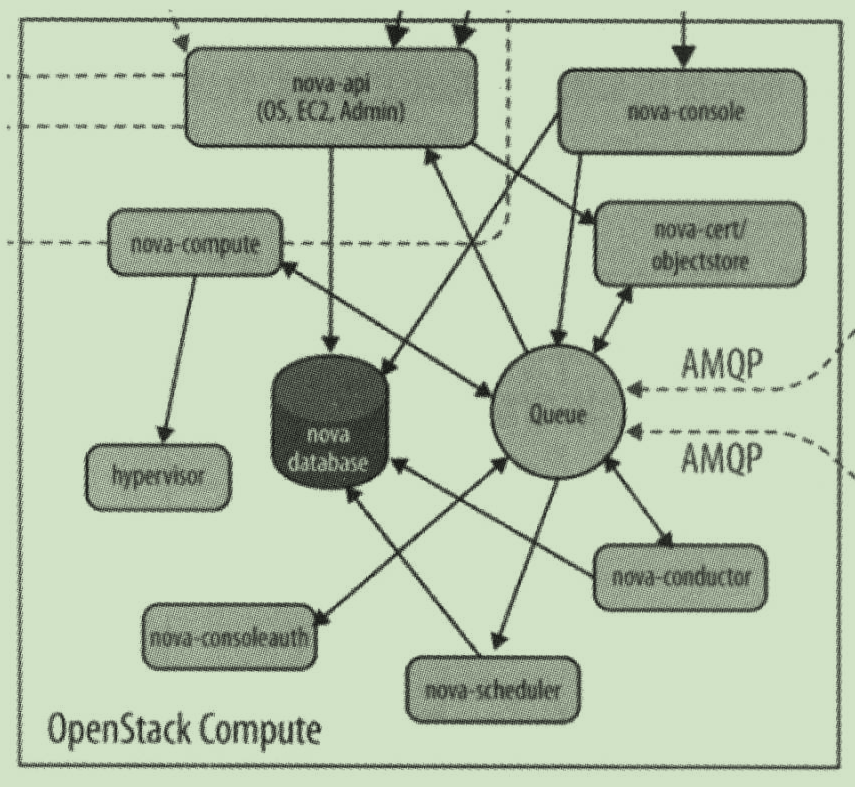
# 一．Nova服务

## （一）Nova架构

### 1.架构图

Nova的架构非常复杂，包含很多组件，这些组件以后台守护进程的方式运行。架构图如下所示：：



### 2.组件介绍

#### （1）API

##### 1) nova-api

接受和响应客户的API调用，除了接受OpenStack自己的API，还支持Amazon EC2 API，也就是说nova-api兼容EC2 API。

#### （2）Compute Core

##### 1) nova-scheduler

虚拟机调度服务，该组件负责决定应该在哪个计算节点上运行虚拟机。

##### 2) nova-compute

该组件是管理虚拟机的核心服务，通过调用Hypervisor API实现虚拟机的生命周期管理。

##### 3) Hypervisor

计算节点上跑的虚拟化管理程序，是虚拟机管理最底层的程序。不同的虚拟化技术提供自己的Hypervisor，常用的有KVM, Xen, VMware等。

##### 4) nova-comductor

计算节点经常需要更新数据库，比如更新虚拟机的状态，处于安全性和伸缩性的考虑，nova-compute并不会直接访问数据库，而是将这个任务交给nova-conductor。

#### （3）Console Interface

##### 1) nova-console

用户可以通过多种方式访问虚机的控制台：

nova-novncproxy:基于Web浏览器的VNC访问；

nova-spicehtml5proxy:基于HTML5浏览器的SPICE访问；

nova-xvpnvncproxy:基于Java客户端的VNC访问；

##### 2) nova-consoleauth

该组件负责对访问虚机控制台的请求进行Token认证。

##### 3) nova-cert

提供x509证书支持。

#### （4）Database

Nova有一些数据需要存放在数据库中，一般使用Mysql。数据库一般安装在控制节点上，Nova服务使用的数据库名称为nova。

#### （5）Message Queue

由于Nova包含众多的子服务，这些子服务之间需要相互协调和通信，为了解耦各个子服务，Nova通过Message Queue作为子服务的信息中转站，默认是RabbitMQ。

