Application Deployment using Docker-Jenkins-AWS ECR & EKS

Assessment

Contents

Title: Application Deployment using Docker-Jenkins-AWS ECR & EKS

Difficulty Level

Expert

Duration

150 minutes

What you will learn

- How to containerize a Spring Boot microservice application using Docker.
- Deploying and managing Docker containers on AWS EKS (Elastic Kubernetes Service).
- Utilizing AWS ECR (Elastic Container Registry) to store Docker images securely.
- Automating deployment pipelines with Jenkins integrated with Maven for CI/CD.
- Configuring and managing EC2 instances as hosts for Docker containers.

What you will be provided

- An AWS account with the necessary resources allowed for this project.
- A Spring Boot application and Jenkins pipeline script are provided through the links in the activity steps.

What you need to know

- Basic understanding of Docker and containerization concepts.
- Familiarity with Spring Boot for Java application development.
- Knowledge of AWS services, especially EKS, ECR, and EC2.
- Experience with Maven for Java build automation and Jenkins for continuous integration.
- Basic Linux commands to manage EC2 instances.

Skill Tags

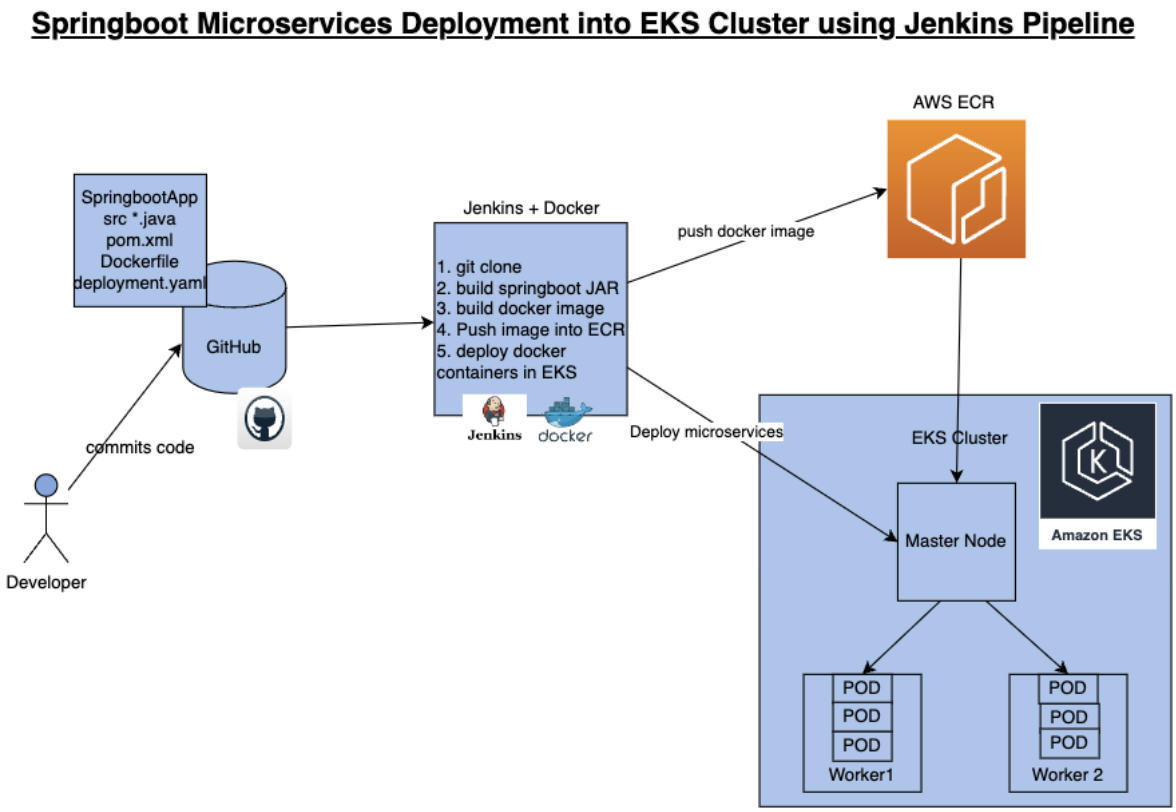
- Docker
- AWS EKS, AWS ECR
- Jenkins CI/CD
- Maven Build Automation, Spring Boot Framework
- EC2 Instance Management

What you will do

Your organization faced challenges deploying microservices due to inconsistent environments. To address this, they adopted Docker to package each microservice with its dependencies into containers, ensuring consistency across platforms. They plan to use AWS as the cloud platform for deployment. As a DevOps engineer, you are tasked with implementing Docker containerization to enable independent scaling, smooth deployment, and streamline the workflow, minimizing troubleshooting issues in the AWS environment.

