Deployment of Spring Boot Web Application in Kubernetes Cluster

Performed by: Prabhash Mishra

Guided by: Akshay Kumar

Resources Required:

- **AWS EC2 Instances:** 3
  - Instance types: 2 `t2.medium`, 1 `t2.micro`
- **Operating System:** Ubuntu Server 22.04 LTS (HVM)
- **Security Groups:** Allow all traffic
- **Terminal:** MobaXterm

Tools Used:

- Docker
- Jenkins
- Kubernetes (`kubeadm`)
- Git
- Maven

Step-by-Step Guide:

Step 1: launch 3 instances 2 t2.medium ,1 t2.micro with Ubuntu Server 22.04 LTS (HVM) with security group allow all traffic

Step 2: connect to mobaxterm terminal to perform ssh

Step 3: enter multi execution mode and be a root user(sudo su -)

To setup a kubernetes cluster i.e to install kubeadm run the following steps: -
Step 1: Disable Swap on All Nodes

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

Step 2: Enable IPv4 Packet Forwarding

sysctl params required by setup, params persist across reboots
```bash
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
EOF
```

Apply sysctl params without reboot
```bash
sudo sysctl --system
```

Step 3: Verify IPv4 Packet Forwarding
```bash
sysctl net.ipv4.ip_forward
```

Step 4: Install containerd
```bash
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
```

```
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update && sudo apt-get install containerd.io && systemctl enable --now containerd
```

Step 5: Install CNI Plugin
```bash
wget https://github.com/containernetworking/plugins/releases/download/v1.4.0/cni-plugins-linux-amd64-v1.4.0.tgz
mkdir -p /opt/cni/bin
tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.4.0.tgz
```

Step 6: Forward IPv4 and Configure iptables
```bash
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter

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

sudo sysctl --system
sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward
modprobe br_netfilter
sysctl -p /etc/sysctl.conf
```

