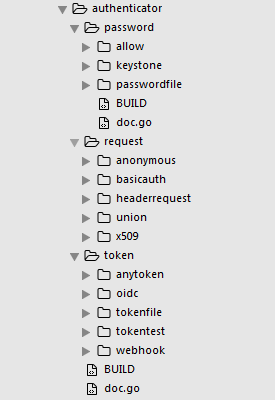
# K8s安全模块源码走读

K8s安全模块通过插件的形式提供服务，好处是方便功能组件的替换。包括认证和授权两部分，认证是鉴权的先决条件。认证就是判断该用户是否是合法用户，授权就是在认证通过的基础上判断该用户是否具有权限。



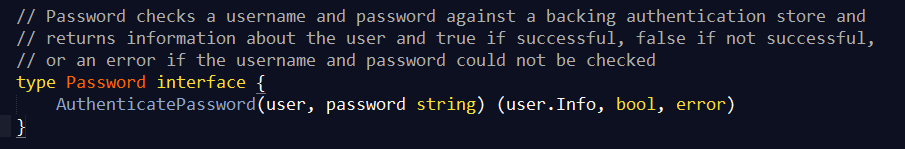
## 认证

认证是对用户合法性的校验，kubernetes提供的认证方式包括用户名密码的方式（password）、请求头方式（request）、token认证方式。



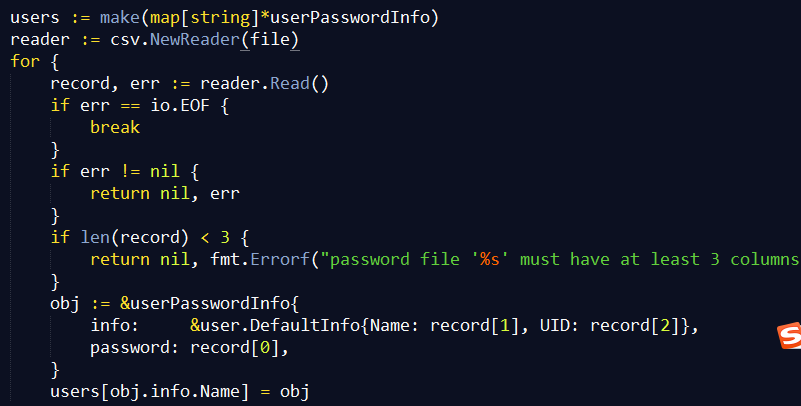
### 密码认证

密码认证过程包含keystone认证方式和密码文件认证方式。采用密码认证的插件需要实现Password接口。

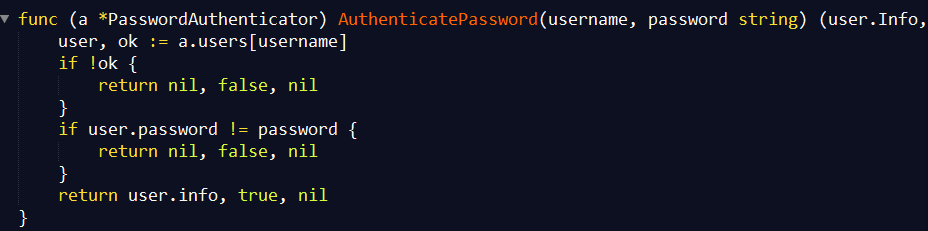


1. 密码文件认证

密码文件认证方式是指用户先指定了密码文件，该文件存储了所有合法用户信息。有效的用户信息存储在csv文件内，存储的格式：密码、用户名、用户标识码。



认证过程中通过传入用户名、密码进行认证。认证也是简单的匹配用户信息是否存在用户预先定义的csv文件。



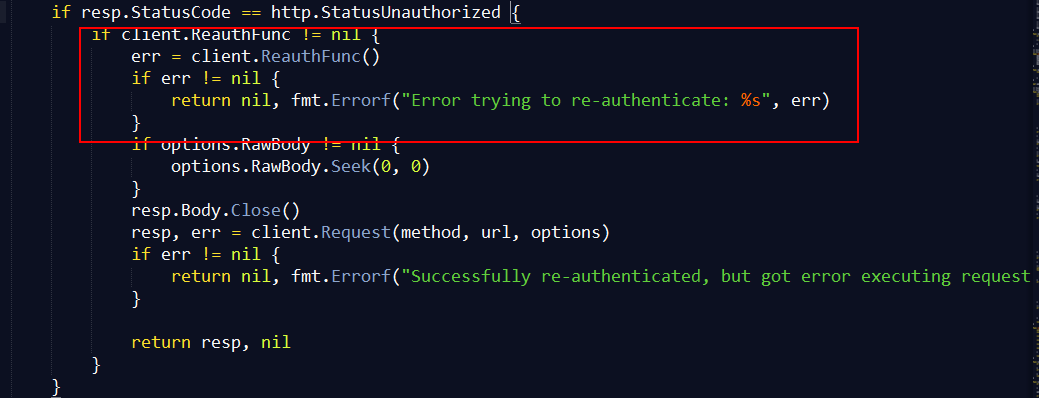
1. Keystone认证

