Exercise 3.4: Deploy A Simple Application

We will test to see if we can deploy a simple application, in this case the **nginx** web server.

1. Create a new `deployment`, which is a Kubernetes object, which will deploy an application in a container. Verify it is running and the desired number of containers matches the available.

```
student@cp:~$ kubectl create deployment nginx --image=nginx
```

```
deployment.apps/nginx created
```

```
student@cp:~$ kubectl get deployments
```

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/1     1            1           8s
```

2. View the details of the deployment. Remember auto-completion will work for sub-commands and resources as well.

```
student@cp:~$ kubectl describe deployment nginx
```

```
Name:                   nginx
Namespace:              default
CreationTimestamp:      Mon, 23 Apr 2019 22:38:32 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 ava....
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
<output_omitted>
```

3. View the basic steps the cluster took in order to pull and deploy the new application. You should see several lines of output. The first column shows the age of each message, note that due to JSON lack of order the time `LAST SEEN` time does not print out chronologically. Eventually older messages will be removed.

```
student@cp:~$ kubectl get events
```

```
<output_omitted>
```

4. You can also view the output in **yaml** format, which could be used to create this deployment again or new deployments. Get the information but change the output to yaml. Note that halfway down there is status information of the current deployment.

```
student@cp:~$ kubectl get deployment nginx -o yaml
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2017-09-27T18:21:25Z
<output_omitted>
```

5. Run the command again and redirect the output to a file. Then edit the file. Remove the `creationTimestamp`, `resourceVersion`, and `uid` lines. Also remove all the lines including and after `status:`, which should be somewhere around line 120, if others have already been removed.

```
student@cp:~$ kubectl get deployment nginx -o yaml > first.yaml

student@cp:~$ vim first.yaml
```

```
<Remove the lines mentioned above>
```

6. Delete the existing deployment.

```
student@cp:~$ kubectl delete deployment nginx
```

```
deployment.apps "nginx" deleted
```

7. Create the deployment again this time using the file.

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

```
deployment.apps/nginx created
```

8. Look at the yaml output of this iteration and compare it against the first. The creation time stamp, resource version and unique ID we had deleted are in the new file. These are generated for each resource we create, so we may need to delete them from yaml files to avoid conflicts or false information. You may notice some time stamp differences as well. The `status` should not be hard-coded either.

```
student@cp:~$ kubectl get deployment nginx -o yaml > second.yaml

student@cp:~$ diff first.yaml second.yaml
```

```
<output_omitted>
```

9. Now that we have worked with the raw output we will explore two other ways of generating useful YAML or JSON. Use the `--dry-run` option and verify no object was created. Only the prior `nginx` deployment should be found. The output lacks the unique information we removed before, but does have the same essential values.

```
student@cp:~$ kubectl create deployment two --image=nginx --dry-run=client -o yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: two
  name: two
spec:
<output_omitted>
```

```
student@cp:~$ kubectl get deployment
```

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/1     1            1           7m
```

10. Existing objects can be viewed in a ready to use YAML output. Take a look at the existing **nginx** deployment.

```
student@cp:~$ kubectl get deployments nginx -o yaml
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: null
  generation: 1
  labels:
    run: nginx
<output_omitted>
```

11. The output can also be viewed in JSON output.

```
student@cp:~$ kubectl get deployment nginx -o json
```

```json
{
    "apiVersion": "apps/v1",
    "kind": "Deployment",
    "metadata": {
        "annotations": {
            "deployment.kubernetes.io/revision": "1"
        },
<output_omitted>
```

12. The newly deployed **nginx** container is a light weight web server. We will need to create a `service` to view the default welcome page. Begin by looking at the help output. Note that there are several examples given, about halfway through the output.

```
student@cp:~$ kubectl expose -h
```

```
<output_omitted>
```

13. Now try to gain access to the web server. As we have not declared a port to use you will receive an error.

```
student@cp:~$ kubectl expose deployment/nginx
```

```
error: couldn't find port via --port flag or introspection
See 'kubectl expose -h' for help and examples.
```

14. To change an object configuration one can use subcommands `apply`, `edit` or `patch` for non-disruptive updates. The `apply` command does a three-way diff of previous, current, and supplied input to determine modifications to make. Fields not mentioned are unaffected. The `edit` function performs a `get`, opens an editor, then an `apply`. You can update API objects in place with JSON patch and merge patch or strategic merge patch functionality.

    If the configuration has resource fields which cannot be updated once initialized then a disruptive update could be done using the `replace --force` option. This deletes first then re-creates a resource.

