

Working with affinity and non-affinity scheduling

Node affinity was introduced as alpha in Kubernetes 1.2. Node affinity is conceptually similar to nodeSelector.

There are currently two types of node affinity
requiredDuringSchedulingIgnoredDuringExecution and
preferredDuringSchedulingIgnoredDuringExecution.

The requiredDuringSchedulingIgnoredDuringExecution works as similar to how nodeSelector works, if labels on a node change at runtime such that the affinity rules on a pod are no longer met, the pod will still continue to run on the node.

The requiredDuringSchedulingRequiredDuringExecution works as similar to how nodeSelector works, except that it will evict pods from nodes that cease to satisfy the pods node affinity requirements.

List the node

```
$ kubectl get nodes
```

	NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	21m	v1.10.2	
worker1	Ready	<none>	17m	v1.10.2	
worker2	Ready	<none>	16m	v1.10.2	

Label the node worker2.

```
$ kubectl label node worker2 name=devserver
```

```
node "worker2" labeled
```

Get the labels of the nodes.

```
$ kubectl get nodes --show-labels
```

NAME	STATUS	ROLES	AGE	VERSION	LABELS
------	--------	-------	-----	---------	--------

```
master Ready master 22m v1.10.2
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,\
kubernetes.io/hostname=master,node-role.kubernetes.io/master=
worker1 Ready <none> 18m v1.10.2
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,\
kubernetes.io/hostname=worker1
worker2 Ready <none> 18m v1.10.2
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,\
kubernetes.io/hostname=worker2,name=devserver
```

Create a pod with Node Affinity-affinity.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: beta.kubernetes.io/os
                operator: In
                values:
                  - linux
            preferredDuringSchedulingIgnoredDuringExecution:
              - weight: 1
                preference:
                  matchExpressions:
                    - key: name
```

```
operator: In
values:
- devserver

containers:
- name: nginx-node-affinity
  image: nginx
  ports:
  - containerPort: 80
```

This node affinity rule says the pod can only be placed on a node with a label whose key is beta.kubernetes.io/os and whose value is linux.

In the field preferredDuringSchedulingIgnoredDuringExecution nodes with a label whose key is name and whose value is devserver should be preferred.

The weight field in preferredDuringSchedulingIgnoredDuringExecution is in the range 1-100. For each node that meets all of the scheduling requirements, the scheduler will compute a sum by iterating through the elements of this field and adding “weight” to the sum if the node matches the corresponding MatchExpressions. This score is then combined with the scores of other priority functions for the node. The node(s) with the highest total score are the most preferred.

Deploy the pod.

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

```
$ kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
nginx-node-affinity	1/1	Running	0	44s	192.168.189.66	worker2

A

Delete the yaml

Inter-pod affinity/anti-affinity.

In a three node cluster, Here is the yaml snippet of a simple mongodb deployment with three replicas and selector label appdb=rsvpdb. The deployment has PodAntiAffinity configured to ensure the scheduler does not co-locate replicas on a single node.

Create a Mongodb deployment.

```
kind: PersistentVolume
```

```
apiVersion: v1
```

```
metadata:
```

```
  name: pv0001
```

```
  labels:
```

```
    type: local
```

```
spec:
```

```
  capacity:
```

```
    storage: 1Gi
```

```
  accessModes:
```

```
    - ReadWriteOnce
```

```
  hostPath:
```

```
    path: "/tmp/data01"
```

```
---
```

```
kind: PersistentVolumeClaim
```

```
apiVersion: v1
```

```
metadata:
```

```
  name: myclaim-1
```

```
spec:
```

```
  accessModes:
```

```
    - ReadWriteOnce
```

```
  resources:
```

```
    requests:
```

storage: 0.5Gi

apiVersion: apps/v1

kind: Deployment

metadata:

name: rsvp-db

spec:

replicas: 2

selector:

matchLabels:

appdb: rsvpdb

template:

metadata:

labels:

appdb: rsvpdb

spec:

affinity:

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: appdb

operator: In

values:

- rsvpdb

topologyKey: "kubernetes.io/hostname"

volumes:

- name: voldb

persistentVolumeClaim:

```
    claimName: myclaim-1
containers:
- name: rsvpd-db
  image: mongo:3.3
  volumeMounts:
  - name: voldb
    mountPath: /data/db
  env:
  - name: MONGODB_DATABASE
    value: rsvpdata
  ports:
  - containerPort: 27017
```

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  labels:
    app: rsvpdb
spec:
  ports:
  - port: 27017
    protocol: TCP
  selector:
    appdb: rsvpdb
```

Deploy the Mongodb.

```
$ kubectl apply -f configs/2-mongodb.yaml
```

Get the list of pods.

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

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
rsvp-db-68bc5b9db9-9dzpn	1/1	Running	0	38s	192.168.189.67	worker2
rsvp-db-68bc5b9db9-fxgmd	1/1	Running	0	38s	192.168.235.131	worker1

Each node is having only one replicas.

Create a RSVP application with podAntiAffinity and podAffinity.

The below yaml snippet of the RSVP deployment has podAntiAffinity and podAffinity configured. This informs the scheduler that all its replicas are to be co-located with pods that have selector label appdb=rsvpdb. This will also ensure that each RSVP replica does not co-locate on a single node.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rsvp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: rsvp
  template:
    metadata:
      labels:
        app: rsvp
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
```

```
- key: app
  operator: In
  values:
    - rsvp
  topologyKey: "kubernetes.io/hostname"
podAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
          - key: appdb
            operator: In
            values:
              - rsvpdb
        topologyKey: "kubernetes.io/hostname"
containers:
- name: rsvp-app
  image: teamcloudyuga/rsvpapp
  livenessProbe:
    httpGet:
      path: /
      port: 5000
    periodSeconds: 30
    timeoutSeconds: 1
    initialDelaySeconds: 50
  env:
    - name: MONGODB_HOST
      value: mongodb
  ports:
    - containerPort: 5000
```



```
    name: web-port
---
apiVersion: v1
kind: Service
metadata:
  name: rsvp
  labels:
    app: rsvp
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: web-port
      protocol: TCP
  selector:
    app: rsvp
```

Deploy the application.

```
$ kubectl apply -f configs/3-rsvp.yaml
```

As we can see, all the 3 replicas of the RSVP are automatically co-located with the RSVP-DB as expected.

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

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
rsvp-8b74dbffc-72qhn	1/1	Running	0	16s	192.168.189.68	worker2
rsvp-8b74dbffc-pc9mf	1/1	Running	0	16s	192.168.235.132	worker1
rsvp-db-68bc5b9db9-9dzpn	1/1	Running	0	12m	192.168.189.67	worker2
rsvp-db-68bc5b9db9-fxgmd	1/1	Running	0	12m	192.168.235.131	worker1

The above example uses PodAntiAffinity and PodAffinity rules with topologyKey: "kubernetes.io/hostname" to deploy the rsvp/rsvp-db cluster so that no two instances of same deployment are located on the same host and Two instance of different deployments are located on same host.

Clean Up

```
$ kubectl delete -f configs/.
```