1) Introduction to AWS EKS (Small Overview)
2) Set up EKS Clusters Using eksctl
3) Deploy Simple NodeJs App using kubectl
4) Setting Up CI/CD With AWS CodePipeline + AWS CodeCommit + AWS CodeBuild
5) Configuring AWS EKS Cluster for CI/CD

A practical guide to AWS EKS CI/CD

By Sandip Das

Contents

AWS EKS

Amazon Elastic Kubernetes Service (Amazon EKS) makes it easy to deploy, manage, and scale containerized applications using Kubernetes on AWS.

Amazon EKS runs the Kubernetes management infrastructure for you across multiple AWS availability zones to eliminate a single point of failure. Amazon EKS is certified Kubernetes conformant so you can use existing tooling and plugins from partners and the Kubernetes community. Applications running on any standard Kubernetes environment are fully compatible and can be easily migrated to Amazon EKS.

Amazon EKS is generally available for all AWS customers.

What is EKS?

- Managed Kubernetes service
- Runs upstream K8s - not an AWS fork
- Use `kubectl` and friends
- ECS is cheaper than EKS for small/simple deployments
- EKS is loosely integrated with other AWS services, but this is changing rapidly
- K8s is more popular than ECS or Elastic Beanstalk
- K8s runs on all major cloud providers, and on-premises
- ~60% of K8s deployments run on AWS!
- EKS is secure by default

EKS-Optimized AMI

- AWS-supplied AMI based on Amazon Linux 2
- Preconfigured with Docker, kubelet, AWS IAM Authenticator
- EC2 User Data bootstrap script
- Allows automatic joining to EKS cluster
- Built using Packer

`https://github.com/awslabs/amazon-eks-ami`

Benefits

Let's discuss in short what features it provides

- ➔ No Control plane to manage
- ➔ Secure by default
- ➔ Comformant and Compatible
- ➔ Optimized for cost
- ➔ Build with the community

Cost

Let's discuss about the cost

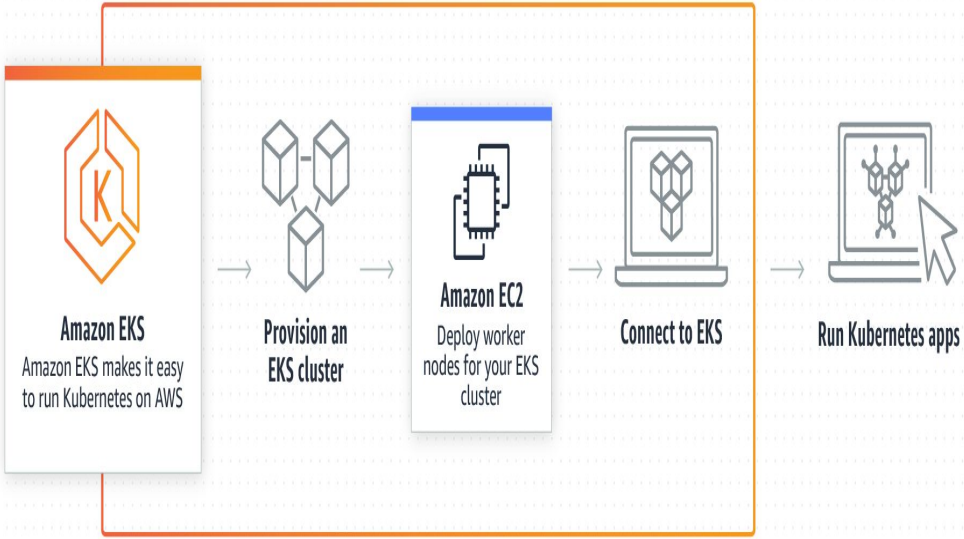
➔ **Each EKS Cluster Cost $0.10/hour i.e. $73/month**

