Exercise 9.1: Deploy A New Service

Overview

Services (also called **microservices**) are objects which declare a policy to access a logical set of Pods. They are typically assigned with `labels` to allow persistent access to a resource, when front or back end containers are terminated and replaced.

Native applications can use the `Endpoints` API for access. Non-native applications can use a Virtual IP-based bridge to access back end pods. `ServiceTypes Type` could be:

- **ClusterIP** default - exposes on a cluster-internal IP. Only reachable within cluster
- **NodePort** Exposes node IP at a static port. A ClusterIP is also automatically created.
- **LoadBalancer** Exposes service externally using cloud providers load balancer. `NodePort` and `ClusterIP` automatically created.
- **ExternalName** Maps service to contents of `externalName` using a `CNAME` record.

We use services as part of decoupling such that any agent or object can be replaced without interruption to access from client to back end application.

1. Deploy two **nginx** servers using **kubectl** and a new `.yaml` file. The kind should be `Deployment` and label it with `nginx`. Create two replicas and expose port 8080. What follows is a well documented file. There is no need to include the comments when you create the file. This file can also be found among the other examples in the tarball.

```
student@cp:~$ vim nginx-one.yaml
```

nginx-one.yaml

```
apiVersion: apps/v1
# Determines YAML versioned schema.
kind: Deployment
# Describes the resource defined in this file.
metadata:
  name: nginx-one
  labels:
    system: secondary
# Required string which defines object within namespace.
  namespace: accounting
# Existing namespace resource will be deployed into.
spec:
  selector:
    matchLabels:
      system: secondary
# Declaration of the label for the deployment to manage
  replicas: 2
# How many Pods of following containers to deploy
  template:
    metadata:
      labels:
        system: secondary
# Some string meaningful to users, not cluster. Keys
```

```
# must be unique for each object. Allows for mapping
# to customer needs.
    spec:
      containers:
# Array of objects describing containerized application with a Pod.
# Referenced with shorthand spec.template.spec.containers
      - image: nginx:1.20.1
# The Docker image to deploy
        imagePullPolicy: Always
        name: nginx
# Unique name for each container, use local or Docker repo image
        ports:
        - containerPort: 8080
          protocol: TCP
# Optional resources this container may need to function.
      nodeSelector:
        system: secondOne
# One method of node affinity.
```

2. View the existing labels on the nodes in the cluster.

```
student@cp:~$ kubectl get nodes --show-labels
```

```
<output_omitted>
```

3. Run the following command and look for the errors. Assuming there is no typo, you should have gotten an error about about the `accounting` namespace.

```
student@cp:~$ kubectl create -f nginx-one.yaml
```

```
Error from server (NotFound): error when creating
"nginx-one.yaml": namespaces "accounting" not found
```

4. Create the namespace and try to create the deployment again. There should be no errors this time.

```
student@cp:~$ kubectl create ns accounting
```

```
namespace/accounting" created
```

```
student@cp:~$ kubectl create -f nginx-one.yaml
```

```
deployment.apps/nginx-one created
```

5. View the status of the new pods. Note they do not show a `Running` status.

```
student@cp:~$ kubectl -n accounting get pods
```

```
NAME                         READY     STATUS    RESTARTS   AGE
nginx-one-74dd9d578d-fcpmv   0/1       Pending   0          4m
nginx-one-74dd9d578d-r2d67   0/1       Pending   0          4m
```

6. View the node each has been assigned to (or not) and the reason, which shows under events at the end of the output.

```
student@cp:~$ kubectl -n accounting describe pod nginx-one-74dd9d578d-fcpmv
```

```
Name:           nginx-one-74dd9d578d-fcpmv
Namespace:      accounting
Node:           <none>
```

```
<output_omitted>

Events:
  Type     Reason            Age         From      ....
  ----     ------            ----        ----
  Warning  FailedScheduling  <unknown>   default-scheduler
0/2 nodes are available: 2 node(s) didn't match node selector.
```

7. Label the secondary node. Note the value is case sensitive. Verify the labels.

```
student@cp:~$ kubectl label node <worker_node_name> system=secondOne
```

```
node/worker labeled
```

```
student@cp:~$ kubectl get nodes --show-labels
```

```
NAME     STATUS   ROLES                AGE   VERSION   LABELS
cp       Ready    control-plane        15h   v1.27.1   beta.kubernetes.io/arch=amd64,
beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=cp,
kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=,node-role.kubernetes.io/master=,
node.kubernetes.io/exclude-from-external-load-balancers=
worker   Ready    <none>               15h   v1.27.1   beta.kubernetes.io/arch=amd64,
beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=worker,
kubernetes.io/os=linux,system=secondOne
```

8. View the pods in the `accounting` namespace. They may still show as `Pending`. Depending on how long it has been since you attempted deployment the system may not have checked for the label. If the Pods show `Pending` after a minute delete one of the pods. They should both show as `Running` after a deletion. A change in state will cause the `Deployment` controller to check the status of both Pods.

```
student@cp:~$  kubectl -n accounting get pods
```

```
NAME                          READY     STATUS    RESTARTS   AGE
nginx-one-74dd9d578d-fcpmv    1/1       Running   0          10m
nginx-one-74dd9d578d-sts5l    1/1       Running   0          3s
```

9. View Pods by the label we set in the YAML file. If you look back the Pods were given a label of `app=nginx`.

```
student@cp:~$ kubectl get pods -l system=secondary --all-namespaces
```

```
NAMESPACE   NAME                          READY   STATUS    RESTARTS   AGE
accounting  nginx-one-74dd9d578d-fcpmv  1/1     Running   0          20m
accounting  nginx-one-74dd9d578d-sts5l  1/1     Running   0          9m
```

10. Recall that we exposed port 8080 in the YAML file. Expose the new deployment.

```
student@cp:~$ kubectl -n accounting expose deployment nginx-one
```

```
service/nginx-one exposed
```

11. View the newly exposed endpoints. Note that port 8080 has been exposed on each Pod.

```
student@cp:~$ kubectl -n accounting get ep nginx-one
```

```
NAME        ENDPOINTS                           AGE
nginx-one   192.168.1.72:8080,192.168.1.73:8080   47s
```

12. Attempt to access the Pod on port 8080, then on port 80. Even though we exposed port 8080 of the container the application within has not been configured to listen on this port. The **nginx** server listens on port 80 by default. A `curl` command to that port should return the typical welcome page.

```
student@cp:~$ curl 192.168.1.72:8080
```

```
curl: (7) Failed to connect to 192.168.1.72 port 8080: Connection refused
```

```
student@cp:~$ curl 192.168.1.72:80
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```

13. Delete the deployment. Edit the YAML file to expose port 80 and create the deployment again.

```
student@cp:~$ kubectl -n accounting delete deploy nginx-one
```

```
deployment.apps "nginx-one" deleted
```

```
student@cp:~$ vim nginx-one.yaml
```

nginx-one.yaml

```
1 ....
2         ports:
3         - containerPort: 8080     #<-- Edit this line
4           protocol: TCP
5 ....
```

```
student@cp:~$ kubectl create -f nginx-one.yaml
```

```
deployment.apps/nginx-one created
```