进阶篇-高级调度

CronJob

介绍

在k8s 里面运行周期性的计划任务，crontab
* * * * * 分时日月周

可以利用 CronJobs 执行基于时间调度的任务。这些自动化任务和 Linux 或者 Unix 系统的 Cron 任务类似
CronJobs在创建周期性以及重复性的任务时很有帮助，例如执行备份操作或者发送邮件。CronJobs 也可以在特定时间调度单个任务，例如你想调度低活跃周期的任务。

创建一个Job

```
[root@k8s-master01 app]# cat  cronjob.yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure

# 创建job
kubectl create -f cronjob.yaml
```

查看job

```
[root@k8s-master01 app]# kubectl get jobs --watch
NAME               COMPLETIONS   DURATION   AGE
hello-1609167960   0/1           7m44s      7m44s
hello-1609168020   0/1           6m42s      6m42s
hello-1609168080   0/1           5m41s      5m41s
hello-1609168140   0/1           4m50s      4m50s
hello-1609168200   0/1           3m49s      3m49s
hello-1609168260   1/1           21s        2m48s
hello-1609168320   1/1           2s         107s
hello-1609168380   1/1           3s         46s

[root@k8s-master01 app]# kubectl get cronjob hello
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     6        4s              8m20s

# 删除
[root@k8s-master01 app]# kubectl delete cronjob hello
cronjob.batch "hello" deleted
```

yaml文件参数介绍

```
kubectl get cj hello -oyaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  labels:
    run: hello
  name: hello
  namespace: default
spec:
  concurrencyPolicy: Allow #并发调度策略：Allow运行同时运行过个任务。
                # Forbid：不运行并发执行。
                # Replace：替换之前的任务
  failedJobsHistoryLimit: 1 # 保留失败的任务数。
  jobTemplate:
    metadata:
      creationTimestamp: null
    spec:
      template:
        metadata:
          creationTimestamp: null
          labels:
            run: hello
        spec:
          containers:
          - args:
```

```
            - /bin/sh
            - -c
            - date
            image: nginx
            imagePullPolicy: IfNotPresent
            name: hello
            resources: {}
            terminationMessagePath: /dev/termination-log
            terminationMessagePolicy: File
          dnsPolicy: ClusterFirst
          restartPolicy: OnFailure
          schedulerName: default-scheduler
          securityContext: {}
          terminationGracePeriodSeconds: 30
  schedule: '*/1 * * * *'  #调度的策略 分时日月周
  successfulJobsHistoryLimit: 3 # 成功的Job保留的次数
  suspend: false # 挂起，true：cronjob不会被执行。
status: {}
```

污点Taint和容忍Toleration

什么是Taint和Toleration

所谓污点就是故意给某个节点服务器上设置个污点参数，那么你就能让生成pod的时候使用相应的参数去避开有污点参数的node服务器。而容忍呢，就是当资源不够用的时候，即使这个node服务器上有污点，那么只要pod的yaml配置文件中写了容忍参数，最终pod还是会容忍的生成在该污点服务器上。默认master节点是NoSchedule

Taint（污点）

污点(Taint)的组成

使用kubectl taint命令可以给某个Node节点设置污点，Node被设置上污点之后就和Pod之间存在了一种相斥的关系，可以让Node拒绝Pod的调度执行，甚至将Node已经存在的Pod驱逐出去。key=value:effect

每个污点有一个key和value作为污点的标签，其中value可以为空，effect描述污点的作用。当前taint effect支持如下三个选项：

NoSchedule：表示k8s将不会将Pod调度到具有该污点的Node上

PreferNoSchedule：表示k8s将尽量避免将Pod调度到具有该污点的Node上

NoExecute：表示k8s将不会将Pod调度到具有该污点的Node上，同时会将Node上已经存在的Pod驱逐出去

查看某个节点的Taint配置情况

```
# 1、查看所有node情况
[root@k8s-master01 ~]# kubectl get node
NAME           STATUS   ROLES    AGE     VERSION
k8s-master01   Ready    matser   7d19h   v1.20.0
k8s-master02   Ready    <none>   7d19h   v1.20.0
k8s-master03   Ready    <none>   7d19h   v1.20.0
k8s-node01     Ready    <none>   7d19h   v1.20.0
k8s-node02     Ready    <none>   7d19h   v1.20.0

# 2、查看某个节点的Taint信息(kubectl describe node nodename)
[root@k8s-master01 ~]# kubectl describe node k8s-node01  (内容太多不贴全了)
Name:               k8s-node01
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    disktype=ssd
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=k8s-node01
                    kubernetes.io/os=linux
                    node.kubernetes.io/node=
Annotations:        node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Mon, 21 Dec 2020 05:13:32 +0800
Taints:             <none>   # 关注这个地方即可 ---没有设置过污点的节点属性中的参数是这样的
Taints:     <none>
Unschedulable:      false

#添加 尽量不调度 PreferNoSchedule
kubectl taint nodes k8s-master02 node-role.kubernetes.io/master:PreferNoSchedule
#去除污点NoSchedule，最后一个"-"代表删除
kubectl taint nodes k8s-master02 node-role.kubernetes.io/master:NoSchedule-
```

给某个节点服务器打上污点标签

```
# 1、先看一下当前pod都分布到哪些节点上
[root@k8s-master01 ~]# kubectl get pod -owide
NAME                        READY   STATUS  RESTARTS     AGE     IP                NODE
          NOMINATED NODE   READINESS GATES
nginx-btl2c                 1/1     Running   3          6d      172.162.195.26    k8s-
master03   <none>           <none>
nginx-c9qf5                 1/1     Running   3          6d      172.161.125.27    k8s-
node01     <none>           <none>
nginx-pl9gs                 1/1     Running   3          6d      172.169.92.103    k8s-
master02   <none>           <none>
```

```
# 2、给节点k8s-node01服务器打上污点标签NoExecute
[root@k8s-master01 ~]# kubectl taint nodes k8s-node01 check=xtaint:NoExecute
`注释:
check-------->键
value: "xtaint"------------>容忍的键对应的键值
"NoExecute"------------>容忍的键对应的影响效果effect
`
# 3、再次查看pod，发现k8s-node01节点上的nginx-c9qf5容器正在被删除，再过一会，就被彻底删除了，这正是我们想要的效果!
[root@k8s-master01 ~]# kubectl get pod -owide
NAME              READY   STATUS        AGE   IP              NODE        NOMINATED NODE
READINESS GATES
nginx-c9qf5       1/1     Terminating   6d   172.161.125.27   k8s-node01   <none>
<none>
```

