**文件编号：**

**实训平台产品线**

**管控系统**

**虚拟机创建源码分析报告**

**中国科学院计算技术研究所**

**无线通信技术研究中心**

**实训平台产品线**

**管控系统组**

修改记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件编号 | 版本号 | 拟制人/  修改人 | 拟制/修改日期 | 更改理由 | 主要更改内容  （写要点即可） |
|  | V1.0 | 任春卉 | 2017/06/12 | 无 | 无 |
|  |  |  |  |  |  |
| 注1：每次更改归档文件（指归档到组内及研究室的文件）时，需填写此表。  注2：文件第一次归档时，“更改理由”、“主要更改内容”栏写“无”。 | | | | | |

修改列表：

后续版本对之前版本所做修改逐条说明清楚。

本文档的程序或内容受版权法的保护，未经中科院计算所&中科晶上公司的书面许可，不得擅自泄漏、拷贝或复制本文档资料的全部或部分。

# 整体描述

Openstack作为一个虚拟机管理平台，核心功能自然是虚拟机的生命周期的管理，而负责虚机管理的模块就是Nova。本文详细地描述openstack创建虚拟机的完整过程，从客户端发起请求到虚拟机成功运行，包括客户端向keystone发送HTTP 请求、keystone完成的功能、keystone通过验证返回token\_id和serviceCatalog、客户端向nova\_api发送虚拟机创建请求、nova\_api接收HTTP请求、nova-api处理模块、nova-schduler处理模块、nova-compute处理模块、nova-compute处理模块两个定时任务、nova-driver处理虚拟机创建。对于每一个模块在创建虚拟机的过程中所负责的功能和执行的操作，进行较为详细描述和讨论。

# 预备知识

## 什么是policy，它是用来做什么的？

在openstack的用户管理中，有三个概念：Users, Tenants, Roles。简单来说，policy就是用来控制某一个User在某个Tenant中的权限的。这个User能执行什么操作，不能执行什么操作，就是通过policy机制来实现的。直观的看，policy就是一个json文件，位于/etc/[SERVICE\_CODENAME]/policy.json中，每一个服务都有一个对应的policy.json文件，通过配置这个文件，实现了对User的权限管理。  
 另外就是还有一个角色(role)的概念，这个概念肯定都很熟悉了，是权限的集合，可以将role赋予某个user，使这个user拥有相应的权限，方便用户权限管理。policy.json文件可以在role的级别配置，不过默认的配置的角色只有admin，如果需要配置其他的角色，需要自己创建，然后在policy.json中进行配置。

## 什么是quota(配额管理)?

简单的讲就是控制用户资源的数量。在**[OpenStack](http://lib.csdn.net/base/openstack" \o "OpenStack知识库" \t "_blank)**里，管理员为每一个工程(project)分配的资源都是有一定限制的，这些资源包括实例（instance）、cpu、内存、存储空间等等，不能让一个工程无限制的使用资源，所以配额管理针对的单位是工程(project)。

管理员给某个工程的资源配额假如最多创建10个实例，最多使用20个vcpu，最多使用5G的内存，只要达到某一个资源的使用上限，就会出现异常，这就是配额管理。

# 虚拟机创建流程源码分析

## 客户端向keystone发送HTTP 请求

一个最简单的创建虚拟机指令如下：nova boot vm\_name --flavor flavor\_id --image image\_uuid，向keystone发送HTTP请求(keystone的服务地址：nova命令一般通过环境变量OS\_AUTH\_URL设置，该变量可通过horizon下载RC文件，RC文件中进行设置)，url为http://localhost:5000/v2.0/tokens 。

## Keystone接收HTTP请求及模块内完成的功能

在客户端发起创建虚拟机请求时，keystone需要对客户端的用户名和密码进行验证。keystone-all与nova-api一样，api的发布没有采用任何框架，而是使用router、paste类库从头写的，但实现风格上又与nova-api有点差异 。keystone-all服务会监听两个端口：localhost:5000，即publicURL，一般用户使用密码可以访问;localhost:35357，即adminURL，只能使用admin账户和密码访问。

http://localhost:5000/v2.0/tokens，客户端请求该api来获取token\_id和serviceCatalog。keystone会将该**api的请求交给keystone.TokenController.authenticate()处理**。该函数主要完成：对于客户端传过来的name、password、tenant\_id进行验证。在keystone的[数据库](http://www.2cto.com/database/" \t "http://www.2cto.com/os/201701/_blank)user表中保存了user相关信息，包括password[加密](http://www.2cto.com/article/jiami/" \t "http://www.2cto.com/os/201701/_blank)后的hash值。password的加密使用了sha512算法，由passlib库提供。密码验证如下：passlib.hash.sha512\_crypt.verify(password, hashed)。

