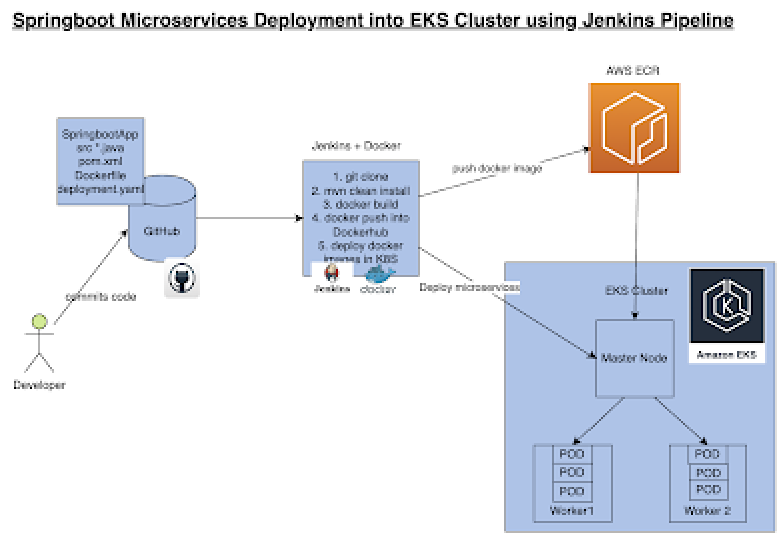
Deploy Springboot Microservices App into Amazon EKS Cluster using Jenkins Pipeline and Kubectl CLI Plug-in | Containerize Springboot App and Deploy into EKS Cluster using Jenkins Pipeline

how to automate springboot microservices builds using Jenkins pipeline and Deploy into AWS EKS Cluster with help of Kubernetes CLI plug-in.

We will use Springboot Microservices based Java application. I have already created a repo with source code + Dockerfile. The repo also have Jenkinsfile for automating the following:

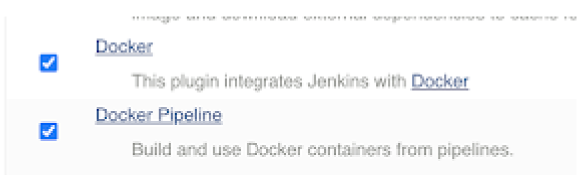
- Automating builds using Jenkins
- Automating Docker image creation
- Automating Docker image upload into AWS ECR
- Automating Deployments to Kubernetes Cluster

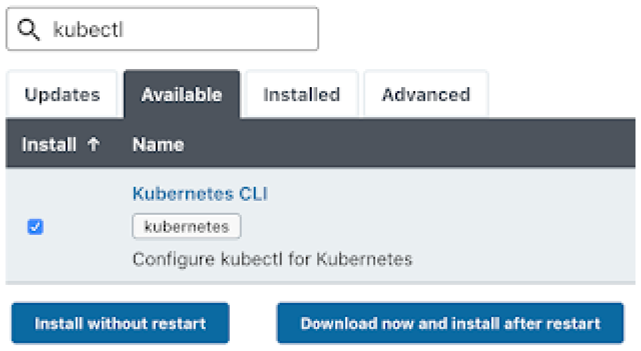


Pre-requistes:
1. Amazon EKS Cluster is setup and running. Click here to learn how to create Amazon EKS cluster.

2. Create ECR repo in AWS
3. Jenkins Master is up and running
4. Docker installed on Jenkins instance
5. Docker, Docker pipeline and Kubernetes CLI plug-ins are installed in Jenkins

Git clone Link https://github.com/awscloudnetwork1/springboot-app

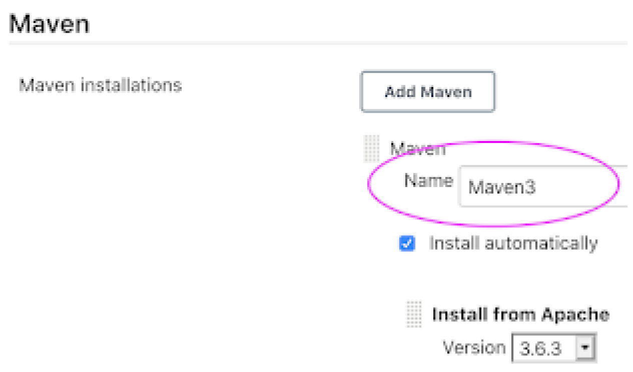

Docker
This plugin integrates Jenkins with Docker
Docker Pipeline
Build and use Docker containers from pipelines.



