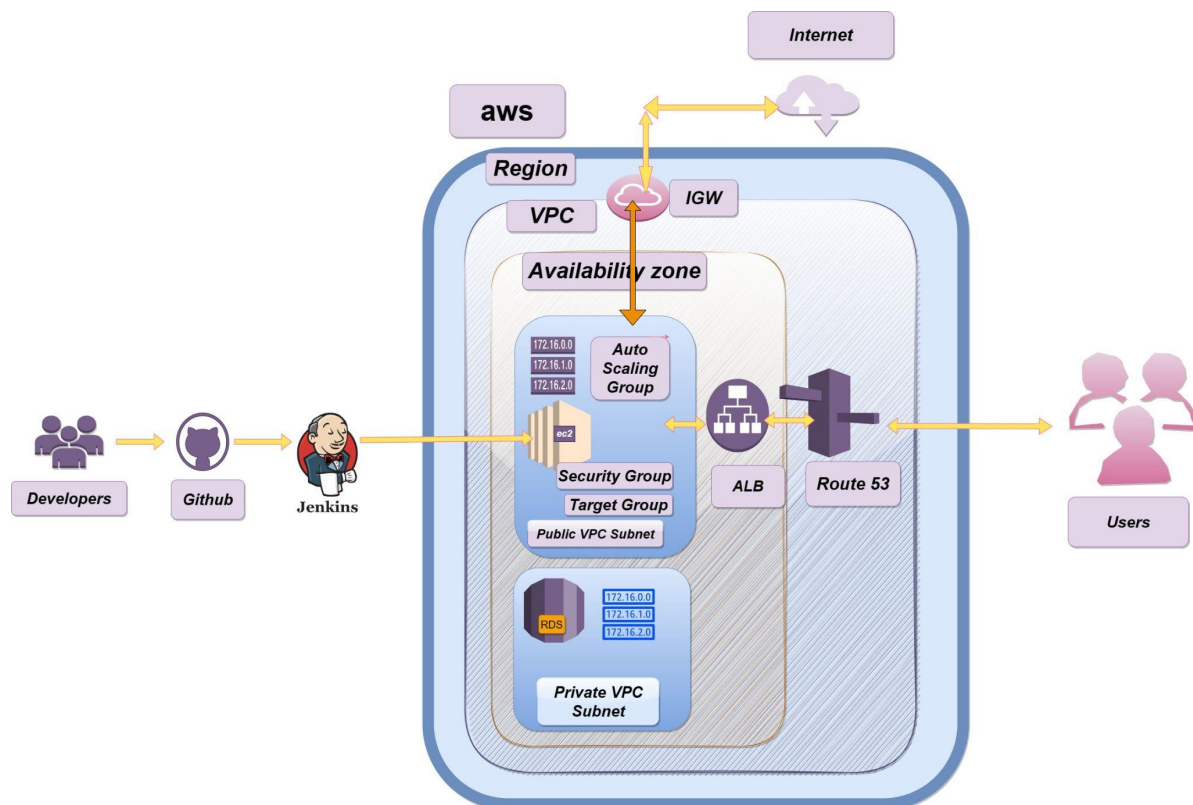


Fig 4.2 : Aws Server Provision and Deployment

4.3 Create VPC network

VPC Create: Make a VPC by visiting the AWS Management Console, selecting VPC, and then clicking "Create VPC." Enter the required VPC name, IPv4 CIDR block, and tenancy choices.

Subnet Create: Create a Public Subnet by selecting "Subnets" and then "Create Subnet" from the VPC dashboard. Select an Availability Zone, the VPC, and the IPv4 CIDR block for the public subnet.

Route Table: To create a route table, go to the VPC dashboard and choose "Route Tables," then "Create Route Table." Connect it to the VPC. Add a route to the internet gateway (0.0.0.0/0) to enable internet access.

Internet Gateway: Make an Internet Gateway (IGW) by clicking "Create Internet Gateway" under "Internet Gateways." Connect it to the VPC you made in step one

4.4 Create RDS

Create an RDS (Relational Database Service) instance on AWS by going to the AWS Management Console, choosing RDS, and then clicking "Create database." Select the database engine (such as MySQL), and then configure the database name, username, and password. Choose the storage options and instance type that you want. Set security parameters and decide whether to enable automated backups and backups on demand. Review the choices once more, then click "Create database." AWS will take care of the RDS instance's provisioning, upkeep, and scalability, giving your application a controlled and dependable database solution.

