Taint and Tolerations

Introduction:

Taints are opposite of NodeAffinity as Taints repel the pods to get deployed on it. Taints are applied on nodes.

Tolerations are applied to pods and it allow the scheduler to schedule pods with matching taints. Tolerations allow scheduling but don't guarantee scheduling: the scheduler also evaluates other parameters as part of its function.

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes.

Objectives:

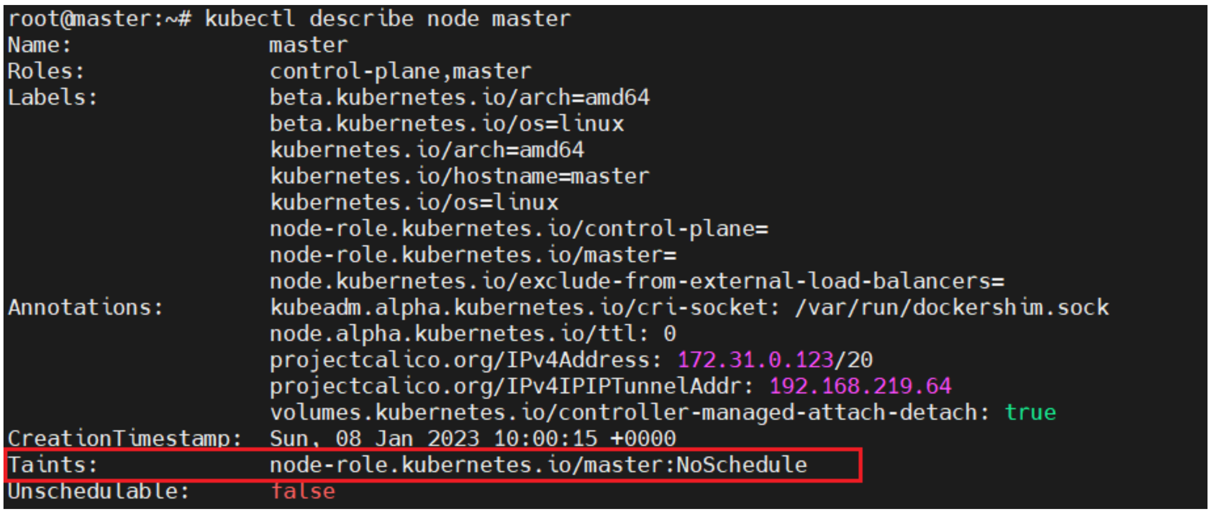
1. Apply Taint on a node
2. Apply Toleration on a Pod
3. Remove the taint

1. Apply Taint on a node:

Let's find out the taints applied on the nodes by default. Use the describe command to check this information.

Master node:

```
kubectl describe node master
```

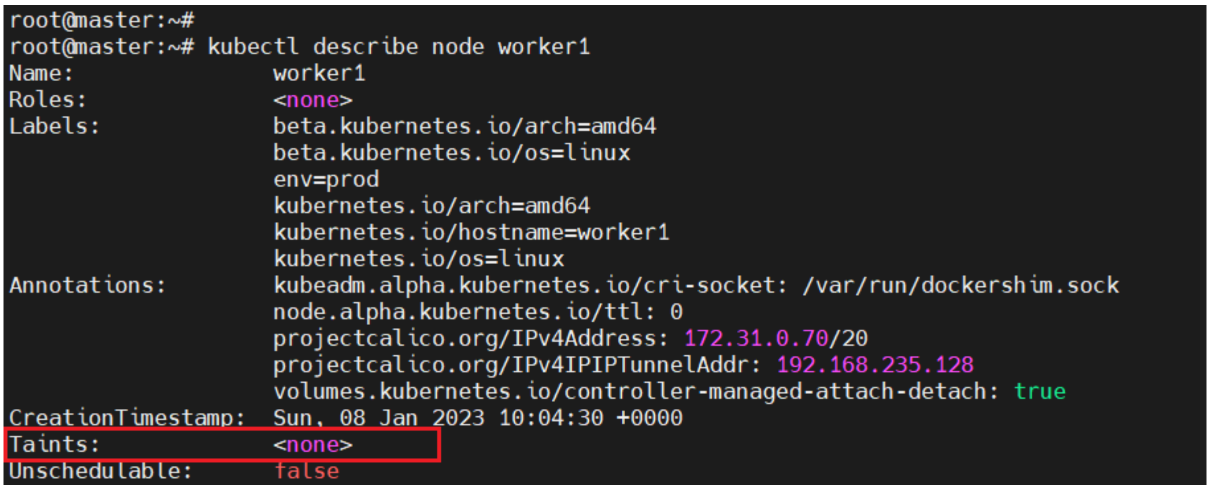


```
root@master:~# kubectl describe node master
Name:               master
Roles:              control-plane,master
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=master
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/control-plane=
                    node-role.kubernetes.io/master=
                    node.kubernetes.io/exclude-from-external-load-balancers=
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    projectcalico.org/IPv4Address: 172.31.0.123/20
                    projectcalico.org/IPv4IPIPTunnelAddr: 192.168.219.64
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Sun, 08 Jan 2023 10:00:15 +0000
Taints:             node-role.kubernetes.io/master:NoSchedule
Unschedulable:      false
```