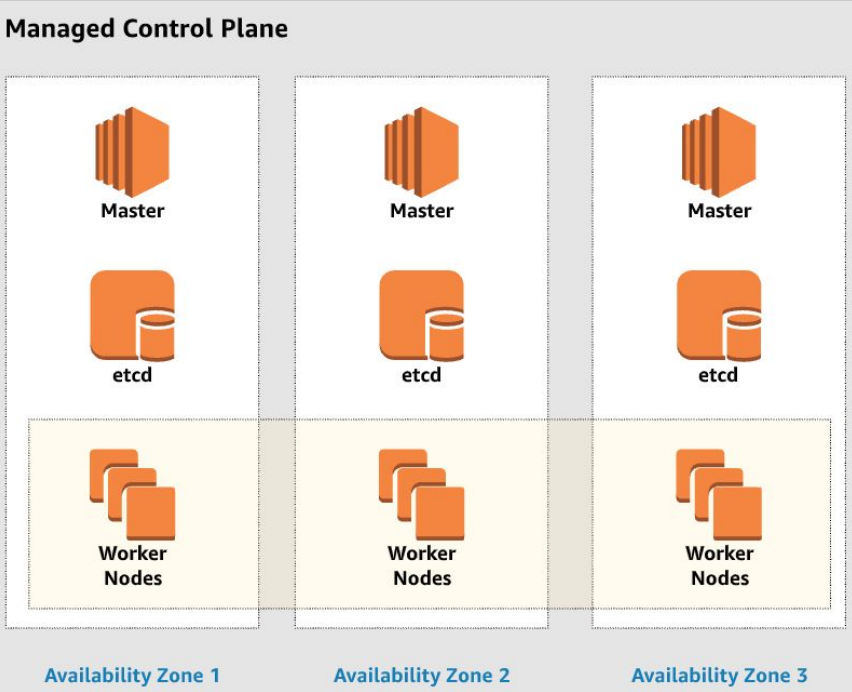
You can use a single Amazon EKS cluster to run multiple applications by taking advantage of Kubernetes namespaces and IAM security policies.

➔ **EC2 Instances & EBS volumes used in AS Worker Nodes**

You pay for AWS resources , you create to run your Kubernetes worker nodes. You only pay for what you use, as you use it; there are no minimum fees and no upfront commitments.