**根据CONF.signing.token\_format配置项，为客户端的每一次请求生成一个token\_id。**目前支持的选择有两个UUID、PKI，默认为UUID。所以在默认情况下，一个token\_id就是一个随机产生的uuid。token\_id有一个有效时间，从token\_id的生成的时刻开始算起。有效时间可通过CONF.token.expiration配置项设置，默认值为86400s（24小时）。

**构建serviceCatalog**，这里面就是一堆endpoints，包括nova-api、glance-api、cinder-api等。根据配置项CONF.catalog.driver，可以从数据库的endpoint表中获取，也可以从模板文件中获取。

**验证通过，keystone则返回token\_id和serviceCatalog。**身份验证通过后，**客户端才可对nova-api发送创建请求。**因为客户端只配置了keystone的服务地址和端口，对于nova-api的服务地址它是不知道的，所以需要先通过keystone验证，然后由keystone将**nova-api的服务地址放在****serviceCatalog(context.py中又如何获取serviceCatalog的方法)**中返回给它。其实serviceCatalog中不仅仅包含nova-api的服务地址，还有glance-api、cinder-api等服务的地址，这些地址在以后的镜像下载、块存储请求时会用到。token\_id被放在请求nova-api的headers中，用作nova-api验证。

## 客户端向nova\_api发送虚拟机创建请求

客户端向nova-api发送满足REST API 的HTTP创建请求，需要有url请求地址，这个url地址需要通过keystone验证后，通过serviceCatalog 中返回给客户端。

创建虚拟机请求的api是：

http://localhost:8774/v2/project\_id/servers。由osapi\_compute服务公布。该api是RESTFUL的，以POST方法请求该api。经过几层middleware处理后，终于交给nova.api.openstack.compute.servers.Controller.create()处理。

http://localhost:35357/v2.0/tokens/token\_id，对请求的token\_id进行验证

## nova\_api接收HTTP请求（HTTP请求映射到Controller对象过程）

如果nova-api想要接收发送过来的满足REST的HTTP 的请求，肯定需要有一个HTTP服务器的，那么[openstack](http://lib.csdn.net/base/openstack" \t "_blank" \o "OpenStack知识库)是如何实现的服务器创建并对外发布服务的呢？

首先程序会找到nova.cmd下的api.py的main()并执行启动。在启动HTTP服务前，会先创建一个WSGIService对象。

**defmain**():  
      ... ...  
    **for**api **in**CONF.enabled\_apis:  
        should\_use\_ssl = api **in**CONF.enabled\_ssl\_apis  
        **if**api =='ec2':  
            server =service.WSGIService(api, use\_ssl=should\_use\_ssl,  
                                         max\_url\_len=16384)  
        **else**:  
            server =service.WSGIService(api, use\_ssl=should\_use\_ssl)  
         ......  
    launcher.wait()

它是基于WSGI标准的服务器类,在启动服务时，启动服务时，会将这个WSGIService对象当做参数传过去。

launcher.launch\_service(server, workers=server.workers **or** 1)

那么这个WSGIService中又引用了哪些类，实现了哪些功能的呢？

WSGIService在构造函数中，通过paste.deploy解析paste.ini配置文件并根据参数name（例如osapi\_compute），经过一系列的middleware（fliter）中间件进行处理，最终走到一个application，并将这个加载的app以及一些host、port作为创建 nova.wsgi.py的Server类的参数，最终nova就是通过这个server类以协程方式来对外提供服务。

那么这个application是怎么来的呢？nova.api.openstack.compute.APIRouter.factory()返回的一个APIRouter实例，这个实例调用其父类的父类nova.wsgi.APIRouter中的成员变量**mapper来实现url与controller的映射**。根据http请求中的地址找到对应的controller，并实现继承了这个Controller类的子类实例，这个子类实例就是我们要找的具体的application。

到这里我们已经准备好了由server接收HTTP 请求后， 交给这个application去处理。根据请求url我们能找到具体的controller中的具体处理方法，例如我们要实现创建虚拟机，那么就会根据接收过来的请求命令，http://localhost:8774/v2/project\_id/servers 。根据/v2 以及paste.ini 配置文件：

配置文件：

**[composite:osapi\_compute]**use = call:nova.api.openstack.urlmap:urlmap\_factory  
/: oscomputeversions  
/v1.1: openstack\_compute\_api\_v2  
/v2: openstack\_compute\_api\_v2  
/v3: openstack\_compute\_api\_v3