4.5 Create ec2

Access the AWS Management Console, go to the EC2 service, click "Launch Instance," pick an Amazon Machine Image (AMI), select an instance type, configure parameters, add storage, and configure security groups. Review the instance one last time, then run it.

4.5.1 Create security group

Create an AWS security group and provide inbound/outbound rules to secure resource traffic. Here I have created three security groups for my project purpose. One is for ec2, one is for rds and one is for application load balancer.

Security group for ec2(team-kernull-ec2-sg):

Inbound-rules:

Type	port
SSH	22
Custom Tcp	8080
HTTP	80

Outbound-rules:

Type	Port Range
All traffic	All

Security group for rds(team-kernull-rds-sg):

Inbound-rules:

Type	port
MYSQL/Aurora	3306

Outbound-rules:

Type	Port Range
All traffic	All

Security group for ec2(nishat-alb-sg):

Inbound-rules:

Type	port
HTTPS	443
Custom Tcp	8080
HTTP	80

Outbound-rules:

Type	Port Range
All traffic	All

4.5.2 Create target group

Select "Instance" as the target type, give the target group a name, pick a VPC, and register targets (instances) for load balancing to create an AWS target group.

EC2 > Target groups > nishat-tg

nishat-tg Actions ▼

Details

arn:aws:elasticloadbalancing:us-west-2:072262450408:targetgroup/nishat-tg/a93acd922f1bbe95

Target type Instance	Protocol : Port HTTP: 8080	Protocol version HTTP1	VPC vpc-0658d3e0bf2c31f3d
IP address type IPv4	Load balancer nishat-alb		

Total targets 2	Healthy ✔ 2	Unhealthy ✘ 0	Unused ⋯ 0	Initial ⌚ 0	Draining ⌚ 0
--------------------	----------------	------------------	---------------	----------------	-----------------

► **Distribution of targets by Availability Zone (AZ)**
Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets							
Monitoring							
Health checks							
Attributes							
Tags							
Registered targets (2)						Refresh	Deregister
<input type="text" value="Filter resources by property or value"/>						1	
<input type="checkbox"/>	Instance ID	Name	Port	Zone	Health status	Health status details	
<input type="checkbox"/>	i-06f284cac13...	nishat-asg	8080	us-west-2b	healthy		
<input type="checkbox"/>	i-01b6d8e623...	nishat-ec2	8080	us-west-2a	healthy		

4.5.3 Configure Application Load Balancer

To effectively distribute traffic among instances, specify the name, VPC, subnet group, security group, and target group when creating an application load balancer.

EC2 > Load balancers > nishat-alb			
nishat-alb			
Refresh Actions			
▼ Details			
Load balancer type	Status	VPC	IP address type
Application	Active	vpc-0658d3e0bf2c31f3d	IPv4
Scheme	Hosted zone	Availability Zones	Date created
Internet-facing	Z1H1FL5HABSF5	subnet-0b7ce09be0d25ef83 us-west-2a (usw2-az1)	July 18, 2023, 08:22 (UTC+06:00)
		subnet-08e55c88443c686a0 us-west-2b (usw2-az2)	
Load balancer ARN		DNS name	
arn:aws:elasticloadbalancing:us-west-2:072262450408:loadbalancer/app/nishat-alb/3a749c08938920c2		nishat-alb-1377647179.us-west-2.elb.amazonaws.com (A Record)	

4.5.4 Configure Auto Scaling group

- ★ Create a Launch Template: Give a name of my asg and specify the setup, including the image made from an EC2 instance that is currently running.
- ★ Choose my AMI: Opt for the Amazon Machine Image (AMI) that best meets the needs of my application.
- ★ Select the VPC: Give the Auto Scaling group the name of the Virtual Private Cloud (VPC) and availability zone.
- ★ Set up Application Load Balancer: The ALB should be configured to distribute traffic.
- ★ Specify the Target Group: Tell the ALB which instances in the Auto Scaling group should receive requests for which target group.

nishat-asg

Details | Activity | Automatic scaling | Instance management | Monitoring | Instance refresh