删除某个节点上的设置的污点

```
# 跟删除标签的方式有点类似，在后面加个 "-"
[root@k8s-master01 ~]# kubectl taint nodes k8s-node01 check=xtaint:NoExecute-
node/k8s-node01 untainted

"
 Taints:    test=xtaint:NoExecute
        check=xtaint:NoSchedule
        test=xtaint:NoSchedule
"
# 以上的Taints这样删，必须得带个"-"
kubectl taint nodes k8s-node02 test=xtaint:NoExecute-
kubectl taint nodes k8s-node02 check=xtaint:NoSchedule-
kubectl taint nodes k8s-node02 test=xtaint:NoSchedule-
```

Toleration（容忍）

先在k8s-node02节点上打上一个NoSchedule

```
# 打上NoExecute，k8s-node02、k8s-master01、k8s-master02节点上的pod都会自动被删除
[root@k8s-master01 ~]# kubectl taint nodes k8s-node02 test=xtaint:NoExecute
node/k8s-node02 tainted
```

创建一个包含有容忍toleration的配置文件

```
cat > test-taint-pod.yaml << EFO
apiVersion: v1
kind: Pod
metadata:
  name: nginx
```

```
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.5.2
  tolerations:
  - key: "check"
    operator: "Equal"
    value: "xtaint"
    effect: "NoExecute"
    tolerationSeconds: 3600
EFO

# create Pod
[root@k8s-master01 app]# kubectl create -f test-taint-pod.yaml
pod/nginx created
```

参数解释

```
tolerations:----------->容忍
- key: "check" ----------->容忍的键
operator: "Equal"----------->操作符"等于"
value: "xtaint"----------->容忍的键对应的键值
effect: "NoExecute"----------->容忍的键对应的影响效果
tolerationSeconds: 3600----------->容忍3600秒。本pod配置文件中有这个参数了，然后再给本服务器设置污点NoExecute，那么这个pod也不会像普通pod那样立即被驱逐，而是再等上3600秒才被删除。
```

toleration配置方式

```
方式一：
tolerations:
- key: "key"
  operator: "Equal"
  value: "value"
  effect: "NoSchedule"
方式二：
tolerations:
- key: "key"
  operator: "Exists"
  effect: "NoSchedule"

    一个Toleration和一个Taint相匹配是指它们有一样的key和effect，并且如果operator是Exists（此时toleration不指定value）或者operator是Equal，则它们的value应该相等。
注意两种情况：
```

如果一个Toleration的key为空且operator为Exists，表示这个Toleration与任意的key、value和effect都匹配，即这个Toleration能容忍任意的Taint：

```
tolerations:
- operator: "Exists"
```

如果一个Toleration的effect为空，则key与之相同的相匹配的Taint的effect可以是任意值：

```
tolerations:
- key: "key"
  operator: "Exists"
```

上述例子使用到effect的一个值NoSchedule，也可以使用PreferNoSchedule，该值定义尽量避免将Pod调度到存在其不能容忍的Taint的节点上，但并不是强制的。effect的值还可以设置为NoExecute。

Kubernetes会自动给Pod添加一个key为node.kubernetes.io/not-ready的Toleration并配置tolerationSeconds=300，同样也会给Pod添加一个key为node.kubernetes.io/unreachable的Toleration并配置tolerationSeconds=300，除非用户自定义了上述key，否则会采用这个默认设置。

一个使用了很多本地状态的应用程序在网络断开时，仍然希望停留在当前节点上运行一段时间，愿意等待网络恢复以避免被驱逐。在这种情况下，Pod的Toleration可以这样配置：

```
tolerations:
- key: "node.alpha.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```

Init Container介绍

什么是Init Container

Init Container就是用来做初始化工作的容器，可以是一个或者多个，如果有多个的话，这些容器会按定义的顺序依次执行，只有所有的Init Container执行完后，主容器才会被启动。我们知道一个Pod里面的所有容器是共享数据卷和网络命名空间的，所以Init Container里面产生的数据可以被主容器使用到的。

Init Container与应用容器本质上是一样的，但他们是仅运行一次就结束的任务，并且必须在成功执行完后，系统才能继续执行下一个容器

Init Container应用场景

- 等待其他量关联组件正确运行（例如solr启动先依赖zookeeper）
- 基于环境变量或配置模板生成配置文件
- 从远程数据库获取本地所需配置，或者将吱声注册到某个中央数据库中
- 下载相关依赖包，或者对系统进行一些预配置操作（可以用python或者bash对系统做初始化操作）

初始容器使用

```
cat > myapp.yaml << EFO
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice;
sleep 2; done;']
  - name: init-mydb
    image: busybox
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2;
done;']
EFO
```

以上pod定义包含两个初始容器,第一个等待`myservice`服务可用,第二个等待`mydb`服务可用,这两个pod执行完成,应用容器开始执行

下面是`myservice`和`mydb`两个服务的yaml文件

```
cat > services.yaml << EFO
kind: Service
apiVersion: v1
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
EFO
#-----------------------#
cat > mydb.yaml << EFO
kind: Service
apiVersion: v1
metadata:
  name: mydb
spec:
```

```
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377
EFO
```

分别构建这些pod、service：

```
# 1、首先构建myapp这个pod、
[root@k8s-master01 app]# kubectl create -f myapp.yaml
pod/myapp-pod created

# 2、查看状态，现在是不会创建成功的因为那2个servicer没初始化
[root@k8s-master01 app]# kubectl get  -f myapp.yaml
NAME        READY   STATUS                  RESTARTS   AGE
myapp-pod   0/1     Init:ImagePullBackOff   0          11s

# 3、构建那2个servier
[root@k8s-master01 app]# kubectl create -f services.yaml
service/myservice created

[root@k8s-master01 app]# kubectl create -f mydb.yaml
service/mydb created

# 4、查看这2个svc
[root@k8s-master01 ~]# kubectl get  svc
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)    AGE
mydb           ClusterIP      10.96.15.77      <none>          80/TCP     38s
myservice      ClusterIP      10.101.111.161   <none>          80/TCP     52s