找到对应的composite：

**[composite:openstack\_compute\_api\_v2]**

use = call:nova.api.auth:pipeline\_factory

noauth = faultwrap sizelimit noauth ratelimit osapi\_compute\_app\_v2

keystone = faultwrap sizelimit authtoken keystonecontext ratelimit osapi\_compute\_app\_v2

keystone\_nolimit = faultwrap sizelimit authtoken keystonecontext osapi\_compute\_app\_v2

这里是根据nova.api.auth.pipeline\_factory() 并经过一系列的filter（middleware）进行处理后，最后找到wsgi app ：osapi\_compute\_app\_v2。

**[app:osapi\_compute\_app\_v2]**

paste.app\_factory = nova.api.openstack.compute:APIRouter.factory

据配置返回一个APIRouter实例，这个实例会通过其父类（nova.api.openstack.APIRouter）的父类（nova,wsgi.APIRouter）的成员变量mapper来维护url到controller的映射。最终找到会根据url找到nova.api.openstack.compute.servers.

Controller()类的create()方法。到这里开始正式对创建虚拟机进行处理。

## nova-api处理模块

1. nova.api.openstack.compute.servers.py：Controller.create()

主要任务是从传递过的req中**获取**各种创建虚拟机所需要的**参数信息**，并做验证，然后将获取的一系列参数（image\_uuid,name,insts\_type等）作为nova/compute/api.py中的API类的create()方法参数，进行下一步处理。

1. nova.compute.api.py:API.create():

主要任务是**对check policies、check quota、创建db记录**，**创建下一步中schedule所需要的调度规则信息filter\_properties**，将部分参数整合，然后将创建请求发送到nova/conductor/api.py 中，进行下一步处理。

3．  nova.conductor.api.py:ComputeTaskAPI.bulid\_instance()接收有上一步的命令，转发的rpcapi，进行下一步：nova.conductor.rpcapi.py:ComputeTaskAPI. ComputeTaskAPI()

**创建rabbitmq消息队列的client对象**，以**cast方式发送rpc消息到conductor.manager**（**如何实现发送需细化**）进行下一步：

cctxt = self.client.prepare(version='1.5')

cctxt.cast(context, 'build\_instances',

           instances=instances, image=image\_p ...)

nova.**conductor.manager**.ComputeTaskManager. build\_instances()

1） 创建request\_spec，供下一步schedule中使用。(request\_spec包含虚拟机的类型、数量、uuid等信息)。

2） 调用nova/scheduler/rpcapi.py的run\_instance()方法，将请求发送至scheduler中。

**context**

## nova-schduler处理模块

1.   nova.**scheduler.rpcapi**.py:SchedulerAPI.run\_instance()

**接收到请求后，创建rabbitmq消息队列的client对象，以cast方式发送rpc消息到scheduler.manager**，进行进一步处理：

cctxt = self.client.prepare(version=version)

**return** cctxt.cast(ctxt, 'run\_instance', \*\*msg\_kwargs)

2.   nova.schduler.manager.py : SchedulerManager.run\_instance()

       收到当SchedulerManager实例化的时候，会根据取得的配置文件取得相应的调度器，默认是FilterScheduler调度器。这个调度器是nova.scheduler.filter\_scheduler.

FilterScheduler类的实例。

3.    nova.scheduler.filter\_scheduler.py:FilterScheduler.\_scheduler()

1)    通过self.schedule(),经过过滤和权重筛选，最终得到满足条件可以创建出虚拟机的所有host集合--weighted\_hosts

weighed\_hosts = self.\_schedule(context, request\_spec,

                                       filter\_properties, instance\_uuids)

2)      通过self.\_provision\_resource()来进行**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**的更新操作，并通过调用nova/compute/rpcapi.py的run\_instance()进行进一步处理

## nova-compute处理模块

1.  nova**.compute.rpcapi**:ComputeAPI.run\_instance():

       创建rabbitmq消息队列的client对象，以cast方式发送rpc消息到compute.manager进行下一步处理。

2 nova.**compute.manager.py**:ComputeManager.run\_instance()

根据context中的信息找到\_prebuild\_instance()方法中得到image信息。

image\_meta =self.\_prebuild\_instance(context,instance)

\_prebuild\_instance中的

image\_meta= \_get\_image\_meta(context,instance['image\_ref'])

\_get\_image\_meta():

**def\_get\_image\_meta**(context,image\_ref):  
    image\_service, image\_id = glance.get\_remote\_image\_service(context,  
                                                             image\_ref)  
    **return**image\_service.show(context,image\_id)

