**K8S的插件机制**

1. **综述**

集群调度管理系统在Google内部已经演进了15年，从最初的Borg，到后来的Omega，再到Docker出现以后的k8s，总体来说k8s的插件机制是非常成熟的。K8s的插件机制几乎在每个组件里都有。下面就针对各个组件，分析一下k8s的插件机制。

1. **API server的插件**
2. 为什么需要插件机制

API server的插件主要是用来做请求的认证&授权、过滤的。设计跟一些web框架的middleware一样，有使用过web 框架的同学会比较容易理解：一个请求要通过多个middleware（又先后顺序），只要有一个middleware把这个请求拦截掉，这个请求就不会成功。这样的话开发者可以根据自己的需求加一些自己的插件，比如用用来集成自己的认证&授权服务，对一些请求进行过滤，重定向等。

1. 默认的插件

* AlwaysAdmit

对所有请求开绿灯。

* AlwaysDeny

对所有请求开红灯，多用于测试环境。

* DenyExecOnPrivileged

它会拦截所有想在privileged container上执行命令的请求。

如果自己的集群支持privileged container，自己又希望限制用户在这些privileged container上执行命令，那么强烈推荐使用它。

* ServiceAccount

这个plug-in将 serviceAccounts实现了自动化，如果想要使用ServiceAccount 对象，那么强烈推荐使用它。

关于serviceAccount的描述如下：

|  |
| --- |
| 一个serviceAccount为运行在pod内的进程添加了相应的认证信息。当准入模块中开启了此插件（默认开启），那么当pod创建或修改时他会做一下事情：  如果pod没有serviceAccount属性，将这个pod的serviceAccount属性设为“default”；   1. 确保pod使用de serviceAccount始终存在； 2. 如果LimitSecretReferences 设置为true，当这个pod引用了Secret对象却没引用ServiceAccount对象，弃置这个pod； 3. 如果这个pod没有包含任何ImagePullSecrets，则serviceAccount的ImagePullSecrets被添加给这个pod； 4. 如果MountServiceAccountToken为true，则将pod中的container添加一个VolumeMount 。 |

* SecurityContextDeny

这个插件将会将使用了 SecurityContext的pod中定义的选项全部失效。

关于 SecurityContext的描述：

|  |
| --- |
| SecurityContext 在container中定义了操作系统级别的安全设定（uid, gid, capabilities, SELinux等等）。 |

* ResourceQuota

它会观察所有的请求，确保在namespace中ResourceQuota对象处列举的container没有任何异常。如果在kubernetes中使用了ResourceQuota对象，就必须使用这个插件来约束container。

推荐在admission control参数列表中，这个插件排最后一个。

* LimitRanger

它会观察所有的请求，确保没有违反已经定义好的约束条件，这些条件定义在namespace中LimitRange对象中。如果在kubernetes中使用LimitRange对象，则必须使用这个插件。

* NamespaceExists

它会观察所有的请求，如果请求尝试创建一个不存在的namespace，则这个请求被拒绝。

1. **Scheduler的插件机制**
2. **调度器原理**

kubernets scheduler主要负责的工作是：接受API Server创建的新Pod，并为其安排一个主机，将信息写入etcd中。当然在这个过程中要处理的事情远远没有这么简单，需要综合考虑多种决策因素，比如把同一个replication controller的Pod分配到不同的主机上，防止因主机节点宕机对业务造成较大冲击；以及如何考虑资源均衡，从而提升整个集群的资源使用率等。

K8s的调度器的架构是插件话的，所有计算的函数都是以插件的形式提供，开发者可以自由选择需要那些计算函数，也可以方便的添加自己的函数，而不用担心改动太大代码

1. **调度过程**

kubernetes调度器通过API Server查找还未分配主机的Pod，并尝试为这些Pod分配主机：

* 客户端提交创建请求，可以通过API Server的Restful API，也可以使用kubectl命令行工具。支持的数据类型包括JSON和YAML。
* API Server处理用户请求，存储Pod数据到etcd。
* 调度器通过API Server查看未绑定的Pod。尝试为Pod分配主机。
* 过滤主机：调度器用一组规则过滤掉不符合要求的主机。比如Pod指定了所需要的资源量，那么可用资源比Pod需要的资源量少的主机会被过滤掉。
* 主机打分：对第一步筛选出的符合要求的主机进行打分，在主机打分阶段，调度器会考虑一些整体优化策略，比如把容一个Replication Controller的副本分布到不同的主机上，使用最低负载的主机等。
* 选择主机：选择打分最高的主机，进行binding操作，结果存储到etcd中。
* 所选主机对于的kubelet根据调度结果执行Pod创建操作。

1. **调度算法**

kubernetes通过一组规则，为每一个未调度的Pod选择一个主机，如调度流程中介绍，kubernetes的调度算法主要包括两个方面，过滤主机和主机打分。

kubernetes调度器的源码位于kubernetes/plugin/中，大体的代码目录结构如下所示：

|  |
| --- |
| —-kubernetes  ——–plugin  ————cmd    //kub-scheduler启动函数在cmd包中  ————pkg    //调度相关的具体实现  —————-scheduler  ——————–algorithm  ————————predicates    //主机筛选策略  ————————priorities    //主机打分策略  ——————–algorithmprovider  ————————defaults       //定义默认的调度器 |

过滤主机的目的是过滤掉不符合Pod要求的主机，现在kubernetes中实现的过滤规则主要包括以下几种（在kubernetes/plugin/pkg/scheduler/algorithm/predicates中实现）：

* NoDiskConflict：检查在此主机上是否存在卷冲突。如果这个主机已经挂载了卷，其它同样使用这个卷的Pod不能调度到这个主机上。GCE,        Amazon EBS, and Ceph RBD使用的规则如下：
  + GCE允许同时挂载多个卷，只要这些卷都是只读的。
  + Amazon EBS不允许不同的Pod挂载同一个卷。
  + Ceph RBD不允许任何两个pods分享相同的monitor，match pool和 image。
* NoVolumeZoneConflict：检查给定的zone限制前提下，检查如果在此主机上部署Pod是否存在卷冲突。假定一些volumes可能有zone调度约束， VolumeZonePredicate根据volumes自身需求来评估pod是否满足条件。必要条件就是任何volumes的zone-labels必须与节点上的zone-labels完全匹配。节点上可以有多个zone-labels的约束（比如一个假设的复制卷可能会允许进行区域范围内的访问）。目前，这个只对PersistentVolumeClaims支持，而且只在PersistentVolume的范围内查找标签。处理在Pod的属性中定义的volumes（即不使用PersistentVolume）有可能会变得更加困难，因为要在调度的过程中确定volume的zone，这很有可能会需要调用云提供商。
* PodFitsResources：检查主机的资源是否满足Pod的需求。根据实际已经分配的资源量做调度，而不是使用已实际使用的资源量做调度。
* PodFitsHostPorts：检查Pod内每一个容器所需的HostPort是否已被其它容器占用。如果有所需的HostPort不满足需求，那么Pod不能调度到这个主机上。
* HostName：检查主机名称是不是Pod指定的HostName。
* MatchNodeSelector：检查主机的标签是否满足Pod的\*nodeSelector\*属性需求。
* MaxEBSVolumeCount：确保已挂载的EBS存储卷不超过设置的最大值。默认值是39。它会检查直接使用的存储卷，和间接使用这种类型存储的PVC。计算不同卷的总目，如果新的Pod部署上去后卷的数目会超过设置的最大值，那么Pod不能调度到这个主机上。
* MaxGCEPDVolumeCount：确保已挂载的GCE存储卷不超过设置的最大值。默认值是16。规则同上。

1. **Volume的插件机制**

**1. Volumes**

volume是一个能够被容器访问的目录，它可能还会包含一些数据。Kubernetes Volumes与Docker Volumes类似，但并不完全相同。一个Pod会在它的ContainerManifest 属性中指明其容器需要哪些Volumes。容器中的进程可见的文件系统视图由两个源组成：一个单独的Docker image和零个或多个Volumes。Docker image位于文件层次结构的根部。所有的Volumes都挂载在Docker image的节点上。Volumes不能挂载在其他的Volumes上，也没有连接其他Volumes的硬链接。Pod中的每个容器都单独地指明了其image挂载的Volume。这会通过VolumeMount属性来确定。

**2.为什么需要插件机制**

Volume的种类是非常多的。可以是aws云盘，也可以是gce云盘，还可以是ceph盘等。为每种volume都单独写一份代码是非常不明智的，这将带来大量的代码冗余和维护的困难。因为最好的做法是，主干代码是一份，然后为每种volume写一份驱动。

1. **NetWork的插件机制**

由于网络的复杂性，以及不同用户对于网络的各种订制化需求，因此需要提供网络方面的插件机制。K8s 1.0的网络插件机制只要是通过一堆Pod的生命周期的钩子函数来实现的。开发者可以自己实现一些钩子函数，在Pod的生命周期的各个阶段调用一些函数，从而实现对网络的一些定制开发。