### 3.nova组件的物理部署方案

Nova的组件会部署在两类节点上：计算节点和控制节点。

计算节点：Hypervisor（运行着虚拟机）以及nova-compute；

控制节点：nova-scheduler,nova- conductor, nova-consleauth, nova-cert, nova-api, nova-novncpoxy, nova-compute, my-server（消息队列）, mysqld（Mysql服务端）

### 4. nova子服务的协同工作

（1）客户向nova-api发送请求，请求创建虚拟机；

（2）nova-api对请求作出处理后，向mq发送消息“让scheduler创建虚机”；

（3）nova-scheduler从mq中获取到API发送给它的消息，然后执行调度算法，从若干个计算节点中选出节点A；

（4）noav-scheduler向mq发送消息“在节点A上创建虚拟”；

（5）计算节点A的nova-compute从mq中获取到nova-scheduler发给它的消息，然后在本节点的Hypervisor上启动虚拟机；

（6）在虚拟机创建的过程中，计算节点A的nova-compute如果需要查询或更新数据库消息，会通过mq向nova-conductor发送消息，nova-conductor负责数据库访问。

## （二）Nova组件介绍

### 1.nova-api

nova-api是整个Nova组件的门户，所有对Nova的请求都首先经由nova-api处理。nova-api向外暴露若干个HTTP REST API的接口。在Keystone中我们可以查询nova-api的endpoints，客户端可以将请求发送到endpoints指定的地址，向nova-api请求操作。只要是和虚拟机生命周期相关的操作，nova-api都可以响应。

### 2.nova-scheduler

#### （1）flavor

创建Instance时，用户会提出资源需求，如CPU，内存，磁盘各需要多少，OpenStack会将这些需求定义在flavor中，用户只需要指定使用哪个flavor就可以了。Flavor中定义了VCPU，RAM，DISK和Metadata这四类细节，nova-scheduler会按照flavor来选择合适的计算节点。

#### （2）nova-scheduler的参数配置

在/etc/nova/nova.conf中，nova通过schedulerdriver, scheduleravailable\_filters和schedulerdefaultfilters这三个参数来配置nova-scheduler。

#### （3）Filter Scheduler

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

通过过滤器filter选择满足条件的计算节点（nova-compute）；

通过权重计算（weighting）选择最大权重的计算节点上创建Instance；

Nova允许使用第三方的scheduler，配置scheduler\_driver即可。

#### （4）Filter

当Filter scheduler需要进行调度操作时，会让filter对计算节点进行判断，返回True或False。nova.conf中的scheduleravaiablefilters选项用于配置scheduler可用的filter，默认所有nova自带的filter都可以用于过滤操作。

另一个选项schedulerdefaultfilters，用于指定scheduler真正使用的filter。有如下的filter：

##### 1)RetryFilter

作用是刷掉之前已经调度过的节点。这种过滤器是为了防止一个被分配的计算节点操作失败，通过RetryFilter可以直接过滤掉失败的节点，避免操作再次失败。

##### 2)AvailabilityZoneFilter

为了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。OpenStack默认有一个Nova的Availability Zone，所有的计算节点初始都放在Nova空间中。创建Instance时，可以指定将Instance部署到指定的Availability Zone中。

AvailabilityZoneFilter会将不属于指定Availability Zone的计算节点过滤掉。

##### 3)RAMFilter

RamFilter将不能满足flavor内存需求的计算节点过滤掉。注意，为了提高系统的资源使用率，计算节点的可用内存是允许overcommit的，即可以超过实际内存大小，nova.conf中通过ramallocationratio控制，默认值是1.5。

##### 4)DiskFilter

DiskFilter将不能满足flavor磁盘需求的计算节点过滤掉。同样磁盘也允许overcommit，通过nova.conf中的diskallocationratio控制，默认值是1。

##### 5)CoreFilter

CoreFilter将不能满足flavor VCPU需求的计算节点过滤器，VCPU同样允许overcommit，通过nova.conf的cpuallocationration控制，默认值是16。