Group details Edit

Auto Scaling group name nishat-asg	Desired capacity 1	Status -	Amazon Resource Name (ARN) arn:aws:autoscaling:us-west-2:072262450408:autoScalingGroup:c6c074ba-435d-4456-a77f-cd984c19d0de:autoScalingGroupName/nishat-asg
Date created Tue Jul 18 2023 14:09:18 GMT+0600 (Bangladesh Standard Time)	Minimum capacity 1		
	Maximum capacity 1		

Launch template Edit

Launch template lt-0b9d5047bc533a23b nishat-template	AMI ID ami-0ddfd690b8aae7f83	Instance type t2.micro	Owner arn:aws:iam::072262450408:user/sadia.nishat
--	---------------------------------	---------------------------	--

4.5.5 Configure Route 53

To direct traffic to the load balancer, use Route 53 to map ALB's DNS to a domain. Create a hosted zone for "sudip.site" in order to control DNS records and connect to resources.

▼ Record 1

Delete

Record name [Info](#)

nishat.sudip.site

Record type [Info](#)

A – Routes traffic to an IPv4 address and some AWS resources

Keep blank to create a record for the root domain.

☒ Alias

Route traffic to [Info](#)

Alias to Application and Classic Load Balancer

US West (Oregon) [us-west-2]

Q dualstack.nishat-alb-1377647179.us-west-2.elb.amazonaws.com

Alias hosted zone ID: Z1H1FL5HABSF5

Routing policy [Info](#)

Simple routing

Evaluate target health

☒ Yes

Add another record

4.6 Create Jenkins Pipeline

4.6.1 Deploy an application in aws

```

pipeline {
    agent any

    triggers { pollSCM 'H/2 * * * *' }

    stages {
        stage('Clone') {
            steps {
                git branch: 'main', credentialsId: 'git-clone', url:
'https://github.com/Nishat792/YSD_Bo2_DevOps_Library_Management_System-aws.git'
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }

        stage('Deploy') {
            steps {

```

```

    script {
        withCredentials([sshUserPrivateKey(credentialsId: '54.202.194.90',
keyFileVariable: 'SSH_KEY')]) {

                                def sourceFilePath =
'/var/lib/jenkins/workspace/aws-test-pipeline/target/spark-lms-0.0.1-SNAPSHOT.jar'
                def destinationPath = '/home/ec2-user/'
                //def sshCmd = "ssh -i $SSH_KEY ec2-user@54.202.194.90 'pkill
java'"

                //sh sshCmd
                def scpCommand = "scp -i $SSH_KEY $sourceFilePath
ec2-user@54.202.194.90:$destinationPath"
                sh scpCommand
                def sshCommand = "ssh -i $SSH_KEY ec2-user@54.202.194.90
'nohup java -jar /home/ec2-user/spark-lms-0.0.1-SNAPSHOT.jar > /dev/null 2>&1 &'"
                sh sshCommand

        }
    }
}
}
}
}
}
}
}

```

Description of AWS Jenkins pipeline:

1. In the trigger section, I used PollSCM to auto trigger.
2. In the 1st stage, I clone the repo from the github repo.
3. Next stage is about maven build for my spring boot application.
4. Lastly, I deploy my application using withCredentials. Here, I used scp command to copy .jar file from source(/var/lib/jenkins/workspace/aws-test-pipeline/target/spark-lms-0.0.1-SNAPSHOT.jar) to destination(/home/ec2-user) and then used ssh command and run the jar file and used nohup to run in background.

5. AWS EKS-Managed Service

5.1 Definition

AWS's managed Kubernetes offering, Amazon Elastic Kubernetes offering (Amazon EKS), makes it easier to develop and manage containerized applications. Without the operational burden, it offers a scalable and secure Kubernetes control plane. EKS provides effective load balancing and auto-scaling as well as easy integration with a number of AWS services. While EKS maintains a dependable and adaptable Kubernetes environment on the AWS cloud, developers can concentrate on developing applications.