# 5、查看pod是否构建完成，可以看到已经构建完成。
[root@k8s-master01 app]# kubectl get -f myapp.yaml
NAME        READY   STATUS    RESTARTS   AGE
myapp-pod   1/1     Running   0          4m46s
# 这样查看也可以
[root@k8s-master01 app]# kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
myapp-pod                   1/1     Running   0          6m46s
```

Affinity

介绍

Kubernetes中的调度策略可以大致分为两种

一种是全局的调度策略，要在启动调度器时配置，包括kubernetes调度器自带的各种predicates和priorities算法

另一种是运行时调度策略，包括nodeAffinity（主机亲和性），podAffinity（POD亲和性）以及podAntiAffinity（POD反亲和性）。

nodeAffinity 主要解决POD要部署在哪些主机，以及POD不能部署在哪些主机上的问题，处理的是POD和主机之间的关系。

podAffinity 主要解决POD可以和哪些POD部署在同一个拓扑域中的问题（拓扑域用主机标签实现，可以是单个主机，也可以是多个主机组成的cluster、zone等。）

podAntiAffinity主要解决POD不能和哪些POD部署在同一个拓扑域中的问题。它们处理的是Kubernetes集群内部POD和POD之间的关系。

三种亲和性和反亲和性策略的比较如下表所示：

策略名称	匹配目标	支持的操作符	支持拓扑域	设计目标
nodeAffinity	主机标签	In，NotIn，Exists，DoesNotExist，Gt，Lt	不支持	决定Pod可以部署在哪些主机上
podAffinity	Pod标签	In，NotIn，Exists，DoesNotExist	支持	决定Pod可以和哪些Pod部署在同一拓扑域
PodAntiAffinity	Pod标签	In，NotIn，Exists，DoesNotExist	支持	决定Pod不可以和哪些Pod部署在同一拓扑域

亲和性：应用A与应用B两个应用频繁交互，所以有必要利用亲和性让两个应用的尽可能的靠近，甚至在一个node上，以减少因网络通信而带来的性能损耗。

反亲和性：当应用的采用多副本部署时，有必要采用反亲和性让各个应用实例打散分布在各个node上，以提高HA。

主要介绍kubernetes的中调度算法中的Node affinity和Pod affinity用法

实际上是对前文提到的优选策略中的`NodeAffinityPriority`策略和`InterPodAffinityPriority`策略的具体应用。

kubectl explain pods.spec.affinity

亲和性策略（Affinity）能够提供比NodeSelector或者Taints更灵活丰富的调度方式，例如：

丰富的匹配表达式（In, NotIn, Exists, DoesNotExist. Gt, and Lt）
软约束和硬约束（Required/Preferred）
以节点上的其他Pod作为参照物进行调度计算
亲和性策略分为NodeAffinityPriority策略和InterPodAffinityPriority策略。

Node亲和力

Node affinity（节点亲和性）

kubectl explain pods.spec.affinity.nodeAffinity

据官方说法未来NodeSeletor策略会被废弃，由NodeAffinityPriority策略中requiredDuringSchedulingIgnoredDuringExecution替代。

NodeAffinityPriority策略和NodeSelector一样，通过Node节点的Label标签进行匹配，匹配的表达式有：In, NotIn, Exists, DoesNotExist. Gt, and Lt。

定义节点亲和性规则有2种：硬亲和性（require）和软亲和性（preferred）

硬亲和性：requiredDuringSchedulingIgnoredDuringExecution
软亲和性：preferredDuringSchedulingIgnoredDuringExecution

- 硬亲和性：实现的是强制性规则，是Pod调度时必须满足的规则，否则Pod对象的状态会一直是Pending
- 软亲和性：实现的是一种柔性调度限制，在Pod调度时可以尽量满足其规则，在无法满足规则时，可以调度到一个不匹配规则的节点之上。

需要注意的是`preferred`和`required`后半段字符串`IgnoredDuringExecution`表示：

在Pod资源基于节点亲和性规则调度到某个节点之后，如果节点的标签发生了改变，调度器不会讲Pod对象从该节点上移除，因为该规则仅对新建的Pod对象有效。

硬亲和性

```
[root@k8s-master nodeAffinity]# pwd
/root/k8s_practice/scheduler/nodeAffinity
[root@k8s-master nodeAffinity]# cat node_required_affinity.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-affinity-deploy
  labels:
    app: nodeaffinity-deploy
spec:
  replicas: 5
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-pod
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
```

```
        - containerPort: 80
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              # 表示node标签存在 disk-type=ssd 或 disk-type=sas
              - key: disk-type
                operator: In
                values:
                - ssd
                - sas
              # 表示node标签存在 cpu-num且值大于6
              - key: cpu-num
                operator: Gt
                values:
                - "6"
```

```
# 运行yaml文件并查看状态
[root@k8s-master nodeAffinity]# kubectl apply -f node_required_affinity.yaml
deployment.apps/node-affinity-deploy created
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get deploy -o wide
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                  SELECTOR
node-affinity-deploy   5/5     5            5           6s    myapp-pod    registry.cn-
beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get rs -o wide
NAME                              DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES
                                                          SELECTOR
node-affinity-deploy-5c88ffb8ff   5         5         5       11s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=5c88ffb8ff
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get pod -o wide
NAME                                    READY   STATUS    RESTARTS   AGE   IP
  NODE         NOMINATED NODE   READINESS GATES
node-affinity-deploy-5c88ffb8ff-2mbfl   1/1     Running   0          15s   10.244.4.237
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-9hjhk   1/1     Running   0          15s   10.244.4.235
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-9rg75   1/1     Running   0          15s   10.244.4.239
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-pqtfh   1/1     Running   0          15s   10.244.4.236
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-zqpl8   1/1     Running   0          15s   10.244.4.238
  k8s-node01   <none>
```

由上可见，再根据之前打的标签，很容易推断出当前pod只能调度在k8s-node01节点。

即使我们删除原来的rs，重新生成rs后pod依旧会调度到k8s-node01节点。如下：

```
[root@k8s-master nodeAffinity]# kubectl delete rs node-affinity-deploy-5c88ffb8ff
replicaset.apps "node-affinity-deploy-5c88ffb8ff" deleted
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get rs -o wide
NAME                              DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES
                                                                  SELECTOR
node-affinity-deploy-5c88ffb8ff   5         5         2       4s    myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=5c88ffb8ff
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get pod -o wide
NAME                                    READY   STATUS    RESTARTS   AGE   IP
  NODE         NOMINATED NODE   READINESS GATES
node-affinity-deploy-5c88ffb8ff-2v2tb   1/1     Running   0          11s   10.244.4.241
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-gl4fm   1/1     Running   0          11s   10.244.4.240
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-j26rg   1/1     Running   0          11s   10.244.4.244
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-vhzmn   1/1     Running   0          11s   10.244.4.243
  k8s-node01   <none>           <none>
node-affinity-deploy-5c88ffb8ff-xxj8m   1/1     Running   0          11s   10.244.4.242
  k8s-node01   <none>
```

软亲和性

优先调度到满足条件的节点，如果都不满足也会调度到其他节点。

要运行的yaml文件

