Taints and Tolerations

In a previous lab, you created "infra" nodes leveraging nodeSelector to control where the workload lands. You saw how you can set nodes to perform certain tasks. The `nodeSelector` in the `nodePlacement` attracts pods to a set of nodes. But what if you want the opposite? What if you want to repel pods?

Having a taint in place will make sure a pod does NOT run on the tainted node.

Some usecases include:

- A node (or a group of nodes) in the cluster is reserved for special purposes
- Nodes having specialized hardware for specific tasks (like a GPU)
- Some nodes may be licensed for some software running on it
- Nodes can be in different network zone for compliance reasons
- You may want to troubleshoot a misbehaving node

In this lab we'll explore how to use taints and tolerations to control where workloads land.

Background

Taints and tolerations work in tandem to ensure that workloads are not scheduled onto certian nodes. Once a node is tainted, that node should not accept any workload that does not tolerate the taint(s). Tolerations that are applied to these workloads will allow (but do not require) the corresponding pods to schedule onto nodes with matching taints.

> You will have to use `nodeSelector` along with taints and tolerations to have workloads ALWAYS land on a specific node.

Examine node labels

In order to examine the nodes, make sure you're running as `kubeadmin`

```
oc login -u kubeadmin -p {{ KUBEADMIN_PASSWORD }}
```

List the current worker nodes, ignoring infra nodes and storage nodes.

```
oc get nodes -l 'node-role.kubernetes.io/worker,!node-role.kubernetes.io/infra,!cluster.ocs.openshift.io/openshift-storage'
```

Depending on which labs you've done, you may have 3 more more nodes with the worker label set.

Now, let's look to see if there are any taints added to these workers.

```
oc describe nodes -l 'node-role.kubernetes.io/worker,!node-role.kubernetes.io/infra,!cluster.ocs.openshift.io/openshift-storage'
| grep Taint
```

There shouldn't be any taints assinged to these workers.

Deploying test application

For this exercise we will be deploying an app to test with.

Before you deploy this app, you'll need to label your nodes for the duration of this exercise. The application deploys on nodes labeled `appnode=welcome-php`.

```
oc get nodes -l 'node-role.kubernetes.io/worker,!node-role.kubernetes.io/infra,!cluster.ocs.openshift.io/openshift-storage' -o
jsonpath='{range .items[*]}{.metadata.name}{"\n"}' | xargs -I{} oc label node {} appnode=welcome-php
```

> To label a node, you pass the node name with a `key=value` pair. Example: `oc label node worker.example.com appnode=welcome-php`

Verify that the nodes were labeled, taking note of the name of the node(s).

```
oc get nodes -l appnode=welcome-php
```

You can now deploy the application using the YAML manifest that is provided in the home directory.

```
oc create -f ~/support/welcome-php.yaml
```

This creates the following objects

- The namespace `tt-lab`
- A deployment called `welcome-php` that's running a sample php app
- A service called `welcome-php` that listens on port `8080`
- A route called `welcome-php`

We will be working on the `tt-lab` project, so switch to it now.

```
oc project tt-lab
```

Now scale the pods to equal to the number of nodes we have. The default scheduler will attempt to evenly distribute the app. Also we'll wait for the rollout to complete.

```
NODE_NUMBER=$(oc get nodes --no-headers -l appnode=welcome-php | wc -l)
oc scale deploy welcome-php --replicas=${NODE_NUMBER}
oc rollout status deploy welcome-php
```

> If the pods are not evenly distributed between nodes, you may have to run `oc delete pods --all` to let them rebalance.

Verify that the pods are spread evenly accross workers. There should be one pod for each node.

```
clear
oc get pods -o=custom-columns=NAME:.metadata.name,NODE:.spec.nodeName
oc get nodes -l appnode=welcome-php
```

Tainting nodes

We will now taint a node, so it'll reject workloads. First, we will go over what each taint does. There are 3 basic taints that you can set.

- `key=value:NoSchedule`
- `key=value:NoExecute`

- `key=value:PreferNoSchedule`

The `key` in this case is any string, up to 253 characters. The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.

The `value` here is any string, up to 63 characters. The value must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.

The 3 "effects" do the following:

- `NoSchedule` - New pods that do not match the taint are not scheduled onto that node. Existing pods on the node remain.
- `NoExecute` - New pods that do not match the taint cannot be scheduled onto that node. Existing pods on the node that do not have a matching toleration are removed.
- `PreferNoSchedule` - New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. Existing pods on the node remain.