Keystone的认证通过发送keystone的URL（/v3/auth/tokens），该api需要用户名、密码、租户ID（v3更名为project）：

vendor\github.com\rackspace\gophercloud\openstack\client.go

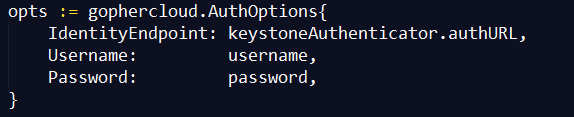


上图描述了向keystone请求认证的接口，如果认证失败，会再次认证，最终认证失败会方法失败信息：



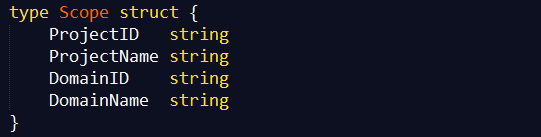
通过Create方法会获取到为用户分配的token信息，请求需要传入用户的基本信息，包括用户名和密码：

vendor\github.com\rackspace\gophercloud\openstack\identity\v3\tokens\requests.go



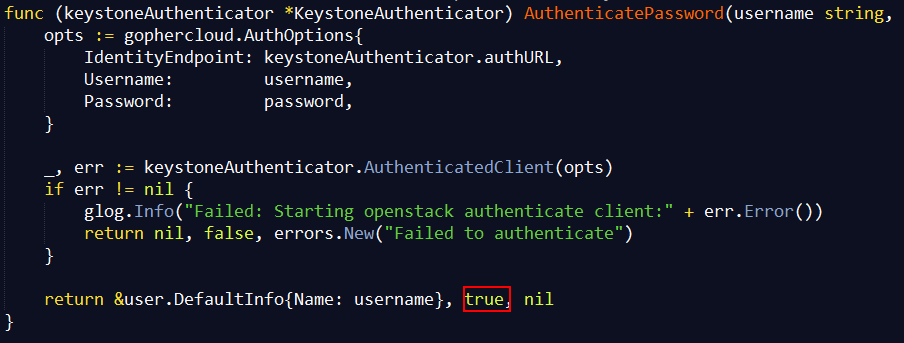
以及用户的scope信息，包括租户信息、域信息

vendor\github.com\rackspace\gophercloud\openstack\identity\v3\tokens\requests.go



Keystone认证通过方法信息：

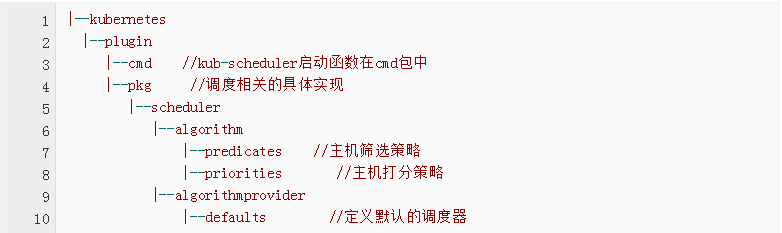
\plugin\pkg\auth\authenticator\password\keystone\keystone.go



