Deploy Spring Boot Application to Elastic Beanstalk, Kubernetes and AWS EKS

1. Deploy Spring Boot Application to Elastic Beanstalk

Beanstalk Deployment: **https://cloudkatha.com/how-to-deploy-spring-boot-application-to-aws-elastic-beanstalk/**

Source code: D:\workspace\lrc_poc**aws_opensearch_spring**

Github:
https://github.com/freeever/aws_opensearch_spring.git

Note:

1. Add the following to the application.properties

```
server.port=5000
```

2. To make the Beanstalk health check working, there MUST BE an endpoint "/" available for health check. e.g.

```
@CrossOrigin
@RestController
public class HealthCheckController {
    @GetMapping("/")
    public String health() {
        return "Hello & Welcome to LRC POC !!!";
    }
}
```

Application name: lrc-spring-db-poc

Under my account.The application was deployed, but failed on all load balancers.

By default, the load balancer listen to port 5000, as per recommendation on the internet:

- Elastic Beanstalk
- => Go to xxxx-env => Configuration => Software
- Add env variable: SERVER_PORT = 5000

Change the environment to single instance:

- Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
- In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
  Note

  If you have many environments, use the search bar to filter the environment list.
- In the navigation pane, choose **Configuration**.
- In the **Capacity** category, choose **Edit**.
- From the **Environment Type** list, select the type of environment that you want

Clean UP for Elastic Beanstalk:

- S3
- EC2
- Load Balancer

2. Deploy Spring Boot Application to Kubernetes

Refer to Full-Stack with Angular 8 + Spring Boot + Mysql CRUD API application on Kubernetes cluster

2.1 Docker - Spring Boot Application

Build and generate the JAR file

> mvn clean install -DskipTests=true

Create **Dockerfile**

```
FROM openjdk:11
ARG JAR_FILE=target/aws_opensearch_spring-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","app.jar"]
```

> docker build -t freeever/aws-opensearch-spring .

> docker run -d --name aws-opensearch-spring -p 8080:5000 freeever/aws-opensearch-spring

http://localhost:8080/api/geo/abandoned-mine/3fa85f64-5717-4562-b3fc-2c963f66afa7 (See **Postman** collection **AWS POC**)

2.2 Docker Compose

docker-compose.yml

```
version: '3'

services:
  spring-app:
    image: 'freeever/aws-opensearch-spring'
    build:
      context: .
      dockerfile: Dockerfile
    container_name: aws-opensearch-spring
    ports:
      - 8080:5000
```

> docker-compose up

http://localhost:8080/api/geo/abandoned-mine/3fa85f64-5717-4562-b3fc-2c963f66afa7 (See **Postman** collection **AWS POC**)

> docker-compose down

2.3 Deploy the Spring Boot App to kubernetes

Create **deployment.yml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aws-opensearch-spring-deployment
  labels:
    app: aws-opensearch-spring-deployment
spec:
  selector:
    matchLabels:
      app: aws-opensearch-spring-deployment
  replicas: 2
  template:
    metadata:
      labels:
        app: aws-opensearch-spring-deployment
    spec:
      containers:
        - name: aws-opensearch-spring-deployment
          image: freeever/aws-opensearch-spring
          imagePullPolicy: Always
          ports:
            - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: aws-opensearch-spring-service
  labels:
    app: aws-opensearch-spring-service
spec:
  ports:
    - nodePort: 31147
      port: 5000
      targetPort: 8080
      protocol: TCP
  selector:
    app: aws-opensearch-spring-deployment
  type: NodePort
```

> kubectl apply -f deployment.yml

> kubectl service list

```
|----------------------|-----------------------------------|--------------|---------------------------|
|      NAMESPACE       |                NAME               | TARGET PORT  |            URL            |
|----------------------|-----------------------------------|--------------|---------------------------|
| default              | aws-opensearch-spring-service     |         8080 | http://172.27.16.237:31148 |
```

http://172.27.16.237:31148/api/geo/abandoned-mine/3fa85f64-5717-4562-b3fc-2c963f66afa7