##### 6)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。

##### 7)ComputeCapabilitiesFilter

该过滤器根据计算节点的特性来筛选，如计算节点的架构是X86还是ARM的，这些都是通过该过滤器过滤的。

##### 8)ImagePropertiesFilter

该过滤器根据所选Image的属性来筛选匹配的计算节点，如Image指定运行在某种Hypervisor，这都是根据Image的Metadata来过滤的，如果Image没有设置Metadata，该过滤器就不会起作用。

#### （5）Weight

经过一堆过滤器的过滤后，nova-scheduler选出了能够部署instance的计算节点，再通过weight来对每个计算节点打分，打分最高的获胜。nova-scheduler默认的计算权重策略是通过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。

#### （6）scheduler日志

整个调度过程都记录在nova-scheduler日志中了，在/var/log/nova/scheduler.log中。如果要查询debug日志，就需要在/etc/nova/nova.conf中打开debug选项。

### 3.nova-compute

#### （1）Hypervisor

nova-compute在计算节点上运行，负责管理节点上的instance，nova-compute与Hypervisor一起实现OpenStack对instance生命周期的管理。nova-compute为不同的Hypervisor定义了统一的接口，Hypervisor只需要实现这些接口，就可以以Driver的形式直接插入到OpenStack系统中。

在计算节点nova-compute的配置文件/etc/nova/nova.conf中配置所对应的compute\_driver，如KVM配置的是Libvirt的driver。

#### （2）nova-compute功能

nova-compute的功能分为两类：

##### 1)定时向OpenStack报告计算节点的状态

要得到计算节点的资源使用情况，需要知道当前节点上所有instance的资源占用信息，这些都是通过Hypervisor的driver驱动获取instance的资源信息。

##### 2)实现instance生命周期的管理

OpenStack对instance的操作都是通过nova-compute实现的，包括instance的启动，停止，重启，暂停，恢复，终止，迁移，快照。

当nova-scheduler选定了部署instance的计算节点之后，会通过RabbitMQ向选定的计算节点发出启动实例的命令，该计算节点上的nova-compute收到消息后会执行instance创建操作。

#### （3）nova-compute创建instance的步骤

分为四步：

##### 1)为instance准备资源；

nova-compute首先会根据指定的flavor依次为instance分配内存，磁盘空间和VCPU和网络资源。

##### 2)创建instance的镜像文件；

资源准备好之后，nova-compute会为instance创建镜像文件。首先选择一个Glance中的Image，检查计算节点中是否已有该Image，如果没有则从Glance下载到计算节点中，然后将其作为backing file创建instance的镜像文件。镜像文件是通过qemu-img命令从Image中创建的，注意，镜像文件是instance启动盘对应的文件，而Image是Glance上保存的模板，即instance运行的模板。

##### 3)创建instance的XML定义文件；

创建instance的XML定义文件。

##### 4)创建虚拟网络并启动虚拟机；

为instance创建虚拟网络设备之后，就可以启动instance了。

### 4.nova-conductor

由于nova-compte需要获取和更新数据库中instance的信息，但是nova-compute并不会直接访问数据库，而是通过nova-conductor实现数据库访问。这样能够保证数据库的访问安全，以及更好的系统伸缩性，对于高并发的数据库访问请求，可以通过配置nova-conductor集群来分摊访问压力。

## （三）Nova 操作详解

### a)常规操作

#### 1.Launch

启动一个实例，通过nova-api，nova-scheduler，rabbitmq，nova-compute，nova-conductor来协调。

1. 向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. nova-scheduler从RabbitMQ获取消息，执行调度选出节点；
4. nova-scheduler向RabbitMQ发送消息；
5. 节点的nova-compute从rabbitMQ中获取消息，通过本节点的Hypervisor Driver创建Instance；

#### 2.Shut Off

停止一个实例 。

1. 向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息，停止Instance；

#### 3.Start

生成一个新的实例。

1. 向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息之后，开始启动，准备虚拟网卡，准备Instance的XML文件，准备Instance的镜像文件，最后启动Instance；

