阿里云 × CLOUD NATIVE COMPUTING FOUNDATION

云原生技术公开课

第 18 讲

Kubernetes调度和资源管理

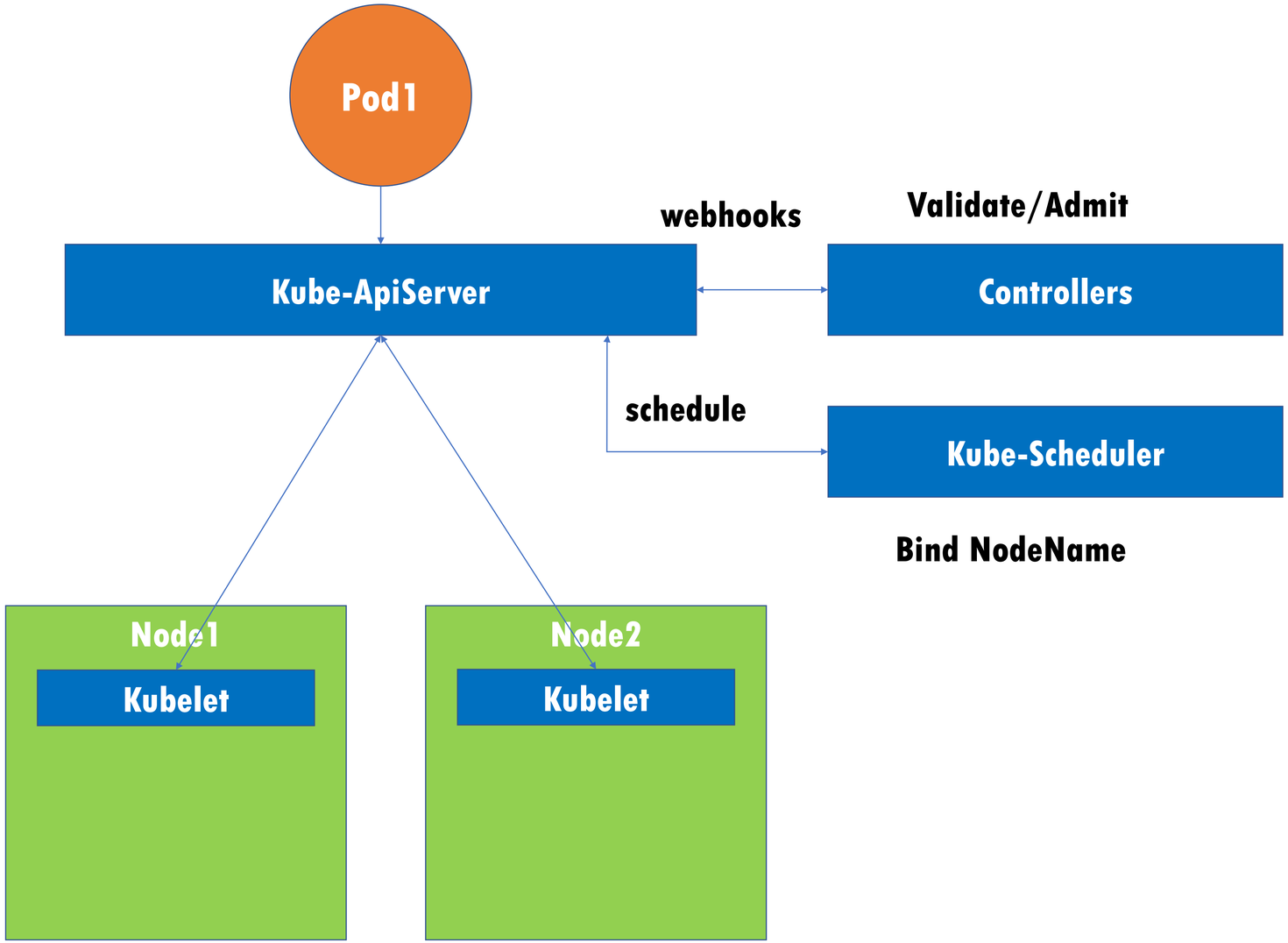
子誉 蚂蚁金服高级技术专家

关注“阿里巴巴云原生”公众号
获取第一手技术资料

1 Kubernetes 调度过程

目录

Kubernetes 调度过程



调度过程 = 把**Pod**放到合适**Node**上去

什么是合适?

