阿里云 × CLOUD NATIVE COMPUTING FOUNDATION

云原生技术公开课

第 19 讲

Kubernetes调度和资源管理

木苏 阿里云工程师

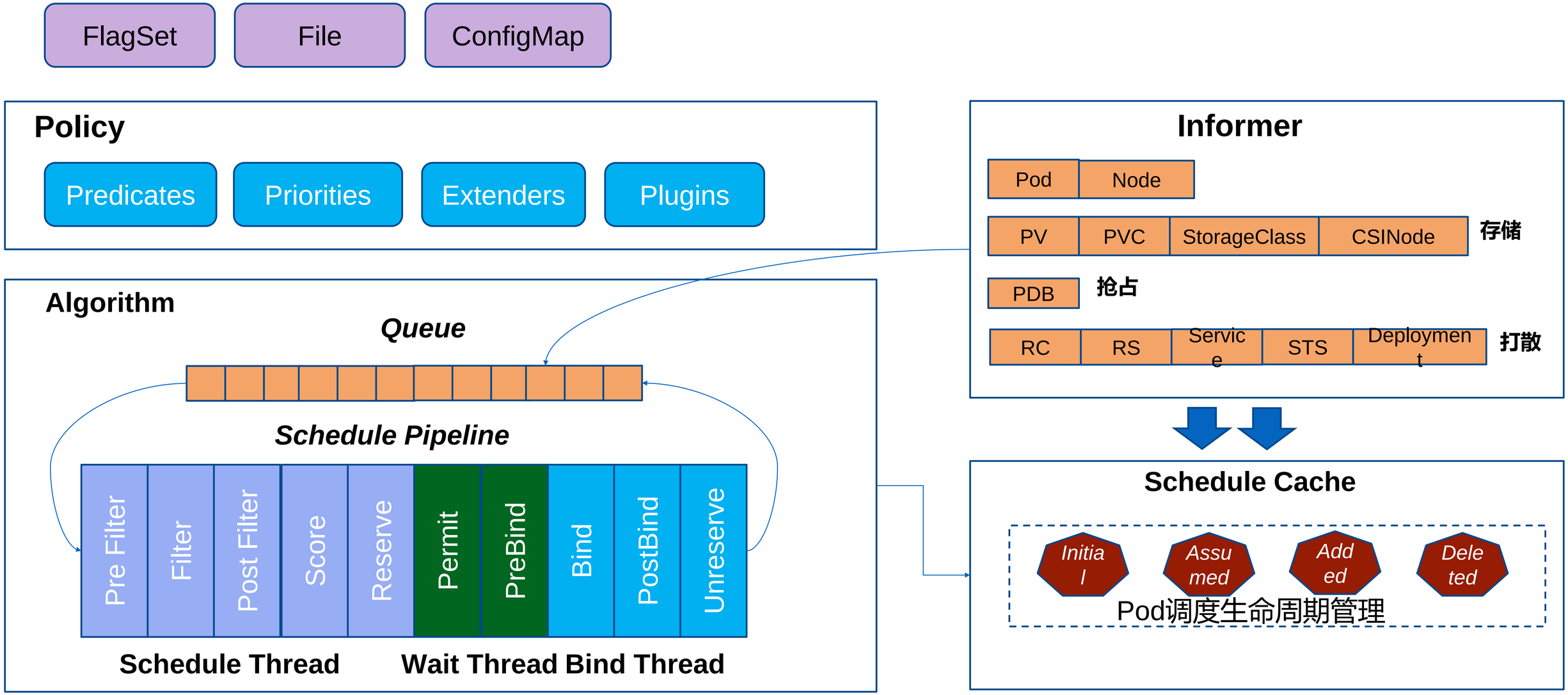
关注“阿里巴巴云原生”公众号
获取第一手技术资料

目录

1. 调度器架构
2. 调度算法实现
    - 调度流程
    - **Predicates**
    - **Priorities**
3. 如何配置调度器
4. **Scheduler Extender**
5. **Scheduler Framework**