The objective of the assessment is to automate Spring Boot microservices builds using Jenkins pipeline and deploy it on AWS EKS Cluster. The major components involved are -

- Automating builds using Jenkins.
- Automating Docker image creation.
- Automating Docker image upload into AWS ECR.
- Automating Docker Containers Deployments to Kubernetes Cluster.



Activities

1. Create EC2 Instance:

- Create an EC2 instance named jenkins-server with Instance Type as t3.small and based on the latest Ubuntu Machine Image.
- Navigate to the EC2 service and create a security group named jenkins-server-security-group.
- Configure the inbound rules to allow traffic from anywhere (IPv4) on:Port 8080 (Custom TCP), Port 22 (SSH for EC2 access).
- Create a key pair named jenkins-ssh-key.

2. Configure IAM Role:

- Create IAM role named ec2-eks-role for the EC2 service.
- Assign a policy that grants full administrative access.
- Attach this IAM role to the EC2 instance.

3. Install Required Software:

- Install Java
- Install AWS CLI and Authenticator
- Install Docker
- Start Docker service
- Install Jenkins

4. Configure Jenkins with Docker Volumes and Networks

- Configure Jenkins with Docker Volumes and Networks
- Create a Docker volume for persistent Jenkins data with name 'jenkins_data'
- Run Jenkins in a Docker container using the created volume.
- Create a Docker network for Jenkins with name 'jenkins_network'
- Attach the Jenkins container to the network

5. Install Kubernetes Tools

- Install kubectl
- Verify the kubectl installation
- Install eksctl

- Verify the eksctl installation

6. Create an ECR Repository

- Go to the Amazon Elastic Container Registry (ECR) service and create a private repository named springboot-jenkins-ecr-repo.

7. Configure Jenkins for Docker

- Add Jenkins User to Docker Group
- Switch to Jenkins User

8. Create EKS Cluster and Node Group

- Create EKS Cluster with name jenkins-cluster in us-east-2 region without a nodegroup
- Save Kube Configuration File.

9. Set Up GitHub Repository

- Create a private GitHub repository named springboot-jenkins-eks in your GitHub account.
- Use the files provided in the project folder and upload them to the repository created.

10. Configure Maven Build Tool and Install Required Jenkins Plugins

- In Jenkins, navigate to Manage Jenkins > Tools and add a Maven installation named Maven3, then save it.
- Install the Kubernetes CLI, Amazon ECR, Docker, and Docker Pipeline plugins without restarting.

11. Configure Jenkins Credentials

- Go to Manage Jenkins > Credentials and configure the following:
- GitHub Credentials: Add your GitHub username and personal access token (PAT) with the ID github-credentials.

- Kubernetes Credentials: Add the kubeconfig.txt file created earlier with the ID K8S (Kind: Secret File).

12. Create and execute the Jenkins Pipeline
    - Create a new Jenkins pipeline named springboot-jenkins-eks with the description: "Automation of Spring Boot microservices builds using Jenkins pipeline and deploy it on AWS EKS Cluster."
    - Upon creating the Jenkins pipeline, trigger a build. A successful run will dockerize the Spring Boot application and deploy it on EKS.

13. Access the Deployed Application
    - A Classic Load Balancer will be created by the Kubernetes Service. Access the DNS of the load balancer to view the deployed Spring Boot application.

Testcases

1. Check for creation of Jenkins EC2 instance(5marks)
2. Checking for the creation of the role ec2-eks-role(5 marks)
3. Check creation of security group creation and association with EC2 instance (5 marks)
4. Check installation and accessibility of Jenkins on the EC2 instance (5 marks)
5. Check installation of plugins: Docker, Docker Pipeline and Kubernetes CLI on Jenkins (10 marks)
6. Check creation of the ECR repository and presence of Dockerized Spring Boot application (10 marks)
7. Check Docker volume creation with the name jenkins_data (10 marks)
8. Check Docker network creation with name jenkins_network (10 marks)
9. Check if the Jenkins pipeline was successful (10 marks)
10. Check creation of the EKS Cluster (10 marks)
11. Check Node group creation (10 marks)

12. Kubernetes deployment is successful (10 marks)