```
{
    "success": true,
    "data": {
        "oriId": "3fa85f64-5717-4562-b3fc-2c963f66afa7",
        "id": "Ivfhe4ABu4ddeLawoKMO",
        "title": "Uranium",
        "description": "Access; Turn Onto Mumford Road Off Of Hwy 648 At Lower Cardiff Lake, Driving East
Approx 600m To Cottage Road, Then South Approx 500m To Mine Road. Road Is Somewhat Overgrown. Continue Walking
Approx 300m To Stripped Area And Partially Covered Shaft. Uranium And Thorium Occurrence; A 45degres Inclined
Shaft 220 M Deep; Underground Devepopment 142 M On 38 M Level Also 2 Pits. Small Mill Foundation; The Year 2000
Survey Reports Partially Covered Inclined Shaft And Possible Second Shaft At South Edge Of Rock Dumps. This
Second Shaft May Be The Mill Foundations.",
        "createdDateTime": null,
        "updatedDateTime": null,
        "abandonedMineIdentifier": "03048",
        "officialName": "Nu-age",
        "new": true,
        "update": false
    }
}
```


3. Deploy Spring Boot Application to AWS EKS

> aws configure
#check configured user

> aws sts get-caller-identity

Prerequisite:

The docker image is ready

3.4 Working with EKS

3.4.1 Install eksctl

1. Install Chocolatey

- Open powershell.exe
- With PowerShell, you must ensure Get-ExecutionPolicy is not Restricted. We suggest using `Bypass` to bypass the policy to get things installed or `AllSigned` for quite a bit more security.
    - Run **Get-ExecutionPolicy**, If it returns `Restricted`, then run `Set-ExecutionPolicy AllSigned` or `Set-ExecutionPolicy Bypass -Scope Process`.
- > Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
- > choco

```
Chocolatey v1.0.0
Please run 'choco -?' or 'choco <command> -?' for help menu.
```

2. Install eksctl

> chocolatey install -y eksctl

> eksctl version

3.4.2 Create repository in Elastic Container Registry (ECR)

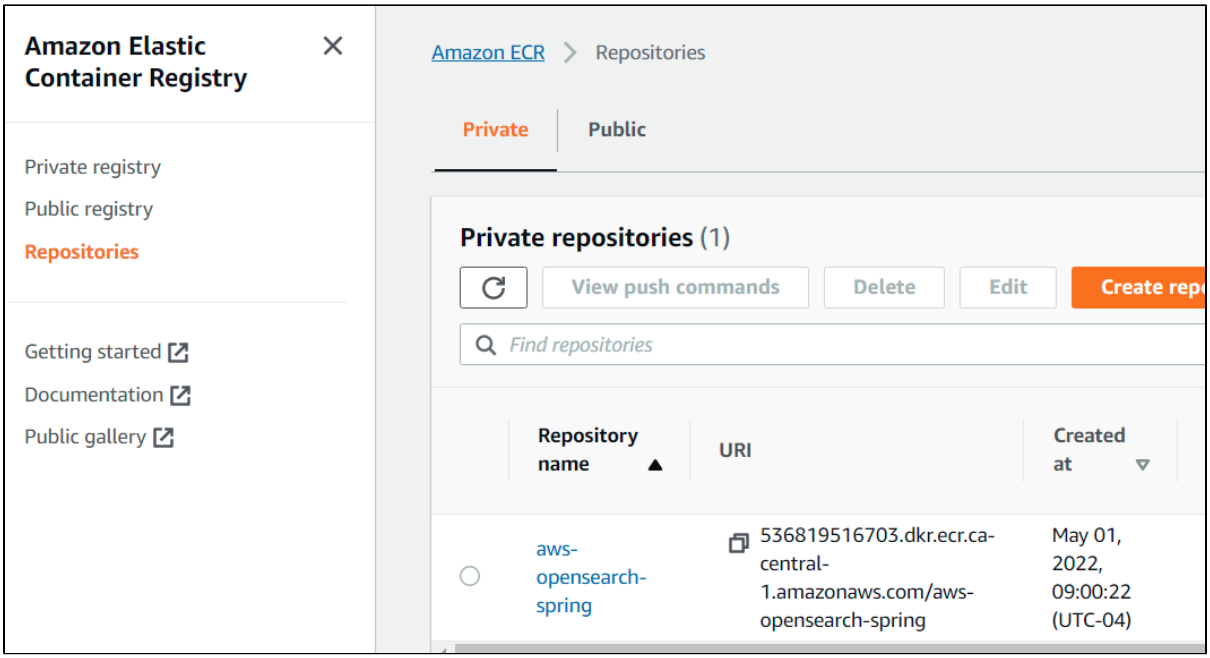
Since we are working on EKS, we need to use image artifactory hosted somewhere. For simplicity, we will leverage ECR and create a repository there to host our demo application images.

To create a repo in ECR:

> aws ecr create-repository --repository-name **aws-opensearch-spring** --image-tag-mutability IMMUTABLE --image-scanning-configuration scanOnPush=true