- 满足**Pod**资源要求
- 满足**Pod**的特殊关系要求
- 满足**Node**限制条件要求
- 做到集群资源合理利用

2 Kubernetes 基础调度能力

Kubernetes 基础调度能力

- 资源调度 – 满足Pod资源要求
  - Resources： CPU/Memory/Storage/GPU/FGPA
  - QoS： Guaranteed/Burstable/BestEffort
  - Resource Quota
- 关系调度 – 满足Pod/Node的特殊关系/条件要求
  - PodAffinity/PodAntiAffinity： Pod和Pod间关系
  - NodeSelector/NodeAffinity： 由Pod决定适合自己的Node
  - Taint/Tolerations： 限制调度到某些Node

如何满足Pod资源要求？ -资源调度用法

- **Pod的资源类型**
  - **cpu**
  - **memory**
  - **ephemeral-storage**
  - **extended-resource: nvidia.com/gpu**
- 怎么用?
  - **Container resources**
  - **request/limit 最小/最大**
  - **Cpu: 1=1000m**
  - **mem/storage: 1Gi=1024Mi**
  - **extended-resource: 整数**

```
apiVersion: v1
kind: Pod
metadata:
  namespace: demo-ns
  name: demo-pod
spec:
  containers:
  - image: nginx:latest
    name: demo-container
    resources:
      requests:
        cpu: 2
        memory: 1Gi
      limits:
        cpu: 2
        memory: 1Gi
```

如何满足Pod资源要求？ - 资源QoS类型

- 为什么会有**request**和**limit**?
- 什么是**Pod QoS**?
  - **Quality of Service**
- 三类**QoS**
  - **Guaranteed** – 高，保障
  - **Burstable** – 中，弹性
  - **BestEffort** – 低，尽力而为
- 隐性的**QoS Class**

```
apiVersion: v1
kind: Pod
metadata:
  namespace: demo-ns
  name: demo-pod
spec:
  containers:
  - image: nginx:latest
    name: demo-container
    resources:
      requests:
        cpu: 2
        memory: 1Gi
      limits:
        cpu: 2
        memory: 1Gi
status:
  qosClass: Guaranteed
```

如何满足Pod资源要求？ - 资源QoS用法

怎么用?

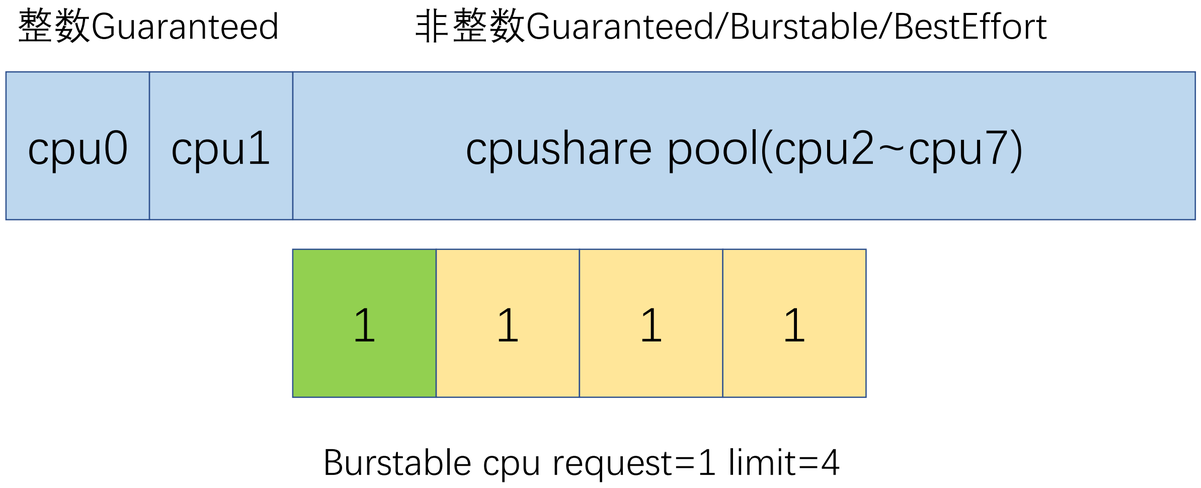
- **Guaranteed: CPU/Memory 必须request==limit，其余资源可不等**
- **Burstable: CPU/Memory request和limit不相等**
- **BestEffort: 所有资源request/limit必须都不填**

