Taints and Tolerations

In this lecture, we will discuss about the pod-to-node relationship and how you can restrict what pods are placed on what nodes. The concept of taints and tolerations can be a bit confusing for beginners. So we will try to understand what they are using an analogy of a bug approaching a person. Now, my apologies in advance, but this is the best I could come up with.

To prevent the bug from landing on the person, we spray the person with a repellent spray or a taint as we will call it in this lecture. The bug is intolerant to the smell so on approaching the person, the taint applied on the person throws the bug off.

Intolerant

Taint

However, there could be other bugs that are tolerant to this smell, and so the taint doesn't really affect them and so they end up landing on the person.

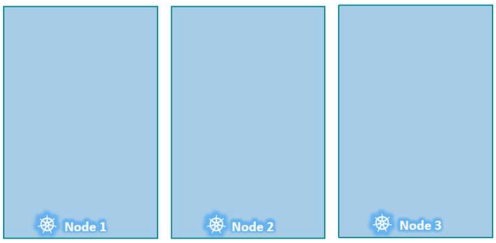
Tolerant

Taint

So there are two things that decide if a bug can land on a person. First, the taint on the person, and second, the bug's toleration level to that particular taint.

Getting back to Kubernetes, the person is a node and the bugs are pods. Now, taints and tolerations have nothing to do with security or intrusion on the cluster. Taints and tolerations are used to set restrictions on what pods can be scheduled on a node.

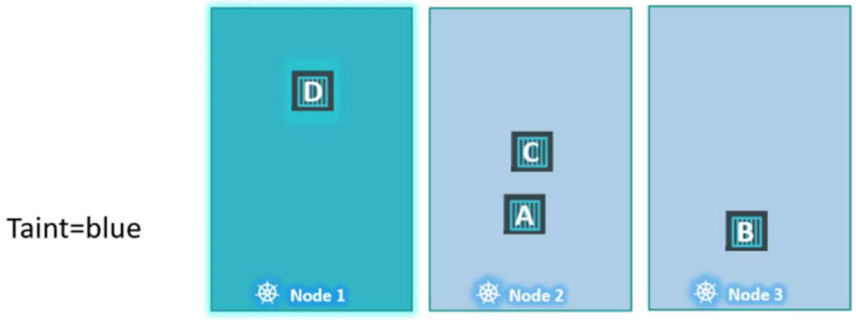
Let us start with a simple cluster with three worker nodes. The nodes are named one, two, and three. We also have a set of pods that are to be deployed on these nodes. Let's call them A, B, C, and D. When the pods are created, Kubernetes scheduler tries to place these pods on the available worker nodes. As of now, there are no restrictions or limitations and as such, the scheduler places the pods across all of the nodes to balance them out equally.



Now, let us assume that we have dedicated resources on node one for a particular use case or application. So we would like only those pods that belong to this application to be placed on node one. First, we prevent all pods from being placed on the node by placing a taint on the node. Let's call it blue. By default, pods have no tolerations, which means unless specified otherwise, none of the pods can tolerate any taint. So in this case, none of the pods can be placed on node one, as none of them can tolerate the taint blue. This solves half of our requirement. No unwanted pods are going to be placed on this node.

The other half is to enable certain pods to be placed on this node. For this we must specify which pods are tolerant to this particular taint. In our case, we would like to allow only pod D to be placed on this node. So we add a toleration to pod D. Pod D is now tolerant to blue. So when the scheduler tries to place this pod on node one, it goes through. Node one can now only accept pods that can tolerate the taint blue. So with all the taints and tolerations in place, this is how the pods would be scheduled.

The scheduler tries to place pod A on node one, but due to the taint it is thrown off and it goes to node two. The scheduler then tries to place pod B on node one, but again, due to the taint, it is thrown off and is placed on node three, which happens to be the next free node. The scheduler then tries to place pod C to the node one. It is thrown off again and ends up on node two. And finally the scheduler tries to place pod D on node one. Since the pod is tolerant to node one, it goes through.



So remember, taints are set on nodes and tolerations are set on pods.

So how do you do this? Use the kubectl taint nodes command to taint a node specify the name of the node to taint followed by the taint itself, which is a key value pair. For example, if you would like to dedicate the node to pods in application blue, then the key value pair would be app=blue. The taint effect defines what would happen to the pods if they do not tolerate the taint. There are three taint effects, no

schedule, which means the pods will not be scheduled on the node, which is what we have been discussing. Prefer no schedule, which means the system will try to avoid placing a pod on the node, but that is not guaranteed. And third is no execute, which means that new pods will not be scheduled on the node and existing pods on the node, if any, will be evicted if they do not tolerate the taint. These pods may have been scheduled on the node before the taint was applied to the node.

