Node Selector

Introduction:

We can *restrict* pod to be scheduled on particular node(s), or to *prefer* to run on particular nodes. By default, the **scheduler** will automatically do a reasonable placement (for example, spreading your Pods across nodes so as not place Pods on a node with insufficient free resources). However, there are some circumstances where you may want to control which node the Pod deploys to, for example, to ensure that a Pod ends up on a node with an SSD attached to it, or to co-locate Pods from two different services that communicate a lot into the same availability zone.

We can use following methods to schedule our pods on particular nodes.

1. nodeName
2. nodeSelector
3. Affinity and anti-affinity

Objectives:

1. Deploy a pod using nodeSelector
2. Deploy a pod using affinity
3. Deploy a pod using nodeName

1. Deploy a pod using nodeSelector:

nodeSelector is the simplest recommended form of node selection constraint. You can add the nodeSelector field to your Pod specification and specify the node labels you want the target node to have. Kubernetes only schedules the Pod onto nodes that have each of the labels you specify.

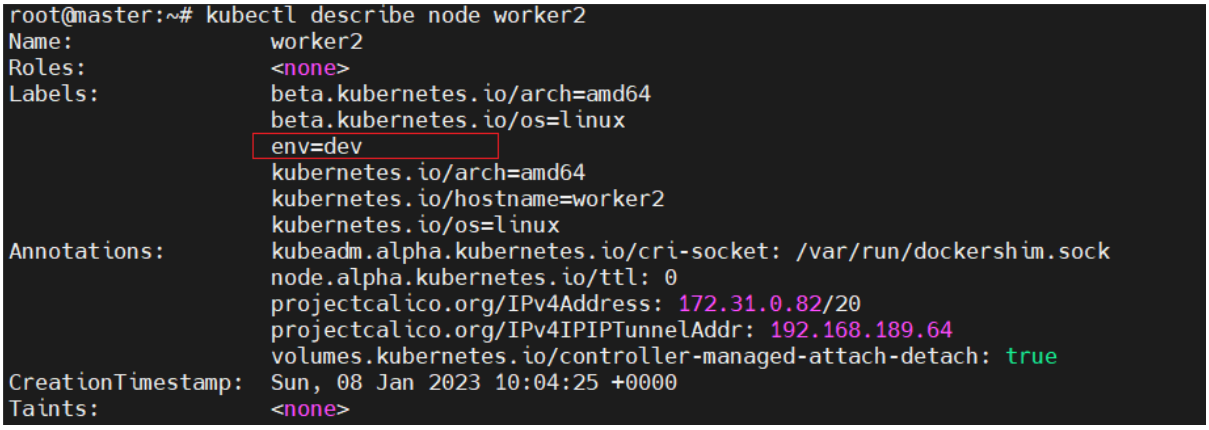
Let’s schedule the pod on a particular node using the nodeSelector.

Step 1: Label the nodes

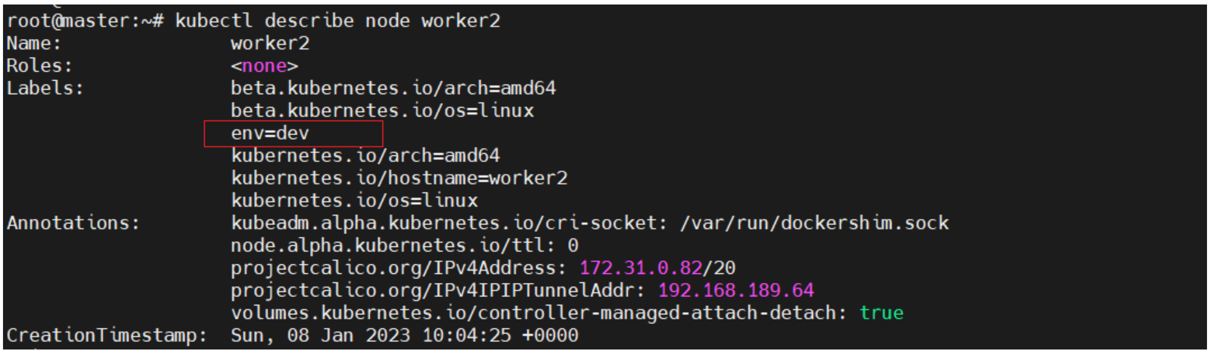
```
kubectl label node worker1 env=prod
```

```
kubectl label node worker2 env=dev
```

We have labelled **worker1** with **env=prod** and **worker2** with **env=dev**. See the output below.



```
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>
```



```
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
```

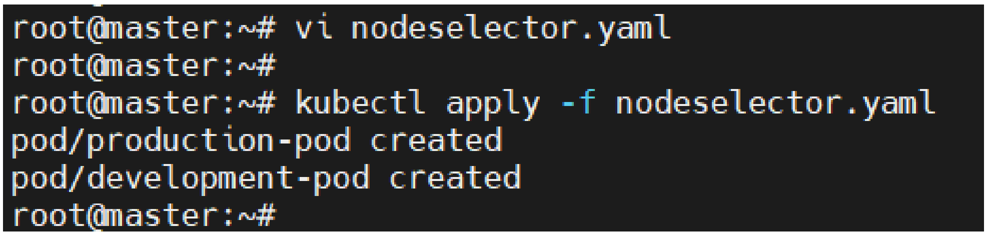
Step2: Schedule a pod

Let’s create a Yaml file to schedule the pods on the basis of **nodeSelector**.

