Create Application Service

Objective: Create a more advanced deployment, using kubectl to update with apply, rollout, and deploy. Plus, work with ReplicationControllers for the first time.

Preparation: You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using Minikube. Also, navigate to the appropriate lab folder and confirm the below mentioned files are present.

Outcome: Once complete the participant will have experience in creating deployments in YAML files, lauching them with kubectl and then updating them.

Data Files: `deployment.yaml, deployment-scale.yaml, deployment-update.yaml, nginx-deployment.yaml`

Step 1. Creating a service for an application running in two pods

1. Run a Hello World application in your cluster:

```
$ kubectl run hello-world --replicas=2 --labels="run=load-balancer-example" --image=gcr.io/google-samples/node-hello:1.0 --port=8080
```

The command creates a Deployment object and an associated ReplicaSet object. The ReplicaSet has two Pods, each of which runs the Hello World application.

2. Display information about the Deployment:

```
$ kubectl get deployments hello-world
$ kubectl describe deployments hello-world
```

3. Display information about your ReplicaSet objects:

```
$ kubectl get replicasets
$ kubectl describe replicasets
```

4. Create a Service object that exposes the deployment:

```
$ kubectl expose deployment hello-world --type=NodePort --nam
e=example-service
```

5. Display information about the Service:

```
$ kubectl describe services example-service
```

The output is similar to this:

```
Name:                   example-service
 Namespace:              default
```

```
Labels:                   run=load-balancer-example
Selector:                 run=load-balancer-example
Type:                     NodePort
IP:                       10.32.0.16s
Port:                     <unset> 8080/TCP
NodePort:                 <unset> 31496/TCP
Endpoints:                10.200.1.4:8080,10.200.2.5:8080
Session Affinity:         None
No events.
```

Make a note of the `NodePort` value for the service. For example, in the preceding output, the `NodePort` value is `31496`.

6. List the pods that are running the Hello World application:

```
$ kubectl get pods --selector="run=load-balancer-example" --o
utput=wide
```

The output is similar to this:

```
NAME                           READY   STATUS    ...  IP
      NODE
hello-world-2895499144-bsbk5   1/1     Running   ...  10.200.
1.4   worker1
hello-world-2895499144-m1pwt   1/1     Running   ...  10.200.
2.5   worker2
```

7. Get the public IP address of one of your nodes that is running a Hello World pod. How you get this address depends on how you set up your cluster. For example, if you are using Minikube, you can see the node address by running `kubectl cluster-info`. If you are using Google Compute Engine instances, you can use the gcloud compute instances list command to see the public addresses of your nodes.

8. [OPTIONAL] On your chosen node, create a firewall rule that allows TCP traffic on your node port. For example, if your Service has a NodePort value of 31568, create a firewall rule that allows TCP traffic on port 31568. (Ask your instructor before doing this step.)

9. Use the node address and node port to access the Hello World application:

```
$ curl http://<public-node-ip>:<node-port>
```

Where `<public-node-ip>` is the public IP address of your node, and `<node-port>` is the NodePort value for your service.

Output:

```
Hello Kubernetes!
```

Step 2. Clean Up

1. To delete the Service, enter this command:

```
$ kubectl delete services example-service
```

2. To delete the Deployment, the ReplicaSet, and the Pods that are running the Hello World application, enter this command:

```
$ kubectl delete deployment hello-world
```