Step 7: Modify containerd Configuration for systemd Support
```bash
sudo cat > /etc/containerd/config.toml
```
Paste the configuration in the file and save it.
```bash
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
version = 2

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_cert = ""
  tcp_tls_key = ""
  uid = 0

[metrics]
  address = ""
  grpc_histogram = false
```

```
[plugins]

  [plugins."io.containerd.gc.v1.scheduler"]
    deletion_threshold = 0
    mutation_threshold = 100
    pause_threshold = 0.02
    schedule_delay = "0s"
    startup_delay = "100ms"

  [plugins."io.containerd.grpc.v1.cri"]
    disable_apparmor = false
    disable_cgroup = false
    disable_hugetlb_controller = true
    disable_proc_mount = false
    disable_tcp_service = true
    enable_selinux = false
    enable_tls_streaming = false
    ignore_image_defined_volumes = false
    max_concurrent_downloads = 3
    max_container_log_line_size = 16384
    netns_mounts_under_state_dir = false
    restrict_oom_score_adj = false
    sandbox_image = "k8s.gcr.io/pause:3.5"
    selinux_category_range = 1024
    stats_collect_period = 10
    stream_idle_timeout = "4h0m0s"
    stream_server_address = "127.0.0.1"
    stream_server_port = "0"
    systemd_cgroup = false
    tolerate_missing_hugetlb_controller = true
    unset_seccomp_profile = ""

    [plugins."io.containerd.grpc.v1.cri".cni]
      bin_dir = "/opt/cni/bin"
      conf_dir = "/etc/cni/net.d"
      conf_template = ""
      max_conf_num = 1

    [plugins."io.containerd.grpc.v1.cri".containerd]
      default_runtime_name = "runc"
      disable_snapshot_annotations = true
      discard_unpacked_layers = false
      no_pivot = false
      snapshotter = "overlayfs"

      [plugins."io.containerd.grpc.v1.cri".containerd.default_runtime]
        base_runtime_spec = ""
        container_annotations = []
        pod_annotations = []
        privileged_without_host_devices = false
        runtime_engine = ""
        runtime_root = ""
        runtime_type = ""

        [plugins."io.containerd.grpc.v1.cri".containerd.default_runtime.options]

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

        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
          base_runtime_spec = ""
          container_annotations = []
          pod_annotations = []
          privileged_without_host_devices = false
          runtime_engine = ""
          runtime_root = ""
          runtime_type = "io.containerd.runc.v2"

          [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
            BinaryName = ""
            CriuImagePath = ""
            CriuPath = ""
            CriuWorkPath = ""
            IoGid = 0
            IoUid = 0
            NoNewKeyring = false
            NoPivotRoot = false
            Root = ""
            ShimCgroup = ""
            SystemdCgroup = true
```

```
      [plugins."io.containerd.grpc.v1.cri".containerd.untrusted_workload_runtime]
        base_runtime_spec = ""
        container_annotations = []
        pod_annotations = []
        privileged_without_host_devices = false
        runtime_engine = ""
        runtime_root = ""
        runtime_type = ""

        [plugins."io.containerd.grpc.v1.cri".containerd.untrusted_workload_runtime.options]

    [plugins."io.containerd.grpc.v1.cri".image_decryption]
      key_model = "node"

    [plugins."io.containerd.grpc.v1.cri".registry]
      config_path = ""

      [plugins."io.containerd.grpc.v1.cri".registry.auths]

      [plugins."io.containerd.grpc.v1.cri".registry.configs]

      [plugins."io.containerd.grpc.v1.cri".registry.headers]

      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

    [plugins."io.containerd.grpc.v1.cri".x509_key_pair_streaming]
      tls_cert_file = ""
      tls_key_file = ""

  [plugins."io.containerd.internal.v1.opt"]
    path = "/opt/containerd"

  [plugins."io.containerd.internal.v1.restart"]
    interval = "10s"

  [plugins."io.containerd.metadata.v1.bolt"]
    content_sharing_policy = "shared"

  [plugins."io.containerd.monitor.v1.cgroups"]
    no_prometheus = false

  [plugins."io.containerd.runtime.v1.linux"]
    no_shim = false
    runtime = "runc"
    runtime_root = ""
    shim = "containerd-shim"
    shim_debug = false

  [plugins."io.containerd.runtime.v2.task"]
    platforms = ["linux/amd64"]

  [plugins."io.containerd.service.v1.diff-service"]
    default = ["walking"]

  [plugins."io.containerd.snapshotter.v1.aufs"]
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.btrfs"]
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.devmapper"]
    async_remove = false
    base_image_size = ""
    pool_name = ""
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.native"]
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.overlayfs"]
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.zfs"]
    root_path = ""

[proxy_plugins]

[stream_processors]

  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar"]
```

```
    accepts = ["application/vnd.oci.image.layer.v1.tar+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar"

  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar.gzip"]
    accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[timeouts]
  "io.containerd.timeout.shim.cleanup" = "5s"
  "io.containerd.timeout.shim.load" = "5s"
  "io.containerd.timeout.shim.shutdown" = "3s"
  "io.containerd.timeout.task.state" = "2s"

[ttrpc]
  address = ""
  gid = 0
  uid = 0

```

Step 8: Restart containerd and Check the Status
```bash
sudo systemctl restart containerd && systemctl status containerd
```

Step 9: Install kubeadm, kubelet, and kubectl
```bash
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg

sudo mkdir -p -m 755 /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' |
sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update -y
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Step 10: Initialize the Cluster and Install CNI In the master node
```bash
sudo kubeadm config images pull
sudo kubeadm init
```
After Initialzing the Cluster Connect to it and apply the CNI yaml in Master node

```bash
#To start using your cluster, you need to run the following as a regular user:

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

#Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf
```
```bash
#Apply the CNI YAML
kubectl apply -f https://reweave.azurewebsites.net/k8s/v1.30/net.yaml
```

Step 11: Join Worker Nodes to the Cluster
Run the command generated after initializing the master node on each worker node. For example:
```bash
kubeadm join 192.168.122.100:6443 --token zcijug.ye3vrct74itrkesp \
        --discovery-token-ca-cert-hash
sha256:e9dd1a0638a5a1aa1850c16f4c9eeaa2e58d03f97fd0403f587c69502570c9cd
```

Note- initialize any one t2.medium as control plane and join other nodes as worker nodes

=> To setup a database server
Use t2.micro instance
=> install mariadb server on it
sudo apt update -y
sudo apt install mariadb-server -y

=>To start the mariadb
Systemctl start mariadb

=>To enble mariadb to automatically restart on boot -
Systemctl restart mariadb

To secure the server by providing a password for root user-
sudo mysql_secure_ installation
New password for root user
Confirm password

As we need to remotely connect to the database:
Edit a file
vi /etc/mysql/mariadb.conf.d/50-server.cnf
Look for the following line:
bind-address = 127.0.0.1
And change it as
bind-address = 0.0.0.0
Now this means that mariadb can accept connection from any ip address

Restart mariadb server
Systemctl restart mariadb

To login:-
Mysql -u root -p
Enter password: your password

Now to provide access for my database remotely for the root user:-
Run a query:-
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '1234' WITH GRANT OPTION ;

To apply reload the privileges:-
FLUSH PRIVILEGES;

Now on your control plane install java 17 as jenkins uses jre
sudo apt install openjdk-17-jdk

Install jenkins for ubuntu

Long Term Support release

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Install git on control plane
sudo apt install git -y

Install docker on control plane
Vi docker .sh
And add script in it

sudo apt update -y
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt update -y
sudo apt install -y docker-ce
sudo systemctl enable docker
sudo systemctl start docker
sudo systemctl status docker
docker --version
sudo apt install -y git
sudo curl -L "https://github.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version

To run the script
sh docker.sh

Now give sudo permission for the jenkins user so that it can execute kubectl commands as by using sudo command is executed as a superuser and super is configured to have control over the entire cluster

Visudo
Jenkins ALL=(ALL : ALL) NOPASSWD: ALL

to provide jenkins user access for docker deamon to execute docker commands :-
Usermod -aG docker jenkins

restart docker and jenkins
Systemctl restart docker
Systemctl restart jenkins

Configure jenkins
Use public ip of control plane instance with portno 8080 to access jenkins
To get the password
 sudo cat /var/lib/jenkins/secrets/initialAdminPassword
Use the password and install suggest plugins
And do other confriguations like username password

To add maven for jenkins
Manage jenkins -> tools -> under maven add maven --> give a name apply and save done

now all confriguation is done
Make sure we have our spring boot project in our github repositoy
In that we will create a docker file to build a image of the project

But before this will we specify the database details for our spring boot project
Edit application.properties file
spring.datasource.url=jdbc:mysql://<private-ip of db server>:3306/your_db_name
spring.datasource.username=your_db_user
spring.datasource.password=your_db_password
server.port=port no on which app should be hosted

Now create a docker file named Dockerfile and add
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY ./target/my-shop-1.0.jar /app/app.jar
ENTRYPOINT ["java", "-jar", "/app/app.jar"]

Create a directory in your github and add you deployment and service script in it
Deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: springboot-app
  labels:
    app: springboot
spec:
  replicas: 1
  selector:
    matchLabels:
      app: springboot
  template:
    metadata:
      labels:
        app: springboot
    spec:
      containers:
        - name: springboot-app
          image: prabhash1710/sample1:latest
          ports:
            - containerPort: 1234
          imagePullPolicy: Always
```

Service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: springboot-service
spec:
  ports:
    - port: 8080
      targetPort: 1234
      protocol: TCP
  selector:
    app: springboot
  type: NodePort
```

Now create a webhook to automate the how the build should be triggered

In your repo setting ->webhook->add webhook-> paste the jenkins server url with port no /github-webhook/

Content type

Application /json

Click on add webhook

come to your jenkins

Create a pipeline job

Build triggers:

Github hok trigger for git scm pollings

Write a pipeline to create a build for the project,build an image of it ,push it for dockerhub , run the kubernetes deployment and service script

```
pipeline {
    agent any

    tools {
        maven 'maven'
    }

    stages {
        stage('Clone Repository') {
            steps {
```

```
            git branch: 'main', url: 'https://github.com/Prabhash1710/Spring_proj.git'
        }
    }

    stage('Build Application') {
        steps {
            sh 'mvn clean package -DskipTests'
            sh 'chmod 777 /var/lib/jenkins/workspace/deploy/target/my-shop-1.0.jar'
        }
    }

    stage('Build Docker Image') {
        steps {
            sh 'docker build -t Prabhash1710/sample1:latest -f Dockerfile.yml .'
        }
    }

    stage('Push Docker Image') {
        steps {
            script {
                // Login to Docker Hub
                sh 'echo "password" | docker login -u "Prabhash1710" --password-stdin'
                // Push the Docker image to Docker Hub
                sh 'docker push Prabhash1710/sample1:latest'
            }
        }
    }

    stage('Run Deployment Scripts') {
        steps {
            sh 'sudo kubectl apply -f .'
        }
    }
  }
}
```

Apply and save
Click on build now

Go to control palne:-
Run a command
Kubectl get svc

You can see a node port

Use that along with the public ip of worker node to access the application