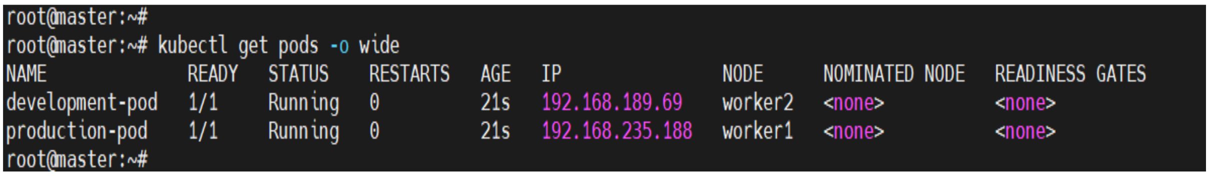
```
vi nodeselector.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: production-pod
  labels:
    env: prod
spec:
  containers:
    - name: container1
      image: nginx
  nodeSelector:
    env: prod
---
apiVersion: v1
kind: Pod
metadata:
  name: development-pod
  labels:
    env: dev
spec:
  containers:
    - name: container1
      image: httpd:2.4
  nodeSelector:
      env: dev
```

Above file mention that we are going to schedule **production-pod** on a node having **env=prod** label whereas **development-pod** on a node having **env=dev** label.



```
root@master:~# vi nodeselector.yaml
root@master:~#
root@master:~# kubectl apply -f nodeselector.yaml
pod/production-pod created
pod/development-pod created
root@master:~#
```



```
root@master:~#
root@master:~# kubectl get pods -o wide
NAME              READY   STATUS    RESTARTS   AGE   IP                NODE      NOMINATED NODE   READINESS GATES
development-pod   1/1     Running   0          21s   192.168.189.69    worker2   <none>           <none>
production-pod    1/1     Running   0          21s   192.168.235.188   worker1   <none>           <none>
root@master:~#
```

from above output we can see that the pods have been scheduled as per our definition file.

2. Deploy a pod using affinity

Affinity/anti-affinity gives you more control over the selection process. We can define a rule which can be *soft* or *preferred*, so that the scheduler still schedules the Pod even if it can't find a matching node.

We can use operators like In, NotIn, Exists, DoesNotExist, Gt and Lt.

Let’s create a Yaml file using the affinity features.

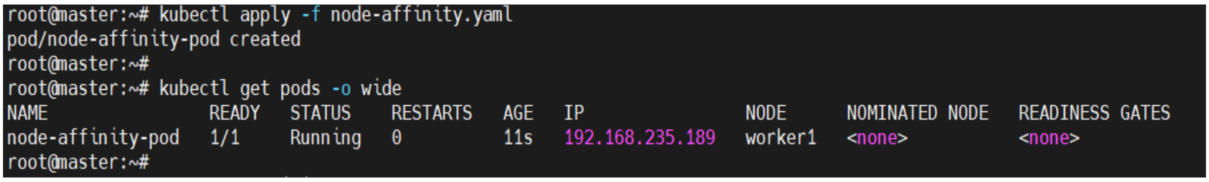
As we are having **worker1** node with **env=prod** and **worker2** with **env=dev** labels. We are using below node affinity features in our file.

- **requiredDuringSchedulingIgnoredDuringExecution**: The scheduler can't schedule the Pod unless the rule is met. This functions like nodeSelector, but with a more expressive syntax.
- **preferredDuringSchedulingIgnoredDuringExecution**: The scheduler tries to find a node that meets the rule. If a matching node is not available, the scheduler still schedules the Pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: node-affinity-pod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: env
            operator: In
            values:
            - prod
            - test
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: env
            operator: In
            values:
            - dev
  containers:
  - name: container1
    image: nginx
```

Above we are giving providing two values **prod** or **test** so scheduler will check for one of the label and will schedule it. Let's apply this definition file.

```
kubectl apply -f node-affinity.yaml
```



```
root@master:~# kubectl apply -f node-affinity.yaml
pod/node-affinity-pod created
root@master:~#
root@master:~# kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP                NODE      NOMINATED NODE   READINESS GATES
node-affinity-pod   1/1     Running   0          11s   192.168.235.189   worker1   <none>           <none>
root@master:~#
```

From above output we can see that the pod has been scheduled as per our affinity rule.

3. Deploy a pod using nodeName:

nodeName is a more direct form of node selection than affinity or nodeSelector. If the nodeName field is not empty, the scheduler ignores the Pod and the kubelet on the named node tries to place the Pod on that node. Using nodeName overrules using nodeSelector or affinity and anti-affinity rules.

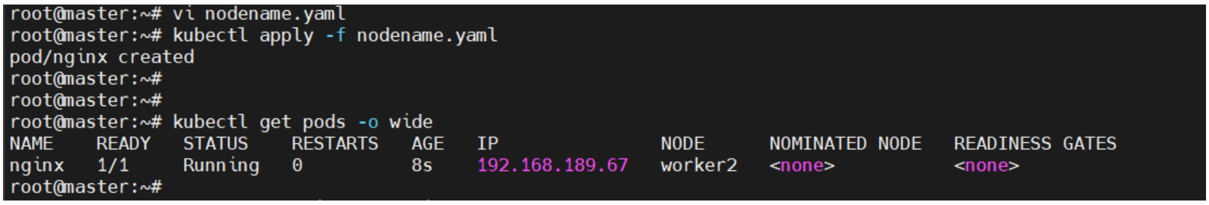
Use the below yaml format to schedule our pod on worker2.

```
# vi nodename.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeName: worker2
  containers:
  - name: nginx
    image: nginx
```

Apply the definition file.

```
Kubectl apply -f nodename.yaml
```



```
root@master:~# vi nodename.yaml
root@master:~# kubectl apply -f nodename.yaml
pod/nginx created
root@master:~#
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          8s    192.168.189.67   worker2   <none>           <none>
root@master:~#
```

So the above output shows that the pod has been scheduled on worker2.