Above we can see the master node has the taints configured which is **node-role.kubernetes.io/master:NoSchedule**.
So, the pods which can tolerate this taint , can be deployed on the master node. By default, the pods do not have any tolerations configured that's why any new pod is not deployed on master node.

Worker1 node:

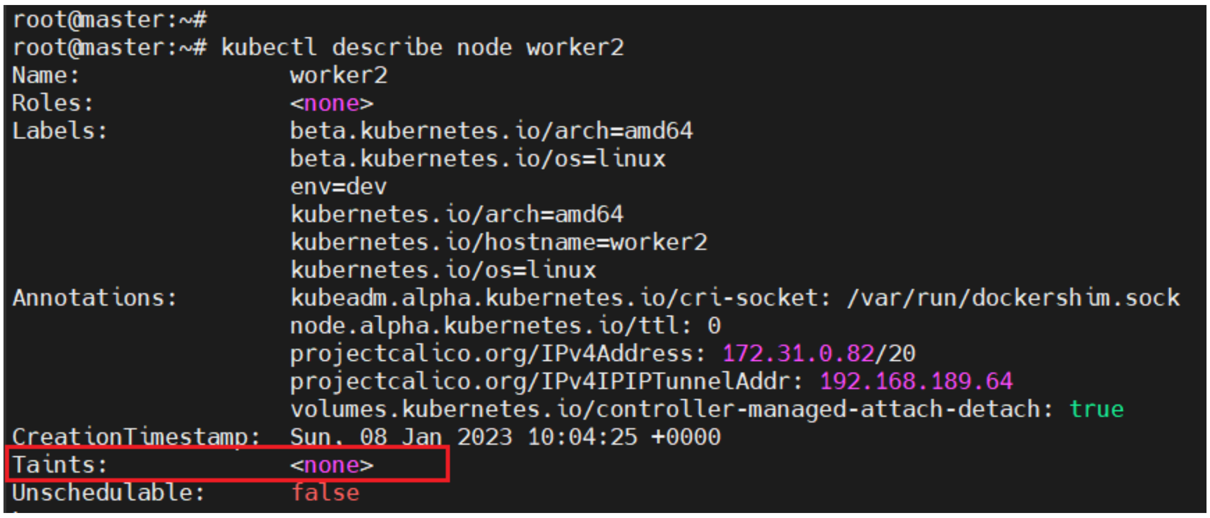
```
kubectl describe node worker1
```



```
root@master:~#
root@master:~# kubectl describe node worker1
Name:               worker1
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    env=prod
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker1
                    kubernetes.io/os=linux
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    projectcalico.org/IPv4Address: 172.31.0.70/20
                    projectcalico.org/IPv4IPIPTunnelAddr: 192.168.235.128
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Sun, 08 Jan 2023 10:04:30 +0000
Taints:             <none>
Unschedulable:      false
```

