

本节讨论 nova-compute，并详细分析 instance 部署的全过程。

先给大家道个歉：今天这篇文章的篇幅比以往要多一些，本来想分两次发，但考虑到文章的完整和系统性，还是一次发了出来，这次可能要超出 5 分钟了，大家见谅。  
  
nova-compute 在计算节点上运行，负责管理节点上的 instance。

OpenStack 对 instance 的操作，最后都是交给 nova-compute 来完成的。

nova-compute 与 Hypervisor 一起实现 OpenStack 对 instance 生命周期的管理。

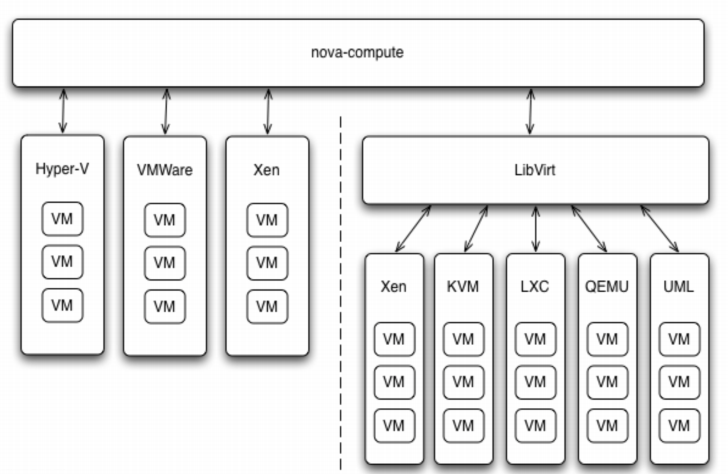
**通过 Driver 架构支持多种 Hypervisor**

接着的问题是：现在市面上有这么多 Hypervisor，nova-compute 如何与它们配合呢？

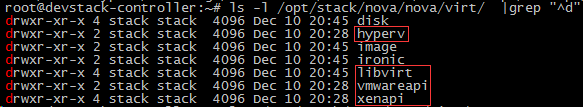
这就是我们之前讨论过的 Driver 架构。

nova-compute 为这些 Hypervisor 定义了统一的接口，Hypervisor 只需要实现这些接口，就可以 Driver 的形式即插即用到 OpenStack 系统中。

下面是Nova Driver的架构示意图



我们可以在 /opt/stack/nova/nova/virt/ 目录下查看到 OpenStack 源代码中已经自带了上面这几个 Hypervisor 的 Driver



某个特定的计算节点上只会运行一种 Hypervisor，只需在该节点 nova-compute 的配置文件 /etc/nova/nova.conf 中配置所对应的 compute\_driver 就可以了。

在我们的环境中因为是 KVM，所以配置的是 Libvirt 的 driver。

image145.png  
nova-compute 的功能可以分为两类：

1. 定时向 OpenStack 报告计算节点的状态
2. 实现 instance 生命周期的管理

下面我们依次介绍。

**定期向 OpenStack 报告计算节点的状态**

前面我们看到 nova-scheduler 的很多 Filter 是根据算节点的资源使用情况进行过滤的。

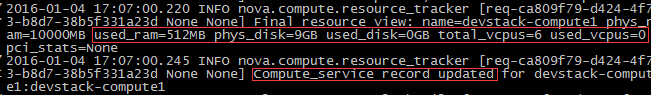
比如 RamFilter 要检查计算节点当前可以的内存量；CoreFilter 检查可用的 vCPU 数量；DiskFilter 则会检查可用的磁盘空间。

那这里有个问题：OpenStack 是如何得知每个计算节点的这些信息呢？

答案就是：nova-compute 会定期向 OpenStack 报告。

从 nova-compute 的日志 /opt/stack/logs/n-cpu.log 可以发现：

每隔一段时间，nova-compute 就会报告当前计算节点的资源使用情况和 nova-compute 服务状态。



如果我们再深入问一个问题：nova-compute 是如何获得当前计算节点的资源使用信息的？

给大家一分钟自己先思考一下？

好，揭晓答案。

要得到计算节点的资源使用详细情况，需要知道当前节点上所有 instance 的资源占用信息。

这些信息谁最清楚？

当然是 Hypervisor。

大家还记得之前我们讨论的 Nova Driver 架构吧，nova-compute 可以通过 Hypervisor 的 driver 拿到这些信息。

举例来说，在我们的实验环境下 Hypervisor 是 KVM，用的 Driver 是 LibvirtDriver。

LibvirtDriver 可以调用相关的 API 获得资源信息，这些 API 的作用相当于我们在 CLI 里执行 virsh nodeinfo、virsh dominfo 等命令。