There is another component called the `operator`. We’ll go over the `operator` in detail in the "tolerations" section.

For this lab we will taint the first node that’s not an infra or storage node with`welcome-php=run:NoSchedule`. This makes it to where all new pods (without a proper toleration) NOT schedule on this node.

```
TTNODE=$(oc get nodes -l 'node-role.kubernetes.io/worker,!node-role.kubernetes.io/infra,!cluster.ocs.openshift.io/openshift-storage' -o jsonpath='{range .items[0]}{.metadata.name}')
oc adm taint node ${TTNODE} welcome-php=run:NoSchedule
```

The syntax is: `oc adm taint node ${nodename} key=value:Effect`

Examine the nodes we deployed on and see that the taint is applied to one of the nodes.

```
oc describe nodes -l appnode=welcome-php | grep Taint
```

We used `NoSchedule` for the effect, so a pod should still be there. Remember that `NoSchedule` only stops new pods from spawning on the node (the command should return a `1`)

```
oc get pods -o wide | grep -c ${TTNODE}
```

Let’s delete the pods and wait for the `replicaSet` to redeploy them.

```
oc delete pods --all
oc rollout status deploy welcome-php
```

Since our deployment doesn’t have a toleration, the scheduler will deploy the pods on all nodes except the one with a taint. This command should return a `0`

```
oc get pods -o wide | grep -c ${TTNODE}
```

Examine where the pods are running.

```
clear
oc get pods -o=custom-columns=NAME:.metadata.name,NODE:.spec.nodeName
oc get nodes -l appnode=welcome-php
```

Tolerations

A `toleration` is a way for pods to "tolerate" (or "ignore") a node's taint during scheduling. Tolerations are applied in the `podSpec`, and is in the following form.

```
tolerations:
- key: "welcome-php"
  operator: "Equal"
  value: "run"
  effect: "NoSchedule"
```

If the toleration "matches" then the scheduler will schedule the workload on this node (if need be…remember, it's not a guarantee). Note that you have to match the `key`, `value`, and `effect`. There is also something called an `operator`.

The `operator` can be set to `Equal` or `Exists`, depending on the fuction you want.

- `Equal` - The `key`, `value`, and `effect` parameters must match. This is the default setting if nothing is provided.
- `Exists` - The `key` and the `effect` parameters must match. You **must** leave a blank value parameter, which matches any.

We'll apply this toleration in the `spec.template.spec` section of the deployment.

```
oc patch deployment welcome-php --patch '{"spec":{"template":{"spec":{"tolerations":[{"key":"welcome-
php","operator":"Equal","value":"run","effect":"NoSchedule"}]}}}}'
```

Patching triggers another deployment so we'll wait for it to finish rolling out.

```
oc rollout status deploy welcome-php
```

You can see the toleration config by inspecting the deployment YAML

```
oc get deploy welcome-php -o yaml
```

Now, since we have the toleration in place, we should be running on the node with the taint (this should return`1`)

```
oc get pods -o wide | grep -c ${TTNODE}
```

Now when you list all pods, they should be now spread evenly.

```
clear
oc get pods -o=custom-columns=NAME:.metadata.name,NODE:.spec.nodeName
oc get nodes -l appnode=welcome-php
```

To read more about taints and tolerations, you can take a look at the Official Documentation.

Clean Up

Make sure you login as `kubeadmin` for the next lab.

```
oc login -u kubeadmin -p {{ KUBEADMIN_PASSWORD }}
```

Other labs may be affected by taints, so let's undo what we did:

```
oc delete project tt-lab
oc adm taint node ${TTNODE} welcome-php-
oc get nodes -l 'node-role.kubernetes.io/worker,!node-role.kubernetes.io/infra,!cluster.ocs.openshift.io/openshift-storage' -o
jsonpath='{range .items[*]}{.metadata.name}{"\n"}' | xargs -I{} oc label node {} appnode-
```

Make sure the nodes have that taint removed

```
oc describe nodes -l 'node-role.kubernetes.io/worker,!node-role.kubernetes.io/infra,!cluster.ocs.openshift.io/openshift-storage'
| grep Taint
```

Also, verify that the label does not exist on the nodes we were working on. This command shouldn’t return any nodes.

```
oc get nodes -l appnode=welcome-php
```