Pod Placement

» Pod Scheduling

Overview

Use the various methods available to help control how Kubernetes schedules pods in the cluster.

First, create a basic deployment with 3 replicas.

```
kubectl run httpd --image httpd --port 80 --replicas 3
```

Verify the worker node that is hosting each Pod in the deployment. The pods should be spread across all (or most) of the available worker nodes.

```
kubectl get pods -l run=httpd -o wide
```

Clean Up

Delete the basic `httpd` deployment.

```
kubectl delete deploy httpd
```

nodeSelector

Get the list of nodes in the cluster.

```
kubectl get nodes -o wide
```

Get the list of worker nodes in the cluster.

```
kubectl get nodes -o wide -l node-role.kubernetes.io/node
```

Save the name of the first worker node in an environment variable.

```
WORKER_NODE=$(kubectl get nodes -l node-role.kubernetes.io/node -o jsonpath={.items[0].metadata.name})
echo $WORKER_NODE
```

Add a label to the worker node indicating that SSD storage is available.

```
kubectl label node $WORKER_NODE disktype=ssd
```

View the list of labels associated with the worker node.

```
kubectl get node $WORKER_NODE --show-labels | grep disktype
```

Define a manifest that will create a deployment that uses the label.

```
cat > node-selector.yaml <<EOF
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  template:
    metadata:
      labels:
        env: test
    spec:
      containers:
      - name: nginx
        image: nginx:1.13
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
      nodeSelector:
        disktype: ssd
EOF
```

Create the Deployment based on the manifest.

```
kubectl create -f node-selector.yaml
```

View the basic information for the pods to confirm the host.

```
kubectl get pods -l env=test -o wide
```

View the detail for the worker node to confirm the pods are running.

Also make note of some of the built-in node labels like `kubernetes.io/hostname` , `beta.kubernetes.io/os` or `beta.kubernetes.io/arch` .

```
kubectl describe node $WORKER_NODE
```

Clean Up

Clean up the `nginx` deployment that we created.

```
kubectl delete -f node-selector.yaml
```

Node Affinity/Anti-Affinity

Create a manifest for a new deployment that uses Node Affinity.

```
cat > node-affinity.yaml <<EOF
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: node-affinity
spec:
  replicas: 3
  template:
    metadata:
      labels:
        env: test
    spec:
      containers:
      - name: nginx
        image: nginx:1.13
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: disktype
                operator: In
                values:
                - ssd
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: beta.kubernetes.io/os
                operator: In
                values:
                - linux
EOF
```

Create the new deployment based on the manifest.

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

Verify that the pods are running on the requested host.

```
kubectl get pods -o wide -l env=test
```

Clean Up

Clean up the `node-affinity` deployment that we created.

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

Taints and Tolerations

Create a new taint for the worker node that we identified earlier.

```
kubectl taint node $WORKER_NODE gpu=present:NoSchedule
```

View the detailed information for the worker node to view the taint.

```
kubectl describe node $WORKER_NODE
```

Define 2 different deployments. One that will tolerate the taint that we set on the worker node and another that does not define a toleration.

```
cat > taint-tolerate.yaml <<EOF
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: tolerate
spec:
  replicas: 3
  template:
    metadata:
      labels:
        env: test
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
      tolerations:
      - key: "gpu"
        operator: "Equal"
        value: "present"
        effect: "NoSchedule"
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: notolerate
spec:
  replicas: 3
  template:
    metadata:
      labels:
        env: test
    spec:
      containers:
      - name: httpd
        image: httpd:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
EOF
```

Create the 2 new deployments based on the manifest.

```
kubectl create -f taint-tolerate.yaml
```

View the basic information for the pods and make note of the host for each.

```
kubectl get pods -o wide
```

Remove the taint from the worker node.

```
kubectl taint node $WORKER_NODE gpu-
```

Now that the taint has been removed, create another simple deployment to verify that node can be once again scheduled for pods.

First, create a basic deployment with 3 replicas.

```
kubectl run nginx --image nginx --port 80 --replicas 3
```

Verify the worker node that is hosting each Pod in the deployment. The pods should be spread across all (or most) of the available worker nodes.

```
kubectl get pods -l run=nginx -o wide
```

Clean Up

Remove the deployments that we created.

```
kubectl delete -f taint-tolerate.yaml
kubectl delete deploy nginx
```

Pod Affinity/Anti-Affinity

First, create a basic deployment with 4 replicas.

```
kubectl run httpd --image httpd --port 80 --replicas 4
```

Verify the worker node that is hosting each Pod in the deployment. The pods should be spread across all (or most) of the available worker nodes.

```
kubectl get pods -l run=httpd -o wide
```

Now create a new manifest that specifies a Redis cache deployment, as well as a web server deployment.

```
cat > pod-affinity.yaml <<EOF
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: redis-cache
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: myapp
        tier: backend
    spec:
      containers:
      - name: redis-server
        image: redis:3.2-alpine
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: web-server
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: myapp
        tier: frontend
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: tier
                operator: In
                values:
                - backend
            topologyKey: "kubernetes.io/hostname"
      containers:
      - name: nginx
        image: nginx:latest
        imagePullPolicy: IfNotPresent
        ports:
```

```
        ports:
        - containerPort: 80
EOF
```

Create the new deployment based on the manifest.

```
kubectl create -f pod-affinity.yaml
```

Verify that the cache and we-server pods are co-located.

```
kubectl get pods -o wide
```

Clean Up

When you are finished, clean up the resources that were created.

```
kubectl delete -f pod-affinity.yaml
kubectl delete deploy httpd
```

View any remaining resources.

```
kubectl get all
```