**实现 instance 生命周期的管理**

OpenStack 对 instance 最主要的操作都是通过 nova-compute 实现的，包括 instance 的 launch、shutdown、reboot、suspend、resume、terminate、resize、migration、snapshot 等。

本小节重点学习 nova-compute 如何实现 instance launch（部署）操作，其它操作将会在后面的章节讨论。

当 nova-scheduler 选定了部署 instance 的计算节点后，会通过消息中间件 rabbitMQ 向选定的计算节点发出 launch instance 的命令。

该计算节点上运行的 nova-compute 收到消息后会执行 instance 创建操作。

日志 /opt/stack/logs/n-cpu.log 记录了整个操作过程。

nova-compute 创建 instance 的过程可以分为 4 步：

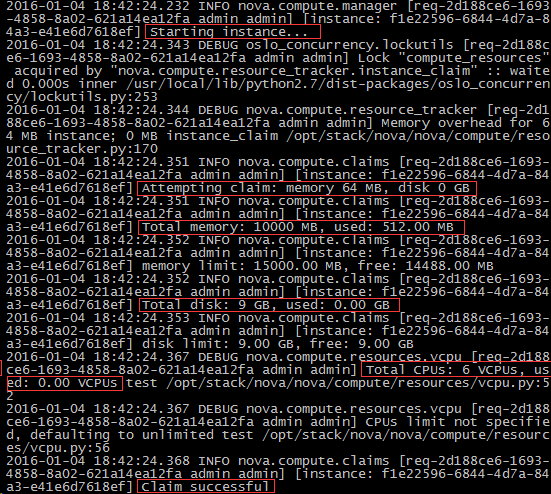
1. 为 instance 准备资源
2. 创建 instance 的镜像文件
3. 创建 instance 的 XML 定义文件
4. 创建虚拟网络并启动虚拟机

下面我们依次讨论每个步骤。

**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。

可以在日志中看到这些细节



网络资源也会提前分配。

http://7xo6kd.com1.z0.glb.clouddn.com/upload-ueditor-image-20160501-1462109863234036929.jpg

**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。

OpenStack 启动一个 instance 时，会选择一个 image，这个 image 由 Glance 管理。

nova-compute会：

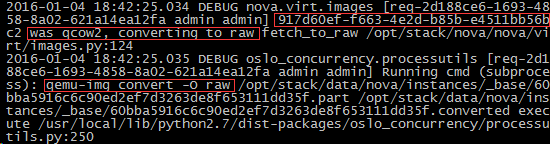
1. 首先将该 image 下载到计算节点
2. 然后将其作为 backing file 创建 instance 的镜像文件。

**从 Glance 下载 image**

nova-compute 首先会检查 image 是否已经下载（比如之前已经创建过基于相同 image 的 instance）。如果没有，就从 Glance 下载 image 到本地。

由此可知，如果计算节点上要运行多个相同 image 的 instance，只会在启动第一个 instance 的时候从 Glance 下载 image，后面的 instance 启动速度就大大加快了。

日志如下：



可以看到

1. image（ID为 917d60ef-f663-4e2d-b85b-e4511bb56bc2）是 qcow2 格式，nova-compute 将其下载，然后通过 qemu-img 转换成 raw 格式。 转换的原因是下一步需要将其作为 instance 的镜像文件的 backing file，而 backing file不能是 qcow2 格式。
2. image 的存放目录是 /opt/stack/data/nova/instances/\_base，这是由 /etc/nova/nova.conf 的下面两个配置选项决定的。

