I choose **AWS** to do this assignment

First, I downloaded command line tool `kubectl` and `eksctl`

- Download kubectl
  ```
  curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.6/2023-01-30/bin/darwin/amd64/kubectl
  ```
- Download eksctl
  ```
  brew upgrade eksctl && { brew link --overwrite eksctl; } || { brew tap weaveworks/tap; brew install weaveworks/tap/eksctl; }
  ```

Once the required command line tool been downloaded, I can move onto the assignment.

Task 1: Launch a Kubenetes cluster. You will need to configure the cluster with one master and at least two workers. You can build your own cluster from scratch on virtual machines, or use cloud-based Kubernetes services such as AWS EKS, Azure Kubernetes Service, or Google Kubernetes Engine.

Create cluster

```
eksctl create cluster --name qcCloud --region us-east-1 --version 1.25 --vpc-private-subnets subnet-xxxxxxxx,subnet-xxxxxxxx --vpc-pub
```

Note: *the creation of cluster takes a few minutes, it took roughly 14 min in my case. Also, 2 private subnets and 2 public subnets should be added. Otherwise, the creation of node group will fail due to no public subnet.*

This one can actually be skipped, and we dont have to create node group ourself because eksctl create node group for us when create cluster. However, the node group created by eksctl has 2 m5.large nodes. Therefore, I used the following command to create a node group that has 3 t2.micro node and delete the previous node group

- Delete the group created by eksctl
  ```
  eksctl delete nodegroup --cluster qcCloud1 --name ng-ff171e0a
  ```
- Create new group that has 3 t2.micro
  ```
   --cluster qcCloud1 \
   --region us-east-1 \
   --name qcCloudPj1NodeGroup1 \
   --node-ami-family AmazonLinux2 \
   --node-type t2.micro \
   --nodes 3 \
   --nodes-min 2 \
   --nodes-max 4 \
   --ssh-access \
   --ssh-public-key qcCloud
  ```

Task 2: Deploy a containerized application which run multiple instances of the same container.

First create a container with docker and deploy an app on it. I used the tutorial app provided in docker's documentation.

1. cd into the app directory, and create a a file named `Dockerfile` with no file extension.
2. Write following content into `Dockerfile`

```
FROM amazonlinux:2

RUN yum update -y && \
    yum install -y curl && \
    curl -sL https://rpm.nodesource.com/setup_14.x | bash - && \
    yum install -y nodejs && \
    yum clean all

WORKDIR /app

COPY package.json yarn.lock ./

RUN curl -sS https://dl.yarnpkg.com/rpm/yarn.repo | tee /etc/yum.repos.d/yarn.repo && \
    rpm --import https://dl.yarnpkg.com/rpm/pubkey.gpg && \
    yum install -y yarn && \
    yarn install --production && \
    yum remove -y yarn && \
    yum clean all

COPY . .

CMD ["node", "./src/index.js"]
```

3. Create docker image. When I deployed the image, it stuck in a crash and loop back cycle. And I found the error to be imcompatible platform. Then I check the platform that my cluster node is amd64, therefore the platform is specified here.
   `docker build --platform linux/amd64 -t getting-started .`
4. Tag the image before push
   `docker tag getting-started jingshiliu/getting-started`
5. Push to docker hub
   `docker push jingshiliu/getting-started`

Second, create a kubernetes deployment config file (kubernetes_deploy_config.yaml)

```
apiVersion: apps/v1 kind: Deployment metadata: name: qc-cloud-pj1 spec: replicas: 3 selector: matchLabels: app: qc-cloud-pj1 template:
```

Then, deploy

```
kubectl apply -f kubernetes_deploy_config.yaml
```

And we can check the status by

```
kubectl get deployment qc-cloud-pj1
```

Task 3: Scale the pod to more container instances.

This task is easy, we can do it simply using the following command

```
kubectl scale deployment/qc-cloud-pj1 --replicas=5
```

We can see the replicas in the deployment, and it is increased from 3 to 5

```
kubectl get pods -l app=qc-cloud-pj1
```

> Also, I want to write down some thought or things I learned in this task. Scaling in cloud computing in general is to adjust the amount of computing resources allocated to an application. And in Kubernetes, there is no such thing called primary node or original node. Instead, a set of identical nodes or copies is called ReplicaSet and all of them are a replica of each other. ReplicaSet is used to ensure the availablity and fault tolerance, so that application is still running if one node is down. Even though they are identical node, they receive request from different user and deals request from different resources. In combination with the idea of ReplicaSet and scaling in cloud computing, we can easily see why `kubectl scale` is to simply change the size of ReplicaSet.

Task 4: Update the application with a new software version.

As mentioned before, I used the Docker tutorial app. I modified the app. So it is now a new version. Since the container is runned on image. When we update software version, we create new image and update that new image to the deployment.
Let's rebuild the image

```
docker build --platform linux/amd64 -t getting-started:latest .
```

Tag it

```
docker tag getting-started jingshiliu/getting-started
```

Push it

```
docker push jingshiliu/getting-started
```

Now update the application with new software version. As simple as scale the app.

```
kubectl set image deployment/qc-cloud-pj1 qc-cloud-pj1=jingshiliu/getting-started:v2
```

Task 5: Delete the application and stop the Kubernetes cluster.

Delete application on Kubernetes

```
kubectl delete deployment qc-cloud-pj1
```

Delete cluster on EKS

```
eksctl delete cluster --name qcCloud1
```