#### 4.Soft/Hartd Reboot

两种重启方式

1. soft reboot

重启操作系统，整个过程中，Instance依然处于运行状态。

1. hard reboot

重启Instance，相当于关机后再开机。

#### 5.Lock/Unlock

为了避免误操作，可以锁定Instance，通过解锁（Lock）操作恢复正常。Lock/Unlock操作都是在nova-api中进行的，操作成功后，nova-api会更新Instance加锁的状态，执行其他操作时，nova-api根据加锁状态来判断是否允许操作。

#### 6.Terminate

删除Instance。

1. 向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息，关闭Instance，删除Instance的镜像文件，释放虚拟网络等其他资源；

#### 7.Pause/Resume

短时间暂停Instance，通过Pause操作将Instance的状态保存到宿主机的内存中，当需要恢复时，执行Resume操作，从内存中读回Instance的状态，然后继续运行Instance。

1. 向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息，暂停Instance后，Instance的状态变为Paused；

#### 8.Suspend/Resume

长时间暂停Instance，可以通过Suspend操作将Instance的状态保存到宿主机的磁盘上，当需要恢复的时候，执行Resume操作，从磁盘中读回Instance的状态，继续运行。

Instance被Suspend后的状态是Shut Down。

#### 9.Snapshot

有时候操作系统损坏地很严重，无法通过Rescue操作恢复，因此考虑使用备份恢复。Nova的备份操作为Snapshot，工作原理是对Instance的镜像文件（系统盘）进行全量备份，生成一个类型为snapshot的Image，然后将其保存在Glance上。

1. 向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息，暂停Instance，对Instance的镜像文件做快照，恢复Instance，将快照上传到Glance中。

#### 10.Resize

Resize操作是用于调整Instance的VCPU，内存和磁盘资源大小的，即为Instance选择新的flavor。通过nova-scheduler重新为Instance选择一个合适的计算节点，如果选择的节点与当前的节点不是同一个，那么就需要做Migrate操作。

### b）计划内的故障处理（系统升级，更换硬件）

#### 1.Sleve

当Instance被Suspend后虽然处于Shut Down的状态，但是Hypervisor依然在宿主机上为其预留了资源，以便能够成功Resume。如果希望释放这些资源，可以使用Shele操作，它会将Instance作为Image保存到Glance中，然后在宿主机上删除该Instance。

1. 向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息，关闭Instance，然后对Instance指定snapshot操作，成功后，snapshot生成的Image会保存在Glance中，最后删除Instance在宿主机上的资源；

#### 2.Unshelve

通过Unshelve操作恢复被Shelve的Instance。它的操作其实就是通过Glance中保存的Image启动一个新的Instance，同时nova-scheduler也会调度合适的计算节点来创建该Instance。

#### 3.Migrate

Migrate操作的作用是将Instance从当前的计算节点迁移到其他的节点上，Migrate操作不要求源节点和目标节点共享存储，但是必须满足一个条件，就是计算节点之间需要配置nova用户无密码访问。

1. 只能由Admin用户向nova-api发送请求；
2. nova-api向RabbitMQ发送消息；
3. nova-scheduler收到消息后，会为Instance选择合适的计算节点，然后通知计算节点迁移Instance；
4. 源节点的nova-compute从rabbitMQ中获取消息，关闭Instance，然后将Instance的镜像我传送到目标节点上。nova-compute尝试通过ssh在目标节点上的Instance目录里touch一个临时文件，然后关闭Instance，将Instance镜像文件通过scp传送到目标节点；（注意，跨节点复制文件时，必须保证nova-compute进程的启动用户能够在计算节点之间无密码访问）
5. 在目标节点上启动Instance，过程与lauch Instance非常类似；
6. 此时的Instance处于“Confirm or Revert Resize/Migrate”状态，需要用户确认或退回当前的迁移操作。
7. 当按下Confirm按钮后，源计算节点会删除Instance的目录，并在Hypervisor上删除Instance；
8. 当按下Revert按钮后，在目标节点上关闭Instance，删除Instance没目录，并在Hypervisor上删除Instance，在源节点上启动Instance即可。’

