CloudNative Lives

Kubernetes管理员实训

Kubernetes调度管理实训

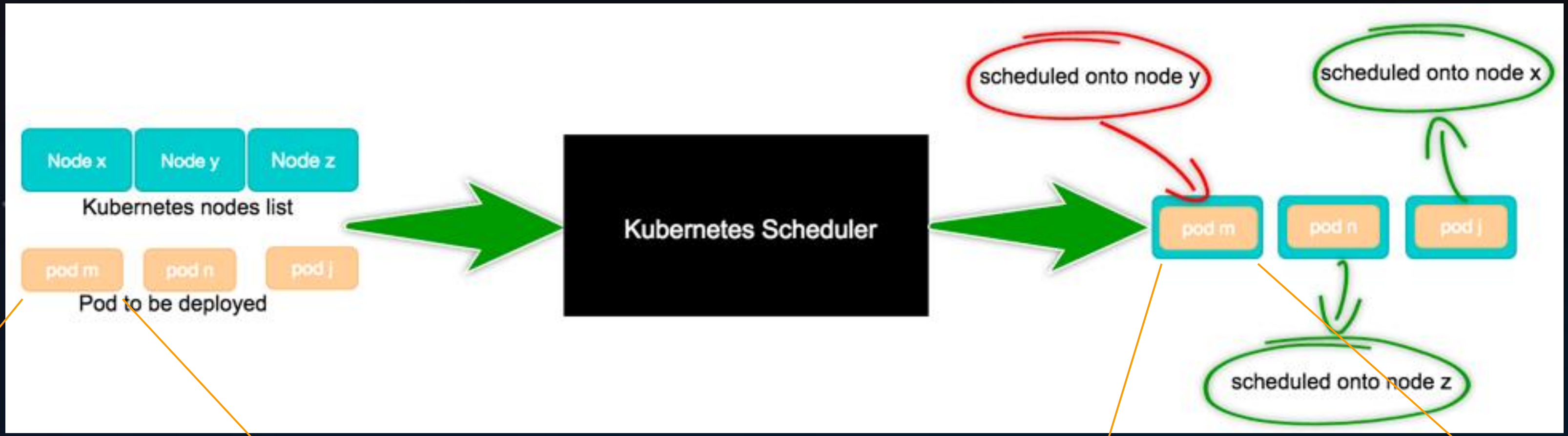
华为云容器团队核心架构师 & CNCF社区主要贡献者倾力打造

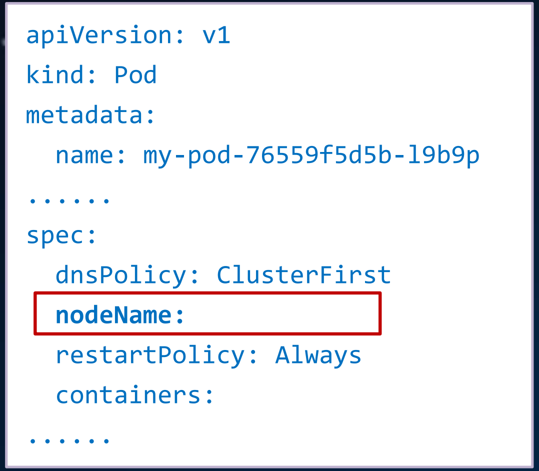
大 纲

- 理解资源限制对Pod调度的影响
- 使用label selector调度Pod
- 手动调度Pod
- 理解DaemonSet
- 调度失败原因分析
- 使用多调度器
- 了解调度器的配置

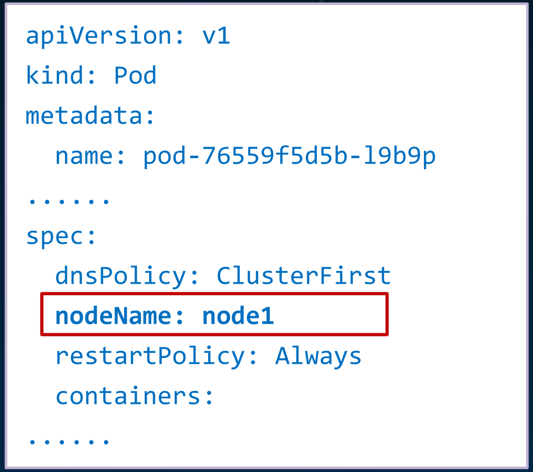
Kubernetes 调度相关基础概念

Scheduling：为Pod找到一个合适的Node





```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-76559f5d5b-l9b9p
......
spec:
  dnsPolicy: ClusterFirst
  nodeName:
  restartPolicy: Always
  containers:
......
```



```
apiVersion: v1
kind: Pod
metadata:
  name: pod-76559f5d5b-l9b9p
......
spec:
  dnsPolicy: ClusterFirst
  nodeName: node1
  restartPolicy: Always
  containers:
......
```