5.2 Design architecture

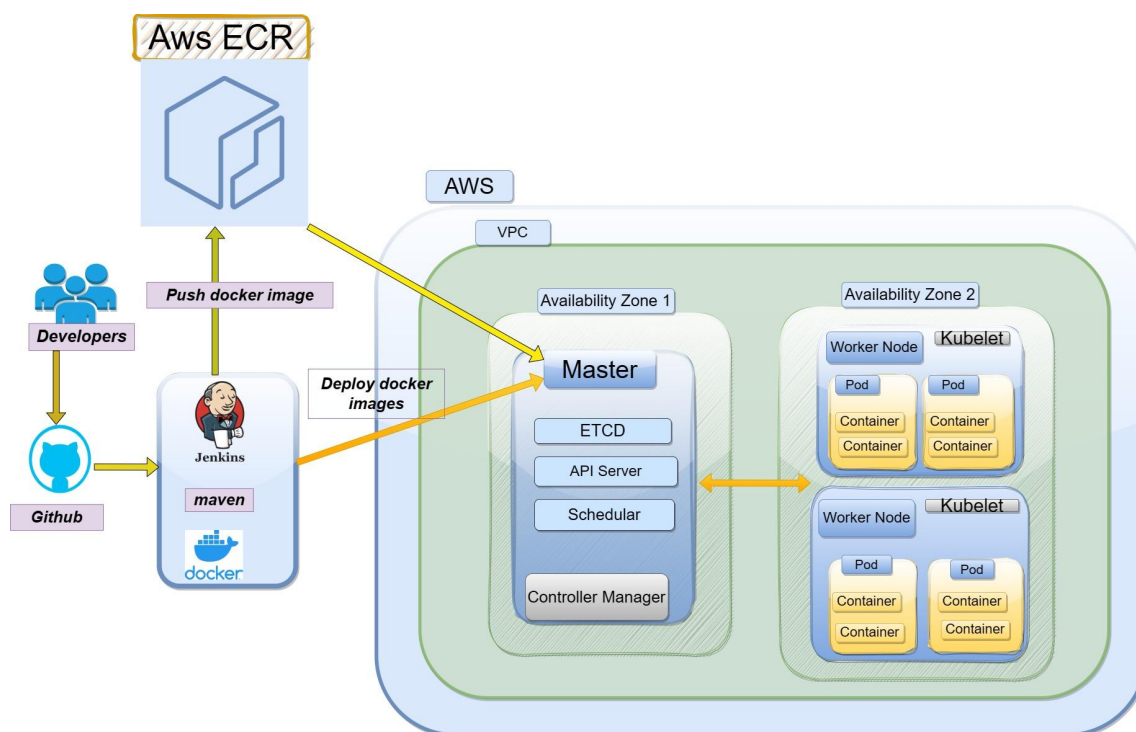


Fig 5.2 : AWS Eks Managed Services

5.3 Explain on deployment process

Containerize Spring Boot Application: The first step is to use Docker containers to package up Spring Boot applications. Using Docker, it may be combined this application's configurations and dependencies into a single container. Then, this containerized programme may be installed and consistently used in a variety of contexts.

Create an ECR Repository (Elastic Container Registry): In order to store Docker images in AWS, we must first create an ECR Repository (Elastic Container Registry). AWS offers ECR, a fully-managed Docker container registry service. Container images can be kept in an ECR repository that can be set up.

Build and Push Docker Image to ECR: Create a Docker image for Spring Boot application after containerizing it, then push the image to ECR. The Docker image will then be pushed to the ECR repository established in the previous step. This step makes sure that the container image for this application is accessible in the AWS cloud.

Configure Kubernetes Deployment Files: Kubernetes uses YAML files to specify and control application deployment. The Spring Boot program must be executed as a container in the EKS cluster, thus we must write Kubernetes deployment files that outline how to do so. These files should contain information such as the name of the container image, the necessary resources, environment variables, etc.

Create an EKS Cluster:

It takes several steps to set up an EKS cluster in AWS. Make sure I have the required AWS CLI (Command Line Interface) installed and set up with the proper permissions before starting creating EKS resources. A step-by-step tutorial for setting up an EKS cluster to deploy a Spring Boot project is provided below:

- ❖ **Install Prerequisites:** Ensure that the AWS CLI is set up with the correct AWS access credentials and that it is installed on the computer. Additionally, it required the Kubernetes command-line tool, kubectl, to be installed.
- ❖ **Create a role in IAM:** Create a role in IAM with the appropriate permissions for EKS cluster. This role needs policies that enable it to work with other AWS services like EC2, VPC, and others as well as create and

maintain EKS resources. For instance, define a role with the managed policies for “**AmazonEKSServicePolicy**” and “**AmazonEKSClusterPolicy**” attached.

- ❖ **Create a VPC (Virtual Private Cloud)** for EKS cluster if I haven't already done so. To provide high availability, make sure the VPC has at least two private subnets and two public subnets dispersed across several Availability Zones.