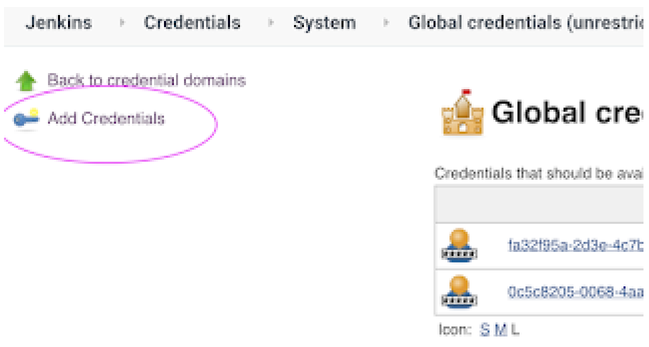
6. Install kubectl on your instance

Step # 1 - Create Maven3 variable under Global tool configuration in Jenkins

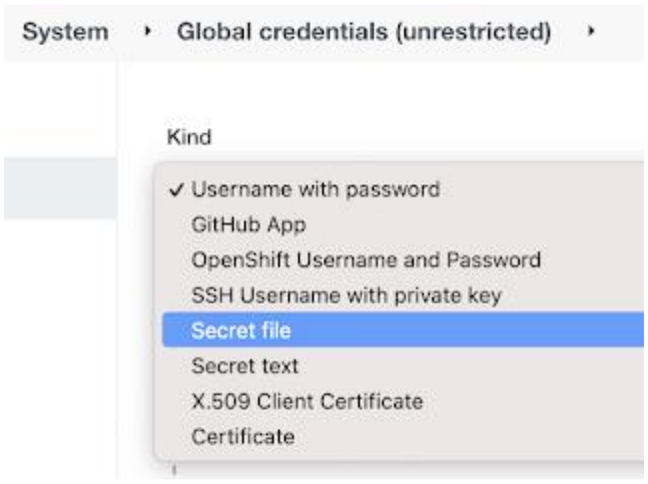
Make sure you create Maven3 variable under Global tool configuration.



Click on Add Credentials, use Kubernetes configuration from drop down.



use secret file from drop down.

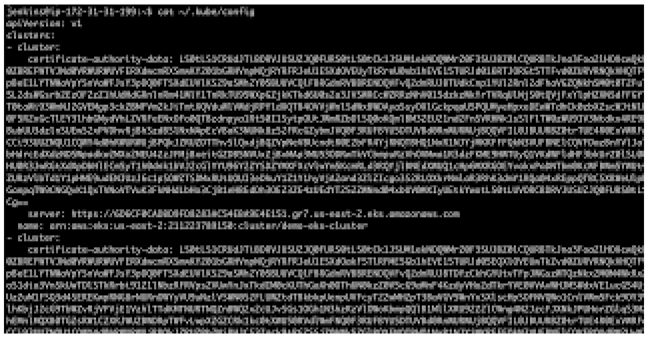


execute the below command to login as jenkins user.
sudo su - jenkins

you should see the nodes running in EKS cluster.

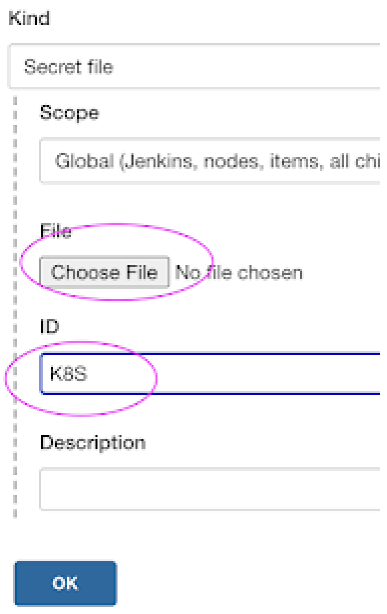
kubectl get nodes

Execute the below command to get kubeconfig info, copy the entire content of the file:
cat /var/lib/jenkins/.kube/config



```
jenkins@ip-172-31-31-199:~$ cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: [illegible]
[illegible]
Cg==
    server: https://[illegible].gr7.us-east-2.eks.amazonaws.com
  name: arn:aws:eks:us-east-2:[illegible]:cluster/demo-eks-cluster
- cluster:
    certificate-authority-data: [illegible]
[illegible]
```

Open your text editor or notepad, copy and paste the entire content and save in a file.
We will upload this file.