1 调度器架构

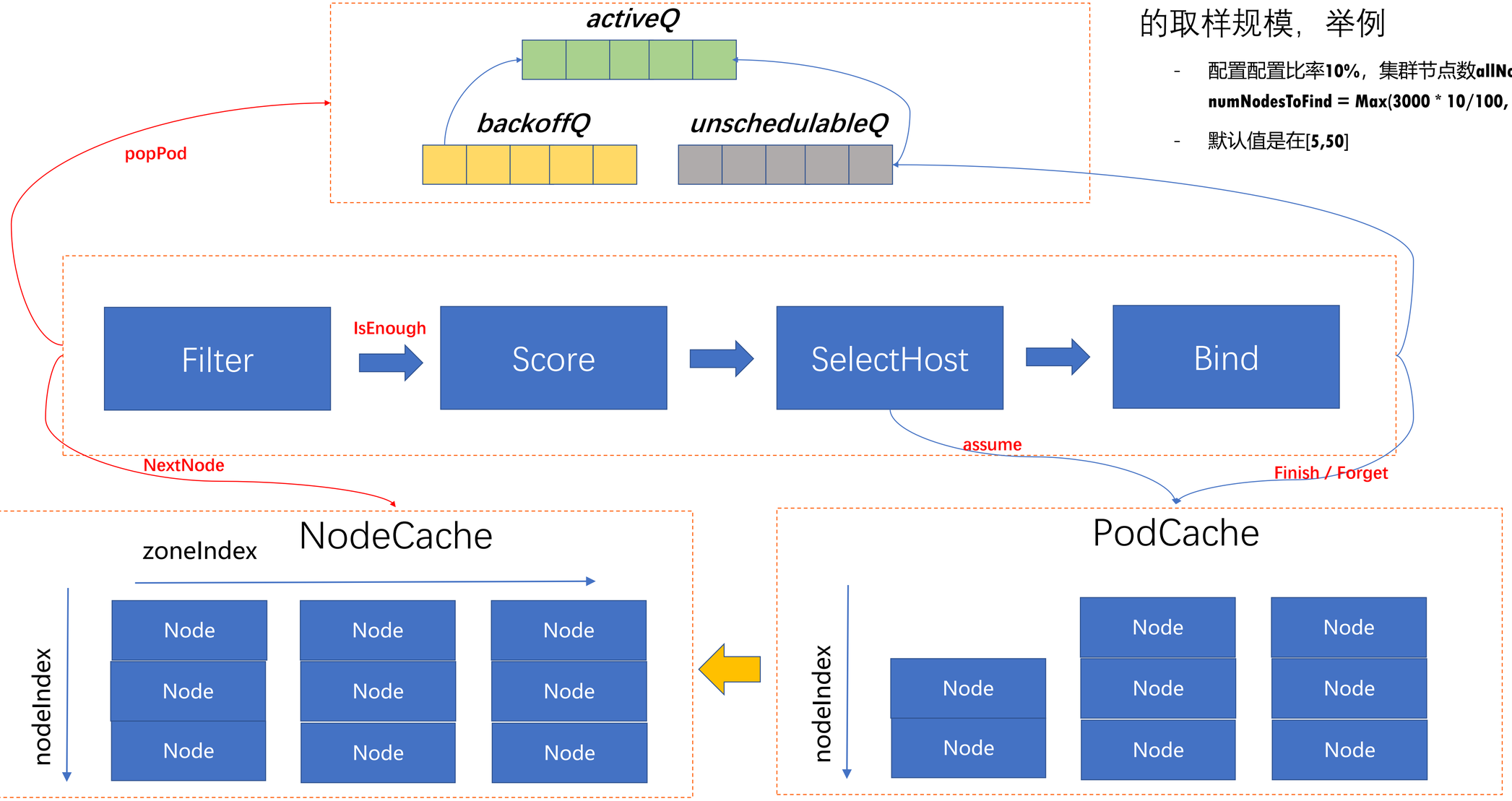
1. 调度器架构


FlagSet
File
ConfigMap
Policy
Predicates
Priorities
Extenders
Plugins
Algorithm
Queue
Schedule Pipeline
Pre Filter
Filter
Post Filter
Score
Reserve
Permit
PreBind
Bind
PostBind
Unreserve
Schedule Thread
Wait Thread Bind Thread
Informer
Pod
Node
PV
PVC
StorageClass
CSINode
存储
PDB
抢占
RC
RS
Service
STS
Deployment
打散
Schedule Cache
Initial
Assumed
Added
Deleted
Pod调度生命周期管理

2 调度算法实现

调度流程

- kube-scheduler 通过一定的策略来缩小 Node 的取样规模，举例
  - 配置配置比率**10%**，集群节点数**allNodes=3000**，那么 **numNodesToFind = Max(3000 * 10/100, 100)**，**100**是默认最小需要值
  - 默认值是在**[5,50]**



Predicates

- 存储相关
  - **NoVolumeZoneConflict:** 校验**pvc**上的可用**zone**是否和**Node**的**zone**匹配
  - **MaxCSIVolumeCountPred:** 校验**pvc**上指定的**Provision**在**CSI plugin**上报的单机最大挂盘数
  - **CheckVolumeBindingPred: pvc**和**pv**的**binding**逻辑校验
  - **NoDiskConflict: SCSI**存储不被重复**volume**
- **Pode**和**Node**匹配相关
  - **CheckNodeCondition, CheckNodeUnschedulable, PodToleratesNodeTaints, PodFitsHost, MatchNodeSelector**
- **Pod**和**Pod**匹配相关
  - **MatchInterPodAffinity:** PodAffinity和PodAntiAffinity的校验逻辑
- **Pod**打散相关
  - **EvenPodsSpread, CheckServiceAffinity**

Predicates- EvenPodsSpread

- **Pod.Spec.TopologySpreadConstraints**新增了支持一组**Pod**按照指定的**TopologyKey**进行打散的描述.
- 案例举例

```
spec:
  topologySpreadConstraint:
  - maxSkew: 1
    topologyKey: k8s.io/zone
    whenUnsatisfiable: DoNotSchedule
    selector:
      matchLabels:
        app: foo

假设pod都带有app=foo的label，集群有三个zone
+-------------+-------------+--------+
|    zone1    |    zone2    | zone3  |
+-------------+-------------+--------+
|     pod     |     pod     |        |
+-------------+-------------+--------+
计算ActualSkew = count[topo] - min(count[topo])
ActualSkew的值为(1/1/0)

假设maxSkew=1,
如果分配到zone1/zone2的话，skew的值为 2 > maxSkew(1),因此，只能分配到zone3

假设maxSkew=2,
如果分配到z1(z2),skew的值为2/1/0(1/2/0)，满足<=maxSkew,
分配到z3的话，skew的值(1/1/1)，满足<=maxSkew，因此z1/z2/z3都可以被选择
```

Priorities

- 解决的问题
  - 碎片、容灾、水位、亲和、反亲和
- **Node水位**
- **Pod打散(topo, service, controller)**
- **Node亲和&反亲和**
- **Pod亲和&反亲和**

Priorities - 资源水位

- **LeastRequestedPriority:** 优先打散
- **MostRequestedPriority**：优先堆叠
- **BalancedResourceAllocation**：碎片率
- **RequestedToCapacityRatioPriority**：指定比率的分数

- 资源的水位公式的概念
  - **Request: pod**中申请的资源数量
  - **Capacity**： **node**中的**Allocatable – sum(pod**的**request)**
- 优先打散的分数
  - **(Capacity-Request) / Capacity * Score**
- 优先堆叠的公式
  - **Request / Capcity * Score**
- 碎片率
  - **(1 - Request/Capacity) * Score**
- 额外说明：
  - 上面的所有计算方式刚好用尽的**Rquest >= Capacity**的场景都为**0**分
- 指定比率
  - 指定比例的**score**的**map**配置 **(0: x, 1:x, 2:x, ……99**： **x,100:x)**
  - **100 – (Capacity – Request) /Capacity**

