Table of Contents

Scheduler Pods Lab

Goals

- Use a node selector to pin a pod to a node
- Use node affinity and anti-affinity to place pods
- Use pod affinity and anti-affinity to place pods
- Use taints and tolerations to place pods

---

1. Use Node Selectors

In this part of the lab, you use a node selector to place a pod on a dedicated node.

In this lab, you use small `hello-openshift` containers rather than real workloads.

In your cluster you have three compute nodes available:

- `node1.$GUID.internal`
- `node2.$GUID.internal`
- `node3.$GUID.internal`

You can also use `oc get nodes|grep compute` to determine your specific nodes. For the next few exercises you need to place labels on these nodes.

1. Label the nodes as follows:

`node1.$GUID.internal`	`ssd=true`, `zone=zone1`
`node2.$GUID.internal`	`ssd=false`, `zone=zone1`

`node3.$GUID.internal`	`ssd=true`, `zone=zone2`

```
oc label node node1.$GUID.internal ssd=true
oc label node node1.$GUID.internal zone=zone1
oc label node node2.$GUID.internal ssd=false
oc label node node2.$GUID.internal zone=zone1
oc label node node3.$GUID.internal ssd=true
oc label node node3.$GUID.internal zone=zone2
```

2. Validate that your nodes have the correct labels:

```
oc get nodes --show-labels
```

Sample Output

```
node1.$GUID.internal        Ready     compute   6h        v1.10.0+b81c8f8   b
node2.$GUID.internal        Ready     compute   6h        v1.10.0+b81c8f8   b
node3.$GUID.internal        Ready     compute   11m       v1.10.0+b81c8f8   b
```

3. Create a new project:

```
oc new-project scheduling
```

4. Deploy two `hello-openshift` applications in the new project:

```
oc new-app openshift/hello-openshift:v3.10 --name=ssd-zone1 -n scheduling
oc new-app openshift/hello-openshift:v3.10 --name=ssd-zone2 -n scheduling
```

5. Set the node selector on the first application so that it runs on a node that has an SSD and is in zone 1 (expect this to be only `node1`).

```
oc edit dc ssd-zone1
[....]
      dnsPolicy: ClusterFirst
      nodeSelector:
        ssd: "true"
        zone: zone1
      restartPolicy: Always
[....]
```

6. Set the node selector on the second application so that it runs on a node that has an SSD and is in zone 2 (expect this to be only `node3`).

```
oc edit dc ssd-zone2
[....]
      dnsPolicy: ClusterFirst
      nodeSelector:
        ssd: "true"
        zone: zone2
      restartPolicy: Always
[....]
```

7. Validate that the pods are on the right nodes:

```
oc get pod -o wide
```

8. Now update the applications to use node affinity instead of node selectors. Delete the node selector from the deployment configuration and replace it with a required `nodeAffinity` rule to place the pods as before:

`ssd-zone1`	`ssd=true`, `zone=zone1`
`ssd-zone2`	`ssd=true`, `zone=zone2`

```
oc edit dc ssd-zone1

[...]
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: ssd
                operator: In
                values:
                - "true"
              - key: zone
                operator: In
                values:
                - "zone1"
      containers:
[...]

oc edit dc ssd-zone2

[...]
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: ssd
                operator: In
                values:
                - "true"
              - key: zone
                operator: In
                values:
                - "zone2"
      containers:
[...]
```

9. Again, validate that the pods are running on the correct hosts:

```
oc get pod -o wide
```

Sample Output

```
NAME                READY     STATUS    RESTARTS   AGE       IP           NODE
ssd-zone1-3-j59gn   1/1       Running   0          3h        10.1.4.12    node
ssd-zone2-5-56929   1/1       Running   0          1m        10.1.8.5     node
```

10. Delete the two applications:

```
oc delete all -lapp=ssd-zone1
oc delete all -lapp=ssd-zone2
```

---

2. Explore Pod Affinity and Anti-Affinity

In this section, you explore pod affinity and anti-affinity rules. You create two applications with two replicas apiece. The first application has anti-affinity rules to spread the two replicas out over all three nodes. This first application could be, for example, a Redis cache.

The second application has anti-affinity rules to spread it out as well. But it also has affinity rules to bind the pods to the same nodes that the first application is running on. This second application could be a web server that needs to have each replica on the same node as a Redis cache pod.

1. Deploy two `hello-openshift` applications in the new project:

```
oc new-app openshift/hello-openshift:v3.10 --name=cache -n scheduling
oc new-app openshift/hello-openshift:v3.10 --name=webserver -n scheduling
```

2. Set up the first application (`cache`) with pod anti-affinity rules to spread it out over the available nodes. You can use the `app=cache` label that the `oc new-app` command automatically added to the deployment configuration.

```
oc edit dc cache

[...]
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - cache
              topologyKey: kubernetes.io/hostname
      containers:
[...]
```

3. Scale the `cache` application to two replicas.

```
oc scale dc cache --replicas=2
```

4. Verify that the three pods are all running on different nodes:

```
oc get pod -o wide
```

Sample Output

```
NAME                READY     STATUS    RESTARTS   AGE       IP           NODE
cache-2-8n2kr       1/1       Running   0          3s        10.1.2.14    node
cache-2-v7cnv       1/1       Running   0          47s       10.1.8.8     node
webserver-1-pqqsz   1/1       Running   0          2m        10.1.8.7     node
```

Your pods may land on different nodes. The important part is that they do not land on the same node.