Kind

Secret file

Scope

Global (Jenkins, nodes, items, all chi

File

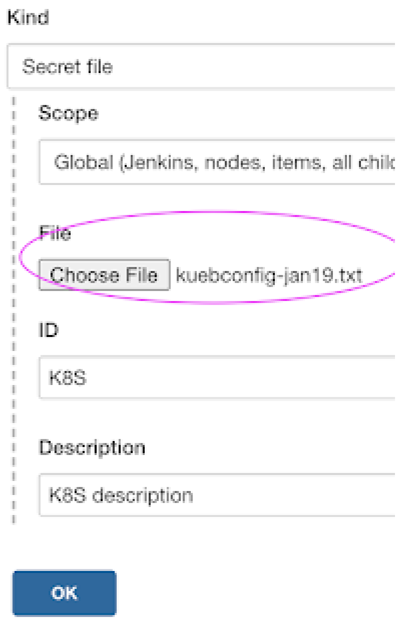
Choose File No file chosen

ID

K8S

Description

Enter ID as K8S and choose File and upload the file and save.


Kind

Secret file

Scope

Global (Jenkins, nodes, items, all chilc

File

Choose File kuebconfig-jan19.txt

ID

K8S

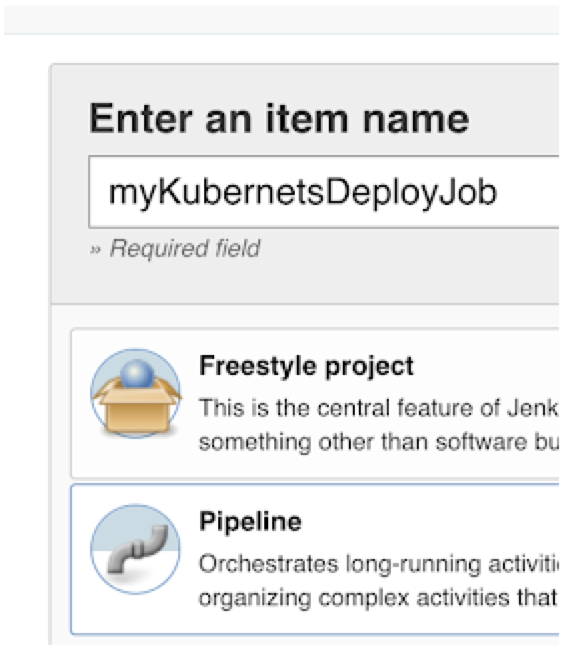
Description

K8S description

Enter ID as K8S and choose enter directly and paste the above file content and save.

Step # 3 - Create a pipeline in Jenkins

Create a new pipeline job.


Enter an item name

myKubernetsDeployJob

» Required field

Freestyle project

This is the central feature of Jenk
something other than software bu

Pipeline

Orchestrates long-running activiti
organizing complex activities that

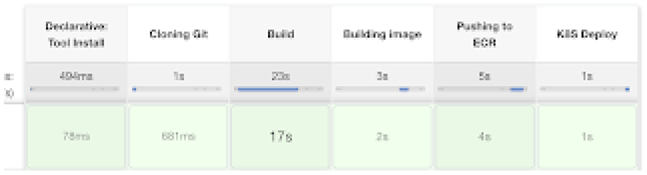
Step # 4 - Copy the pipeline code from below

Make sure you change red highlighted values below as per your settings:
Your docker user id should be updated.
your registry credentials ID from Jenkins from step # 1 should be copied

```
pipeline {
  tools {
      maven 'Maven3'
   }
   agent any
   environment {
      registry = "account_id.dkr.ecr.us-east-2.amazonaws.com/my-docker-repo"
   }

   stages {
      stage('Cloning Git') {
          steps {
              checkout([$class: 'GitSCM', branches: [[name: '*/main']], doGenerateSubmoduleConfigurations: false,
extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '', url:
'https://github.com/akannan1087/springboot-app']]])
          }
      }
     stage ('Build') {
        steps {
          sh 'mvn clean install'
         }
     }
   // Building Docker images
      }
     }
   }

   // Uploading Docker images into AWS ECR
   stage('Pushing to ECR') {
    steps{
        script {
              sh 'aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-
stdin account_id.dkr.ecr.us-east-2.amazonaws.com'
              sh 'docker push account_id.dkr.ecr.us-east-2.amazonaws.com/my-docker-repo:latest'
        }
      }
     }

      stage('K8S Deploy') {
      steps{
          script {
              withKubeConfig([credentialsId: 'K8S', serverUrl: '']) {
              sh ('kubectl apply -f  eks-deploy-k8s.yaml')
              }
          }
      }
     }
   }
}
```