❖ **Making an EKS Cluster:**

To construct the EKS cluster, issue the command “aws eks create-cluster”. Indicate the cluster name, Kubernetes version, previously defined IAM role ARN, VPC and subnets IDs, and other relevant information. A sample command is provided below:

```
aws eks create-cluster --name my-eks-cluster --kubernetes-version 1.21 --role-arn arn:aws:iam::123456789012:role/my-eks-cluster-role --resources-vpc-config subnetIds=subnet-12345678,subnet-87654321,securityGroupIds=sg-12345678
```

Fill in the necessary values for the placeholders.

- ❖ **Set up kubectl:** Once the cluster has been built, kubectl needs to be set up to communicate with the EKS cluster. Execute the upcoming command:

```
aws eks update-kubeconfig --name my-eks-cluster
```

By using the newly built EKS cluster, this command modifies kubectl settings.

❖ **Check the Cluster:**

The following command can be used to check the status of your cluster and make sure it is operational:

```
kubectl get nodes
```



EKS cluster's nodes should be listed in the response.

Create the Worker Nodes (NodeGroup):

Create the NodeGroup with the appropriate configurations, including the instance type, instance count, IAM role, and other parameters, using the AWS CLI.

Ensure that the NodeGroup is created in the same VPC as the EKS cluster.

AWS CLI Steps:

1. Start a command prompt or terminal.
2. To create the NodeGroup, use the create-nodegroup command. Fill in the blanks with the values you actually have:

```
aws eks create-nodegroup --cluster-name CLUSTER_NAME \
    --nodegroup-name NODEGROUP_NAME \
    --node-role IAM_ROLE_ARN \
    --subnets SUBNET_IDS \
    --instance-types INSTANCE_TYPES \
    --scaling-config minSize=MIN_SIZE,maxSize=MAX_SIZE,desiredSize=DESIRED_SIZE \
    --ami-type AL2_x86_64 \
    --tags _TAGS
```

3. Following the execution of the command, AWS will construct the NodeGroup and add the worker nodes to your EKS cluster.

Verify the Worker Nodes:

4. Run the following AWS CLI command to view the worker nodes' status and specifics:

```
aws eks describe-nodegroup --cluster-name CLUSTER_NAME --nodegroup-name NODEGROUP_NAME
```

Deploy spring boot Application to EKS: We can deploy your Spring Boot application to the EKS cluster once the Kubernetes deployment files are prepared and the cluster is up and running. To apply the deployment files and build the required Kubernetes resources, use the kubectl command-line tool.

Automating the deployment process through a Jenkins-managed CI/CD (Continuous Integration/Continuous Deployment) pipeline is required to deploy a Spring Boot project to AWS EKS (Elastic Kubernetes Service). Here is a detailed explanation on how to do it:

Prerequisites:

A functional Jenkins server should be set up and configured.
To connect with AWS EKS and other required AWS services, the Jenkins server should have the appropriate IAM rights.

Step 1: Configure Jenkins with AWS credentials:

Access Jenkins by going to "Manage Jenkins" > "Manage Credentials."
Include your secret access key and AWS access key ID as credentials.

Step 2: Setup of the Spring Boot project's Jenkins pipeline:

In order to develop and deploy your Spring Boot project, create a Jenkins pipeline job. You'll specify the pipeline configuration.

step 3: Create a Jenkinsfile

Make a Jenkinsfile file in the repository for your Spring Boot project. This file will outline the phases and procedures for the Jenkins pipeline used to develop and deploy the application.

Step 4: Define the stages of the Jenkins pipeline:

Specify the pipeline stages in the Jenkinsfile. "Build," "Test," "Dockerize," "Push to ECR," "Deploy to EKS," etc. are typical steps.

Step 5: Construct the Jenkins Pipeline:

Implement the appropriate procedures to create, test, containerize, and deploy Spring Boot applications within each stage.

6. References

1. [How to Install Ansible on Ubuntu 20.04 - VITUX](#)
2. [Install Docker Engine on Ubuntu | Docker Documentation](#)
3. [Linux \(jenkins.io\)](#)
4. [How to Install a Kubernetes Cluster with Kubeadm on Rocky Linux \(howtoforge.com\)](#)
5. [Managed Kubernetes Service – Amazon EKS – Amazon Web Services](#)