Priorities – Pod打散

- 解决问题
  - 支持一类**Pod**在不同**topology**上部署的**spread**需求
- SelectorSpreadPriority
  - 在**Node**上计数
  - **TopoPods = Exists Pod**匹配**Income Pod**的**controller**的**workload**的**selector**条件
  - **(Sum(TopoPods)– TopoPods)/ Sum(TopoPods)**
- ServiceSpreadingPriority
  - 官方注释上说大概率会用来替换SelectorSpreadPriority
  - 在**Node**内计数
  - **TopoPods :** 满足**Pod**所在的**Service**的**Selector**条件
  - **(Sum(TopoPods)– TopoPods)/ Sum(TopoPods)**
- EvenPodsSpreadPriority
  - Spec指定的topologyKey
  - TopoPods=满足spec的labelSelector
  - 算分公式
    - NodeTopoPods = 按照Node级别累计TopoPods
    - MaxDif = Max(NodeTopoPods) – Min(NodeTopoPods)
    - NodeTopoPods - MaxDif / MaxDif

Priorities - Node亲和&反亲和

- 关键点
  - 以什么分组进行计数
  - **TopoPods:** 选择的**Pod**的条件
  - 算分公式
- NodeAffinityPriority
  - Node
  - TopoPods: 累计满足affnity次数
  - TopoPods /Max(TopoPods) * MaxPriority
- ServiceAntiAffinity
- NodeLabelPrioritizer
- ImageLocalityPriority: 镜像亲和调度

Priorities - Pod亲和&反亲和

- InterPodAffinityPriority
  - 如果应用A是提供数据，应用B提供服务，如果A和B部署在一起可以走本地网络，优化网络传输
  - 如果应用A和应用B之间都是CPU密集型应用，而且证明它们之间是会互相干扰的，那么可以通过这个规则设置尽量让它们不在一个节点上
- NodePreferAvoidPodsPriority
  - 针对RC(Replicas Controller)和RS(Replica Set)的在宿主机上打表对哪些RC/RS反亲和

3 如何配置调度器

如何配置调度器

- 怎么启动一个调度器
  - 默认配置(--write-config-to)
  - 配置文件(--config)
- 配置文件解释
  - **schedulerName:** 负责**Pod.SchedulerName**的调度
  - **algorithmSource:** 配置算法
  - **hardPodAffinitySymmetricWeight**： 配置**Affnity**权重
  - **percentageOfNodesToScore: filter**到的节点数跟总节点的数的比率达到这个值的时候退出**filter**
  - **bindTimeoutSeconds**： **binding**阶段的超时时间

```yaml
#默认配置文件
apiVersion: kubescheduler.config.k8s.io/v1alpha1
kind: KubeSchedulerConfiguration
# 算法配置信息
algorithmSource:
  provider: DefaultProvider
percentageOfNodesToScore: 0
schedulerName: default-scheduler
bindTimeoutSeconds: 600
clientConnection:
  acceptContentTypes: ""
  # 跟kube-apiserver交互的序列化协议
  contentType: application/vnd.kubernetes.protobuf
  qps: 50
  burst: 100
  kubeconfig: ""
disablePreemption: false
enableContentionProfiling: false
enableProfiling: false
hardPodAffinitySymmetricWeight: 1
healthzBindAddress: 0.0.0.0:10251
leaderElection:
  leaderElect: true
  leaseDuration: 15s
  lockObjectName: kube-scheduler
  lockObjectNamespace: kube-system
  renewDeadline: 10s
  resourceLock: endpoints
  resourceName: kube-scheduler
  resourceNamespace: kube-system
  retryPeriod: 2s
metricsBindAddress: 0.0.0.0:10251
```