Node 定义

```
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node-n1
  name: node-n1
spec:
  externalID: node-n1
status:
  addresses:
  - address: 10.162.197.135
    type: InternalIP
  allocatable:
    cpu: "8"
    memory: 16309412Ki
    pods: "110"
  capacity:
    cpu: "8"
    memory: 16411812Ki
    pods: "110"
  conditions: {...}
  daemonEndpoints:
    kubeletEndpoint:
      Port: 10250
  images: {...}
  nodeInfo: {...}
```

执行 `kubectl get node <node-name> -o yaml` 查看一个完整的node

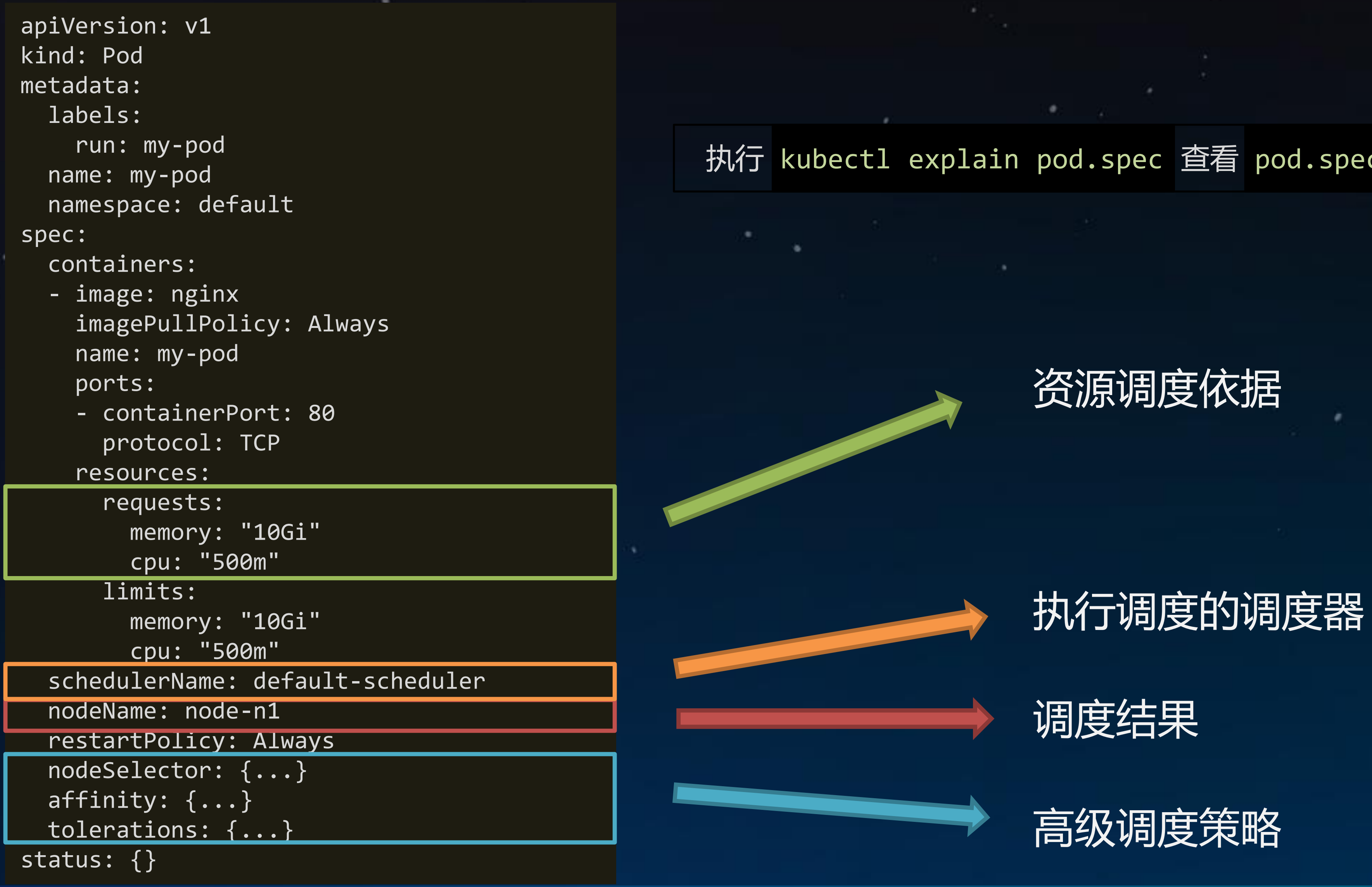
→ 一个node的可分配资源量

Pod 定义

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      requests:
        memory: "10Gi"
        cpu: "500m"
      limits:
        memory: "10Gi"
        cpu: "500m"
  schedulerName: default-scheduler
  nodeName: node-n1
  restartPolicy: Always
  nodeSelector: {...}
  affinity: {...}
  tolerations: {...}
status: {}
```

执行 `kubectl explain pod.spec` 查看 `pod.spec` 提供的完整配置字段

Pod 中影响调度的主要属性字段



```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      requests:
        memory: "10Gi"
        cpu: "500m"
      limits:
        memory: "10Gi"
        cpu: "500m"
  schedulerName: default-scheduler
  nodeName: node-n1
  restartPolicy: Always
  nodeSelector: {...}
  affinity: {...}