5. Set up the **`webserver`** deployment configuration with pod anti-affinity rules for itself and an affinity rule for the **`cache`** application. You can use the application labels again (**`app=cache`** and **`app=webserver`**) to set up the correct affinity and anti-affinity rules.

```
oc edit dc webserver

[...]
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - cache
            topologyKey: kubernetes.io/hostname
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - webserver
              topologyKey: kubernetes.io/hostname
            weight: 100
      containers:
[...]
```

6. Scale the **`webserver`** application to two replicas.

```
oc scale dc webserver --replicas=2
```

7. Validate that the pods are all running on the correct nodes:

```
oc get pod -o wide
```

Sample Output

```
NAME                READY     STATUS    RESTARTS   AGE       IP           NODE
cache-3-k9lr9       1/1       Running   0          58m       10.1.2.15    node
cache-3-rzbf5       1/1       Running   0          58m       10.1.4.17    node
webserver-4-pdtj8   1/1       Running   0          13m       10.1.2.17    node
webserver-4-q48x7   1/1       Running   0          13m       10.1.4.22    node
```

- Expect to see a `cache` and associated `webserver` pod on the same node. This means that one node does not have any pods scheduled—in the example above, the empty node is `node3`.

8. Once again delete the two applications:

```
oc delete all -lapp=cache
oc delete all -lapp=webserver
```

---

3. Work with Taints and Tolerations

In this section, you set up taints on the three nodes to influence pod placement. Then you set up tolerations on some pods to schedule them on the tainted nodes.

Remember that you have two nodes (`node1` and `node3`) labeled as `ssd=true`. These nodes will now also be tainted as `ssd=true`. This means that pods that do not specify a toleration of `ssd=true` will not land on these nodes.

1. Add the taint to the two nodes using the `NoSchedule` effects.

```
oc adm taint nodes node1.$GUID.internal node3.$GUID.internal
ssd=true:NoSchedule
```

2. Deploy two `hello-openshift` applications again as `cache` and `webserver`:

```
oc new-app openshift/hello-openshift:v3.10 --name=cache -n scheduling
oc new-app openshift/hello-openshift:v3.10 --name=webserver -n scheduling
```

3. Scale both applications to two replicas.

```
oc scale dc cache --replicas=2
oc scale dc webserver --replicas=2
```

4. Confirm where the pods are running:

```
oc get pods -o wide
```

Sample Output

```
NAME                READY     STATUS    RESTARTS   AGE       IP
NODE
cache-1-4m4dn       1/1       Running   0          38s       10.1.2.24
node2.$GUID.internal
cache-1-5q8gn       1/1       Running   0          3s        10.1.2.26
node2.$GUID.internal
webserver-1-9jbfw   1/1       Running   0          37s       10.1.2.25
node2.$GUID.internal
webserver-1-hqtvs   1/1       Running   0          3s        10.1.2.27
node2.$GUID.internal
```

   - Every single pod is running on **`node2`**, which is the one node that did not get a taint. The other two nodes are *repelling* the pods.

5. Add a toleration to the web server application to allow it to use nodes with SSDs. (The cache is usually memory-based and does not benefit from the faster disk, so it is fine to leave it on non-SSD nodes).

```
oc edit dc webserver

[...]
      terminationGracePeriodSeconds: 30
      tolerations:
      - effect: NoSchedule
        key: ssd
        operator: Equal
        value: "true"
  test: false
[...]
```

6. Verify that the newly deployed web server pods are now running on the SSD nodes:

```
oc get pod -o wide
```

Sample Output

```
NAME                READY     STATUS    RESTARTS   AGE       IP
NODE
cache-1-4m4dn       1/1       Running   0          6h        10.1.2.24
node2.$GUID.internal
cache-1-5q8gn       1/1       Running   0          6h        10.1.2.26
node2.$GUID.internal
webserver-3-cm8k6   1/1       Running   0          5h        10.1.8.15
node3.$GUID.internal
webserver-3-twm2j   1/1       Running   0          5h        10.1.4.29
node1.$GUID.internal
```

4. Clean Up Environment

1. Remove the scheduling project:

```
oc delete project scheduling
```

2. Remove the taints and labels from the three nodes.

```
oc adm taint nodes node1.$GUID.internal node3.$GUID.internal ssd-
oc label nodes node1.$GUID.internal node2.$GUID.internal
node3.$GUID.internal ssd- zone-
```

3. Validate that the nodes no longer have the labels:

```
oc get nodes --show-labels
```

Sample Output

```
node1.$GUID.internal        Ready     compute   1d        v1.10.0+b81c8f8   b
node2.$GUID.internal        Ready     compute   1d        v1.10.0+b81c8f8   b
node3.$GUID.internal        Ready     compute   1d        v1.10.0+b81c8f8   b
```

4. Validate that the nodes no longer have taints:

```
oc describe nodes node1.$GUID.internal node2.$GUID.internal
node3.$GUID.internal|grep Taints
```

Sample Output

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

This concludes this lab.

Build Version: 2.12 : Last updated 2019-03-27 09:54:17 EDT