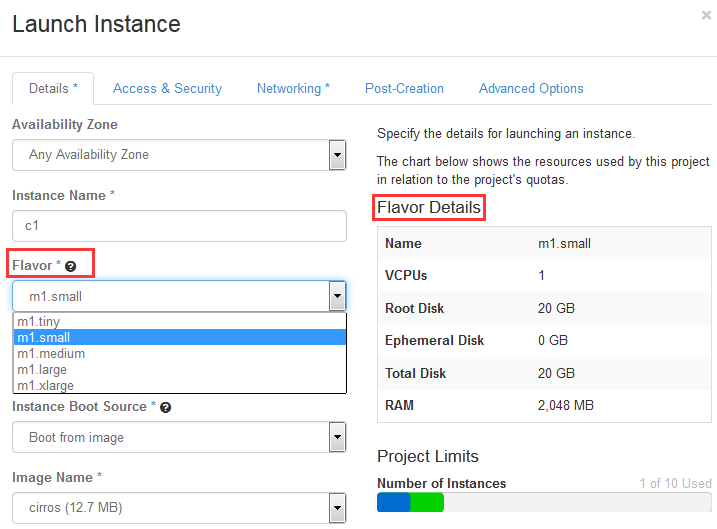


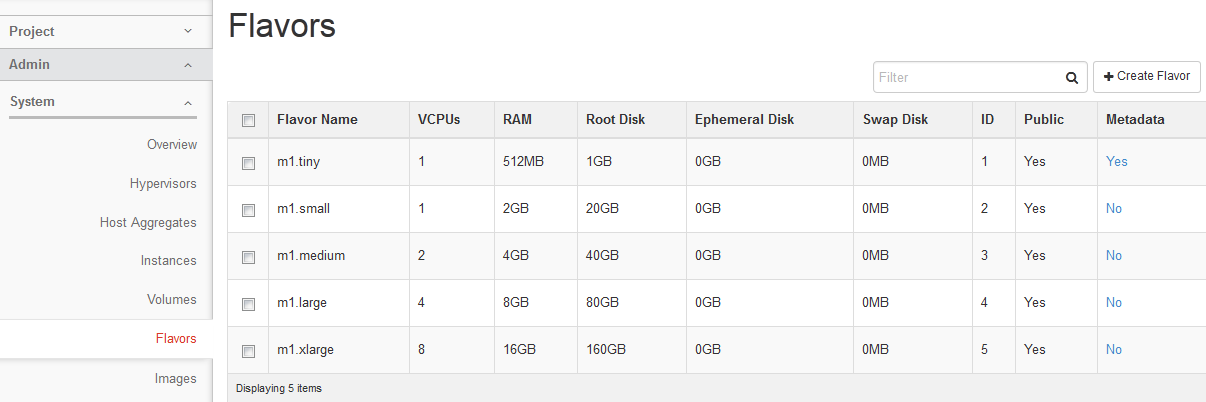
本节重点介绍 nova-scheduler 的调度机制和实现方法：即解决如何选择在哪个计算节点上启动 instance 的问题。

创建 Instance 时，用户会提出资源需求，例如 CPU、内存、磁盘各需要多少。

OpenStack 将这些需求定义在 flavor 中，用户只需要指定用哪个 flavor 就可以了。



可用的 flavor 在 System->Flavors 中管理。



Flavor 主要定义了 VCPU，RAM，DISK 和 Metadata 这四类。 nova-scheduler 会按照 flavor 去选择合适的计算节点。 VCPU，RAM，DISK 比较好理解，而 Metatdata 比较有意思，我们后面会具体讨论。

下面介绍 nova-scheduler 是如何实现调度的。

在 /etc/nova/nova.conf 中，nova 通过 scheduler\_driver，scheduler\_available\_filters 和 scheduler\_default\_filters 这三个参数来配置 nova-scheduler。

**Filter scheduler**

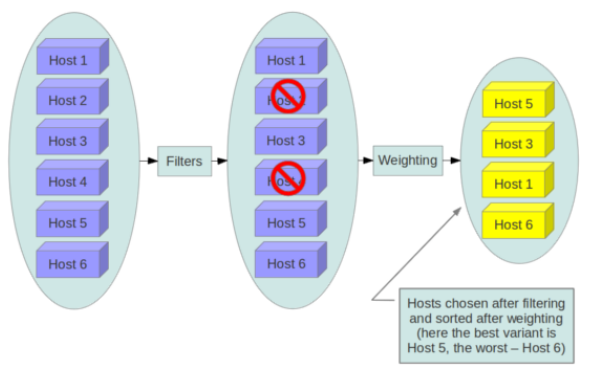
Filter scheduler 是 nova-scheduler 默认的调度器，调度过程分为两步：

