**Spring Boot CRUD (Create, Read, Update, Delete) application and deploy it using Docker on Ubuntu 22.04 with PostgreSQL, Apache2, Apache Tomcat, and a load balancer setup using Kubernetes,**

Q1🡺

Understanding, deploy and create springboot project like “CRUDE Operation “Apache2, Apache tomcat 8.05 with PostgreSQL 9.5 help of dcoker how to LoadBalancer, and ClusterIP,Ingress withe command and code and example on ubutnu 22.04 ?

To create a Spring Boot CRUD (Create, Read, Update, Delete) application and deploy it using Docker on Ubuntu 22.04 with PostgreSQL, Apache2, Apache Tomcat, and a load balancer setup using Kubernetes, follow these steps:

**Step 1: Create a Spring Boot CRUD Application**

1. **Initialize a Spring Boot Project:**
   * Use Spring Initializr (<https://start.spring.io/>) to generate a new project with the following dependencies:
     + Spring Web
     + Spring Data JPA
     + PostgreSQL Driver
   * Alternatively, you can use the command line to generate the project:

sh

Copy code

curl https://start.spring.io/starter.zip \

-d dependencies=web,data-jpa,postgresql \

-d name=CrudDemo \

-d type=maven-project \

-d packageName=com.example.crud \

-o CrudDemo.zip

unzip CrudDemo.zip -d CrudDemo

1. **Create a Model Class:**

java

Copy code

package com.example.crud.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String role;

// Getters and Setters

}

1. **Create a Repository Interface:**

java

Copy code

package com.example.crud.repository;

import com.example.crud.model.Employee;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

1. **Create a Service Class:**

java

Copy code

package com.example.crud.service;

import com.example.crud.model.Employee;

import com.example.crud.repository.EmployeeRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository repository;

public List<Employee> findAll() {

return repository.findAll();

}

public Employee save(Employee employee) {

return repository.save(employee);

}

public void deleteById(Long id) {

repository.deleteById(id);

}

public Employee findById(Long id) {

return repository.findById(id).orElse(null);

}

}

1. **Create a Controller Class:**

java

Copy code

package com.example.crud.controller;

import com.example.crud.model.Employee;

import com.example.crud.service.EmployeeService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeService service;

@GetMapping

public List<Employee> findAll() {

return service.findAll();

}

@PostMapping

public Employee save(@RequestBody Employee employee) {

return service.save(employee);

}

@GetMapping("/{id}")

public Employee findById(@PathVariable Long id) {

return service.findById(id);

}

@DeleteMapping("/{id}")

public void deleteById(@PathVariable Long id) {

service.deleteById(id);

}

}

1. **Configure PostgreSQL in application.properties:**

properties

Copy code

spring.datasource.url=jdbc:postgresql://localhost:5432/cruddb

spring.datasource.username=your\_username

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**Step 2: Dockerize the Spring Boot Application**

1. **Create a Dockerfile:**

dockerfile

Copy code

FROM openjdk:11-jre-slim

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]

1. **Build and Run the Docker Image:**

sh

Copy code

mvn clean package

docker build -t springboot-crud-demo .

docker run -p 8080:8080 springboot-crud-demo

**Step 3: Setup PostgreSQL in Docker**

1. **Create a docker-compose.yml file:**

yaml

Copy code

version: '3.8'

services:

db:

image: postgres:9.5

environment:

POSTGRES\_DB: cruddb

POSTGRES\_USER: your\_username

POSTGRES\_PASSWORD: your\_password

ports:

- "5432:5432"

app:

image: springboot-crud-demo

depends\_on:

- db

environment:

SPRING\_DATASOURCE\_URL: jdbc:postgresql://db:5432/cruddb

SPRING\_DATASOURCE\_USERNAME: your\_username

SPRING\_DATASOURCE\_PASSWORD: your\_password

ports:

- "8080:8080"

1. **Run Docker Compose:**

sh

Copy code

docker-compose up

**Step 4: Deploy on Kubernetes**

1. **Create Kubernetes Deployment and Service Files:**

**deployment.yml:**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: springboot-crud

spec:

replicas: 3

selector:

matchLabels:

app: springboot-crud

template:

metadata:

labels:

app: springboot-crud

spec:

containers:

- name: springboot-crud

image: springboot-crud-demo:latest

ports:

- containerPort: 8080

env:

- name: SPRING\_DATASOURCE\_URL

value: jdbc:postgresql://postgres-service:5432/cruddb

- name: SPRING\_DATASOURCE\_USERNAME

value: your\_username

- name: SPRING\_DATASOURCE\_PASSWORD

value: your\_password

**service.yml:**

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: springboot-crud-service

spec:

type: ClusterIP

selector:

app: springboot-crud

ports:

- protocol: TCP

port: 80

targetPort: 8080

**postgres-deployment.yml:**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: postgres

spec:

replicas: 1

selector:

matchLabels:

app: postgres

template:

metadata:

labels:

app: postgres

spec:

containers:

- name: postgres

image: postgres:9.5

ports:

- containerPort: 5432

env:

- name: POSTGRES\_DB

value: cruddb

- name: POSTGRES\_USER

value: your\_username

- name: POSTGRES\_PASSWORD

value: your\_password

**postgres-service.yml:**

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: postgres-service

spec:

type: ClusterIP

selector:

app: postgres

ports:

- protocol: TCP

port: 5432

targetPort: 5432

1. **Apply the Kubernetes Configurations:**

sh

Copy code

kubectl apply -f postgres-deployment.yml

kubectl apply -f postgres-service.yml

kubectl apply -f deployment.yml

kubectl apply -f service.yml

**Step 5: Setup Ingress Controller for Load Balancing**

1. **Install NGINX Ingress Controller:**

sh

Copy code

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml

1. **Create an Ingress Resource:**

**ingress.yml:**

yaml

Copy code

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: springboot-crud-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: your-domain.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: springboot-crud-service

port:

number: 80

1. **Apply the Ingress Configuration:**

sh

Copy code

kubectl apply -f ingress.yml

1. **Update DNS to point to the Ingress Controller's external IP.**

**Step 6: Access Your Application**

Once everything is set up, you should be able to access your Spring Boot CRUD application via the domain specified in the Ingress resource.

**Summary**

This guide walks you through setting up a Spring Boot CRUD application with PostgreSQL, Dockerizing it, deploying on Kubernetes, and configuring a load balancer with NGINX Ingress. Make sure to replace placeholders like your\_username, your\_password, and your-domain.com with actual values.

For more advanced configurations, you might want to dive deeper into Kubernetes resources, Docker networking, and Spring Boot configuration properties.

4o

Top of Form



Bottom of Form

ChatGPT can make mistakes. Check important info.