### 请求信息认证

请求信息认证包括基本信息认证和证书认证

# Scheduler



## 工作流程

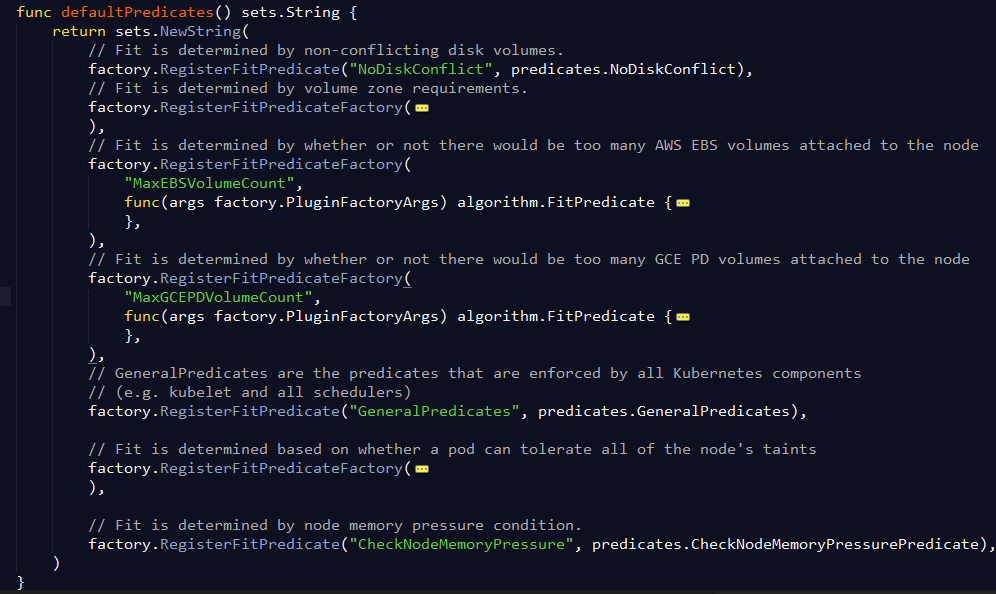
* 客户端提交创建请求，可以通过API Server的Restful API，也可以使用kubectl命令行工具。支持的数据类型包括JSON和YAML。
* API Server处理用户请求，存储Pod数据到etcd。
* 调度器通过API Server查看未绑定的Pod。尝试为Pod分配主机。
* 过滤主机：调度器用一组规则过滤掉不符合要求的主机。比如Pod指定了所需要的资源量，那么可用资源比Pod需要的资源量少的主机会被过滤掉。
* 主机打分：对第一步筛选出的符合要求的主机进行打分，在主机打分阶段，调度器会考虑一些整体优化策略，比如把容一个Replication Controller的副本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。

### 主机过滤

plugin\pkg\scheduler\algorithmprovider\defaults\defaults.go

过滤主机的目的是过滤掉不符合Pod要求的主机

* 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。



过滤规则和打分规则：



### 主机打分

Kubernetes用一组优先级函数处理每一个待选的主机，每一个优先级函数会返回一个0-10的分数，分数越高表示主机越“好”， 同时每一个函数也会对应一个表示权重的值。



打分函数：

* LeastRequestedPriority：如果新的pod要分配给一个节点，这个节点的优先级就由节点空闲的那部分与总容量的比值（即（总容量-节点上pod的容量总和-新pod的容量）/总容量）来决定。CPU和memory权重相当，比值最大的节点的得分最高。需要注意的是，这个优先级函数起到了按照资源消耗来跨节点分配pods的作用。计算公式如下：cpu((capacity - sum(requested)) \* 10 / capacity) + memory((capacity - sum(requested)) \* 10 / capacity) / 2
* BalancedResourceAllocation：尽量选择在部署Pod后各项资源更均衡的机器。*BalancedResourceAllocation*不能单独使用，而且必须和*LeastRequestedPriority*同时使用，它分别计算主机上的cpu和memory的比重，主机的分值由cpu比重和memory比重的“距离”决定。计算公式如下：
* score = 10 - abs(cpuFraction-memoryFraction)\*10
* SelectorSpreadPriority：对于属于同一个service、replication controller的Pod，尽量分散在不同的主机上。如果指定了区域，则会尽量把Pod分散在不同区域的不同主机上。调度一个Pod的时候，先查找Pod对于的service或者replication controller，然后查找service或replication controller中已存在的Pod，主机上运行的已存在的Pod越少，主机的打分越高。
* CalculateAntiAffinityPriority：对于属于同一个service的Pod，尽量分散在不同的具有指定标签的主机上。
* ImageLocalityPriority：根据主机上是否已具备Pod运行的环境来打分。ImageLocalityPriority会判断主机上是否已存在Pod运行所需的镜像，根据已有镜像的大小返回一个0-10的打分。如果主机上不存在Pod所需的镜像，返回0；如果主机上存在部分所需镜像，则根据这些镜像的大小来决定分值，镜像越大，打分就越高。
* NodeAffinityPriority（Kubernetes1.2实验中的新特性）：Kubernetes调度中的亲和性机制。Node Selectors（调度时将pod限定在指定节点上），支持多种操作符（In, NotIn, Exists, DoesNotExist, Gt, Lt），而不限于对节点labels的精确匹配。另外，Kubernetes支持两种类型的选择器，一种是“hard（requiredDuringSchedulingIgnoredDuringExecution）”选择器，它保证所选的主机必须满足所有Pod对主机的规则要求。这种选择器更像是之前的nodeselector，在nodeselector的基础上增加了更合适的表现语法。另一种是“soft（preferresDuringSchedulingIgnoredDuringExecution）”选择器，它作为对调度器的提示，调度器会尽量但不保证满足NodeSelector的所有要求。

# Kube-apiserver