```
[root@k8s-master nodeAffinity]# pwd
/root/k8s_practice/scheduler/nodeAffinity
[root@k8s-master nodeAffinity]# cat node_preferred_affinity.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-affinity-deploy
  labels:
    app: nodeaffinity-deploy
```

```yaml
spec:
  replicas: 5
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-pod
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              # 表示node标签存在 disk-type=ssd 或 disk-type=sas
              - key: disk-type
                operator: In
                values:
                - ssd
                - sas
          - weight: 50
            preference:
              matchExpressions:
              # 表示node标签存在 cpu-num且值大于16
              - key: cpu-num
                operator: Gt
                values:
                - "16"
```

```
[root@k8s-master nodeAffinity]# kubectl apply -f node_preferred_affinity.yaml
deployment.apps/node-affinity-deploy created
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get deploy -o wide
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                  SELECTOR
node-affinity-deploy   5/5     5            5           9s    myapp-pod    registry.cn-
beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get rs -o wide
NAME                              DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES
                                                          SELECTOR
```

```
node-affinity-deploy-d5d9cbc8d   5         5         5       13s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=d5d9cbc8d
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get pod -o wide
NAME                                    READY   STATUS    RESTARTS   AGE   IP
  NODE          NOMINATED NODE   READINESS GATES
node-affinity-deploy-d5d9cbc8d-bv86t   1/1     Running   0          18s   10.244.2.243
  k8s-node02   <none>           <none>
node-affinity-deploy-d5d9cbc8d-dnbr8   1/1     Running   0          18s   10.244.2.244
  k8s-node02   <none>           <none>
node-affinity-deploy-d5d9cbc8d-ldq82   1/1     Running   0          18s   10.244.2.246
  k8s-node02   <none>           <none>
node-affinity-deploy-d5d9cbc8d-nt74q   1/1     Running   0          18s   10.244.4.2
  k8s-node01   <none>           <none>
node-affinity-deploy-d5d9cbc8d-rt5nb   1/1     Running   0          18s   10.244.2.245
  k8s-node02   <none>       <none>
```

由上可见，再根据之前打的标签，很容易推断出当前pod会【优先】调度在k8s-node02节点。

node软硬亲和性联合示例

硬亲和性与软亲和性一起使用

要运行的yaml文件

```
[root@k8s-master nodeAffinity]# pwd
/root/k8s_practice/scheduler/nodeAffinity
[root@k8s-master nodeAffinity]# cat node_affinity.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-affinity-deploy
  labels:
    app: nodeaffinity-deploy
spec:
  replicas: 5
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-pod
```

```
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              # 表示node标签存在 cpu-num且值大于10
            - matchExpressions:
              - key: cpu-num
                operator: Gt
                values:
                - "10"
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 50
            preference:
              matchExpressions:
                # 表示node标签存在 disk-type=ssd 或 disk-type=sas
              - key: disk-type
                operator: In
                values:
                - ssd
                - sas
```

运行yaml文件并查看状态

```
[root@k8s-master nodeAffinity]# kubectl apply -f node_affinity.yaml
deployment.apps/node-affinity-deploy created
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get deploy -o wide
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                 SELECTOR
node-affinity-deploy   5/5     5            5           9s    myapp-pod    registry.cn-
beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get rs -o wide
NAME                             DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES
                                                     SELECTOR
node-affinity-deploy-f9cb9b99b   5         5         5       13s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=f9cb9b99b
[root@k8s-master nodeAffinity]#
[root@k8s-master nodeAffinity]# kubectl get pod -o wide
NAME                                   READY   STATUS    RESTARTS   AGE   IP
  NODE         NOMINATED NODE   READINESS GATES
node-affinity-deploy-f9cb9b99b-8w2nc   1/1     Running   0          17s   10.244.4.10
k8s-node01   <none>           <none>
```

```
node-affinity-deploy-f9cb9b99b-csk2s   1/1     Running   0          17s   10.244.4.9
  k8s-node01   <none>           <none>
node-affinity-deploy-f9cb9b99b-g42kq   1/1     Running   0          17s   10.244.4.8
  k8s-node01   <none>           <none>
node-affinity-deploy-f9cb9b99b-m6xbv   1/1     Running   0          17s   10.244.4.7
  k8s-node01   <none>           <none>
node-affinity-deploy-f9cb9b99b-mxbdp   1/1     Running   0          17s   10.244.2.253
  k8s-node02   <none>           <none>
```

由上可见，再根据之前打的标签，很容易推断出k8s-node01、k8s-node02都满足必要条件，但当前pod会【优先】调度在k8s-node01节点。

Pod亲和力和反亲和力

文档：https://kubernetes.io/zh/docs/tasks/configure-pod-container/assign-pods-nodes-using-node-affinity/

与节点亲和性一样，当前有Pod亲和性/反亲和性都有两种类型，称为 requiredDuringSchedulingIgnoredDuringExecution和 preferredDuringSchedulingIgnoredDuringExecution，分别表示“硬”与“软”要求。对于硬要求，如果不满足则pod会一直处于Pending状态。

Pod的亲和性与反亲和性是基于Node节点上已经运行pod的标签(而不是节点上的标签)决定的，从而约束哪些节点适合调度你的pod。

规则的形式是：如果X已经运行了一个或多个符合规则Y的pod，则此pod应该在X中运行(如果是反亲和的情况下，则不应该在X中运行）。当然pod必须处在同一名称空间，不然亲和性/反亲和性无作用。从概念上讲，X是一个拓扑域。我们可以使用topologyKey来表示它，topologyKey 的值是node节点标签的键以便系统用来表示这样的拓扑域。当然这里也有个隐藏条件，就是node节点标签的键值相同时，才是在同一拓扑域中；如果只是节点标签名相同，但是值不同，那么也不在同一拓扑域。★★★★★

也就是说：Pod的亲和性/反亲和性调度是根据拓扑域来界定调度的，而不是根据node节点。★★★★★

注意事项
1、pod之间亲和性/反亲和性需要大量的处理，这会明显降低大型集群中的调度速度。不建议在大于几百个节点的集群中使用它们。

2、Pod反亲和性要求对节点进行一致的标记。换句话说，集群中的每个节点都必须有一个匹配topologyKey的适当标签。如果某些或所有节点缺少指定的topologyKey标签，可能会导致意外行为。

requiredDuringSchedulingIgnoredDuringExecution中亲和性的一个示例是“将服务A和服务B的Pod放置在同一区域【拓扑域】中，因为它们之间有很多交流”；preferredDuringSchedulingIgnoredDuringExecution中反亲和性的示例是“将此服务的 pod 跨区域【拓扑域】分布”【此时硬性要求是说不通的，因为你可能拥有的 pod 数多于区域数】。

Pod亲和性/反亲和性语法支持以下运算符：In，NotIn，Exists，DoesNotExist。

原则上，topologyKey可以是任何合法的标签键。但是，出于性能和安全方面的原因，topologyKey有一些限制：

1、对于Pod亲和性，在requiredDuringSchedulingIgnoredDuringExecution和 preferredDuringSchedulingIgnoredDuringExecution中topologyKey都不允许为空。

2、对于Pod反亲和性，在requiredDuringSchedulingIgnoredDuringExecution和preferredDuringSchedulingIgnoredDuringExecution中topologyKey也都不允许为空。

3、对于requiredDuringSchedulingIgnoredDuringExecution的pod反亲和性，引入了允许控制器LimitPodHardAntiAffinityTopology来限制topologyKey的kubernet.io/hostname。如果你想让它对自定义拓扑可用，你可以修改许可控制器，或者干脆禁用它。

4、除上述情况外，topologyKey可以是任何合法的标签键。

Pod 间亲和通过 PodSpec 中 affinity 字段下的 podAffinity 字段进行指定。而 pod 间反亲和通过 PodSpec 中 affinity 字段下的 podAntiAffinity 字段进行指定。

Pod亲和性/反亲和性的requiredDuringSchedulingIgnoredDuringExecution所关联的matchExpressions下有多个key列表，那么只有当所有key满足时，才能将pod调度到某个区域【针对Pod硬亲和】。

准备事项

给node节点打label标签