如何配置调度器

- algorithmSource
  - provider
  - file
  - configMap

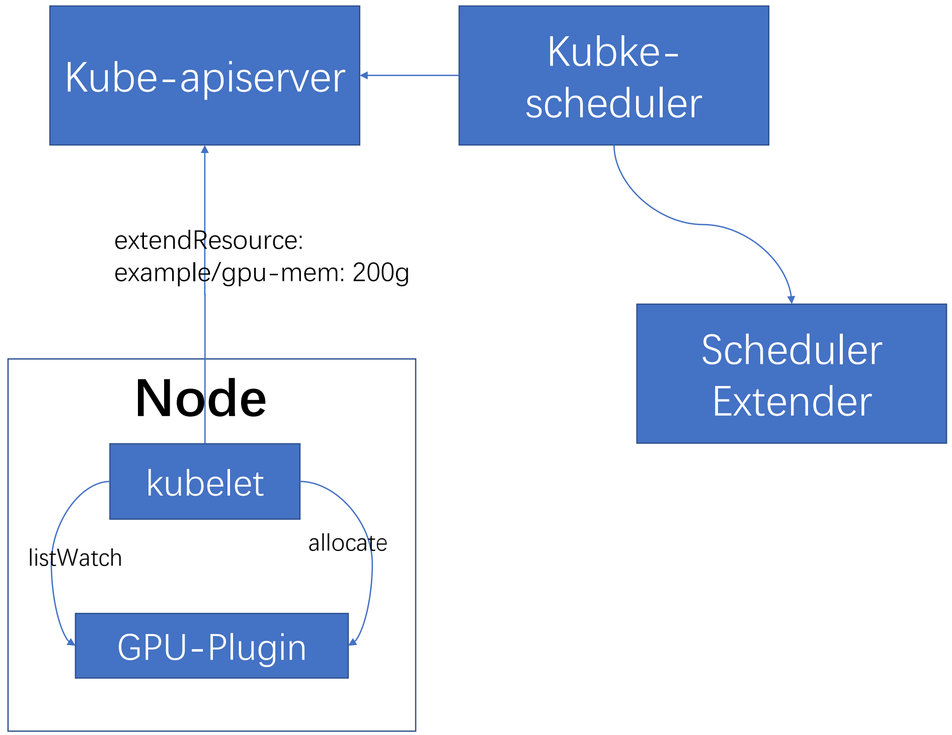
```
apiVersion: v1
kind: Policy
# 不管每个predicate的结果，所有的predidate都要走一遍
alwaysCheckAllPredicates: false
# 如果不配置表示，使用默认的predicates
# 如果未空列表，表示跳过所有配置
predicates: ['GeneralPredicates','……']
# PreferredDuringScheduling的权重值
hardPodAffinitySymmetricWeight:
# 配置打分算法插件
# 如果不配置表示，使用默认的打分插件
# 如果未空列表，表示跳过所有打分插件
priorities:
- name: LeastRequestedPriority
  weight: 1
# 用来配置scheduler extender
extenders:
```

4 Scheduler Extender

Scheduler Extender

- 能做什么?
  - 不改变原先调度的代码，直接在调度外起服务作为插件给调度器调用（类似**webhook**）
  - 支持**predicate,preempt,priority,bind**的注入
  - 一种**ExtendResource,Bind**只能一个**extender**
- 如何配置
  - 配置文件解释
- 案例
  - 申请**gpu**的显存，但是具体的卡有多少显存只有**Extender**知道，所以增加**Extender**的**Filter**