  tolerations: {...}
status: {}
```

执行 `kubectl explain pod.spec` 查看 `pod.spec` 提供的完整配置字段

资源调度依据

执行调度的调度器

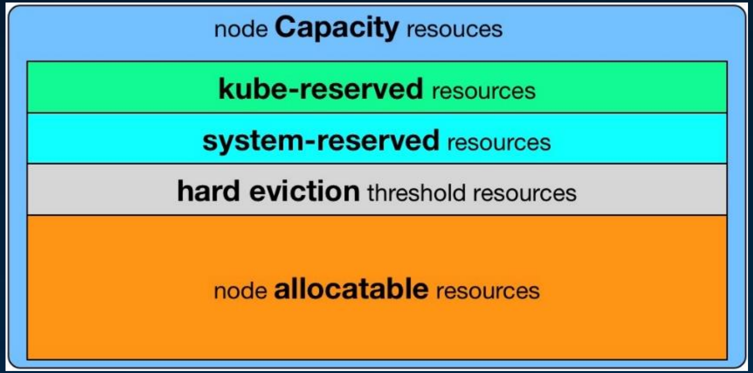
调度结果

高级调度策略

Kubernetes 中的资源分配

K8S 调度器的资源分配机制

- 基于Pod中容器request资源“总和”调度
  - resoureces.limits影响pod的运行资源上限，不影响调度
  - initContainer取最大值，container取累加值，最后取大者
    即 Max(Max(initContainers.requests), Sum(containers.requests))
  - 未指定request资源时， 按0资源需求进行调度
- 基于资源声明量的调度，而非实际占用
  - 不依赖监控，系统不会过于敏感
  - 能否调度成功：pod.request < node.allocatable - node.requested
- Kubernetes node 资源的盒子模型



node **Capacity** resouces

kube-reserved resources

system-reserved resources

hard eviction threshold resources

node **allocatable** resources

- 资源分配相关算法
  - GeneralPredicates（主要是PodFitsResources）
  - LeastRequestedPriority
  - BalancedResourceAllocation，平衡cpu/mem的消耗比例

Pod 所需资源的计算

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  initContainers:
  - name: ic1
    resources:
      requests:
        cpu: "1"
        memory: "1G"
  - name: ic2
    resources:
      requests:
        cpu: "1"
        memory: "3G"
  containers:
  - name: container1
    resources:
      requests:
        cpu: "500m"
        memory: "1G"
  - name: container2
    resources:
      requests:
        cpu: "500m"
        memory: "1G"
```