```
# 删除已存在标签
kubectl label nodes k8s-node01 cpu-num-
kubectl label nodes k8s-node01 disk-type-
kubectl label nodes k8s-node02 cpu-num-
kubectl label nodes k8s-node02 disk-type-


###  --overwrite覆盖已存在的标签信息
# k8s-master 标签添加
kubectl label nodes k8s-master busi-use=www --overwrite
kubectl label nodes k8s-master disk-type=ssd --overwrite
kubectl label nodes k8s-master busi-db=redis

# k8s-node01 标签添加
kubectl label nodes k8s-node01 busi-use=www
kubectl label nodes k8s-node01 disk-type=sata
kubectl label nodes k8s-node01 busi-db=redis

# k8s-node02 标签添加
kubectl label nodes k8s-node02 busi-use=www
kubectl label nodes k8s-node02 disk-type=ssd
kubectl label nodes k8s-node02 busi-db=etcd
```

查询所有节点标签信息

```
[root@k8s-master ~]# kubectl get node -o wide --show-labels
NAME         STATUS   ROLES    AGE   VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE
             KERNEL-VERSION           CONTAINER-RUNTIME   LABELS
k8s-master   Ready    master   28d   v1.17.4   172.16.1.110   <none>        CentOS
Linux 7 (Core)   3.10.0-1062.el7.x86_64   docker://19.3.8
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,busi-db=redis,busi-
use=www,disk-type=ssd,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-
master,kubernetes.io/os=linux,node-role.kubernetes.io/master=
k8s-node01   Ready    <none>   28d   v1.17.4   172.16.1.111   <none>        CentOS
Linux 7 (Core)   3.10.0-1062.el7.x86_64   docker://19.3.8
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,busi-db=redis,busi-
use=www,disk-type=sata,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-
node01,kubernetes.io/os=linux
k8s-node02   Ready    <none>   28d   v1.17.4   172.16.1.112   <none>        CentOS
Linux 7 (Core)   3.10.0-1062.el7.x86_64   docker://19.3.8
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,busi-db=etcd,busi-
use=www,disk-type=ssd,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-
node02,kubernetes.io/os=linux
```

如上所述：k8s-master添加了disk-type=ssd,busi-db=redis,busi-use=www标签

k8s-node01添加了disk-type=sata,busi-db=redis,busi-use=www标签

k8s-node02添加了disk-type=ssd,busi-db=etcd,busi-use=www标签

通过deployment运行一个pod，或者直接运行一个pod也可以。为后续的Pod亲和性与反亲和性测验做基础。

```
### yaml文件
[root@k8s-master podAffinity]# pwd
/root/k8s_practice/scheduler/podAffinity
[root@k8s-master podAffinity]# cat web_deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deploy
  labels:
    app: myweb-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp-web
  template:
    metadata:
      labels:
        app: myapp-web
        version: v1
    spec:
      containers:
```

```
      - name: myapp-pod
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
[root@k8s-master podAffinity]#
### 运行yaml文件
[root@k8s-master podAffinity]# kubectl apply -f web_deploy.yaml
deployment.apps/web-deploy created
[root@k8s-master podAffinity]#
### 查看pod标签
[root@k8s-master podAffinity]# kubectl get pod -o wide --show-labels
NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE
    NOMINATED NODE   READINESS GATES   LABELS
web-deploy-5ccc9d7c55-kkwst   1/1     Running   0          15m   10.244.2.4   k8s-
node02   <none>           <none>            app=myapp-web,pod-template-
hash=5ccc9d7c55,version=v1
```

当前pod在k8s-node02节点；其中pod的标签app=myapp-web,version=v1会在后面pod亲和性/反亲和性示例中使用。

pod硬亲和性示例

```
[root@k8s-master podAffinity]# pwd
/root/k8s_practice/scheduler/podAffinity
[root@k8s-master podAffinity]# cat pod_required_affinity.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-podaffinity-deploy
  labels:
    app: podaffinity-deploy
spec:
  replicas: 6
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      # 允许在master节点运行
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: myapp-pod
```

```
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              # 由于是Pod亲和性/反亲和性；因此这里匹配规则写的是Pod的标签信息
              matchExpressions:
              - key: app
                operator: In
                values:
                - myapp-web
            # 拓扑域  若多个node节点具有相同的标签信息【标签键值相同】，则表示这些node节点就在同一拓扑域
            # 请对比如下两个不同的拓扑域，Pod的调度结果
            #topologyKey: busi-use
            topologyKey: disk-type
```

运行yaml文件并查看状态

```
[root@k8s-master podAffinity]# kubectl apply -f pod_required_affinity.yaml
deployment.apps/pod-podaffinity-deploy created
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get deploy -o wide
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                SELECTOR
pod-podaffinity-deploy   6/6     6            6           48s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
web-deploy               1/1     1            1           22h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get rs -o wide
NAME                                DESIRED   CURRENT   READY   AGE   CONTAINERS
IMAGES                                                    SELECTOR
pod-podaffinity-deploy-848559bf5b   6         6         6       52s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=848559bf5b
web-deploy-5ccc9d7c55               1         1         1       22h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web,pod-template-
hash=5ccc9d7c55
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get pod -o wide
NAME                                     READY   STATUS    RESTARTS   AGE   IP
    NODE         NOMINATED NODE   READINESS GATES
pod-podaffinity-deploy-848559bf5b-8kkwm   1/1     Running   0          54s
10.244.0.80    k8s-master   <none>           <none>
```