Worker2 node:

```
kubectl describe node worker2
```



```
root@master:~#
root@master:~# kubectl describe node worker2
Name:               worker2
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    env=dev
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker2
                    kubernetes.io/os=linux
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    projectcalico.org/IPv4Address: 172.31.0.82/20
                    projectcalico.org/IPv4IPIPTunnelAddr: 192.168.189.64
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Sun, 08 Jan 2023 10:04:25 +0000
Taints:             <none>
Unschedulable:      false
```

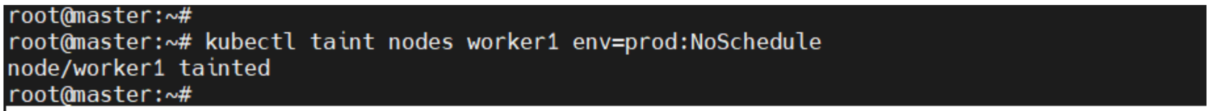
In above output, worker1 and worker2 nodes do not have any taints so new pods get deployed on these nodes only.

Let's apply the taint on one of the nodes using the below command.

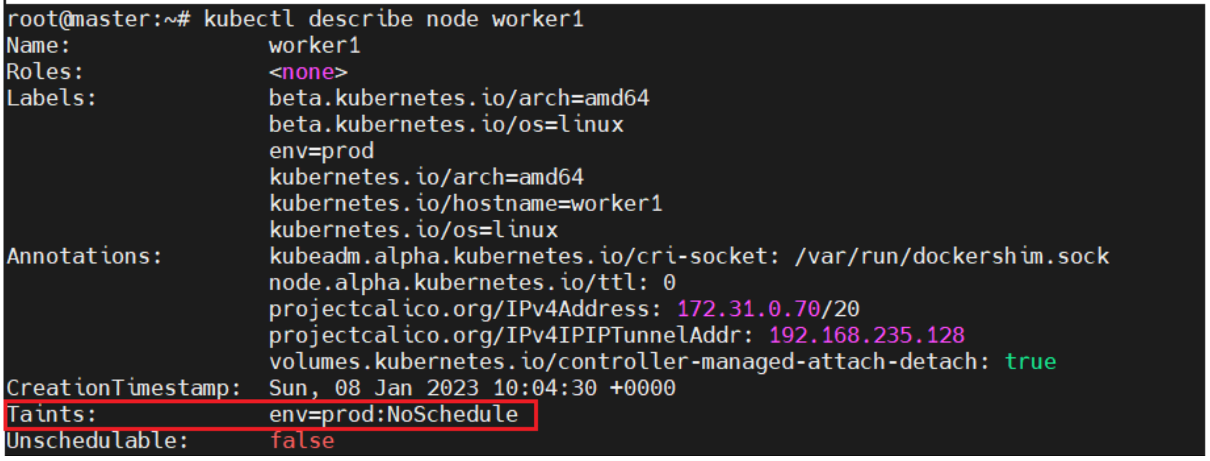
```
kubectl taint nodes worker3 env=prod:NoSchedule
```

Using above command, we have tainted **worker1** node with **env=prod:NoSchedule**. **env=prod** is the **key=value** whereas **NoSchedule** is an effect. There are three types of effect we can use which are:

NoSchedule: Kubernetes will not schedule the pod onto that node.
PreferNoSchedule: Kubernetes will *try* to not schedule the pod onto the node.
NoExecute: then the pod will be evicted from the node (if it is already running on the node), and will not be scheduled onto the node (if it is not yet running on the node).



```
root@master:~#
root@master:~# kubectl taint nodes worker1 env=prod:NoSchedule
node/worker1 tainted
root@master:~#
```



```
root@master:~# kubectl describe node worker1
Name:               worker1
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    env=prod
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker1
                    kubernetes.io/os=linux
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    projectcalico.org/IPv4Address: 172.31.0.70/20
                    projectcalico.org/IPv4IPIPTunnelAddr: 192.168.235.128
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Sun, 08 Jan 2023 10:04:30 +0000
Taints:             env=prod:NoSchedule
Unschedulable:      false
```

Above output shows that the taint has been applied on the worker1 node.