1 Guaranteed Pod:
1 Burstable Pod:
1 BestEffort Pod:

```
apiVersion: v1
apiVersion: v1
kind: Pod
metadata:
  namespace: demo-ns
  name: demo-burstable-pod
spec:
  containers:
  - image: nginx:latest
    name: demo-container
      requests:
        cpu: 1
      limits:
        cpu: 2
        memory: 1Gi
```

如何满足Pod资源要求？ - 资源和QoS

- 不同**QoS**有什么区别呢?
- 调度表现不同
  - 调度器会使用**request**进行调度
- 底层表现不同
  - **CPU** 按照**request**划分权重
    - **--cpu-manager-policy=static guaranteed**整数会绑核
  - **Memory** 按**QoS**划分**OOMScore**
    - **Guaranteed -998**
    - **Burstable 2~999**
    - **BestEffort 1000**
  - **Eviction**
    - 优先**BestEffort**
    - **Kubelet – CPUManager**



如何满足Pod资源要求？ - 资源Quota

- **ResourceQuota**
- 限制每个**Namespace**资源用量
- 用法
  - **per namespace**
  - **Scope**：
    - **Terminating/NotTerminating**
    - **BestEffort/NotBestEffort**
    - **PriorityClass**
- 当**Quota**用超过后会禁止创建：
  - **forbidden: exceeded quota**

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: demo-quota
  namespace: demo-ns
spec:
  hard:
    cpu: "1000"
    memory: 200Gi
    pods: "10”
  scopeSelector:    # 可以不填
    matchExpressions:
    - operator: Exists
      scopeName: NotBestEffort
```

限制**demo-ns Namespace**下非**BestEffort QoS**的**Quota**

- **cpu**只能使用**1000**个
- **memory**只能使用**200Gi**
- **pods**只能创建**10**个

小结：如何满足Pod资源要求？

- **Pod**要配置合理的资源要求
  - **CPU/Memory/EphemeralStorage/GPU**
- 通过**request**和**limit**来为不同业务特点的**Pod**选择不同的**QoS**
  - **Guaranteed –** 敏感型、需要保障业务
  - **Burstable –** 次敏感型、需要弹性业务
  - **BestEffort –** 可容忍型业务
- 为每个**NS**配置**ResourceQuota**来防止过量使用，保障其他人的资源可用

如何满足Pod与Pod关系要求？ – Pod亲和调度

- **Pod亲和调度 *PodAffinity***
  - 必须和某些**Pod**调度到一起

    requiredDuringSchedulingIgnoredDuringExecution
  - 优先和某些**Pod**调度到一起

    preferredDuringSchedulingIgnoredDuringExecution
- **Pod反亲和调度 *PodAntiAffinity***
  - 禁止和某些**Pod**调度到一起

    requiredDuringSchedulingIgnoredDuringExecution
  - 优先不和某些**Pod**调度到一起

    preferredDuringSchedulingIgnoredDuringExecution
- **Operator**
  - **In/NotIn/Exists/DoesNotExist**

禁止调度到带了k2=v2标签的Pod所在节点

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: demo-ns
  name: demo-pod
spec:
  containers:
  - image: nginx:latest
    name: demo-container
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: k2
              operator: In
              values:
              - v2
          topologyKey: "kubernetes.io/hostname"
```

如何满足Pod与Node关系要求？ – Node亲和调度

- ***NodeSelector***
  - 必须调度到带了某些标签的**Node**
  - **Map[string]string**
- ***NodeAffinity***
  - 必须调度到某些**Node**上

    requiredDuringSchedulingIgnoredDuringExecution
  - 优先调度到某些**Node**上

    preferredDuringSchedulingIgnoredDuringExecution
  - **Operator**
    - **In/NotIn/Exists/DoesNotExist/Gt/Lt**

