Exercise 12.2: Using Taints to Control Pod Deployment

Use taints to manage where Pods are deployed or allowed to run. In addition to assigning a Pod to a group of nodes, you may also want to limit usage on a node or fully evacuate Pods. Using taints is one way to achieve this. You may remember that the cp node begins with a `NoSchedule` taint. We will work with three taints to limit or remove running pods.

1. Create a deployment which will deploy eight **nginx** containers. Begin by creating a `YAML` file.

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

taint.yaml

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: taint-deployment
5  spec:
6    replicas: 8
7    selector:
8      matchLabels:
9        app: nginx
10   template:
11     metadata:
12       labels:
13         app: nginx
14     spec:
15       containers:
16       - name:  nginx
17         image: nginx:1.20.1
18         ports:
19         - containerPort: 80
```

2. Apply the file to create the deployment.

```
student@cp:~$ kubectl apply -f taint.yaml
```

```
1  deployment.apps/taint-deployment created
```

3. Determine where the containers are running. In the following example three have been deployed on the cp node and five on the secondary node. Remember there will be other housekeeping containers created as well. Your numbers may be different, the actual number is not important, we are tracking the change in numbers.

```
student@cp:~$ sudo docker ps |grep nginx   #<-- For Docker systems
```

```
student@cp:~$ sudo crictl ps |grep nginx   #<-- For crictl systems
```

```
1  00c1be5df1e7        nginx@sha256:e3456c851a152494c3e.....
2  <output_omitted>
```

```
student@cp:~$ sudo docker ps |wc -l
```

```
1  27
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
17
```

4. Delete the deployment. Verify the containers are gone.

```
student@cp:~$ kubectl delete deployment taint-deployment
```

```
deployment.apps "taint-deployment" deleted
```

```
student@cp:~$ sudo docker ps |wc -l
```

```
21
```

5. Now we will use a taint to affect the deployment of new containers. There are three taints, `NoSchedule`, `PreferNoSchedule` and `NoExecute`. The taints having to do with schedules will be used to determine newly deployed containers, but will not affect running containers. The use of `NoExecute` will cause running containers to move.

   Taint the secondary node, verify it has the taint then create the deployment again. We will use the key of `bubba` to illustrate the key name is just some string an admin can use to track Pods.

```
student@cp:~$ kubectl taint nodes worker \
          bubba=value:PreferNoSchedule
```

```
node/worker tainted
```

```
student@cp:~$ kubectl describe node |grep Taint
```

```
Taints:                     bubba=value:PreferNoSchedule
Taints:                     <none>
```

```
student@cp:~$ kubectl apply -f taint.yaml
```

```
deployment.apps/taint-deployment created
```

6. Locate where the containers are running. We can see that more containers are on the cp, but there still were some created on the secondary. Delete the deployment when you have gathered the numbers.

```
student@cp:~$ sudo docker ps |wc -l
```

```
21
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
23
```

```
student@cp:~$ kubectl delete deployment taint-deployment
```

```
deployment.apps "taint-deployment" deleted
```

7. Remove the taint, verify it has been removed. Note that the key is used with a minus sign appended to the end.

```
student@cp:~$ kubectl taint nodes worker bubba-
```

```
node/worker untainted
```

```
student@cp:~$ kubectl describe node |grep Taint
```

```
Taints:                         <none>
Taints:                         <none>
```

8. This time use the `NoSchedule` taint, then create the deployment again. The secondary node should not have any new containers, with only daemonsets and other essential pods running.

```
student@cp:~$ kubectl taint nodes worker \
          bubba=value:NoSchedule
```

```
node/worker tainted
```

```
student@cp:~$ kubectl apply -f taint.yaml
```

```
deployment.apps/taint-deployment created
```

```
student@cp:~$ sudo docker ps |wc -l
```

```
21
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
23
```

9. Remove the taint and delete the deployment. When you have determined that all the containers are terminated create the deployment again. Without any taint the containers should be spread across both nodes.

```
student@cp:~$ kubectl delete deployment taint-deployment
```

```
deployment.apps "taint-deployment" deleted
```

```
student@cp:~$ kubectl taint nodes worker bubba-
```

```
node/worker untainted
```

```
student@cp:~$ kubectl apply -f taint.yaml
```

```
deployment.apps/taint-deployment created
```

```
student@cp:~$ sudo docker ps |wc -l
```

```
27
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
17
```

10. Now use the `NoExecute` to taint the secondary (**worker**) node. Wait a minute then determine if the containers have moved. The DNS containers can take a while to shutdown. Some containers will remain on the worker node to continue communication from the cluster.

```
student@cp:~$ kubectl taint nodes worker \
         bubba=value:NoExecute
```

```
node "worker" tainted
```

```
student@cp:~$ sudo docker ps |wc -l
```

```
37
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
5
```

11. Remove the taint. Wait a minute. Note that all of the containers did not return to their previous placement.

```
student@cp:~$ kubectl taint nodes worker bubba-
```

```
node/worker untainted
```

```
student@cp:~$ sudo docker ps |wc -l
```

```
32
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
6
```

12. Remove the deployment a final time to free up resources.

```
student@cp:~$ kubectl delete deployment taint-deployment
```

```
deployment.apps "taint-deployment" deleted
```