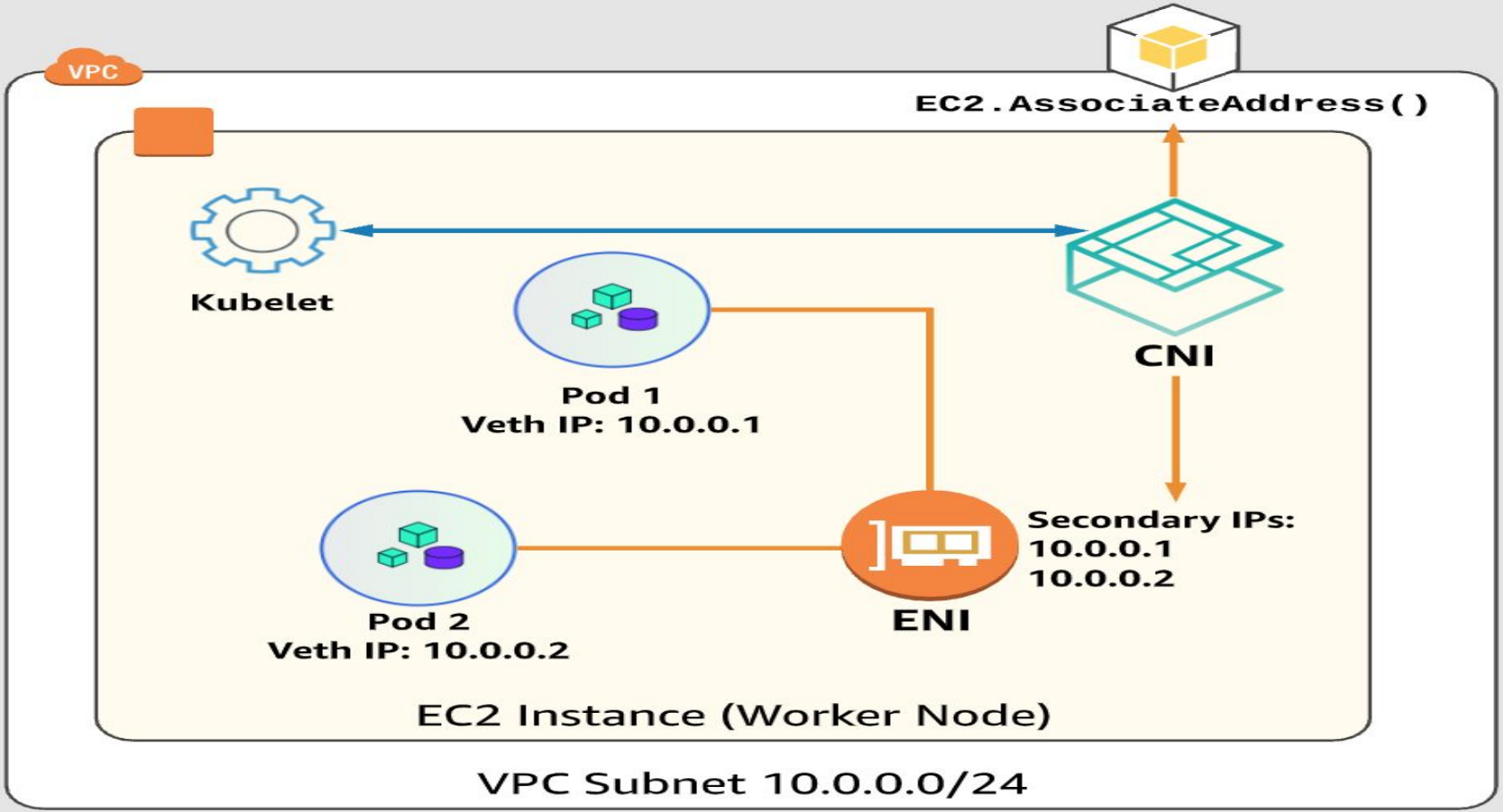
EKS Architecture





- EKS provieds K8s master nodes, API servers, etcd layer
- 3 master and 3 etcd nodes by default
- Backups, etcd snapshots, autoscaling included
- You provision and manage the EC2 worker nodes
- Unlike kops, you don't roll your own master
  (https://github.com/kubernetes/kops)
- Masters and etcd are Multi-AZ
- EKS scales master nodes for you

Kubernetes and VPC Networking



Companies Using AWS EKS

Let’s Create An AWS EKS Cluster & Deploy A Simple NodeJs Application

Make sure to install awscli, eksctl, kubectl,aws-iam-authenticator

To make sure everything installed and all set, run:

sh prereqs.sh

Project GitHub url:
https://github.com/sd031/aws_codebuild_codedeploy_nodeJs_demo

All Kubernetes related config files are in “eks_cicd” folder

Install Essential Tools

We will need below tools to be installed

- **Install aws cli**
  https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html *(install cli v2)*
- **Install kubectl**
  https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html
- **Install aws-iam-authenticator**
  https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html
- **Install eksctl**

  https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html or
  https://eksctl.io/introduction/installation/

Create AWS EKS Cluster

Create Cluster:

```
eksctl create cluster -f cluster.yaml
```

Then check for the message:

```
 EKS cluster "<cluster name>" in
"<aws region>" region is ready
```

Check that kubectl client get auto set properly or not by:

```
cat /home/ec2-user/.kube/config
```

If want to Delete Cluster anytime:

```
eksctl delete cluster -f cluster.yaml
```

Note: Running delete command will remove all the resources

cluster.yaml

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

metadata:
 name: cicd-demo #cluster name
 region: us-west-2 #desired region

nodeGroups:
 - name: ng-1 #cluster node group name
   instanceType: t2.medium #desired instance type
   desiredCapacity: 3 #desired nodes count / capacity
   ssh:
     allow: false
```

Build the docker image using Dockerfile

Run Below command from

Project Directory to build the image

Before that install docker:

https://docs.docker.com/engine/install/

To Make Docker Build

docker image build -t <image_name>:tag .

(“.” refer to current directory)

e.g.

```
docker image build -t cicd-demo:v1 .
```

Test image running fine or not:

```
docker run -d --name cicd-demo -p 3000:3000 cicd-demo:v1
```

Dockerfile

```
FROM node:14

# Setting working directory. All the path will be
relative to WORKDIR
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json
AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 3000
CMD [ "node", "index.js" ]
```

Publishing Image to AWS ECR

It’s just like Docker Hub, where we can push and Pull image and we can use it with AWS ECR same way. To get the docker login and auto execute the login the command is:

aws ecr get-login-password --region <region name> | docker login --username AWS --password-stdin <account_id>.dkr.ecr.us-west-2.amazonaws.com

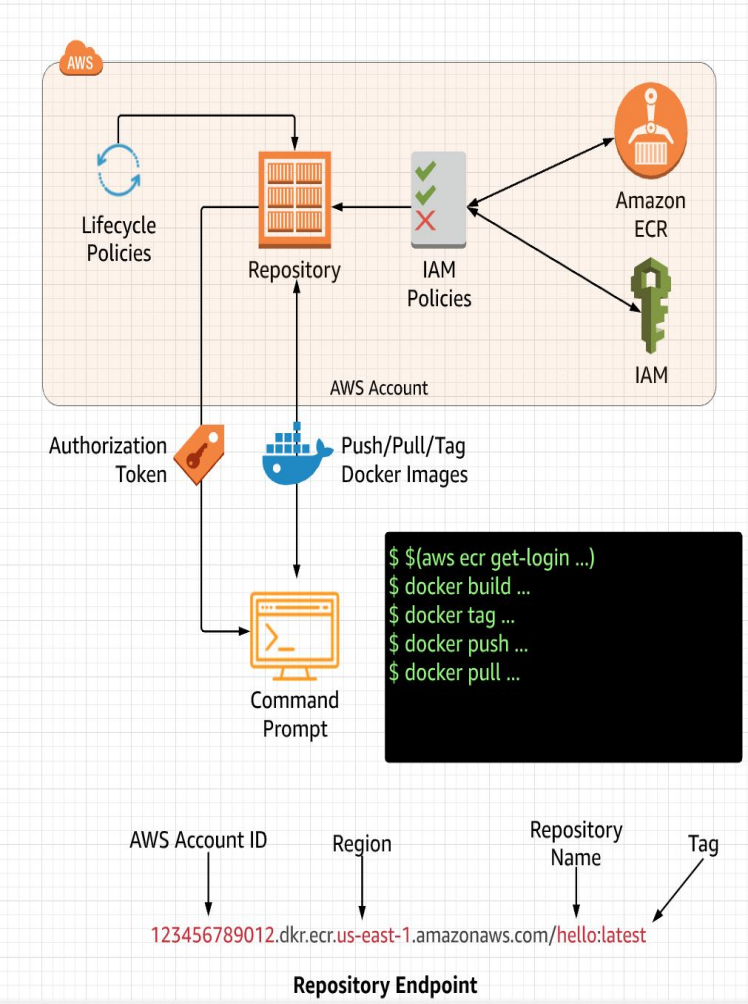
E.g aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 123456789012.dkr.ecr.us-west-2.amazonaws.com

After this create a ECR repo from AWS Management console, it will give a url like Showing in right side diagram, after that tag The image: (All command are already given in AWS ECR UI, just use the same)

`docker tag cicd-demo:latest 123456789012.dkr.ecr.us-west-2.amazonaws.com/cicd-demo:latest`

`docker push 123456789012.dkr.ecr.us-west-2.amazonaws.com/cicd-demo:latest`

Make sure you use the right tags



Finally Let's deploy the Application

```
kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

If any issue check logs:

kubectl get pods

kubectl describe pod pod_name_here
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
 labels:
   app.kubernetes.io/name: cicd-demo
   app.kubernetes.io/instance: cicd-demo-instance
   app.kubernetes.io/version: "1.0.0"
   app.kubernetes.io/component: backend
   app.kubernetes.io/managed-by: kubectl
 name: cicd-demo
spec:
 selector:
   app: cicd-demo
 type: LoadBalancer
 ports:
  -  protocol: TCP
     port: 80
     targetPort: 3000
```

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
   app.kubernetes.io/name: cicd-demo
   app.kubernetes.io/instance: cicd-demo-instance
   app.kubernetes.io/version: '1.0.0'
   app.kubernetes.io/managed-by: kubectl
 name: cicd-demo-deployment
spec:
 replicas: 1
 selector:
   matchLabels:
     app: cicd-demo
 template:
   metadata:
     labels:
       app: cicd-demo
   spec:
     containers:
       - image: 120717539064.dkr.ecr.us-west-2.amazonaws.com/cicd-demo:latest
         imagePullPolicy: Always
         name: cicd-demo
         ports:
           - containerPort: 3000
```

Let’s set-up the pipeline

We will be creating one AWS Pipeline project e.g. cicd-demo

Then Configure AWS CodeCommit as source
And then using AWS CodeBuild We will make Build and Deploy changes in AWS EKS

While Configuring AWS CodeBuild , make sure to set the build spec file properly and to enter the required environment variables as follow:

AWS_DEFAULT_REGION

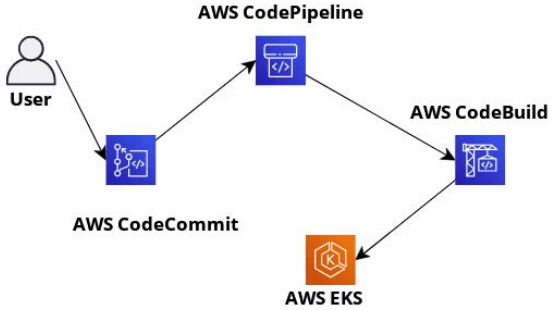
AWS_CLUSTER_NAME

AWS_ACCOUNT_ID

IMAGE_REPO_NAME

IMAGE_TAG

Also add, ecr, eks, s3 access to the build role



buildspec.yml

```
version: 0.2
run-as: root

phases:

  install:
    commands:
      - echo Installing app dependencies...
      - curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/b
      - chmod +x ./kubectl
      - mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/b
      - echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
      - source ~/.bashrc
      - echo 'Check kubectl version'
      - kubectl version --short --client
      - chmod +x k8s/Configuring-Cluster-cloud-formation/prereqs.sh
      - sh k8s/Configuring-Cluster-cloud-formation/prereqs.sh

  pre_build:
    commands:
      - echo Logging in to Amazon EKS...
      - aws eks --region $AWS_DEFAULT_REGION update-kubeconfig --name $AWS_CLUSTER_NAME -
      - echo check config
      - kubectl config view --minify
      - echo check kubectl access
      - kubectl get svc
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --username
      - REPOSITORY_URI=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_R
      - docker pull $REPOSITORY_URI:$IMAGE_TAG


  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build --cache-from $REPOSITORY_URI:$IMAGE_TAG -t $IMAGE_REPO_NAME:$IMAGE_T
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGIO

  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO
      - echo Push the latest image to cluster
      - kubectl apply -f eks_cicd/deployment.yaml
      - kubectl rollout restart -f eks_cicd/deployment.yaml
```

Configuring AWS EKS Cluster for CI/CD

Even though the CodeBuild role has permission to authenticate to the cluster, it doesn't have the requisite RBAC access to do any other action on the cluster. You can even list pods in the cluster. You should read the following quote from EKS documentation:

"When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted system:masters permissions in the cluster's RBAC configuration. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the aws-auth ConfigMap within Kubernetes."

So, we need to **edit the aws-auth configmap.**

How do you do that?

If you are the one who created the cluster, you need to run the following in your local terminal to create a copy of the `aws-auth` configMap.

Option 1:

Since we are using eksctl tool, we can simply do the role mapping by running this command:

eksctl create iamidentitymapping --cluster cluster_name_here --arn role_arn_here --group system:masters --username intended_user_name_here

E.g.

eksctl create iamidentitymapping --cluster my-cluster-1 --arn arn:aws:iam::123456:role/testing --group system:masters --username admin

Check below URL for documentation

https://eksctl.io/usage/iam-identity-mappings/

Configuring AWS EKS Cluster for CI/CD

Option 2:

```
aws eks update-kubeconfig --name eks-cluster-name --region aws-region
```

```
kubectl get configmaps aws-auth -n kube-system -o yaml > aws-auth.yaml
```

Now, edit your aws-auth.yaml, and add the following under **data.mapRoles**

```
-rolearn: arn:aws:iam::510442909921:role/role-name
 username: role-name
 groups:
   -system:masters
```

Apply this configuration from your terminal:

```
kubectl apply -f aws-auth.yaml
```

If face any issue, follow this debug steps:

https://aws.amazon.com/premiumsupport/knowledge-center/amazon-eks-cluster-access/

aws-auth.yaml sample file

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn:
arn:aws:iam::11122223333:role/EKS-Worker-NodeInstance
Role-1I00GBC9U4U7B
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn:
arn:aws:iam::11122223333:role/designated_role
      username: designated_role
      groups:
        - system:masters
```

Clean up everything

Since we have used eksctl, it’s a lot easier

Delete cluster:

```
eksctl delete cluster -f cluster.yaml
```

Behind the scene the cloud formation stack will get deleted and accordingly resources will be deleted as well, must do it if you are doing in development or test as a temporary deployment otherwise it will cost you a lot

There are few more things you need to know

This demo is just the start points of CICD and there is a lot more out there, the more you use it , the more experience you will gather, so I will highly suggest try by yourself and deploy your own AWS EKS Cluster.

After trying the basic app deployments , the next thing you might be interested to learn are:

1) Using Spot instances with Kubernetes and save 90% of cost
2) Types of Deployments available in Kubernetes Deployment and how to configure it
3) Types of Services available in Kubernetes Service and how to configure it.
4) CI/CD with Kubernetes
5) Logging with Cloud Trail
6) Logging to Cloudwatch with Fluentd
7) Complex Authentication , Roles etc

Good luck!

I hope you'll use this knowledge and build awesome solutions.

If any issue contact me in Linkedin:
https://www.linkedin.com/in/sandip-das-developer/