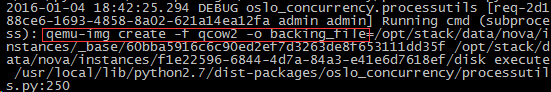
instances\_path = /opt/stack/data/nova/instances

base\_dir\_name = \_base

1. 下载的 image 文件被命名为 60bba5916c6c90ed2ef7d3263de8f653111dd35f，这是 image id 的 SHA1 哈希值。

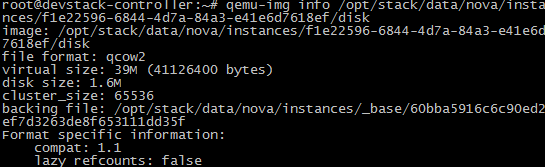
**为 instance 创建镜像文件**

有了 image 之后，instance 的镜像文件直接通过 qemu-img 命令创建，backing file 就是下载的 image。



这里 instance 的镜像文件位于 /opt/stack/data/nova/instances/f1e22596-6844-4d7a-84a3-e41e6d7618ef/disk，格式为 qcow2，其中 f1e22596-6844-4d7a-84a3-e41e6d7618ef 就是 instance 的 id。

可以通过 qemu-info 查看 disk 文件的属性

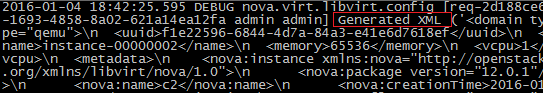


这里有两个容易搞混淆的术语，在此特别说明一下：

1. image，指的是 Glance 上保存的镜像，作为 instance 运行的模板。 计算节点将下载的 image 存放在 /opt/stack/data/nova/instances/\_base 目录下。
2. 镜像文件，指的是 instance 启动盘所对应的文件
3. 二者的关系是：image 是镜像文件 的 backing file。image 不会变，而镜像文件会发生变化。比如安装新的软件后，镜像文件会变大。

因为英文中两者都叫 “image”，为避免混淆，我们用 “image” 和 “镜像文件” 作区分。

**创建 instance 的 XML 定义文件**



创建的 XML 文件会保存到该 instance 目录 /opt/stack/data/nova/instances/f1e22596-6844-4d7a-84a3-e41e6d7618ef，命名为 libvirt.xml

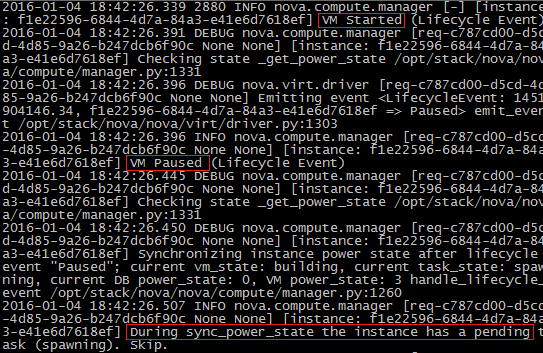
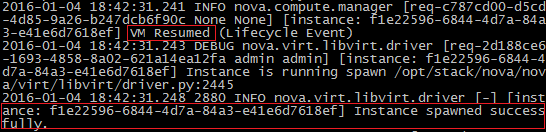
**创建虚拟网络并启动 instance**

接下来便是为 instance 创建虚拟网络设备

http://7xo6kd.com1.z0.glb.clouddn.com/upload-ueditor-image-20160501-1462109863768082566.jpg

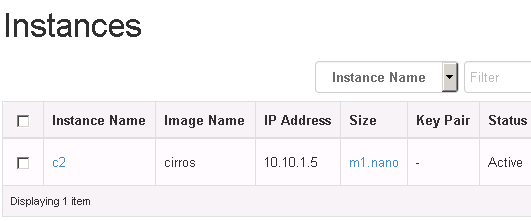
本环境用的是 linux-bridge 实现的虚拟网络，在 Neutron 章节我们会详细讨论 OpenStack 虚拟网络的不同实现方式。

一切就绪，接下来可以启动 instance 了。

至此，instance 已经成功启动。

OpenStack 图形界面和 KVM CLI 都可以查看到 instance 的运行状态。



http://7xo6kd.com1.z0.glb.clouddn.com/upload-ueditor-image-20160501-1462109864369078950.jpg

在计算节点上，instance 并不是以 OpenStack上 的名字命名，而是用 instance-xxxxx 的格式。