An example command would be to taint node one with the key value pair app=blue and an effect of no schedule.

```
kubectl taint nodes node-name key=value:taint-effect
```

What happens to PODs that do not tolerate this taint?

NoSchedule | PreferNoSchedule | NoExecute

```
kubectl taint nodes node1 app=myapp:NoSchedule
```

Tolerations are added to pods. To add a toleration to a pod, first pull up the pod definition file. In the spec section of the pod definition file, add a section called, 'tolerations', move the same values used while creating the taint. Under this section, the key is app operator is equal value is blue, and the effect is no schedule. And remember, all of these values need to be encoded in double codes. When the pods are now created or updated with the new tolerations, they are either not scheduled on nodes or evicted from the existing nodes depending on the effect set.

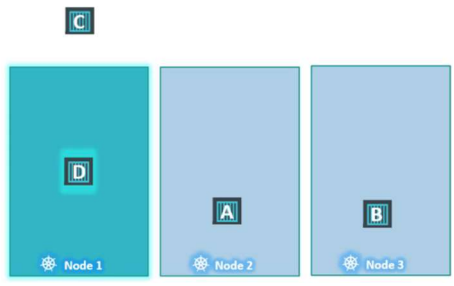
```
kubectl taint nodes node1          :
```

pod-definition.yml

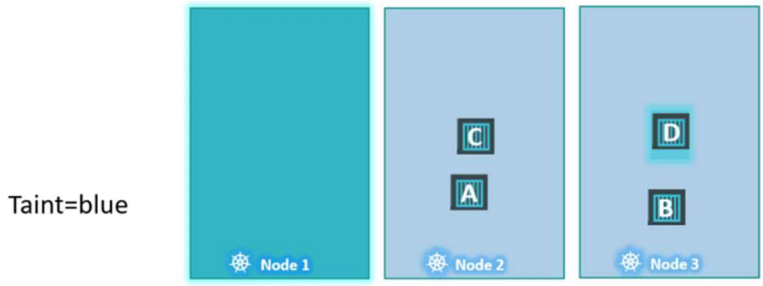
```yaml
apiVersion:
kind: Pod
metadata:
 name: myapp-pod
spec:
  containers:
  - name: nginx-container
    image: nginx

  tolerations:
  - key:"app"
    operator:"Equal"
    value:" blue"
    effect:" NoSchedule"
```

Let us try to understand the no execute taint effect in a bit more depth. In this example, we have three nodes running some workload. We do not have any taints or tolerations at this point, so they're scheduled this way. We then decided to dedicate node one for a special application, and as such, we taint the node with the application name and add a toleration to the pod that belongs to the application, which happens to be pod D in this case. While tainting the node, we set the taint effect to no execute, and as such, once the taint on the node takes effect, it evicts pod C from the node, which simply means that the pod is killed. The pod D continues to run on the node as it has a toleration to the taint blue.



Now, going back to our original scenario where we have taints and tolerations configured. **Remember taints and tolerations are only meant to restrict nodes from accepting certain pods.** In this case, node one can only accept pod D, but it does not guarantee that pod D will always be placed on node one. Since there are no taints or restrictions applied on the other two nodes, pod D may very well be placed on any of the other two nodes.



So remember, taints and tolerations does not tell the pod to go to a particular node. Instead, it tells the node to only accept pods with certain tolerations. If your requirement is to restrict a pod to certain nodes, it is achieved through another concept called as node affinity, which we will discuss in the next lecture.

Finally, while we are on this topic, let us also take a look at an interesting fact. So far we have only been referring to the worker nodes. But we also have master nodes in the cluster, which is technically just another node that has all the capabilities of hosting a pod, plus it runs all the management software.

Now, I'm not sure if you noticed the scheduler does not schedule any pods on the master node. Why is that? When the Kubernetes cluster is first set up, a taint is set on the master node automatically that prevents any pods from being scheduled on this node. You can see this, as well as modify this behavior, if required. However, a best practice is to not deploy application workloads on a master server.

To see this taint, run a kubectl describe node command with kube master as the node name and look for the taint section. You will see a taint set to not schedule any pods on the master node.

```
kubectl describe node kubemaster | grep Taint
Taints:             node-role.kubernetes.io/master:NoSchedule
```