1. 通过过滤器（filter）选择满足条件的计算节点（运行 nova-compute）
2. 通过权重计算（weighting）选择在最优（权重值最大）的计算节点上创建 Instance。

scheduler\_driver=nova.scheduler.filter\_scheduler.FilterScheduler

Nova 允许使用第三方 scheduler，配置 scheduler\_driver 即可。 这又一次体现了OpenStack的开放性。

Scheduler 可以使用多个 filter 依次进行过滤，过滤之后的节点再通过计算权重选出最适合的节点。



上图是调度过程的一个示例：

1. 最开始有 6 个计算节点 Host1-Host6
2. 通过多个 filter 层层过滤，Host2 和 Host4 没有通过，被刷掉了
3. Host1，Host3，Host5，Host6 计算权重，结果 Host5 得分最高，最终入选

**Filter**

当 Filter scheduler 需要执行调度操作时，会让 filter 对计算节点进行判断，filter 返回 True 或 False。

Nova.conf 中的 scheduler\_available\_filters 选项用于配置 scheduler 可用的 filter，默认是所有 nova 自带的 filter 都可以用于滤操作。

scheduler\_available\_filters = nova.scheduler.filters.all\_filters

另外还有一个选项 scheduler\_default\_filters，用于指定 scheduler 真正使用的 filter，默认值如下

scheduler\_default\_filters = RetryFilter, AvailabilityZoneFilter, RamFilter, DiskFilter, ComputeFilter, ComputeCapabilitiesFilter, ImagePropertiesFilter, ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter

Filter scheduler 将按照列表中的顺序依次过滤。 下面依次介绍每个 filter。

**RetryFilter**

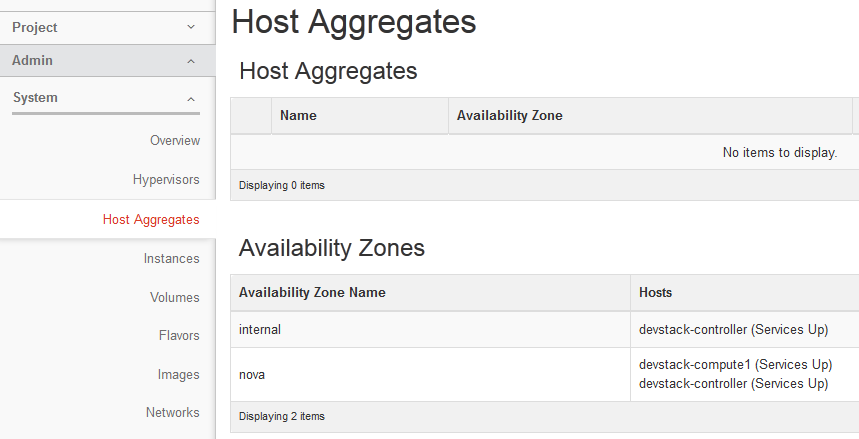
RetryFilter 的作用是刷掉之前已经调度过的节点。

举个例子方便大家理解： 假设 A,B,C 三个节点都通过了过滤，最终 A 因为权重值最大被选中执行操作。 但由于某个原因，操作在 A 上失败了。 默认情况下，nova-scheduler 会重新执行过滤操作（重复次数由 scheduler\_max\_attempts 选项指定，默认是 3）。 那么这时候 RetryFilter 就会将 A 直接刷掉，避免操作再次失败。 RetryFilter 通常作为第一个 filter。

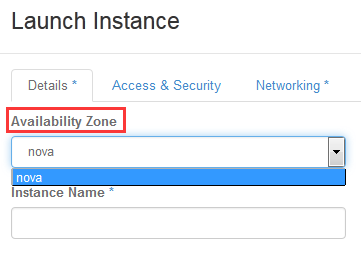
**AvailabilityZoneFilter**

为提高容灾性和提供隔离服务，可以将计算节点划分到不同的Availability Zone中。

例如把一个机架上的机器划分在一个 Availability Zone 中。 OpenStack 默认有一个命名为 “Nova” 的 Availability Zone，所有的计算节点初始都是放在 “Nova” 中。 用户可以根据需要创建自己的 Availability Zone。



创建 Instance 时，需要指定将 Instance 部署到在哪个 Availability Zone中。



nova-scheduler 在做 filtering 时，会使用 AvailabilityZoneFilter 将不属于指定 Availability Zone 的计算节点过滤掉。