InitContainers：

逐个运行并退出，之后才拉起containers

资源需求取单个容器的最大值

Containers：

同时运行，资源需求为所有容器累加

最终结果：cpu 1，memory 3G

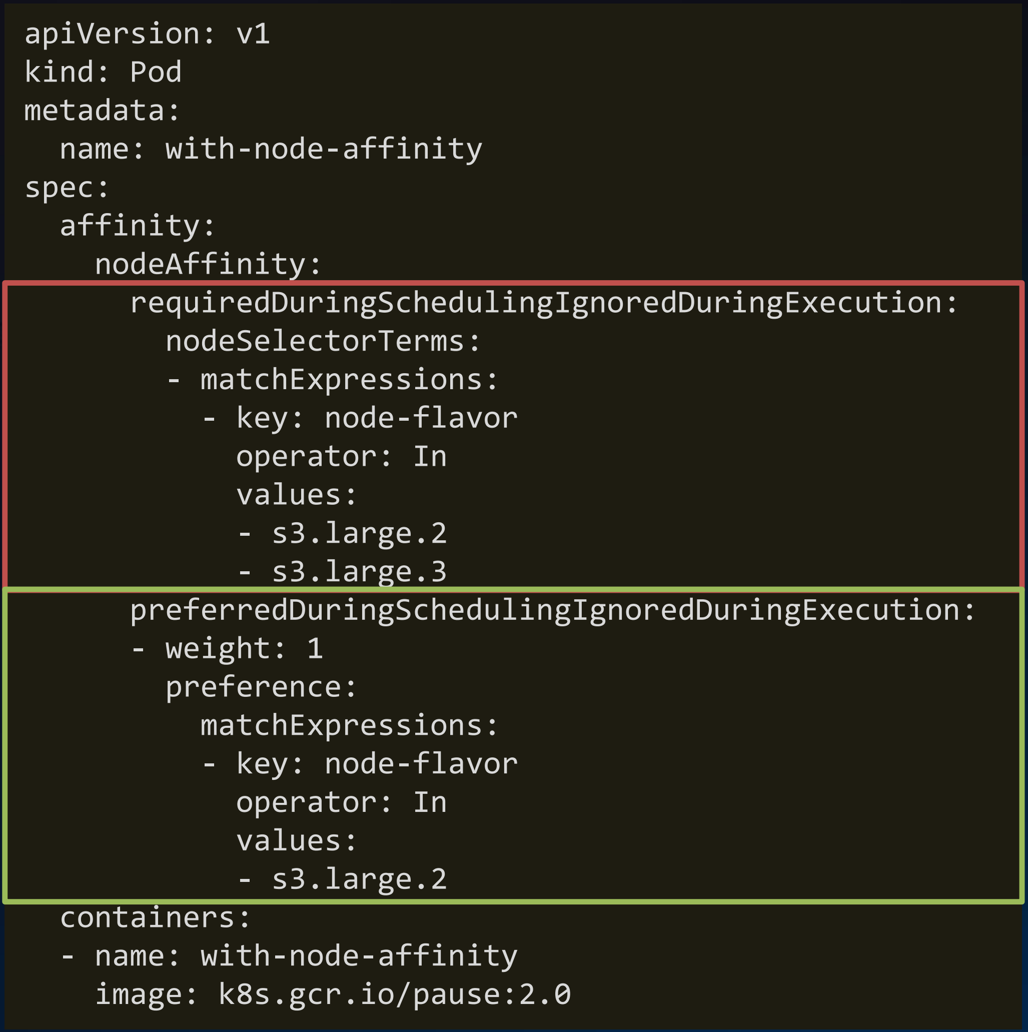
Kubernetes 中的高级调度及用法

nodeSelector：将 Pod 调度到特定的 Node 上

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    pod-template-hash: "4173307778"
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
    resources: {}
  nodeSelector:
    disktype: ssd
    node-flavor: s3.large.2
```

- 语法格式：map[string]string
- 作用：
  - 匹配node.labels
  - 排除不包含nodeSelector中指定label的所有node
  - 匹配机制 —— 完全匹配

nodeAffinity：nodeSelector 升级版



```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: node-flavor
            operator: In
            values:
            - s3.large.2
            - s3.large.3
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: node-flavor
            operator: In
            values:
            - s3.large.2
  containers:
  - name: with-node-affinity
    image: k8s.gcr.io/pause:2.0
```

- 与nodeSelector关键差异
  - 引入运算符：In，NotIn（labelselector语法）
  - 支持枚举label可能的取值，如 zone in [az1, az2, az3...]
  - 支持硬性过滤和软性评分
  - 硬性过滤规则支持指定 多条件之间的逻辑或运算
  - 软性评分规则支持 设置条件权重值

硬性过滤：

排除不具备指定label的node

软性评分：

不具备指定label的node打低分，

降低node被选中的几率

podAffinity：让某些 Pod 分布在同一组 Node 上

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: kubernetes.io/zone
      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
```