**创建glanceClient对象，通过http call到glance中请求镜像并下载到本地**。

**获取网络信息**：(这里是通过nova-network方式获取网络信息，到neutron请求类似于glance)

         通过调用self.\_run\_instance()中的\_build\_instance()获取instance和网络信息network\_info。

instance, network\_info = self.\_build\_instance(context,

        request\_spec, filter\_properties, requested\_networks,

        injected\_files, admin\_password, is\_first\_time, node,

        instance, image\_meta, legacy\_bdm\_in\_spec)

         在build\_instance()通过调用\_allocate\_network()获得网络信息network\_info

network\_info = self.\_allocate\_network(context, instance,

        requested\_networks, macs, security\_groups,

        dhcp\_options)

         通过**调用self.\_spawn()并将network\_info当做参数，进行虚拟机创建**

下面看下如何在\_spawn()方法中创建instance:

self.driver = driver.load\_compute\_driver(self.virtapi, compute\_driver)

**首先会根据虚拟技术（KVM [Docker](http://lib.csdn.net/base/docker" \t "_blank" \o "Docker知识库) vmware等），先获得对应的driver，这里我们是选择kvm，所以driver会找到nova/virt/libvirt/dirver.py**

self.driver.spawn(context, instance, image\_meta,

                  injected\_files, admin\_password,

                   … …)

## nova-compute处理模块两个定时任务详解

nova-scheduler选定host后，随即通过rpc调用，调用host的nova-compute服务的nova.compute.manager.ComputeManager.run\_instance()。由它执行真正的创建虚拟机操作。在此，需要简单介绍下nova-compute的两个定时任务：

1. **update\_available\_resource**。该任务首先**获取当前计算节点的cpu个数、总内存大小、总磁盘大小，**数据由nova.virt.libvirt..driver.LibvirtDriver.get\_available\_resource()获取。然后从数据库中查找到运行在该节点上的所有虚拟机信息，统计出这些虚拟机所使用的vcpu个数、内存大小、磁盘大小。并将计算节点的总资源数量减去虚拟机使用的数量，得到空闲内存大小和空闲磁盘大小。然后更新数据库compute\_node信息。这里需要注意从数据库中获取的虚拟机使用资源数量并不一定是计算节点消耗的资源数量，如1）虚拟机使用磁盘镜像为qcow2格式时，它的大小要小于或等于实际分配的大小；2）也有可能因为数据不同步，使得统计的数据与实际的资源使用不一致。

2. **\_report\_driver\_status**。该**任务是用于定时更新当前计算节点的****capabilities信息**，同样也是通过LibvirtDriver来获取资源数据的，不过在计算已使用资源方面，是直接使用通过调用multiprocessing、libvirt、os等返回的cpu个数、内存、磁盘使用情况。并附加上host\_ip信息。并由定时任务\_publish\_service\_capabilities通过rpc call转发到nova.scheduler.host\_manager.HostManager.service\_states中。

        这两个定时任务为nova-scheduler提供了实时的host信息，所以才能实现准确调度。由于capabilities信息与compute\_node表中信息有很大的相似度，所以调度过程中很少用到。nova-scheduler调度到nova-compute的run\_instance()主要完成什么功能呢：

* 检查虚拟机是否已创建，及instance的image大小是否超过root的大小，超过则报错。
* 更新数据库，将instance的vm\_state=BUILDING，task\_state=NULL状态，horizon上面会有反应，但时间很短。
* 给虚拟机分配资源，包括cpu、内存、磁盘等。更新数据库，设置instance的host字段。
* 分配网络，首先将instance的tast\_state=NETWORKING，然后通过rpc调用nova-network的nova.network.manager.NetworkManager.allocate\_for\_instance()，返回网络信息。nova-network处理的详细将在下面的nova-network模块讨论。
* 建立块设备，将task\_state=BLOCK\_DEVICE\_MAPPING。因为未给分配块设备，这步将不进行操作，有待后面讨论。
* 将task\_state=SPAWNING，如果该类型的instance第一次在该计算节点上创建，该状态要持续几分钟，因为需要下载镜像。
* 根据上面配置的虚拟机信息，生成xml，写入libvirt,xml文件，生成console.log文件。
* 下载镜像，kernel、ramdisk、image，通过glance api下载。它们首先会被放在FLAGS.instances\_path的\_base目录下，然后copy一份到instance的目录下面。这样，如果这个计算节点上创建相同虚拟机时，首先查找\_base中是否已经下载，如果已经下载过了，则直接copy就可以了。对于image，一般采用qcow2格式，作为qemu-img的backing\_file新建一个image使用，这样可以节约空间。
* 向下载过后的磁盘文件，注入指定的内容，如public\_key、/etc/network/interfaces、root密码、指定的文件路径和内容、/etc/vmuuid等。原理比较简单，将下载的image使用mount命令进行挂载，然后将要写入的内容下到特定的位置。
* 最后使用上面生成的xml，调用libvirt创建虚拟机，等待虚拟机正常运行。
* 更新数据库，将instance的vm\_state=ACTIVE、task\_state=None。