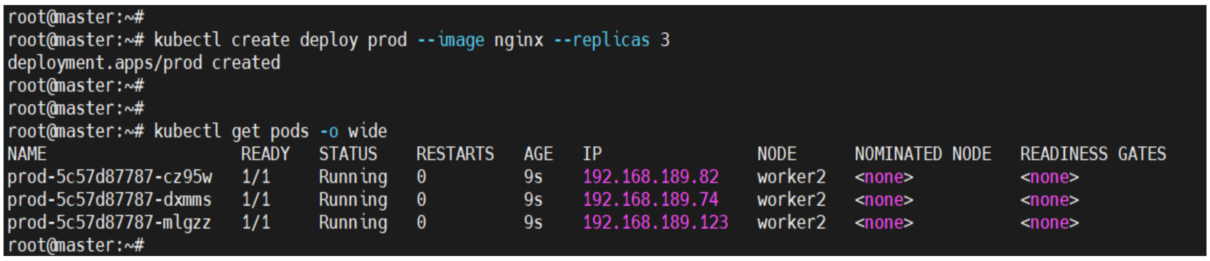
Now if we create any pod then it will not be deployed on worker1 unless the pod has the toleration for that taint.

2. Apply Toleration on a Pod:

Let's create a deployment with toleration and we will check where they have been deployed.

```
kubectl create deploy prod --image nginx --replicas 3
```

You will that the pods will be deployed only on **worker2** node as **worker2** does not have any taint configured. Since the pods do not have any tolerations set so they will not be deployed on **worker1** node.



```
root@master:~#
root@master:~# kubectl create deploy prod --image nginx --replicas 3
deployment.apps/prod created
root@master:~#
root@master:~#
root@master:~# kubectl get pods -o wide
NAME                    READY   STATUS    RESTARTS   AGE   IP                NODE      NOMINATED NODE   READINESS GATES
prod-5c57d87787-cz95w   1/1     Running   0          9s    192.168.189.82    worker2   <none>           <none>
prod-5c57d87787-dxmms   1/1     Running   0          9s    192.168.189.74    worker2   <none>           <none>
prod-5c57d87787-mlgzz   1/1     Running   0          9s    192.168.189.123   worker2   <none>           <none>
root@master:~#
```

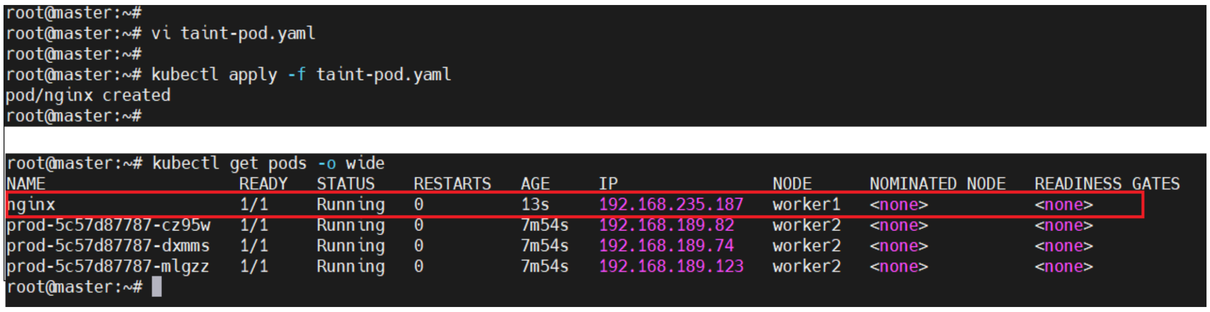
Let’s create a Pod definition file and apply the tolerations. Use the below Yaml file.

```
vi taint-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
  tolerations:
  - key: "env"
    operator: "Equal"
    value: "prod"
    effect: "NoSchedule"
```

Apply the above definition file using below command.

```
kubectl apply -f taint-pod.yaml
```