- 与nodeAffinity的关键差异
  - 定义在PodSpec中，亲和与反亲和规则具有对称性
  - labelSelector的匹配对象为Pod
  - 对node分组，依据label-key = topologyKey，每个label-value取值为一组
  - 硬性过滤规则，条件间只有逻辑与运算

硬性过滤：

排除不具备指定pod的node组

软性评分：

不具备指定pod的node组打低分，降低该组node被选中的几率

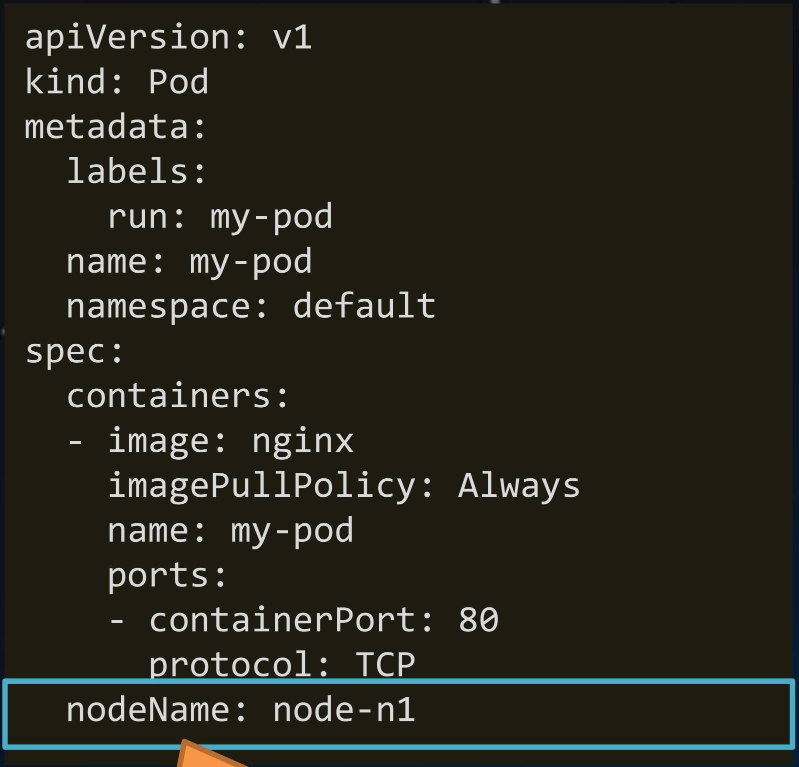
podAntiAffinity：避免某些 Pod 分布在同一组 Node 上

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: kubernetes.io/zone
      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
```

- 与podAffinity的差异
  - 匹配过程相同
  - 最终处理调度结果时取反
- 即
  - podAffinity中可调度节点，在podAntiAffinity中为不可调度
  - podAffinity中高分节点，在podAntiAffinity中为低分

手动调度和DaemonSet

手动调度Pod（不经过调度器）



```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
  nodeName: node-n1
```

创建Pod时直接指定nodeName

- 适用场景：
  - 调度器不工作时，临时救急
  - 封装实现自定义调度器
- 小故事：
  - 过去几个版本的Daemonset都是由controller直接指定pod的运行节点，不经过调度器。
  - 直到1.11版本，DaemonSet的pod由scheduler调度才作为alpha特性引入

DaemonSet：每个节点来一份

- 每个node上部署一个相同的pod
- 通常用来部署集群中的agent，如果网络插件

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-daemonset
spec:
  selector:
    matchLabels:
      name: my-daemonset
  template:
    metadata:
      labels:
        name: my-daemonset
    spec:
      containers:
      - name: container
        image: k8s.gcr.io/pause:2.0
```

等价于

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
spec:
  replicas: <# of nodes>
  selector:
    matchLabels:
      podlabel: daemonset
  teplate:
    metadata:
      labels:
        podlabel: daemonset
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: podlabel
                operator: In
                values:
                - daemonset
            topologyKey: kubernetes.io/hostname
      containers:
      - name: container
        image: k8s.gcr.io/pause:2.0
```

Taints：避免 Pod 调度到特定 Node 上

```yaml
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node-n1
  name: node-n1
spec:
  externalID: node-n1
  taints:
  - effect: NoSchedule
    key: accelerator
    timeAdded: null
    value: gpu
status: {...}
```

- 带effect的特殊label，对Pod有排斥性
  - 硬性排斥 NoSchedule
  - 软性排斥 PreferNoSchedule
- 系统创建的taint附带时间戳
  - effect为NoExecute
  - 便于触发对Pod的超时驱逐
- 典型用法：预留特殊节点做特殊用途

给node添加taint