**RamFilter**

RamFilter 将不能满足 flavor 内存需求的计算节点过滤掉。

对于内存有一点需要注意： 为了提高系统的资源使用率，OpenStack 在计算节点可用内存时允许 overcommit，也就是可以超过实际内存大小。 超过的程度是通过 nova.conf 中 ram\_allocation\_ratio 这个参数来控制的，默认值为 1.5

ram\_allocation\_ratio = 1.5

其含义是：如果计算节点的内存有 10GB，OpenStack 则会认为它有 15GB（10\*1.5）的内存。

**DiskFilter**

DiskFilter 将不能满足 flavor 磁盘需求的计算节点过滤掉。

Disk 同样允许 overcommit，通过 nova.conf 中 disk\_allocation\_ratio 控制，默认值为 1

disk\_allocation\_ratio = 1.0

**CoreFilter**

CoreFilter 将不能满足 flavor vCPU 需求的计算节点过滤掉。

vCPU 同样允许 overcommit，通过 nova.conf 中 cpu\_allocation\_ratio 控制，默认值为 16

cpu\_allocation\_ratio = 16.0

这意味着一个 8 vCPU 的计算节点，nova-scheduler 在调度时认为它有 128 个 vCPU。 需要提醒的是： nova-scheduler 默认使用的 filter 并没有包含 CoreFilter。 如果要用，可以将 CoreFilter 添加到 nova.conf 的 scheduler\_default\_filters 配置选项中。

**ComputeFilter**

ComputeFilter 保证只有 nova-compute 服务正常工作的计算节点才能够被 nova-scheduler调度。  
ComputeFilter 显然是必选的 filter。

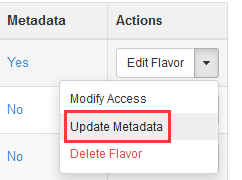
**ComputeCapabilitiesFilter**

ComputeCapabilitiesFilter 根据计算节点的特性来筛选。

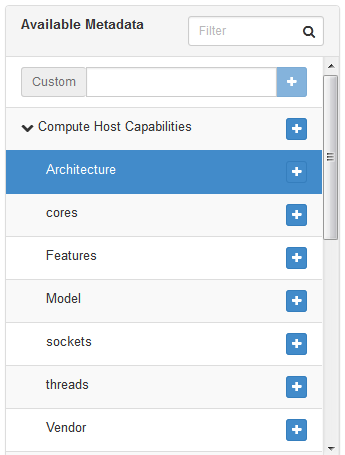
这个比较高级，我们举例说明。

例如我们的节点有 x86\_64 和 ARM 架构的，如果想将 Instance 指定部署到 x86\_64 架构的节点上，就可以利用到 ComputeCapabilitiesFilter。

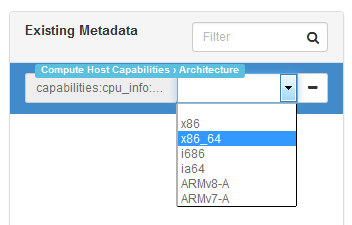
还记得 flavor 中有个 Metadata 吗，Compute 的 Capabilitie s就在 Metadata中 指定。



“Compute Host Capabilities” 列出了所有可设置 Capabilities。



点击 “Architecture” 后面的 “+”，就可以在右边的列表中指定具体的架构。



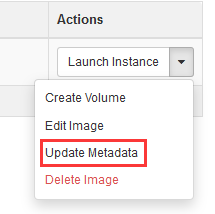
配置好后，ComputeCapabilitiesFilter 在调度时只会筛选出 x86\_64 的节点。

如果没有设置 Metadata，ComputeCapabilitiesFilter 不会起作用，所有节点都会通过筛选。

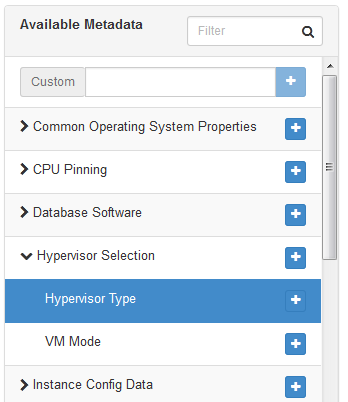
**ImagePropertiesFilter**

ImagePropertiesFilter 根据所选 image 的属性来筛选匹配的计算节点。

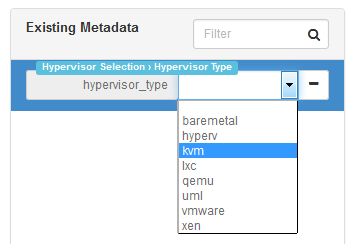
跟 flavor 类似，image 也有 metadata，用于指定其属性。



例如希望某个 image 只能运行在 kvm 的 hypervisor 上，可以通过 “Hypervisor Type” 属性来指定。



点击 “+”，然后在右边的列表中选择 “kvm”。



配置好后，ImagePropertiesFilter 在调度时只会筛选出 kvm 的节点。

如果没有设置 Image 的Metadata，ImagePropertiesFilter 不会起作用，所有节点都会通过筛选。

**ServerGroupAntiAffinityFilter**

ServerGroupAntiAffinityFilter 可以尽量将 Instance 分散部署到不同的节点上。

例如有 inst1，inst2 和 inst3 三个 instance，计算节点有 A,B 和 C。

为保证分散部署，进行如下操作：

1. 创建一个 anti-affinity 策略的 server group “group-1”