```
pod-podaffinity-deploy-848559bf5b-8s59f   1/1     Running   0          54s
10.244.2.252   k8s-node02   <none>           <none>
pod-podaffinity-deploy-848559bf5b-8z4dv   1/1     Running   0          54s
10.244.2.253   k8s-node02   <none>           <none>
pod-podaffinity-deploy-848559bf5b-gs7sb   1/1     Running   0          54s
10.244.0.79    k8s-master   <none>           <none>
pod-podaffinity-deploy-848559bf5b-sm6nz   1/1     Running   0          54s
10.244.0.78    k8s-master   <none>           <none>
pod-podaffinity-deploy-848559bf5b-zbr6v   1/1     Running   0          54s
10.244.2.251   k8s-node02   <none>           <none>
web-deploy-5ccc9d7c55-khhrr               1/1     Running   3          22h
10.244.2.245   k8s-node02   <none>           <none>
```

由上可见，yaml文件中为topologyKey: disk-type；虽然k8s-master、k8s-node01、k8s-node02都有disk-type标签；但是k8s-master和k8s-node02节点的disk-type标签值为ssd；而k8s-node01节点的disk-type标签值为sata。因此k8s-master和k8s-node02节点属于同一拓扑域，Pod只会调度到这两个节点上。

pod软亲和性示例

```
[root@k8s-master podAffinity]# pwd
/root/k8s_practice/scheduler/podAffinity
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# cat pod_preferred_affinity.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-podaffinity-deploy
  labels:
    app: podaffinity-deploy
spec:
  replicas: 6
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      # 允许在master节点运行
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: myapp-pod
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
```

```
        - containerPort: 80
      affinity:
        podAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                # 由于是Pod亲和性/反亲和性；因此这里匹配规则写的是Pod的标签信息
                matchExpressions:
                - key: version
                  operator: In
                  values:
                  - v1
                  - v2
              # 拓扑域  若多个node节点具有相同的标签信息【标签键值相同】，则表示这些node节点就在同一拓扑域
              topologyKey: disk-type
```

运行yaml文件并查看状态

```
[root@k8s-master podAffinity]# kubectl apply -f pod_preferred_affinity.yaml
deployment.apps/pod-podaffinity-deploy created
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get deploy -o wide
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                SELECTOR
pod-podaffinity-deploy   6/6     6            6           75s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
web-deploy               1/1     1            1           25h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get rs -o wide
NAME                                DESIRED   CURRENT   READY   AGE   CONTAINERS
IMAGES                                                      SELECTOR
pod-podaffinity-deploy-8474b4b586   6         6         6       79s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=8474b4b586
web-deploy-5ccc9d7c55               1         1         1       25h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web,pod-template-
hash=5ccc9d7c55
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get pod -o wide
NAME                                     READY   STATUS    RESTARTS   AGE   IP
    NODE         NOMINATED NODE   READINESS GATES
pod-podaffinity-deploy-8474b4b586-57gxh   1/1     Running   0          83s   10.244.2.4
    k8s-node02   <none>           <none>
pod-podaffinity-deploy-8474b4b586-kd5l4   1/1     Running   0          83s   10.244.2.3
    k8s-node02   <none>           <none>
```

```
pod-podaffinity-deploy-8474b4b586-mlvv7   1/1     Running   0          83s
10.244.0.84    k8s-master   <none>           <none>
pod-podaffinity-deploy-8474b4b586-mtk6r   1/1     Running   0          83s
10.244.0.86    k8s-master   <none>           <none>
pod-podaffinity-deploy-8474b4b586-n5jpj   1/1     Running   0          83s
10.244.0.85    k8s-master   <none>           <none>
pod-podaffinity-deploy-8474b4b586-q2xdl   1/1     Running   0          83s
10.244.3.22    k8s-node01   <none>           <none>
web-deploy-5ccc9d7c55-khhrr               1/1     Running   3          25h
10.244.2.245   k8s-node02   <none>           <none>
```

由上可见，再根据k8s-master、k8s-node01、k8s-node02的标签信息；很容易推断出Pod会优先调度到k8s-master、k8s-node02节点。

pod硬反亲和性示例

```
[root@k8s-master podAffinity]# pwd
/root/k8s_practice/scheduler/podAffinity
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# cat pod_required_AntiAffinity.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-podantiaffinity-deploy
  labels:
    app: podantiaffinity-deploy
spec:
  replicas: 6
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      # 允许在master节点运行
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: myapp-pod
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
```

```
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              # 由于是Pod亲和性/反亲和性；因此这里匹配规则写的是Pod的标签信息
              matchExpressions:
              - key: app
                operator: In
                values:
                - myapp-web
            # 拓扑域  若多个node节点具有相同的标签信息【标签键值相同】，则表示这些node节点就在同一拓扑域
            topologyKey: disk-type
```

```
[root@k8s-master podAffinity]# kubectl apply -f pod_required_AntiAffinity.yaml
deployment.apps/pod-podantiaffinity-deploy created
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get deploy -o wide
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                                SELECTOR
pod-podantiaffinity-deploy    6/6     6            6           68s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
web-deploy                    1/1     1            1           25h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get rs -o wide
NAME                                     DESIRED   CURRENT   READY   AGE   CONTAINERS
IMAGES                                                       SELECTOR
pod-podantiaffinity-deploy-5fb4764b6b    6         6         6       72s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=5fb4764b6b
web-deploy-5ccc9d7c55                    1         1         1       25h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web,pod-template-
hash=5ccc9d7c55
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get pod -o wide
NAME                                          READY   STATUS    RESTARTS   AGE   IP
        NODE          NOMINATED NODE   READINESS GATES
pod-podantiaffinity-deploy-5fb4764b6b-b5bzd   1/1     Running   0          75s
10.244.3.28    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-5fb4764b6b-b6qjg   1/1     Running   0          75s
10.244.3.23    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-5fb4764b6b-h262g   1/1     Running   0          75s
10.244.3.27    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-5fb4764b6b-q98gt   1/1     Running   0          75s
10.244.3.24    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-5fb4764b6b-v6kpm   1/1     Running   0          75s
10.244.3.25    k8s-node01   <none>           <none>
```