    Edit the file. Find the container name, somewhere around line 31 and add the port information as shown below.

```
student@cp:~$ vim first.yaml
```

first.yaml

```yaml
....
   spec:
      containers:
```

```
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        ports:                                  # Add these
        - containerPort: 80                     # three
          protocol: TCP                         # lines
        resources: {}
....
```

15. Due to how the object was created we will need to use `replace` to terminate and create a new deployment.

```
student@cp:~$ kubectl replace -f first.yaml
```

```
deployment.apps/nginx replaced
```

16. View the Pod and Deployment. Note the `AGE` shows the Pod was re-created.

```
student@cp:~$ kubectl get deploy,pod
```

```
NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx   1/1     1            1           2m4s

NAME                           READY   STATUS    RESTARTS   AGE
pod/nginx-7db75b8b78-qjffm   1/1     Running   0          8s
```

17. Try to expose the resource again. This time it should work.

```
student@cp:~$ kubectl expose deployment/nginx
```

```
service/nginx exposed
```

18. Verify the service configuration. First look at the service, then the endpoint information. Note the `ClusterIP` is not the current endpoint. Calico provides the `ClusterIP`. The `Endpoint` is provided by `kubelet` and `kube-proxy`. Take note of the current endpoint IP. In the example below it is `192.168.1.5:80`. We will use this information in a few steps.

```
student@cp:~$ kubectl get svc nginx
```

```
NAME      TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
nginx     ClusterIP    10.100.61.122   <none>        80/TCP    3m
```

```
student@cp:~$ kubectl get ep nginx
```

```
NAME    ENDPOINTS        AGE
nginx   192.168.1.5:80   26s
```

19. Determine which node the container is running on. Log into that node and use **tcpdump**, which you may need to install using **apt-get install**, to view traffic on the `tunl0`, as in tunnel zero, interface. The second node in this example. You may also see traffic on an interface which starts with `cali` and some string. Leave that command running while you run **curl** in the following step. You should see several messages go back and forth, including a `HTTP` HTTP/1.1 200 OK: and a `ack` response to the same sequence.

```
student@cp:~$ kubectl describe pod nginx-7cbc4b4d9c-d27xw \
                | grep Node:
```

```
Node:   worker/10.128.0.5
```

```
student@worker:~$  sudo tcpdump -i tunl0
```

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol...
listening on tunl0, link-type EN10MB (Ethernet), capture size...
<output_omitted>
```

20. Test access to the Cluster IP, port 80. You should see the generic `nginx` installed and working page. The output should be the same when you look at the `ENDPOINTS` IP address. If the **curl** command times out the pod may be running on the other node. Run the same command on that node and it should work.

```
student@cp:~$ curl 10.100.61.122:80
```

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

```
student@cp:~$ curl 192.168.1.5:80
```

21. Now scale up the deployment from one to three web servers.

```
student@cp:~$ kubectl get deployment nginx
```

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/1     1            1           12m
```

```
student@cp:~$ kubectl scale deployment nginx --replicas=3
```

```
deployment.apps/nginx scaled
```

```
student@cp:~$ kubectl get deployment nginx
```

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   3/3     3            3           12m
```

22. View the current endpoints. There now should be three. If the `UP-TO-DATE` above said three, but `AVAILABLE` said two wait a few seconds and try again, it could be slow to fully deploy.

```
student@cp:~$ kubectl get ep nginx
```

```
NAME    ENDPOINTS                                      AGE
nginx   192.168.0.3:80,192.168.1.5:80,192.168.1.6:80   7m40s
```

23. Find the oldest pod of the **nginx** deployment and delete it. The `Tab` key can be helpful for the long names. Use the `AGE` field to determine which was running the longest. You may notice activity in the other terminal where **tcpdump** is running, when you delete the pod. The pods with `192.168.0` addresses are probably on the cp and the `192.168.1` addresses are probably on the worker

```
student@cp:~$ kubectl get pod -o wide
```

```
NAME                     READY     STATUS    RESTARTS   AGE   IP
nginx-1423793266-7f1qw   1/1       Running   0          14m   192.168.1.5
nginx-1423793266-8w2nk   1/1       Running   0          86s   192.168.1.6
nginx-1423793266-fbt4b   1/1       Running   0          86s   192.168.0.3
```

```
student@cp:~$ kubectl delete pod nginx-1423793266-7f1qw
```

```
pod "nginx-1423793266-7f1qw" deleted
```

24. Wait a minute or two then view the pods again. One should be newer than the others. In the following example nine seconds instead of four minutes. If your **tcpdump** was using the `veth` interface of that container it will error out. Also note we are using a short name for the object.

```
student@cp:~$  kubectl get po
```

```
NAME                     READY     STATUS    RESTARTS   AGE
nginx-1423793266-13p69   1/1       Running   0          9s
nginx-1423793266-8w2nk   1/1       Running   0          4m1s
nginx-1423793266-fbt4b   1/1       Running   0          4m1s
```

25. View the endpoints again. The original endpoint IP is no longer in use. You can delete any of the pods and the `service` will forward traffic to the existing backend pods.

```
student@cp:~$ kubectl get ep nginx
```

```
NAME    ENDPOINTS                                            AGE
nginx   192.168.0.3:80,192.168.1.6:80,192.168.1.7:80   12m
```

26. Test access to the web server again, using the `ClusterIP` address, then any of the endpoint IP addresses. Even though the endpoints have changed you still have access to the web server. This access is only from within the cluster. When done use **ctrl-c** to stop the **tcpdump** command.

```
student@cp:~$ curl 10.100.61.122:80
```

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