Lab 1 – Kubernetes Intro (EKS)

Task 1 – Create your EKS Cluster

Prerequisite – AWS IAM account configured to a specific region (Ireland – eu-west-1)

1) Ensure you are logged into the IAM Account and not a root user

2) Navigate to IAM Roles within AWS

3) Create a role called `eksClusterRole` which uses EKS as a service (Select from the dropdown). Choose `EKS-Cluster` as a use case. The only permission it needs is `AmazonEKSClusterPolicy`. No need for a description and create this role.

4) Create a role called `AmazonEKSNodeRole`, uses EC2 and has permissions:

AmazonEKSWorkerNodePolicy
AmazonEC2ContainerRegistry
AmazonEKS_CNI_Policy

5) Navigate to EKS within AWS GUI and start the steps to Create a Cluster (Add a Cluster)

6) Give the Cluster a memorable and simple name, set the version to 1.22 and choose your ekcClusterRole then click `next`

7) No need to change any settings on the Networking page, check the correct VPC (default) and subnets are set as normal

8) Follow the steps clicking `Next` and `Review and Create`

Task 2 Create node group

1) Wait 10+ minutes for the cluster to be created before accessing the cluster and then clicking on the `Compute` subgroup

2) Create a `Node Group` by clicking on the `Add Node Group` that is composed of `t3.micro` machines using Amazon Linux and the `AmazonEKSNodeRole` created earlier

3) Specify the desired, minimum and maximum number of nodes to 2 (this will change through further labs however)

4) Review the steps and Click `Create`

Lab 2 – Kubectl Setup

Task – Install and set up Kubectl

Prerequisites – EKS Cluster setup with Node group running 2x t3.micro EC2 machines. Separate EC2 machine (t2.micro) that has ports 22, 80 open (not part of node group)

1) SSH to the separate EC2, this will be referred to as `controller`

2) Run the following commands in this EC2

```
sudo apt update
sudo apt install awscli
aws configure
# (enter access key, secret key, region eu-west-1)

curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-
09/bin/linux/amd64/kubectl

chmod +x ./kubectl

mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export
PATH=$PATH:$HOME/bin

echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc

kubectl version --short --client

aws eks update-kubeconfig --region eu-west-1 --name <name of cluster>

kubectl get svc
```

3) If the Kubectl get svc shows a ClusterIP and doesn't fail, you have successfully configured and connected to the EKS cluster

Troubleshooting: Ensure the region code and name are correct, login to AWS GUI and check the health and status of your Cluster and Node Groups

Lab 3 – Pod Configuration

Task – Create an nginx pod running a basic image

Prerequisites – EKS cluster with 2x t3.micro in node group. Controller EC2 with Kubectl configured to EKS cluster

1) Create a directory for Kubernetes work and create a file called nginx-pod.yaml
2) Enter the following in nginx-pod.yaml (via nano, vim, VSCode, Git Clone down etc.) :

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:alpine
    ports:
    - containerPort: 80
```

3) Run the Kubectl command to create the pod `kubectl create -f nginx-pod.yaml`
4) Exec into the running pod with `kubectl exec -it nginx –sh`
5) While in the nginx pod `curl localhost` to see nginx working
6) The pod can be safely deleted after exiting the pod with `kubectl delete pod nginx`

Stretch goal – Create a MySQL pod passing in MYSQL_ROOT_PASSWORD and MYSQL_DATABASE as env vars, check the DockerHub docs for info. Exec into the pod and login to the MySQL Server

Lab 4 – Service Configuration

Task – Create a Service configuration using ClusterIP for internal and Load balancer for external

Prerequisites – EKS Cluster with 3x t3.micro Node Group (1 more than previous exercises, node group can be edited) Controller EC2 with kubectl setup

Using prebuilt Python front and backend images saved to DockerHub (Source code available here https://gitlab.com/Reece-Elder/devops-pythonfront-back-docker). ClusterIP used to connect backend to front end and load balancer used to connect open internet to front end.

Create a frontend.yaml and backend.yaml, within the frontend and backend the image to be used for the pod will be reeceqa/python-frontend and reeceqa/python-backend respectively.

Frontend

Add the following to the frontend.yaml code, you will have to configure the pod based on the nginx pod and source code:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: LoadBalancer  # set the type of Service
  selector:
    app: frontend     # referencing the Pod's label
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 5000
---
# Pod config below
```