```
kubectl taint node node-n1 foo=bar:NoSchedule
```

删除taint

```
kubectl taint node node-n1 foo:NoSchedule-
```

Tolerations：允许 Pod 调度到有特定 taints 的 Node 上

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - name: my-pod
    image: nginx
  tolerations:
  - key: accelerator
    operator: Equal
    value: gpu
    effect: NoSchedule
```

无视排斥

```
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node-n1
  name: node-n1
spec:
  externalID: node-n1
  taints:
  - effect: NoSchedule
    key: accelerator
    timeAdded: null
    value: gpu
status: {...}
```

- 完全匹配
  - 例：<key>=<value>:<effect>
- 匹配任意taint value
  - Operator为Exists，value为空
  - 例： <key>:<effect>

- 匹配任意 taint effect
  - effect为空
  - 例： <key>=<value>

注：<key>=<value>:<effect>为 kubectl describe pod中的写法

调度结果和失败原因分析

调度失败原因分析

- 查看调度结果

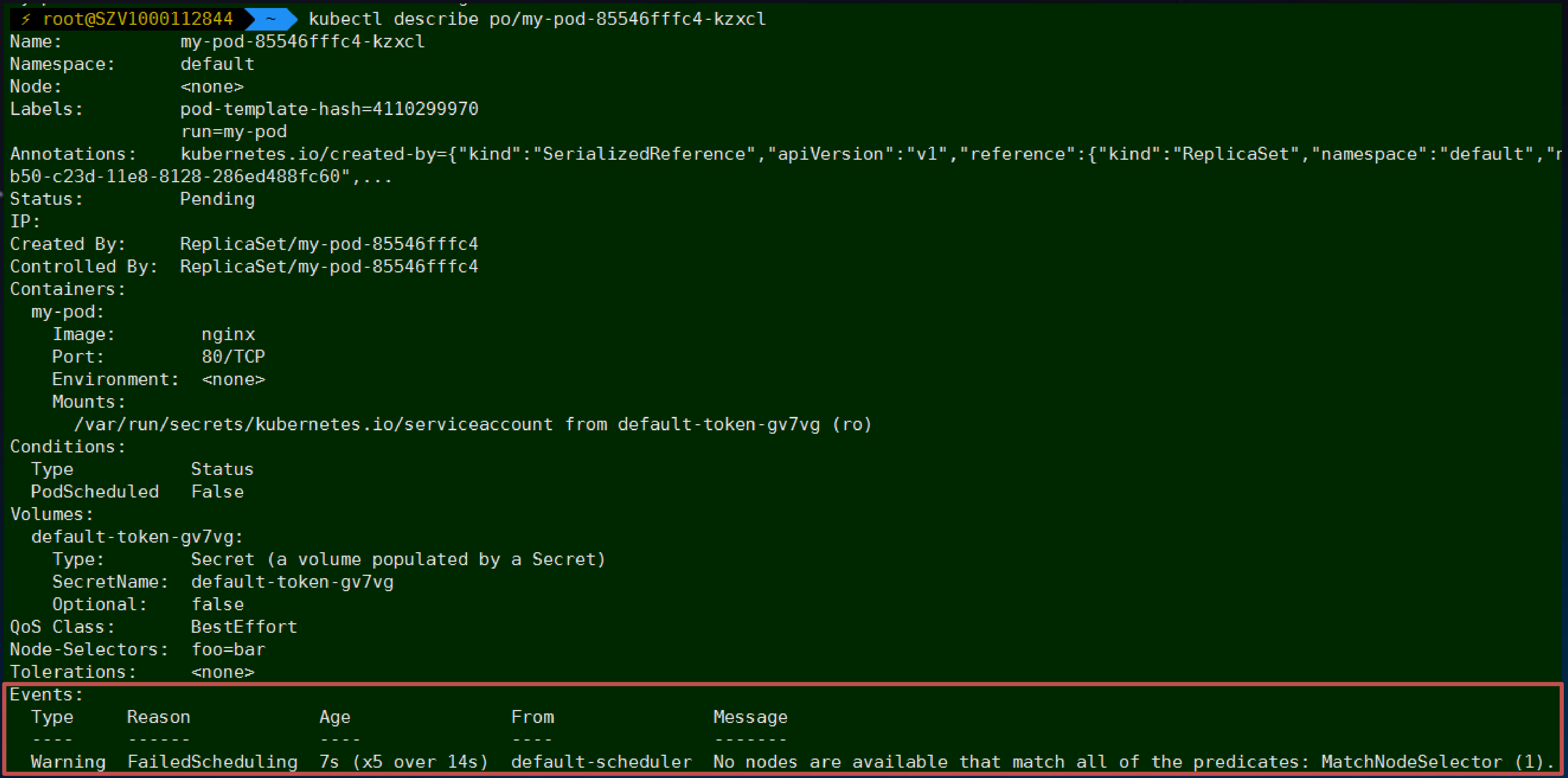
```
kubectl get pod [podname] -o wide
```