Step # 5 - Build the pipeline

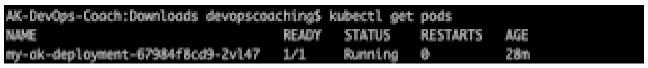
Once you create the pipeline and changes values per your configuration, click on Build now:



Declarative: Tool Install	Cloning Git	Build	Building image	Pushing to ECR	K8S Deploy
494ms	1s	23s	3s	5s	1s
78ms	681ms	17s	2s	4s	1s

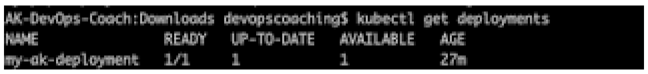
Step # 6 - Verify deployments to K8S

kubectl get pods



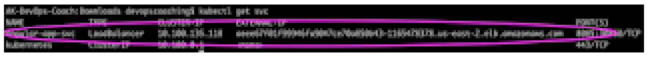
```
AK-DevOps-Coach:Downloads devopscoaching$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
my-ak-deployment-67984f8cd9-2vl47    1/1     Running   0          28m
```

kubectl get deployments



```
AK-DevOps-Coach:Downloads devopscoaching$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
my-ak-deployment   1/1     1            1           27m
```

kubectl get services

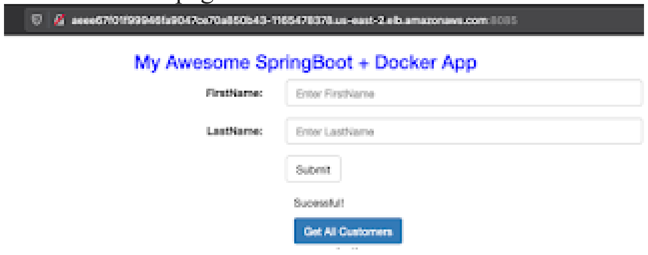


```
AK-DevOps-Coach:Downloads devopscoaching$ kubectl get svc
NAME                 TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)
[illegible]-app-svc  LoadBalancer   10.100.135.118   aeee67f01f99946fa9047ca70a850b43-1165478378.us-east-2.elb.amazonaws.com   8085:[illegible]/TCP
kubernetes           ClusterIP      10.100.0.1       <none>                                                                     443/TCP
```

Steps # 7 - Access SpringBoot App in K8S cluster

Once build is successful, go to browser and enter master or worker node public ip address along with port number mentioned above
http://loadbalancer_ip_address

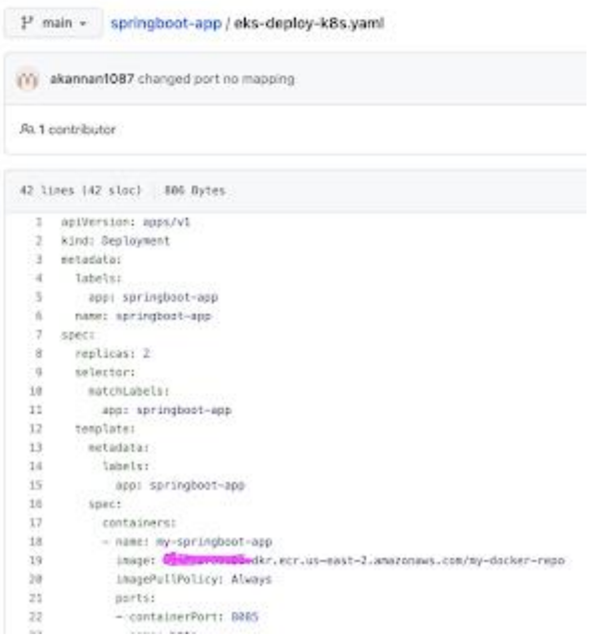
You should see page like below:



aeee67f01f99946fa9047ca70a850b43-1165478378.us-east-2.elb.amazonaws.com:8085

Note:

Make sure you fork my repo https://github.com/awscloudnetwork1/springboot-app
and make changes in eks-deploy-k8s.yaml to pull Docker image from your AWS ECR repo.



main
springboot-app / eks-deploy-k8s.yaml
akannan1087 changed port no mapping
1 contributor
42 lines (42 sloc) 806 Bytes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: springboot-app
  name: springboot-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: springboot-app
  template:
    metadata:
      labels:
        app: springboot-app
    spec:
      containers:
      - name: my-springboot-app
        image: [redacted]dkr.ecr.us-east-2.amazonaws.com/my-docker-repo
        imagePullPolicy: Always
        ports:
        - containerPort: 8085
```