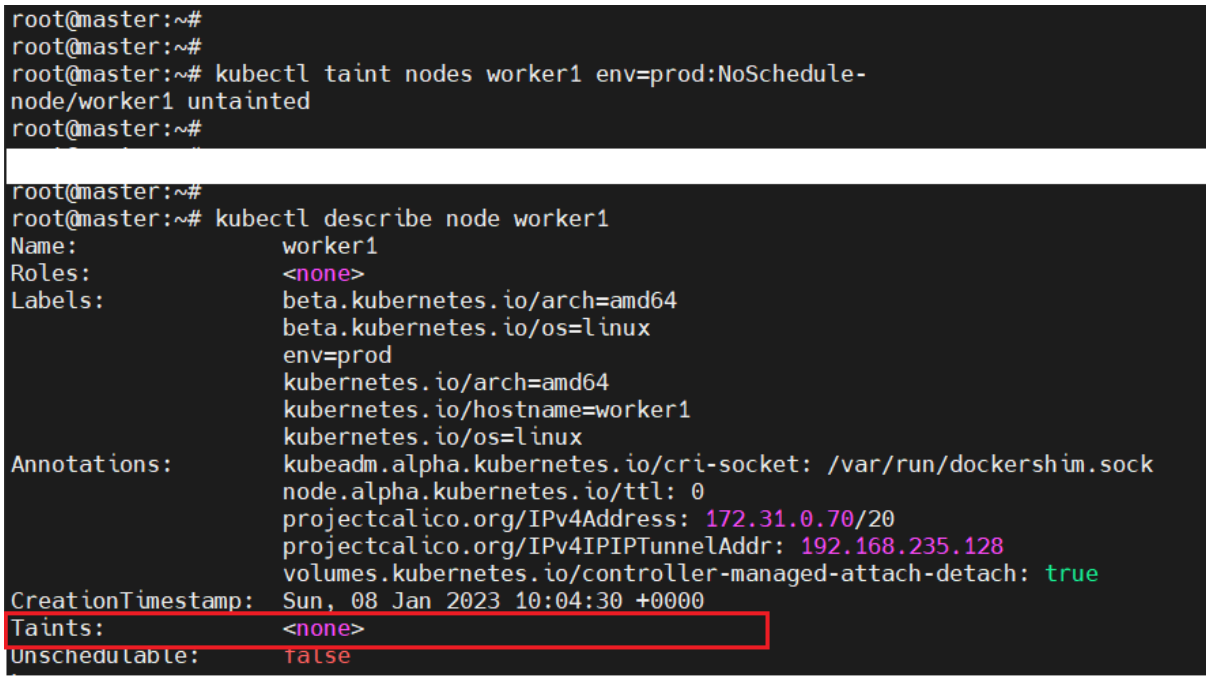
```
root@master:~#
root@master:~# vi taint-pod.yaml
root@master:~#
root@master:~# kubectl apply -f taint-pod.yaml
pod/nginx created
root@master:~#

root@master:~# kubectl get pods -o wide
NAME                    READY   STATUS    RESTARTS   AGE     IP                NODE      NOMINATED NODE   READINESS GATES
nginx                   1/1     Running   0          13s     192.168.235.187   worker1   <none>           <none>
prod-5c57d87787-cz95w   1/1     Running   0          7m54s   192.168.189.82    worker2   <none>           <none>
prod-5c57d87787-dxmms   1/1     Running   0          7m54s   192.168.189.74    worker2   <none>           <none>
prod-5c57d87787-mlgzz   1/1     Running   0          7m54s   192.168.189.123   worker2   <none>           <none>
root@master:~#
```

Once we applied the toleration on our pod, it can be deployed on both the nodes. Pod has the tolerations set so it can easily be deployed on **worker1** node whereas **worker2** has no taint so pod can be deployed on it as well. We can see the nginx pod has been deployed on **worker1** as the pod has the toleration set.

3. Remove the taint

We can use below command to remove the taint on worker1 node.

```
kubectl taint nodes worker1 env=prod:NoSchedule-
```



```
root@master:~#
root@master:~#
root@master:~# kubectl taint nodes worker1 env=prod:NoSchedule-
node/worker1 untainted
root@master:~#
```

```
root@master:~#
root@master:~# kubectl describe node worker1
Name:               worker1
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    env=prod
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker1
                    kubernetes.io/os=linux
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    projectcalico.org/IPv4Address: 172.31.0.70/20
                    projectcalico.org/IPv4IPIPTunnelAddr: 192.168.235.128
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Sun, 08 Jan 2023 10:04:30 +0000
Taints:             <none>
Unschedulable:      false
```

Above output shows that the taint has been removed from worker1 node.