优先调度到带了k2=v2标签的Node上：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: demo-ns
  name: demo-pod
spec:
  containers:
  - image: nginx:latest
    name: demo-container
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 10
        preference:
          matchExpressions:
          - key: k2
            operator: In
            values:
            - v2
```

如何限制调度到某些Node？ – Node 标记/容忍

- **Node *Taints***
  - 一个**Node**可以有多个**Taints**
  - **Effect**（不能为空）
    - **NoSchedule** – 只是禁止新**Pods**调度上来
    - **PreferNoSchedule** – 尽量不调度到这台
    - **NoExecute** – 会**evict**没有对应**toleration**的**Pods**，并且也不会调度新的上来
- **Pod *Tolerations***
  - 一个**Pod**可以有多个**Tolerations**
  - **Effect**可以为空，匹配所有
    - 取值和**Taints**的**Effect**一致
  - **Operator**
    - **Exists/Equal**

Node上带了k1=v1且效果是NoSchedule的Taints：

```
apiVersion: v1
kind: Node
metadata:
  name: demo-node
spec:
  taints:
    - key: "k1"
      value: "v1"
      effect: "NoSchedule"
```

Pod上必须带有k1=v1且效果是NoSchedule的Tolerations：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: demo-ns
  name: demo-pod
spec:
  containers:
  - image: nginx:latest
    name: demo-container
  tolerations:
  - key: "k1"
    operator: "Equal"
    value: “v1”                    # 当op=Exists可为空
    effect: "NoSchedule"           # 可以为空，匹配所有
```

小结：如何满足Pod/Node的特殊关系/条件要求？

- 给**Pod**配置和其他**Pod**的亲和/互斥关系
  - ***PodAffinity***
  - ***PodAntiAffinity***
- 给**Pod**配置和**Node**的亲和关系
  - ***NodeSelector***
  - ***NodeAffinity***
- 给**Node**配置一些限制条件标记，给**Pods**配置容忍标记
  - ***Node – Taints***
  - ***Pod - Tolerations***

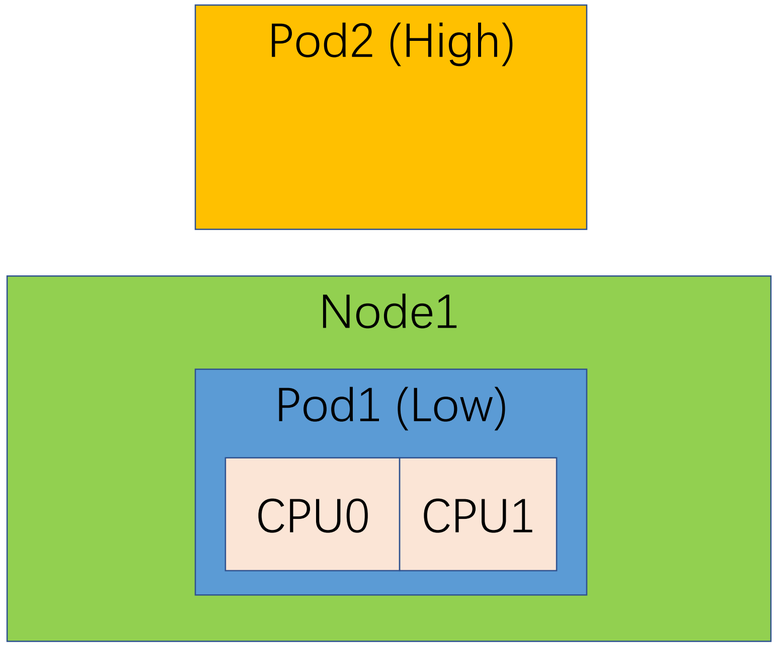
3 Kubernetes 高级调度能力

Kubernetes 高级调度能力

- 优先级调度和抢占
  - **Priority**
  - **Preemption**

如何做到集群资源合理利用？ – 优先级调度

- 集群资源不够用时候怎么办?
  - 先到先得策略（**FIFO**） – 简单、相对公平、上手快
  - 优先级策略（**Priority**） – 符合日常公司业务特点
- **PodPriority和Preemption**
  - **v1.14 – stable**
  - **PodPriority & Preemption default is ON**



如何做到集群资源合理利用？ – 优先级调度配置

- 怎么使用?
  - 创建**PriorityClass**
  - 为各个**Pod**配置上不同的**priorityClassName**
- 一些内置优先级
  - 内置默认优先级**DefaultPriorityWhenNoDefaultClassExists=0**
  - 用户可配置的最大优先级限制