```
pod-podantiaffinity-deploy-5fb4764b6b-wtmm6   1/1     Running   0          75s
10.244.3.26    k8s-node01   <none>           <none>
web-deploy-5ccc9d7c55-khhrr                   1/1     Running   3          25h
10.244.2.245   k8s-node02   <none>           <none>
```

由上可见，由于是Pod反亲和测验，再根据k8s-master、k8s-node01、k8s-node02的标签信息；很容易推断出Pod只能调度到k8s-node01节点。

pod软反亲和性示例

```
[root@k8s-master podAffinity]# pwd
/root/k8s_practice/scheduler/podAffinity
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# cat pod_preferred_AntiAffinity.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-podantiaffinity-deploy
  labels:
    app: podantiaffinity-deploy
spec:
  replicas: 6
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      # 允许在master节点运行
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: myapp-pod
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                # 由于是Pod亲和性/反亲和性；因此这里匹配规则写的是Pod的标签信息
                matchExpressions:
```

```
                - key: version
                  operator: In
                  values:
                  - v1
                  - v2
              # 拓扑域  若多个node节点具有相同的标签信息【标签键值相同】，则表示这些node节点就在同一拓扑域
              topologyKey: disk-type
```

```
[root@k8s-master podAffinity]# kubectl apply -f pod_preferred_AntiAffinity.yaml
deployment.apps/pod-podantiaffinity-deploy created
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get deploy -o wide
NAME                         READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                                                SELECTOR
pod-podantiaffinity-deploy   6/6     6            6           9s    myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
web-deploy                   1/1     1            1           26h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get rs -o wide
NAME                                    DESIRED   CURRENT   READY   AGE   CONTAINERS
IMAGES                                                      SELECTOR
pod-podantiaffinity-deploy-54d758ddb4   6         6         6       13s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=54d758ddb4
web-deploy-5ccc9d7c55                   1         1         1       26h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web,pod-template-
hash=5ccc9d7c55
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get pod -o wide
NAME                                          READY   STATUS    RESTARTS   AGE   IP
        NODE         NOMINATED NODE   READINESS GATES
pod-podantiaffinity-deploy-54d758ddb4-58t9p   1/1     Running   0          17s
10.244.3.31    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-54d758ddb4-9ntd7   1/1     Running   0          17s
10.244.3.32    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-54d758ddb4-9wr6p   1/1     Running   0          17s
10.244.2.5     k8s-node02   <none>           <none>
pod-podantiaffinity-deploy-54d758ddb4-gnls4   1/1     Running   0          17s
10.244.3.30    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-54d758ddb4-jlftn   1/1     Running   0          17s
10.244.3.29    k8s-node01   <none>           <none>
pod-podantiaffinity-deploy-54d758ddb4-mvplv   1/1     Running   0          17s
10.244.0.87    k8s-master   <none>           <none>
web-deploy-5ccc9d7c55-khhrr                   1/1     Running   3          26h
10.244.2.245   k8s-node02   <none>           <none>
```

由上可见，由于是Pod反亲和测验，再根据k8s-master、k8s-node01、k8s-node02的标签信息；很容易推断出Pod会优先调度到k8s-node01节点。

pod亲和性与反亲和性联合示例

```
[root@k8s-master podAffinity]# pwd
/root/k8s_practice/scheduler/podAffinity
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# cat pod_podAffinity_all.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-podaffinity-all-deploy
  labels:
    app: podaffinity-all-deploy
spec:
  replicas: 6
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      # 允许在master节点运行
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: myapp-pod
        image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              # 由于是Pod亲和性/反亲和性；因此这里匹配规则写的是Pod的标签信息
              matchExpressions:
              - key: app
                operator: In
                values:
                - myapp-web
            # 拓扑域  若多个node节点具有相同的标签信息【标签键值相同】，则表示这些node节点就在同一拓扑域
            topologyKey: disk-type
```

```
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: version
                  operator: In
                  values:
                  - v1
                  - v2
              topologyKey: busi-db
```

```
[root@k8s-master podAffinity]# kubectl apply -f pod_podAffinity_all.yaml
deployment.apps/pod-podaffinity-all-deploy created
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get deploy -o wide
NAME                         READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
                                                    SELECTOR
pod-podaffinity-all-deploy   6/6     6            1           5s    myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp
web-deploy                   1/1     1            1           28h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get rs -o wide
NAME                                    DESIRED   CURRENT   READY   AGE   CONTAINERS
IMAGES                                                      SELECTOR
pod-podaffinity-all-deploy-5ddbf9cbf8   6         6         6       10s   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp,pod-template-
hash=5ddbf9cbf8
web-deploy-5ccc9d7c55                   1         1         1       28h   myapp-pod
registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1   app=myapp-web,pod-template-
hash=5ccc9d7c55
[root@k8s-master podAffinity]#
[root@k8s-master podAffinity]# kubectl get pod -o wide
NAME                                          READY   STATUS    RESTARTS   AGE   IP
        NODE         NOMINATED NODE   READINESS GATES
pod-podaffinity-all-deploy-5ddbf9cbf8-5w5b7   1/1     Running   0          15s
10.244.0.91    k8s-master   <none>           <none>
pod-podaffinity-all-deploy-5ddbf9cbf8-j57g9   1/1     Running   0          15s
10.244.0.90    k8s-master   <none>           <none>
pod-podaffinity-all-deploy-5ddbf9cbf8-kwz6w   1/1     Running   0          15s
10.244.0.92    k8s-master   <none>           <none>
pod-podaffinity-all-deploy-5ddbf9cbf8-l8spj   1/1     Running   0          15s
10.244.2.6     k8s-node02   <none>           <none>
pod-podaffinity-all-deploy-5ddbf9cbf8-lf22c   1/1     Running   0          15s
10.244.0.89    k8s-master   <none>           <none>
```

```
pod-podaffinity-all-deploy-5ddbf9cbf8-r2fgl   1/1     Running   0          15s
10.244.0.88    k8s-master   <none>           <none>
web-deploy-5ccc9d7c55-khhrr                   1/1     Running   3          28h
10.244.2.245   k8s-node02   <none>           <none>
```

由上可见，根据k8s-master、k8s-node01、k8s-node02的标签信息；很容易推断出Pod只能调度到k8s-master、k8s-node02节点，且会优先调度到k8s-master节点。

Topology拓扑域

什么是topologyKey

顾名思义，topology 就是 拓扑 的意思，这里指的是一个 拓扑域，是指一个范围的概念，比如一个 Node、一个机柜、一个机房或者是一个地区（如杭州、上海）等，实际上对应的还是 Node 上的标签。这里的 topologyKey 对应的是 Node 上的标签的 Key（没有Value），可以看出，其实 topologyKey 就是用于筛选 Node 的。通过这种方式，我们就可以将各个 Pod 进行跨集群、跨机房、跨地区的调度了。

如何使用topologyKey

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: failure-domain.beta.kubernetes.io/zone
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey: kubernetes.io/hostname
  containers:
```

```
  - name: with-pod-affinity
    image: k8s.gcr.io/pause:2.0
```

这里 Pod 的亲和性规则是：这个 Pod 要调度到的 Node 必须有一个标签为 security: S1 的 Pod，且该 Node 必须有一个 Key 为 failure-domain.beta.kubernetes.io/zone 的 标签，即 Node 必须属于 failure-domain.beta.kubernetes.io/zone 拓扑域。

Pod 的反亲和性规则是：这个 Pod 尽量不要调度到这样的 Node，其包含一个 Key 为 kubernetes.io/hostname 的标签，且该 Node 上有标签为 security: S2 的 Pod。

topologyKey详解

既然 topologyKey 是拓扑域，那 Pod 之间怎样才是属于同一个拓扑域?