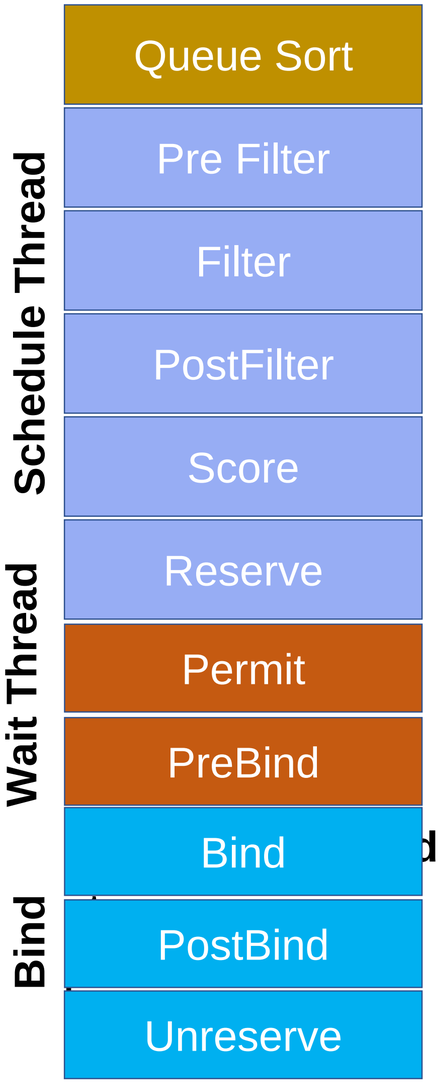
```yaml
kind: Policy
apiVersion: v1
extenders:
- urlPrefix: "http://xxxx:xxxx/scheduler-gpu-extender"
  filterVerb: filter
  weight: 1
  enableHttps: false
  # 调度器传递nodenames列表，而不是nodeinfo 列表
  nodeCacheCapable: true
  # 调用extender服务报错或者网络不可达的时候，是否可忽略extender
  ignorable: false
  managedResources:
  - name: "example/gpu-mem"
  # resourceFit阶段是否忽略这个资源的校验
    ignoredByScheduler: false
```



5 Scheduler Framework

Scheduler Framework

- 扩展点用途
- 并发模型



- **QueueSort**： 支持自定义的**Pod**的排序
- **PreFilter:** 对**Pod**的请求做预处理
- **Filter:** 自定义**filter**逻辑
- **PostFilter:** 可以用于**logs/metircs**，或者对**Score**之前做数据预处理
- **Score:** 自定义的**Score**逻辑
- **Reserve:** 有状态的**plugin**可以对资源做内存记账
- **Permit: wait, deny, approve,**可以作为**gang**的插入点
- **PreBind:** 在真正**bind node**之前，执行一些操作，例如：云盘挂载盘到**Node**上
- **Bind**： 一个**Pod**只会被一个**BindPlugin**处理
- **PostBind**： **bind**成功之后执行的逻辑
- **Unreserve:** 在**permit**到**Bind**这几个阶段只要报错就回退

Scheduler Framework

- 编写注册自定义Plugin
- 启动自定义Plugin的调度器
  - vendor
  - fork

```
# 在scheduler-plicy.config新增
#默认配置文件
apiVersion: kubescheduler.config.k8s.io/v1alpha1
kind: KubeSchedulerConfiguration
schedulerName: default-scheduler
plugins:
  bind:
    enable: [default-binder-plugin]
```

```
// Name returns default binder's name.
func (p DefaultBinder) Name() string {
    return Name
}
```

```
# 按照指定配置文件启动，如果需要自定义参数推荐这种方式
third-scheduler \
    --config=/home/admin/scheduler/conf/scheduler-plicy.config \
    --v=3
```

```
// kube-scheduler作为vendor启动调度器
func main() {
    command := scheduler.NewSchedulerCommand(
            scheduler.WithPlugin(defaultbinder.Name, defaultbinder.New),
    if err := command.Execute(); err != nil {
        fmt.Fprintf(os.Stderr, "%v\n", err)
        os.Exit(1)
    }
}
```

```
}

// New returns a default binder plugin.
func New(_ *runtime.Unknown, handle framework.FrameworkHandle) (framework.Plugin, error) {
    return &DefaultBinder{
        Client: handle.Clientset(),
    }, nil
}
```

```
nodeName string) *framework.Status {

pod.Name, UID: pod.UID},

err != nil {
```

关注“阿里巴巴云原生”公众号
获取第一手技术资料