    HighestUserDefinablePriority= 1000000000
  - 系统级别优先级 **SystemCriticalPriority=2000000000**
  - 内置系统级别优先级
    - **system-cluster-critical**
    - **system-node-critical**

创建一个名为**high**的**priorityClass**，得分为**10000**：

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high
value: 10000
globalDefault: false
```

创建一个名为**low**的**priorityClass**，得分为**100**：

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low
value: 100
globalDefault: false
```

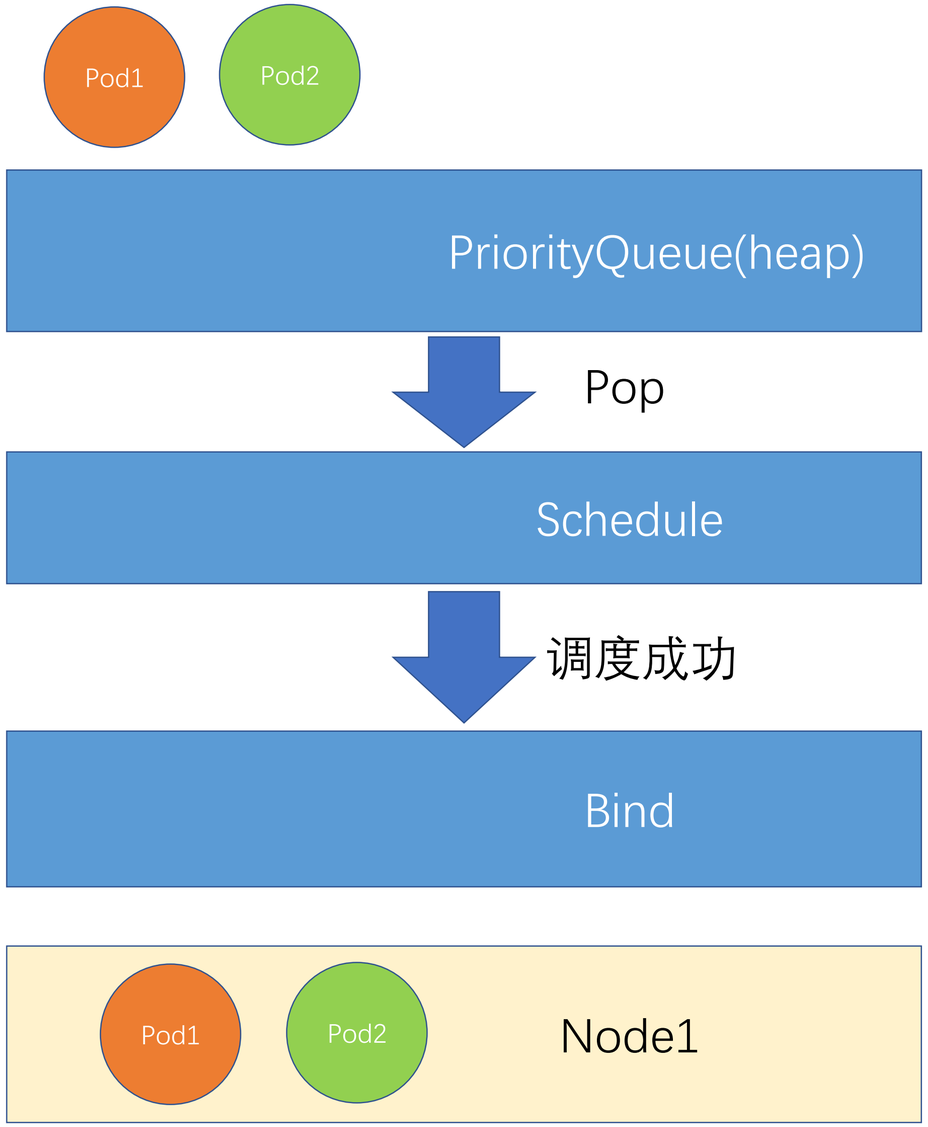
为**Pod1**配置上**high**、**Pod2**配置上**low priorityClassName**：

```
spec:
  priorityClassName: high       # 或者 low
```