```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:ca-central-1:536819516703:repository/aws-opensearch-spring",
        "registryId": "536819516703",
        "repositoryName": "aws-opensearch-spring",
        "repositoryUri": "536819516703.dkr.ecr.ca-central-1.amazonaws.com/aws-opensearch-spring",
        "createdAt": 1651410022.0,
        "imageTagMutability": "IMMUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```


Now go to AWS Console => Elastic Container Registry => Make sure region is "ca-central-1" => Repositories



3.4.3 Push local image to ECR

1. Get temp token from ECR
> aws ecr get-login-password --region ca-central-1 | docker login --username AWS --password-stdin **536819516703.dkr.ecr.ca-central-1.amazonaws.com**

```
Login Succeeded
Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
```

After we get the token, next is to **build an image for this ECR**
> docker tag **freeever/aws-opensearch-spring** 536819516703.dkr.ecr.ca-central-1.amazonaws.com/**aws-opensearch-spring**:latest
Push image to ECR
> docker push **536819516703.dkr.ecr.ca-central-1.amazonaws.com/aws-opensearch-spring**:latest
You can confirm that push is a success by logging into AWS Console and choose ECR service

3.4.4 Working with EKS

> eksctl version

0.95.0

1. Create cluster

Create **cluster.yaml**

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: aws-opensearch-spring-cluster
  region: ca-central-1

vpc:
  subnets:
    private:
      us-east-1c: { id: subnet-03111d20473ad03b3 }
      us-east-1d: { id: subnet-09d05a8a02bf37f16 }
      us-east-1b: { id: subnet-0e700de1a7991d27f }

nodeGroups:
  - name: ng-1-workers
    labels: { role: workers }
    instanceType: t3.small
    desiredCapacity: 2
    privateNetworking: true
  - name: ng-2-builders
    labels: { role: builders }
    instanceType: t3.small
    desiredCapacity: 2
    privateNetworking: true
    iam:
      withAddonPolicies:
        imageBuilder: true
```

> eksctl create cluster -f cluster.yaml

To check cluster creation, you can run below

> kubectl get svc

> kubectl get pods --all-namespaces -o wide

2. Deploy the image

Create **eks-deployment.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aws-opensearch-spring-deployment
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: aws-opensearch-spring
  template:
    metadata:
      labels:
        app: aws-opensearch-spring
    spec:
      containers:
        - name: back-end
          image: 536819516703.dkr.ecr.ca-central-1.amazonaws.com/aws-opensearch-spring:latest
          ports:
            - containerPort: 8080
```

> kubectl apply -f eks-deployment.yaml

> kubectl get deployments

```
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
aws-opensearch-spring-deployment   2/2     2            2           17m
```

3. Create service to expose the deployment

Option 1 - Command line

> kubectl expose deployment aws-opensearch-spring-deployment --type=LoadBalancer --name=aws-opensearch-spring-service

> kubectl get service aws-opensearch-spring-service

```
NAME                              TYPE           CLUSTER-IP       EXTERNAL-
IP                                                                PORT(S)          AGE
aws-opensearch-spring-service   LoadBalancer   10.100.188.170   aabb1574cbd40481f8ddecf539832714-1388902096.ca-
central-1.elb.amazonaws.com   5000:32164/TCP   32m
```

http://aabb1574cbd40481f8ddecf539832714-1388902096.ca-central-1.elb.amazonaws.com:5000/api/geo/abandoned-mine/3fa85f64-5717-4562-b3fc-2c963f66afa7

Option 2 - YAML file

Create service to expose the deployment to other members via the NodePort service

- **nodePort** - used **by external members** (non-cluster resources),
- **port** - used **by cluster resources**
- **targetPort** is where our app (container) is currently running

Create **eks-service.yaml**

```
apiVersion: v1
kind: Service
metadata:
  name: aws-opensearch-spring-service
spec:
  selector:
    app: aws-opensearch-spring
  ports:
  - nodePort: 32164    #external traffic
    port: 8080         #port of this service. Cluster members talk via this port
    protocol: TCP
    targetPort: 5000   #where container is actually running
  type: LoadBalancer
```

> kubectl apply -f eks-service.yaml

> kubectl get service

```
aws-opensearch-spring-service   LoadBalancer   10.100.133.21   a2589d5fbca7b43239752822bc893c01-2143273162.ca-
central-1.elb.amazonaws.com   8080:31479/TCP   7s
kubernetes                      ClusterIP      10.100.0.1
<none>                                                                  443/TCP          142m
```

http://a2589d5fbca7b43239752822bc893c01-2143273162.ca-central-1.elb.amazonaws.com:8080/api/geo/abandoned-mine/3fa85f64-5717-4562-b3fc-2c963f66afa7