简介

Scheduler 是 kubernetes 的调度器，主要的任务是把定义的 pod 分配到集群的节点上。听起来非常简单，但有很多要考虑的问题：

- 公平：如何保证每个节点都能被分配资源
- 资源高效利用：集群所有资源最大化被使用
- 效率：调度的性能要好，能够尽快地对大批量的 pod 完成调度工作
- 灵活：允许用户根据自己的需求控制调度的逻辑

Sheduler 是作为单独的程序运行的，启动之后会一直坚挺 API Server，获取 `PodSpec.NodeName` 为空的 pod，对每个 pod 都会创建一个 binding，表明该 pod 应该放到哪个节点上

调度过程

调度分为几个部分：首先是过滤掉不满足条件的节点，这个过程称为 `predicate`；然后对通过的节点按照优先级排序，这个是 `priority`；最后从中选择优先级最高的节点。如果中间任何一步骤有错误，就直接返回错误

Predicate 有一系列的算法可以使用：

- `PodFitsResources`：节点上剩余的资源是否大于 pod 请求的资源
- `PodFitsHost`：如果 pod 指定了 NodeName，检查节点名称是否和 NodeName 匹配
- `PodFitsHostPorts`：节点上已经使用的 port 是否和 pod 申请的 port 冲突
- `PodSelectorMatches`：过滤掉和 pod 指定的 label 不匹配的节点
- `NoDiskConflict`：已经 mount 的 volume 和 pod 指定的 volume 不冲突，除非它们都是只读

如果在 predicate 过程中没有合适的节点，pod 会一直在 `pending` 状态，不断重试调度，直到有节点满足条件。经过这个步骤，如果有多个节点满足条件，就继续 priorities 过程： 按照优先级大小对节点排序

优先级由一系列键值对组成，键是该优先级项的名称，值是它的权重（该项的重要性）。这些优先级选项包括：

- `LeastRequestedPriority`：通过计算 CPU 和 Memory 的使用率来决定权重，使用率越低权重越高。换句话说，这个优先级指标倾向于资源使用比例更低的节点
- `BalancedResourceAllocation`：节点上 CPU 和 Memory 使用率越接近，权重越高。这个应该和上面的一起使用，不应该单独使用
- `ImageLocalityPriority`：倾向于已经有要使用镜像的节点，镜像总大小值越大，权重越高

通过算法对所有的优先级项目和权重进行计算，得出最终的结果

自定义调度器

除了 kubernetes 自带的调度器，你也可以编写自己的调度器。通过 `spec:schedulername` 参数指定调度器的名字，可以为 pod 选择某个调度器进行调度。比如下面的 pod 选择 `my-scheduler` 进行调度，而不是默认的 `default-scheduler`：

```
apiVersion: v1
kind: Pod
metadata:
  name: annotation-second-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulername: my-scheduler   讲的不是很清楚
  containers:
  - name: pod-with-second-annotation-container
    image: gcr.io/google_containers/pause:2.0
```