如何做到集群资源合理利用？ – 优先级调度过程

优先级调度过程（发生在**Pending Pod**出队列开始调度）：

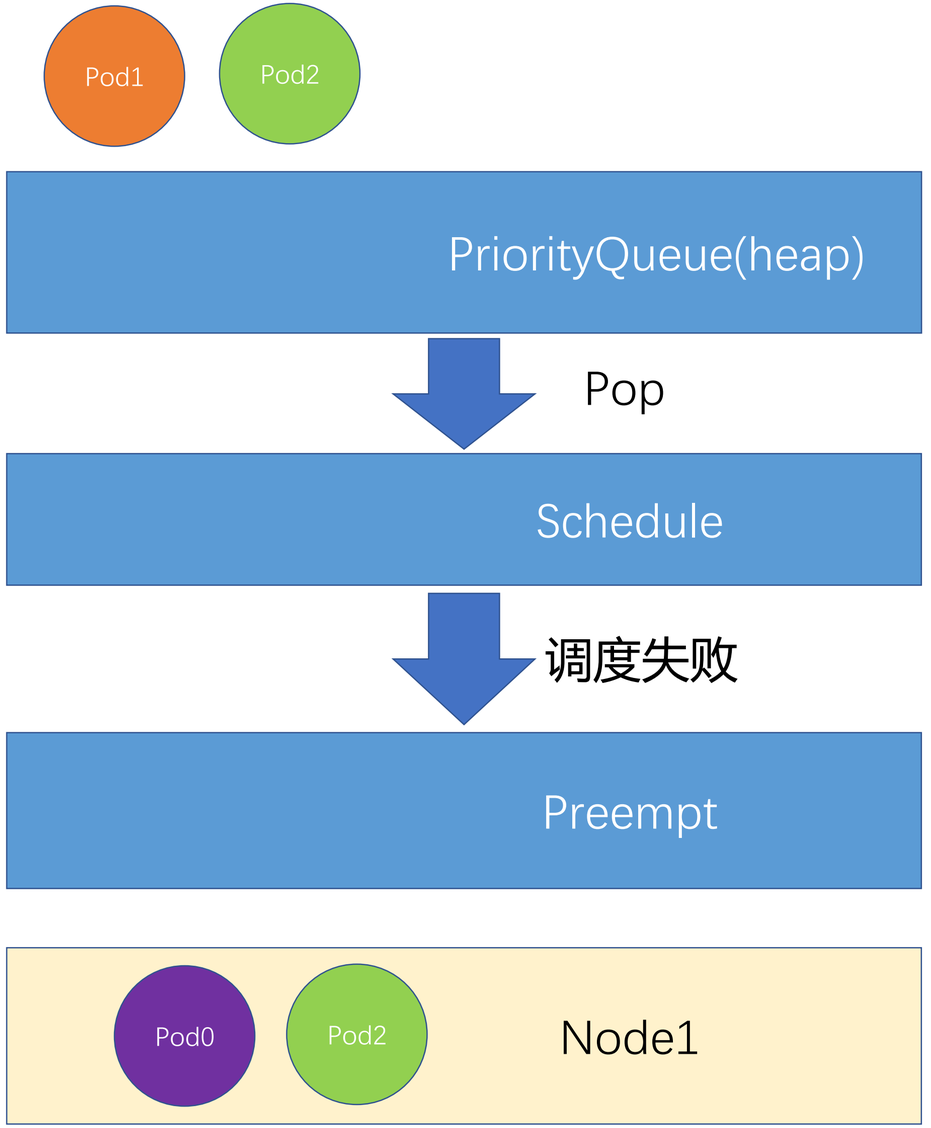
1. **Pod2**和**Pod1**先后进入调度队列，但均未被调度
2. 当进行一轮调度时，**PriorityQueue**会优先**Pop**优先级更大的**Pod1**出队列镜像调度
3. 调度成功后，对此**Pod1**进行**Bind**，并开始下一轮调度**Pod2**