如果使用 k8s.io/hostname，则表示拓扑域为 Node 范围，那么 k8s.io/hostname 对应的值不一样就是不同的拓扑域。比如 Pod1 在 k8s.io/hostname=node1 的 Node 上，Pod2 在 k8s.io/hostname=node2 的 Node 上，Pod3 在 k8s.io/hostname=node1 的 Node 上，则 Pod2 和 Pod1、Pod3 不在同一个拓扑域，而Pod1 和 Pod3在同一个拓扑域。

如果使用 failure-domain.k8s.io/zone ，则表示拓扑域为一个区域。同样，Node 的标签 failure-domain.k8s.io/zone 对应的值不一样也不是同一个拓扑域，比如 Pod1 在 failure-domain.k8s.io/zone=beijing 的 Node 上，Pod2 在 failure-domain.k8s.io/zone=hangzhou 的 Node 上，则 Pod1 和 Pod2 不属于同一个拓扑域。

当然，topologyKey 也可以使用自定义标签。比如可以给一组 Node 打上标签 custom_topology，那么拓扑域就是针对这个标签了，则该标签相同的 Node 上的 Pod 属于同一个拓扑域。

注意事项

原则上，topologyKey 可以是任何合法的标签 Key。但是出于性能和安全原因，对 topologyKey 有一些限制：

- 对于亲和性和 requiredDuringSchedulingIgnoredDuringExecution 的 Pod 反亲和性，topologyKey 不能为空。
- 对于 requiredDuringSchedulingIgnoredDuringExecution 的 Pod 反亲和性，引入 LimitPodHardAntiAffinityTopology 准入控制器来限制 topologyKey 只能是 kubernetes.io/hostname。如果要使用自定义拓扑域，则可以修改准入控制器，或者直接禁用它。
- 对于 preferredDuringSchedulingIgnoredDuringExecution 的 Pod 反亲和性，空的 topologyKey 表示所有拓扑域。截止 v1.12 版本，所有拓扑域还只能是 kubernetes.io/hostname、failure-domain.beta.kubernetes.io/zone 和 failure-domain.beta.kubernetes.io/region 的组合。
- 除上述情况外，topologyKey 可以是任何合法的标签 key。

临时容器

概念和配置

什么是临时容器

临时容器：一种特殊的容器，该容器在现有 Pod 中临时运行，以便完成用户发起的操作，例如故障排查。 你会使用临时容器来检查服务，而不是用它来构建应用程序。

临时容器与其他容器的不同之处在于，它们缺少对资源或执行的保证，并且永远不会自动重启， 因此不适用于构建应用程序。 临时容器使用与常规容器相同的 `ContainerSpec` 节来描述，但许多字段是不兼容和不允许的。

- 临时容器没有端口配置，因此像 `ports`， `livenessProbe`， `readinessProbe` 这样的字段是不允许的。
- Pod 资源分配是不可变的，因此 `resources` 配置是不允许的。
- 有关允许字段的完整列表，请参见 EphemeralContainer 参考文档。

临时容器是使用 API 中的一种特殊的 `ephemeralcontainers` 处理器进行创建的， 而不是直接添加到 `pod.spec` 段，因此无法使用 `kubectl edit` 来添加一个临时容器。

与常规容器一样，将临时容器添加到 Pod 后，将不能更改或删除临时容器。

临时容器是使用 Pod 的 `ephemeralcontainers` 子资源创建的，可以使用 `kubectl --raw` 命令进行显示。 首先描述临时容器被添加为一个 `EphemeralContainers` 列表：

```
{
    "apiVersion": "v1",
    "kind": "EphemeralContainers",
    "metadata": {
        "name": "example-pod"
    },
    "ephemeralContainers": [{
        "command": [
            "sh"
        ],
        "image": "busybox",
        "imagePullPolicy": "IfNotPresent",
        "name": "debugger",
        "stdin": true,
        "tty": true,
        "terminationMessagePolicy": "File"
    }]
}
```

使用如下命令更新已运行的临时容器 `example-pod`：

```
kubectl replace --raw /api/v1/namespaces/default/pods/example-pod/ephemeralcontainers
 -f ec.json
```

这将返回临时容器的新列表：

```
{
```

```
    "kind":"EphemeralContainers",
    "apiVersion":"v1",
    "metadata":{
        "name":"example-pod",
        "namespace":"default",
        "selfLink":"/api/v1/namespaces/default/pods/example-pod/ephemeralcontainers",
        "uid":"a14a6d9b-62f2-4119-9d8e-e2ed6bc3a47c",
        "resourceVersion":"15886",
        "creationTimestamp":"2019-08-29T06:41:42Z"
    },
    "ephemeralContainers":[
        {
            "name":"debugger",
            "image":"busybox",
            "command":[
                "sh"
            ],
            "resources":{

            },
            "terminationMessagePolicy":"File",
            "imagePullPolicy":"IfNotPresent",
            "stdin":true,
            "tty":true
        }
    ]
}
```

可以使用以下命令查看新创建的临时容器的状态：

```
kubectl describe pod example-pod
```

输出为：

```
...
Ephemeral Containers:
  debugger:
    Container ID:
docker://cf81908f149e7e9213d3c3644eda55c72efaff67652a2685c1146f0ce151e80f
    Image:          busybox
    Image ID:       docker-
pullable://busybox@sha256:9f1003c480699be56815db0f8146ad2e22efea85129b5b5983d0e0fb52d9a
b70
    Port:           <none>
    Host Port:      <none>
    Command:
      sh
    State:          Running
```

```
      Started:      Thu, 29 Aug 2019 06:42:21 +0000
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:         <none>
...
```

可以使用以下命令连接到新的临时容器：

```
kubectl attach -it example-pod -c debugger
```