- 查看调度失败原因

```
kubectl describe pod [podname]
```

- 调度失败错误列表（kubernetes 1.9版本）
  - https://github.com/kubernetes/kubernetes/blob/release-1.9/plugin/pkg/scheduler/algorithm/predicates/error.go#L25-L58

调度失败原因分析



```
⚡ root@SZV1000112844  ~  kubectl describe po/my-pod-85546fffc4-kzxcl
Name:           my-pod-85546fffc4-kzxcl
Namespace:      default
Node:           <none>
Labels:         pod-template-hash=4110299970
                run=my-pod
Annotations:    kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind":"ReplicaSet","namespace":"default","r
b50-c23d-11e8-8128-286ed488fc60",...
Status:         Pending
IP:
Created By:     ReplicaSet/my-pod-85546fffc4
Controlled By:  ReplicaSet/my-pod-85546fffc4
Containers:
  my-pod:
    Image:        nginx
    Port:         80/TCP
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-gv7vg (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  default-token-gv7vg:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-gv7vg
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  foo=bar
Tolerations:     <none>
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  7s (x5 over 14s)   default-scheduler  No nodes are available that match all of the predicates: MatchNodeSelector (1).
```

多调度器及调度器配置

多调度器

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
  schedulerName: my-custom-scheduler
```

使用scheduleName指定调度器

- 适用场景：
  - 集群中存在多个调度器，分别处理不同类型的作业调度
- 使用限制：
  - 建议对node做资源池划分，避免调度结果写入冲突

自定义调度器配置

--policy-config-file自定义调度器加载的算法，或者调整排序算法权重

```
{
  "kind" : "Policy",
  "apiVersion" : "v1",
  "predicates" : [
    {"name" : "PodFitsHostPorts"},
    {"name" : "PodFitsResources"},
    {"name" : "NoDiskConflict"},
    {"name" : "NoVolumeZoneConflict"},
    {"name" : "MatchNodeSelector"},
    {"name" : "HostName"}
  ],
  "priorities" : [
    {"name" : "LeastRequestedPriority", "weight" : 1},
    {"name" : "BalancedResourceAllocation", "weight" : 1},
    {"name" : "ServiceSpreadingPriority", "weight" : 1},
    {"name" : "EqualPriority", "weight" : 1}
  ],
  "hardPodAffinitySymmetricWeight" : 10,
  "alwaysCheckAllPredicates" : false
}
```

执行 `kube-scheduler --help` 查看更多调度器配置项

课后作业

课后作业

1. 通过命令行，使用nginx镜像创建一个pod并手动调度到集群中的一个节点。
   - Pod的名称为<hwcka-002-你的华为云id>
   - 将所用命令、创建的Pod完整yaml截图上传

2. 通过命令行，创建两个个deployment。
   - 需要集群中有2个节点
   - 第1个deployment名称为<hwcka-002-app1-你的华为云id>，使用nginx镜像，用有2个pod，并配置该deployment自身的pod之间在节点级别反亲和
   - 第2个deployment名称为<hwcka-002-app2-你的华为云id>，使用nginx镜像，用有2个pod，并配置该deployment的pod与第1个deployment的pod在节点级别亲和
   - 将所用命令、创建的deployment完整yaml截图上传

- 作业完成后，提交到论坛，包括完整的浏览器截图、华为云账号，作业中所创建的集群、应用名称要带hwcka前缀
- **提交作业且答对的前50名，可获得满100减50的优惠券一张**

HUAWEI

Thank You

直播 每周四 晚20:00

CloudNativeLives

Kubernetes Docker Istio