nova server-group-create --policy anti-affinity group-1

请注意，**这里的 server group 其实是 instance group，并不是计算节点的 group**。

1. 依次创建 Instance，将inst1, inst2和inst3放到group-1中

nova boot --image IMAGE\_ID --flavor 1 --hint group=group-1 inst1

nova boot --image IMAGE\_ID --flavor 1 --hint group=group-1 inst2

nova boot --image IMAGE\_ID --flavor 1 --hint group=group-1 inst3

因为 group-1 的策略是 AntiAffinity，调度时 ServerGroupAntiAffinityFilter 会将 inst1, inst2 和 inst3 部署到不同计算节点 A, B 和 C。

目前只能在 CLI 中指定 server group 来创建 instance。

创建 instance 时如果没有指定 server group，ServerGroupAntiAffinityFilter 会直接通过，不做任何过滤。

**ServerGroupAffinityFilter**

与 ServerGroupAntiAffinityFilter 的作用相反，ServerGroupAffinityFilter 会尽量将 instance 部署到同一个计算节点上。

方法类似

1. 创建一个 affinity 策略的 server group “group-2”

nova server-group-create --policy affinity group-2

1. 依次创建 instance，将 inst1, inst2 和 inst3 放到 group-2 中

nova boot --image IMAGE\_ID --flavor 1 --hint group=group-2 inst1

nova boot --image IMAGE\_ID --flavor 1 --hint group=group-2 inst2

nova boot --image IMAGE\_ID --flavor 1 --hint group=group-2 inst3

因为 group-2 的策略是 Affinity，调度时 ServerGroupAffinityFilter 会将 inst1, inst2 和 inst3 部署到同一个计算节点。

创建 instance 时如果没有指定 server group，ServerGroupAffinityFilter 会直接通过，不做任何过滤。

**Weight**

经过前面一堆 filter 的过滤，nova-scheduler 选出了能够部署 instance 的计算节点。

如果有多个计算节点通过了过滤，那么最终选择哪个节点呢？

Scheduler 会对每个计算节点打分，得分最高的获胜。

打分的过程就是 weight，翻译过来就是计算权重值，那么 scheduler 是根据什么来计算权重值呢？

目前 nova-scheduler 的默认实现是根据计算节点空闲的内存量计算权重值：

空闲内存越多，权重越大，instance 将被部署到当前空闲内存最多的计算节点上。

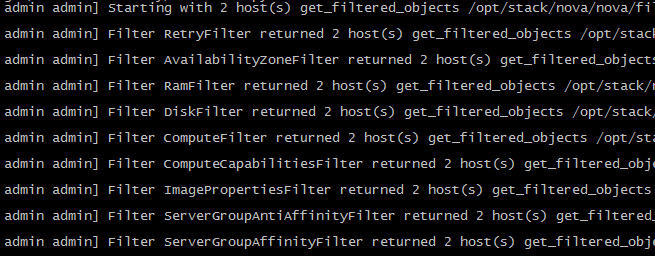
**日志**

是时候完整的回顾一下 nova-scheduler 的工作过程了。

整个过程都被记录到 nova-scheduler 的日志中。

比如当我们部署一个 instance 时

打开 nova-scheduler 的日志 /opt/stack/logs/n-sch.log（非 devstack 安装其日志在 /var/log/nova/scheduler.log）

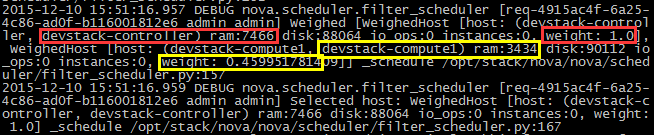


日志显示初始有两个 host（在我们的实验环境中就是 devstack-controller 和 devstack-compute1），依次经过 9 个 filter 的过滤（RetryFilter, AvailabilityZoneFilter, RamFilter,

DiskFilter, ComputeFilter, ComputeCapabilitiesFilter, ImagePropertiesFilter,

ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter），两个计算节点都通过了。

那么接下来就该 weight 了：



可以看到因为 devstack-controller 的空闲内存比 devstack-compute1 多（7466 > 3434），权重值更大（1.0 > 0.4599），最终选择 devstack-controller。

注：要显示 DEBUG 日志，需要在 /etc/nova/nova.conf 中打开 debug 选项

[DEFAULT]

debug = True

nova-scheduler 就是这些内容了，稍微有些复杂哈（因为灵活嘛），大家这两天可以好好消化一下。

下节我们讨论 nova-compute。