#### 4.Live Migrate

Migrate操作会将Instance停止，即冷迁移。Live Migrate是热迁移，Instance不会停机。Live Migrate分为两种：

1. 源和目标节点没有共享存储，Instance在迁移的时候需要将其镜像文件从源节点传到目标节点，这叫块迁移；
2. 源和目标节点共享存储，Instance的镜像文件不需要迁移，只需要将Instance的装填迁移到目标节点；

源和目标节点需要满足一定的条件才能实现Live Migrate：

1. 源和目标节点的CPU类型一致；
2. 源和目标节点的Libvirt版本一致；
3. 源和目标节点能够互相识别对方的主机名称，如在/etc/hosts中加入对方的条目；
4. 在源和目标节点的/etc/nova/nova.conf中指明在线迁移时使用 TCP协议；
5. 由于Instance使用config driver保存其metadata，因此在块迁移的过程中，该config driver也需要迁移到目标节点，由于目前的libvirt只支持vfat类型的config driver，所以必须在/etc/nova/nova.conf指出lauch instance时创建vfat类型的config driver；；
6. 源和目标节点的libvirt TCP远程监听服务需要打开；

共享存储有多种方式实现，如NFS服务器，NAS服务器，分布式文件系统都可以实现。不需要传输镜像文件，只需要传输Instance的状态，速度比块迁移快很多。

### c）计划外的故障处理（文件损坏，硬件故障）

#### 1.Rescue/Unrescue

当操作系统故障时，无法重启，我们可以使用一张系统盘先将系统引导起来，然后再尝试恢复，这种处理方式适用于不太严重的故障，称为Rescue。

Rescue用指定的Image作为启动盘引导In斯坦猜测，将Instance本身的系统盘作为第二个磁盘挂载到操作系统上。

1. 向nova-api发送请求，nova默认使用Instance部署时使用的Image；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息，关闭Instance，通过Image创建新的引导盘，命名为disk.rescue，启动Instance。当Rescue执行成功后，可以通过virsh edit<instance>查看Instance的XML文件，可以看到disk.rescue是启动盘vda，原始的启动盘disk是第二个磁盘vdb。

修复完毕后，使用Unrescue操作从原启动盘重新引导Instance。

#### 2.Rebuild

Rebuild操作可以恢复snapshot，用snapshot替换当前Instance的镜像文件，同时保持Instance的其他，诸如网络，资源分配属性不变。

1. 向nova-api发送请求，选择用于恢复的Image；
2. nova-api向RabbitMQ发送消息；
3. 节点的nova-compute从rabbitMQ中获取消息，关闭Instance，下载新的Image，并准备Instance的镜像文件，启动Instance；

#### 3.Evacuate

宿主机损坏，怎样恢复宿主机上的Instance。通过Evacuate操作，在nova-compute无法工作的情况下将节点的Instance迁移到其他的计算节点上，前提是Instance的镜像文件必须放在共享存储上。

## （四）OpenStack日志阅读

OpenStack的日志记录了详细的信息，对于我们排错有很大的帮助。

### 1.目录位置

对于非devstack安装的OpenStack，日志一般放在/var/log/xxx目录下。

### 2.日志格式

OpenStack的日志格式都是统一的，如下所示：

<时间戳><日志等级> <Request ID> <日志内容> <源代码位置>

日志戳：日志记录的时间；

日志等级：有INFO，WARNING，ERROR，DEBUG；

Request ID：每个操作都会被分配唯一的Request ID，便于查找；

日志内容：日志的主体，记录当前执行的操作和结果；

源代码位置：日志代码的位置，包括方法名称，源代码文件的目录位置和行号；

### 3.查看日志

首先需要掌握OpenStack的运行机制，才能针对性地查看日志，不然日志内容太多容易让人眼花缭乱。分析清楚一个操作的运行流程，然后再到相应的节点上查看日志。

可以先确定大的范围，如在操作之前用tail -f 打印日志文件，这样需要查看的日志一定在操作之后打印的内容中；

另外也可以通过时间戳来确定需要的日志范围；