如何做到集群资源合理利用？ – 优先级抢占过程

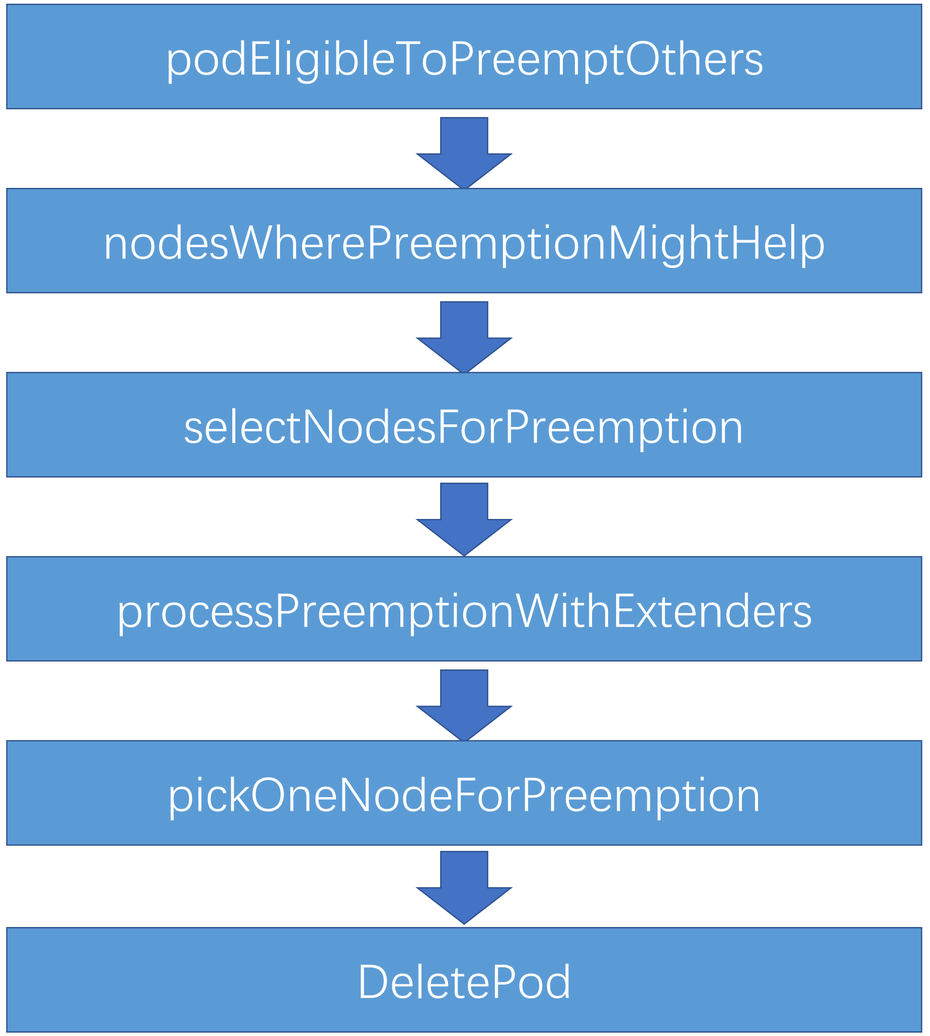
优先级抢占过程（发生在调度失败）：

1. **Pod2**先进行调度，调度成功后分配至**Node1**上运行
2. 之后**Pod1**再进行调度，由于**Node1**资源不足出现了调度失败，此时进入抢占流程
3. 在经过抢占算法计算后，选中了**Pod2**作为**Pod1**的让渡者
4. 驱逐了**Node1**上运行的**Pod2**，并将**Pod1**调度至**Node1**



如何做到集群资源合理利用？ – 优先级抢占策略

- 优先级抢占流程
- 挑选被抢占节点策略
  1. 优先选择打破**PDB**最少的节点
  2. 其次选择待抢占**Pods**中最大优先级最小的节点
  3. 再次选择待抢占**Pods**优先级加和最小的节点
  4. 接下来选择待抢占**Pods**数目最少的节点
  5. 最后选择拥有最晚启动**Pod**的节点



小结：如何做到集群资源合理利用？

- 创建自定义的一些优先级类别（**PriorityClass**）
- 给不同类型**Pods**配置不同的优先级（**PriorityClassName**）
- 通过组合不同类型**Pods**运行和优先级抢占让集群资源和调度弹性起来

关注“阿里巴巴云原生”公众号
获取第一手技术资料