Apply the frontend configuration with `kubectl apply -f frontend.yaml`

Backend

Add the following to the backend.yaml code, you will have to configure the pod based on the nginx pod and source code.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  type: ClusterIP     # set the type of Service
  selector:
    app: backend      # referencing the Pod's label
  ports:
  - protocol: TCP
    port: 5001
    targetPort: 5001
---
```

Apply the backend configuration with `kubectl apply -f backend.yaml`

Config

After setting the back and front end pods up you can run the following command `kubectl get svc` to get the IP addresses of the Service resources. To access the app you need to enter the LoadBalancer External IP into a browser.

Troubleshooting: The LoadBalancer External IP may show <pending>, EKS creates load balancer resources and takes a little time to be setup. Check the status of the pods `kubectl describe pod <name>` as well as `kubectl show pods` to check all pods are working and if it's just waiting

Lab 5 – Kubernetes Deployments

Task 1 – Create Deployment of nginx pods

Prerequisites – EKS Cluster with 3x t3.micro Node Group (1 more than previous exercises, node group can be edited) Controller EC2 with kubectl setup

1) Create a file called deployment-nginx.yaml in a new directory and add the following:

```yaml
apiVersion: apps/v1
kind: deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nginx        # must be the same value as the label in the template
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
  template:
    metadata:
      labels:
        app: nginx      # this label is assigned to each pod in the set
    spec:
      containers:
      - name: nginx
        image: nginx:alpine
```

2) This .yaml creates a deployment of 4 replicas of the nginx pod created earlier.

3) To run the deployment use `kubectl apply -f nginx-deployment.yaml`

4) To change the number of replicas use the below:

```
kubectl scale deployment/nginx-deployment --replicas=5
```

5) We can update the images using the rolling update function with the below:

```
kubectl set image deployment/nginx nginx=nginx:latest
```

We won't see any change in the code with this update, but it should be working.

Task 2 – Deploy Python front and backend using deployment

Prerequisites – EKS Cluster with 3x t3.micro Node Group (1 more than previous exercises, node group can be edited) Controller EC2 with kubectl setup

Deploy the python front and backend flask app docker images from Lab 4 using a deployment of 2 front and backend pods.

Use the RollingUpdate to change the front end image to reeceqa/python-frontend-purple. If configured properly there should be no down time between building a new pod and the background won't be purple immediately when accessing the browser.

Stretch goal – Deploy Task 1 and 2 from https://gitlab.com/Reece-Elder/dockerfileexercise as a Deployment of pods.

Lab 6 – Namespaces

Task – Configure deployments using new namespaces and deleting resources via namespace

Prerequisites – EKS Cluster with 3x t3.micro Node Group (1 more than previous exercises, node group can be edited) Controller EC2 with kubectl setup

1) Access all current Namespaces with `kubectl get ns`
2) Create a new namespace `kubectl create ns dev-cli`
3) Create a file called namespace.yaml that contains the following and then apply it as normal

```yaml
kind: Namespace
apiVersion: v1
metadata:
  name: dev-manifest
  labels:
    name: dev-manifest
```

4) Create an nginx manifest yaml file as normal but within the `metadata` add the key value pair `namespace: dev-cli`
5) Create a 2nd nginx manifest with the namespace being dev-manifest
6) Apply the nginx manifests and view the pods
7) Delete a namespace with `kubectl delete ns dev-cli` and see how this deletes all resources associated with this namespace. A quick and simple way to remove all resources (but is very destructive)