## nova-driver处理模块

nova.virt.libvirt.driver.py:LibvirtDriver.spawn()

1）  获取磁盘相关信息

disk\_info = blockinfo.get\_disk\_info(CONF.libvirt.virt\_type, instance,...

2）  创建实例镜像

self.\_create\_image(context, instance,...

3）  为新建的实例参数获取配置数据conf，并把conf 的配置信息文件转化成所需xml格式的文件。

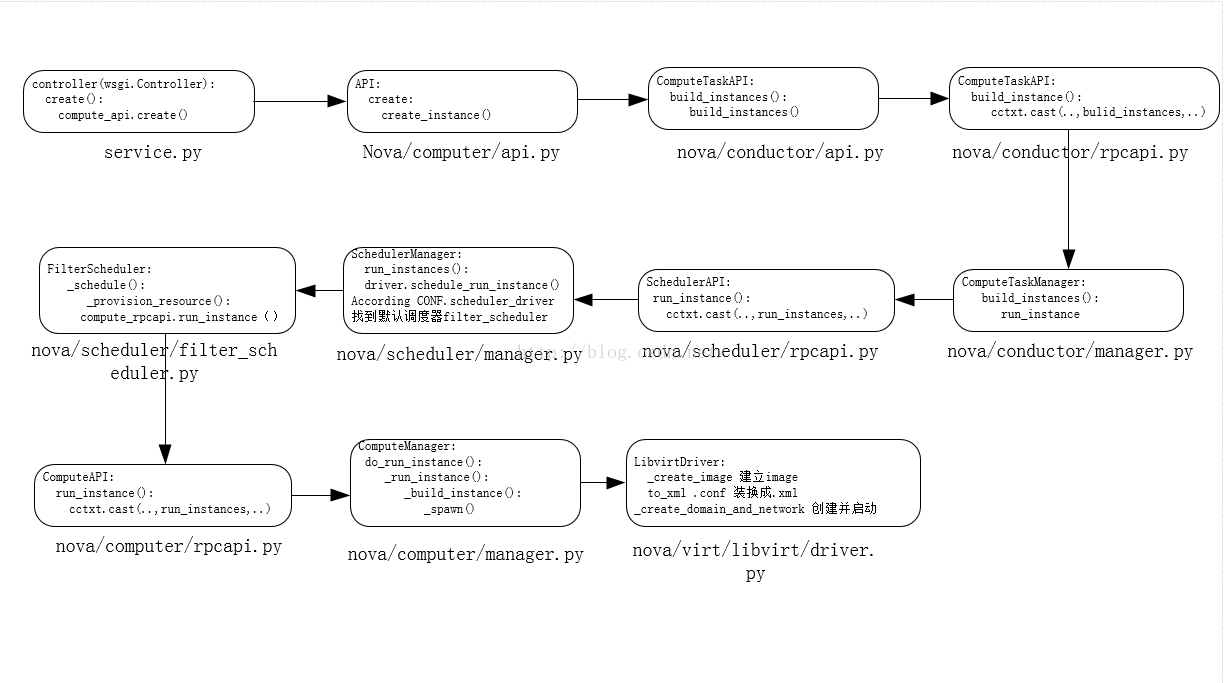
xml = self.to\_xml(context, instance, network\_info ,...

4）  获取网络和域等相关信息

self.\_create\_domain\_and\_network(context, xml, instance, network\_info,...

5）\_wait\_for\_boot函数等待libvirt进行虚拟机启动操作。

# 创建虚拟机函数调用关系



# 结论

初步理清创建虚拟机流程及模块内和模块间调用方法，后续还需要了解各模块内部完成的工作及运行机制，开发新的模块及其API。

# 调研过程记录

按时间推进，将调研过程中所做的主要工作，对外交流的过程，项目总体组对于方案的整体讨论决策，调研方案的推进，或者影响调研方向的主要时间点及结论记录下来。

|  |  |  |  |
| --- | --- | --- | --- |
| **时间** | **所做工作描述** | **参与者** | **主要结论** |
| 2017/06/12 | 初步理清创建虚拟机过程函数调用关系 | 任春卉 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |