13.3 制作OpenStack镜像

OpenStack所提供的虚拟机是通过镜像（image）部署的，可以直接使用官方提供的标准镜像（<https://docs.openstack.org/image-guide/obtain-images.html>），但是这种镜像难以满足生产环境的需求，因此必须定制所需的OpenStack镜像。

制作镜像的方法主要有两种：

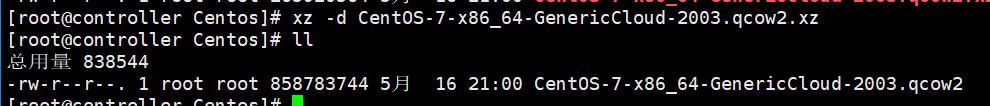
1. 基于标准镜像进行定制；
2. 使用KVM自己制作镜像。

13.3.1 制作OpenStack Linux镜像

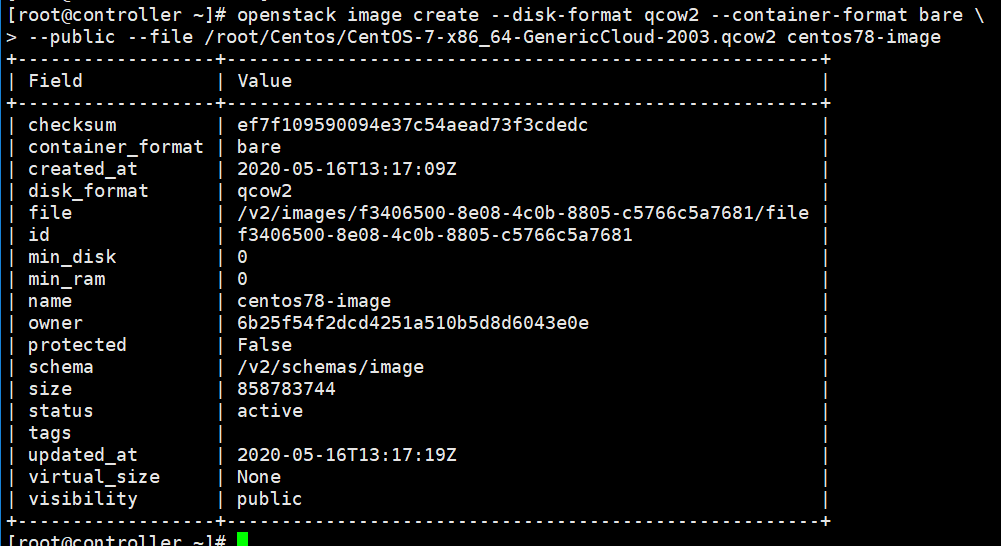
示例：基于官方标准镜像制作OpenStack Linux镜像，基本方法是首先基于标准镜像创建一个实例，然后对实例进行修改定制，最后基于该实例创建一个快照。

1. 下载镜像[CentOS-7-x86\_64-GenericCloud-2003.qcow2](http://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud-2003.qcow2)并创建Glance镜像

如果下载的镜像为CentOS-7-x86\_64-GenericCloud-2003.qcow2.xz的压缩文件，则需先解压：



通过CLI方式或者web方式创建镜像。



由于下载的标准镜像，没有图像界面，默认时区为美国时区，且只能通过SSH密钥对登录，所以需要对该镜像镜像定制，如设置中国时区和语言，设置SSH登录等。

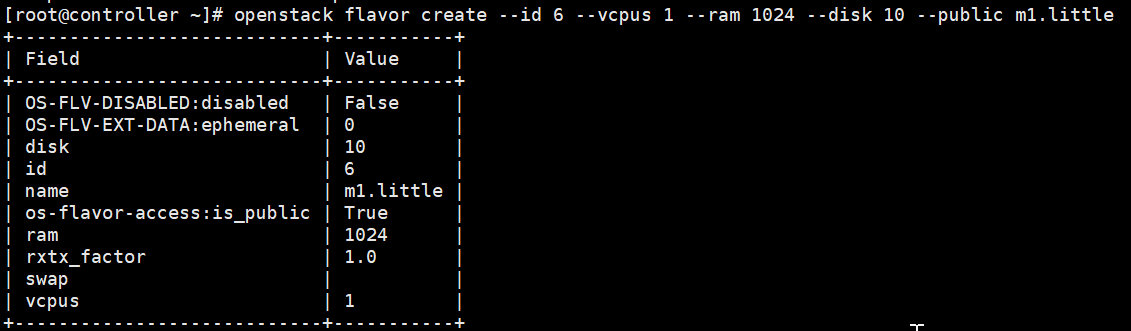
1. 基于[CentOS-7-x86\_64-GenericCloud-2003.qcow2](http://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud-2003.qcow2)创建一个实例

admin用户登录dashboard基于刚创建的centso78-image创建一个实例。

Centos最小化安装要求：

[https://docs.centos.org/en-US/centos/install-guide/Steps-x86/#sect-installation-planning-disk-space-memory-x86](https://docs.centos.org/en-US/centos/install-guide/Steps-x86/" \l "sect-installation-planning-disk-space-memory-x86)

所以需要新建一个虚机flavor，内存为1G，硬盘10GiB，1个vcpu。



1. 详细信息



1. 源



1. Flavor



1. 网络



1. 安全组



1. 密钥对



1. 创建实例



1. 对实例进行定制
2. 登录

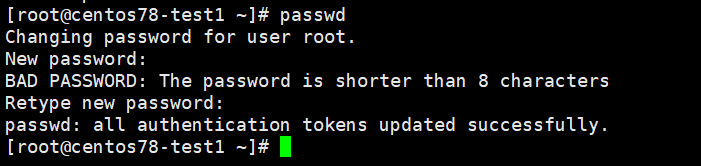
OpenStack提供的CentOS镜像内置了用户centos，实例绑定了浮动IP为192.168.10.154,使用私钥通过SSH登录该虚拟机实例。

密钥对为my-key，私钥文件位于controller节点上/root/.ssh/id\_rsa。



1. 切换root用户，并设置root密码

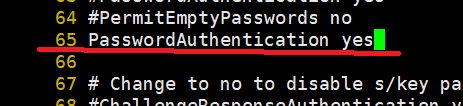




两次输入密码确认。

1. 更改ssh登录方式





重启不中断ssh服务



1. 更改时区

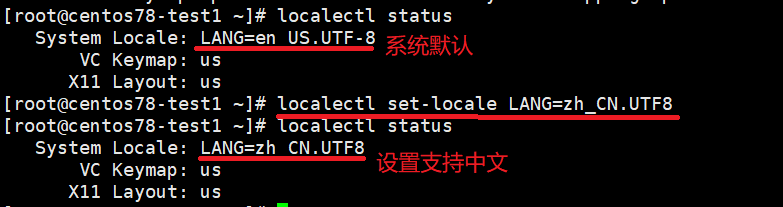


由上可以看出系统默认时间为UTC时间，更改为CST时间。





1. 设置系统支持中文

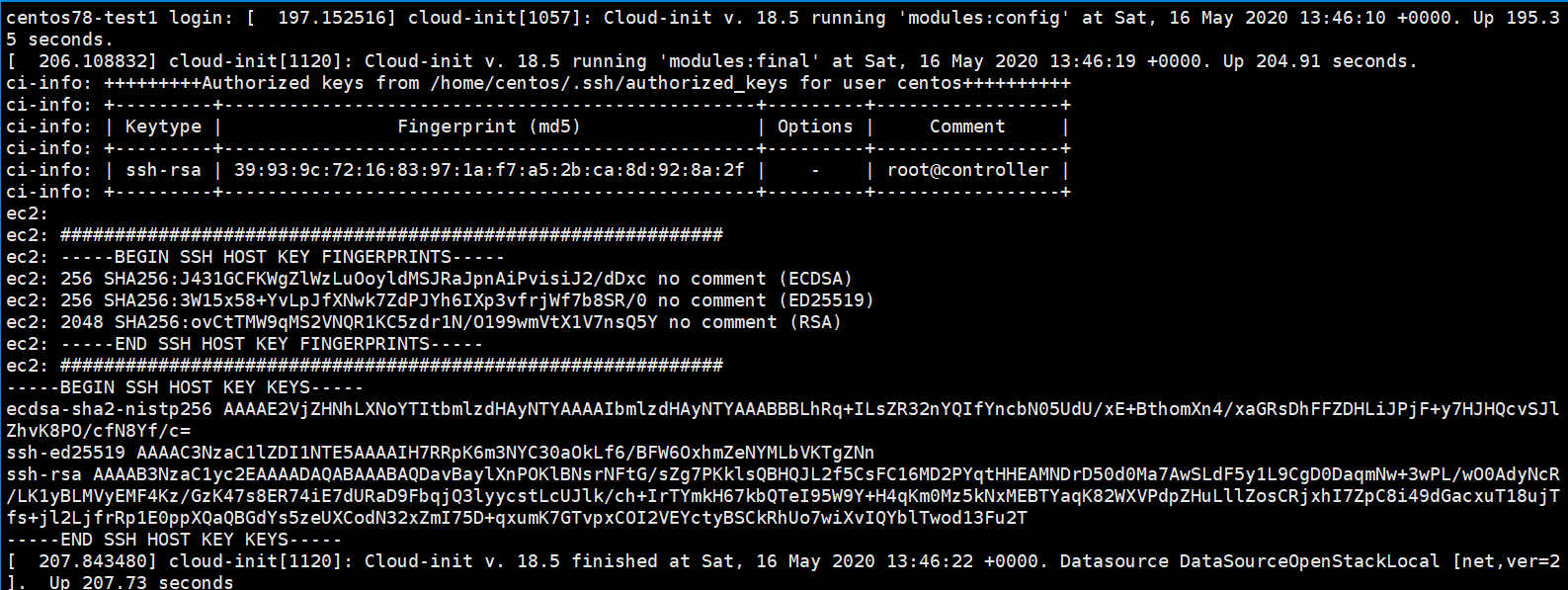


1. 安装一些有用的工具软件包



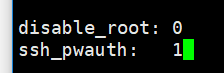
1. 定制cloud-init初始化行为

Linux的镜像都预置了cloud-init软件，由cloud-init负责实例的初始化工作。如下图为利用标准镜像创建实例的cloud-init初始化信息：



创建的实例默认不允许root用户直接登录实例，为了让root可以直接登录实例，需将/etc/cloud/cloud.cfg文件中的，disable\_root参数设为0，并将ssh\_pwauth参数值设为1，以启用SSH密码方式登录（默认只允许私钥通过SSH登录）。





1. 创建快照

重启上述实例，并创建快照。

1. 重启后xshell root登录测试：



登录成功。

1. 创建快照







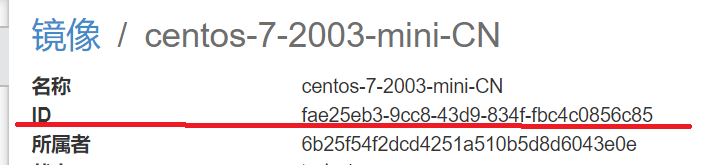
1. 测试实例快照

基于以上实例快照，创建实例。以root用户登录，查看定制功能。

1. 将快照转换成镜像

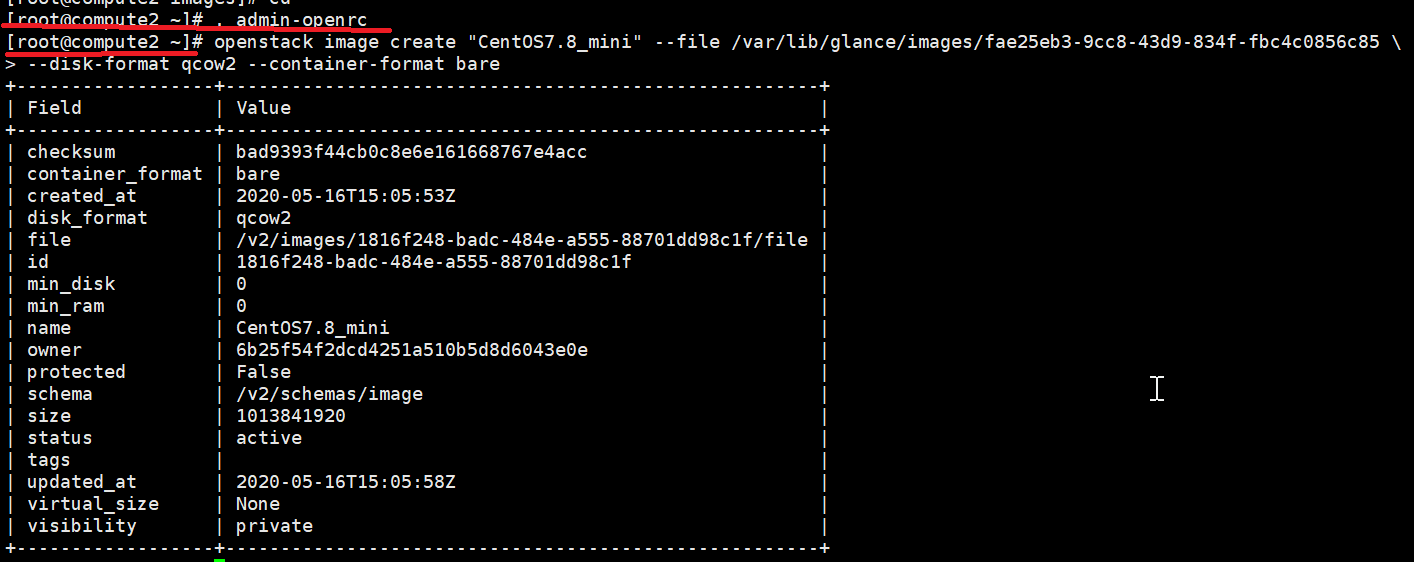
Glance的镜像有两类：镜像（image）和快照（Snapshot）。生成环境中一般使用镜像，所以需将创建的快照转化成为镜像。

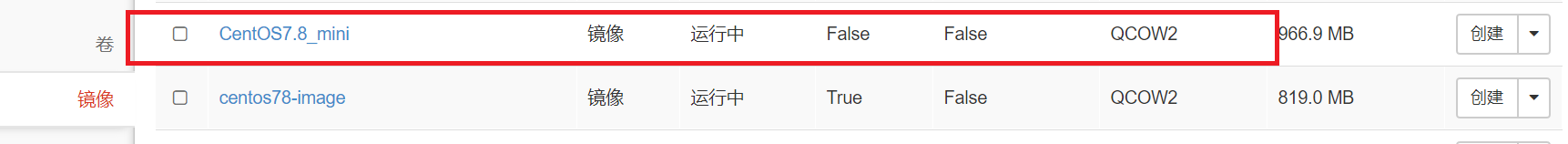
先查看实例ID，再获取实例快照的文件路径，最后利用openstack image create命令转化为镜像。



那么实例快照文件路径为：

/var/lib/glance/images/fae25eb3-9cc8-43d9-834f-fbc4c0856c85。注意glance服务已经迁移到compute2上，则在compute2上创建镜像。





OpenStack的实例是基于镜像部署的，镜像中包含了操作系统、最常见的软件（SSH），以及最通用的配置（如网卡设置）。在实际应用中，在创建实例的时候通过要对实例进行一些额外的自定义配置都加到镜像中，而每次部署虚拟机实例都要制作定制化的镜像，不仅费时费力，而且大量镜像不便于管理，违背了镜像本来是作为模板提供通用配置的初衷。当然也可以像上述示例中在安装完镜像后，手工完成相应的个性化设置，但是这不符合云服务自动化的基本要求。可行的方案是通过向虚拟机实例注入元数据信息，实例启动时获得自己的元数据，实例中的cloud-init（或cloudbase-init）工具根据元数据完成个性化配置工作。

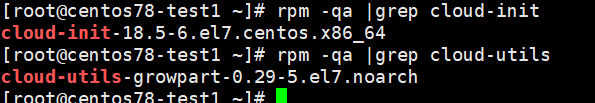
13.3.2 cloud-init工作机制及其应用

Cloud-init是一组Python脚本的集合，是一个能够定制云镜像（cloud images）的实用工具。可以完成默认区域设置、主机名设置、用户密码和SSH密钥注入、网络设备配置、临时装载点设置、软件包安装等虚拟机初始化任务。这些功能是通过修改/etc/cloud/cloud.cfg配置文件来实现的。

Cloud-init一般会被包含在用于启动虚拟机实例的镜像文件中，基于该镜像文件部署虚拟机实例，cloud-init随虚拟机启动而自动启动，对虚拟机进行自定义的初始化配置。它目前支持Ubuntu、Fedora、Debian、RHEL、CentOS等主流的Linux发行版。

1. 安装cloud-init

OpenStack在制作镜像时一定要安装cloud-init软件。否则无法实现元数据注入。OpenStack提供的官方Linux云镜像中大多预装cloud-init。



默认已经安装了cloud-init。Cloud-utils-growpart是管理磁盘分区的软件包，用来实现分区自动扩展。该功能仅在linux内核版本高于3.8才能支持，因此Centos6的cloud-init安装需执行以下命令变通处理。

yum -y install cloud-init cloud-utils-growpart dracut-modules-growroot

dracut -f ###该命令将growroot的脚本封装到initramfs里，便于系统启动时initramfs利用growpart命令对根分区进行扩展。内核启动完成后cloud-init服务会自动扩展文件系统。

1. 数据源

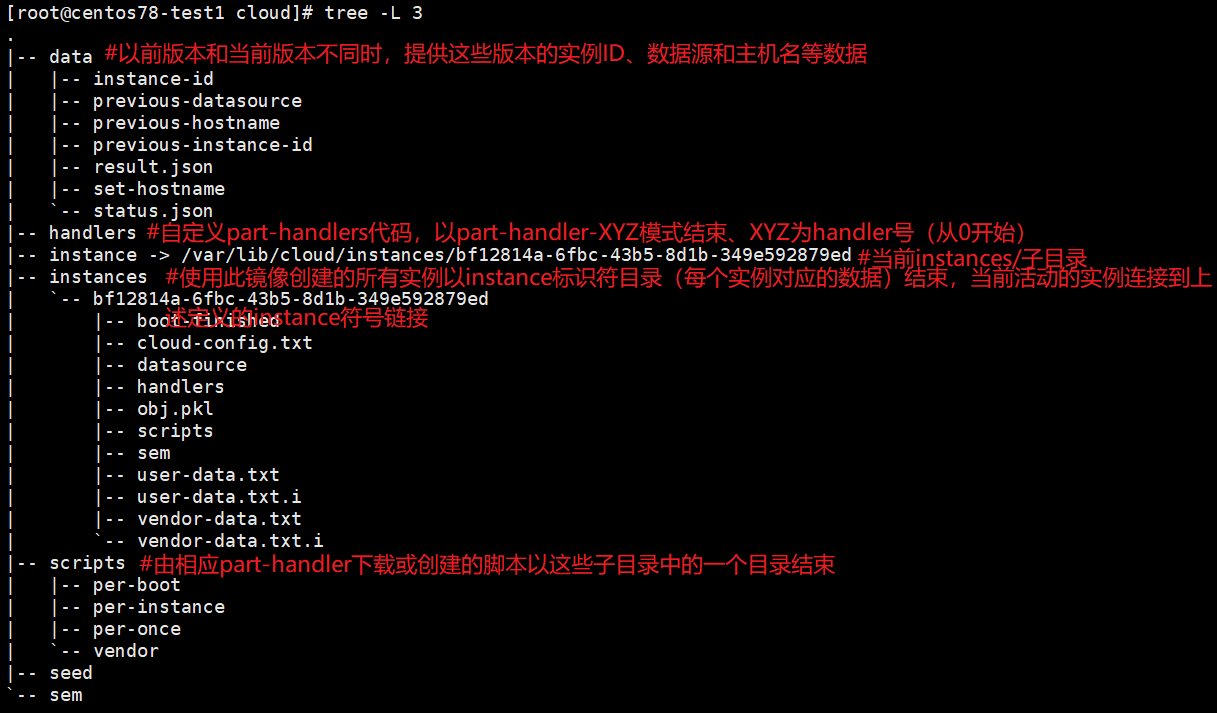
数据源是指用户cloud-init的配置数据，包括用户数据Userdata，和来自创建配置驱动器的栈称为元数据（Metadata）

* Userdata：files，yaml和Shell脚本。其中cloud config files以“#cloud-config”开头，而shell脚本以“#！”开头。
* Metadata：服务器名（Server Name）、实例ID（instance ID）、显示名称（Display Name）和云的其它细节。

Cloud-init以JSON文件存储所有的元数据、提供商数据（Vendordata）、和用户数据。

1. Cloud-init目录

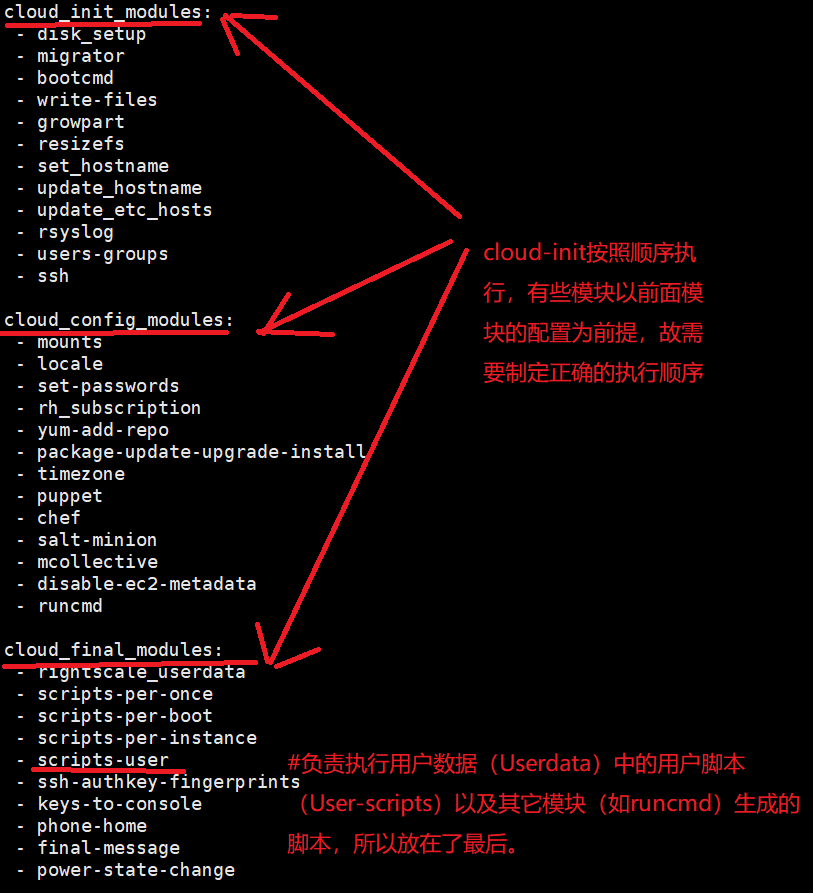
cloud-init的目录位于/var/lib/cloud下。



1. Cloud-init配置文件

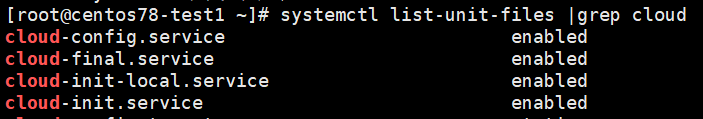
使用cloud-init主要是在镜像制作过程中定义配置文件/etc/cloud/cloud.cfg。

该文件主要分成两部分：位于开头的参数和变量定义部分和模块列表部分。各模块在运行时会根据之前定义的变量和参数值，或者通过数据源获取的用户数据来配置虚拟机实例。



1. Cloud-init工作过程

Cloud-init是驻留在虚拟机实例中的一个代理程序，它仅仅在系统启动时运行，不会常驻系统中。与cloud-init相关的服务显示如下：



为实现定制功能。cloud-init必须以适当的受控方式整合到实例的系统启动中。它在实例启动的运行过程分为4个阶段。

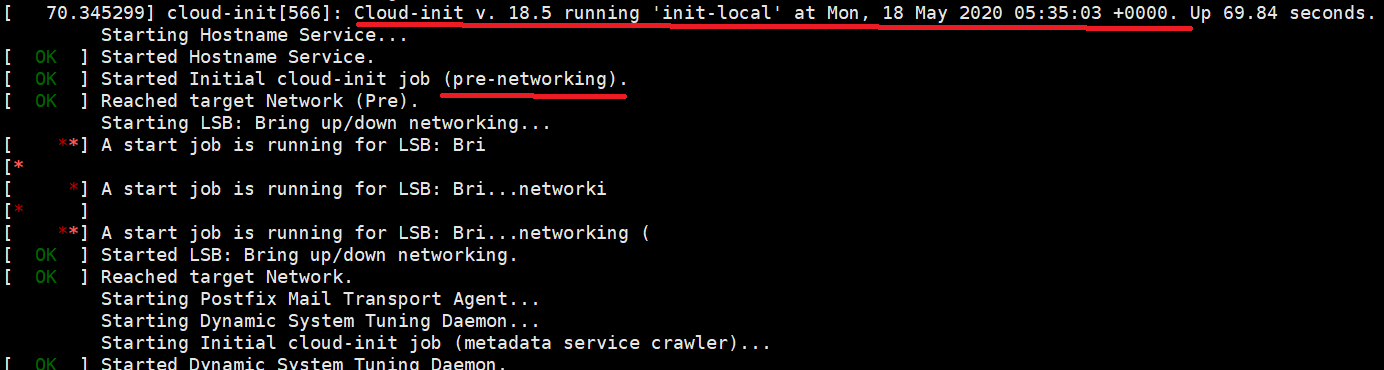
在systemd（Linux最新的系统启动初始化方式）支持的启动过程中会运行一个生成器来决定cloud-init.target目标是否被嵌入启动（Boot）中。默认将启用该生成器。

若存在文件/etc/cloud/cloud-init.disabled和有“cloud-init=disabled”这样的内核命令行时则不启用该生成器。

1. Init-local

systemd服务：cloud-init-local.service。

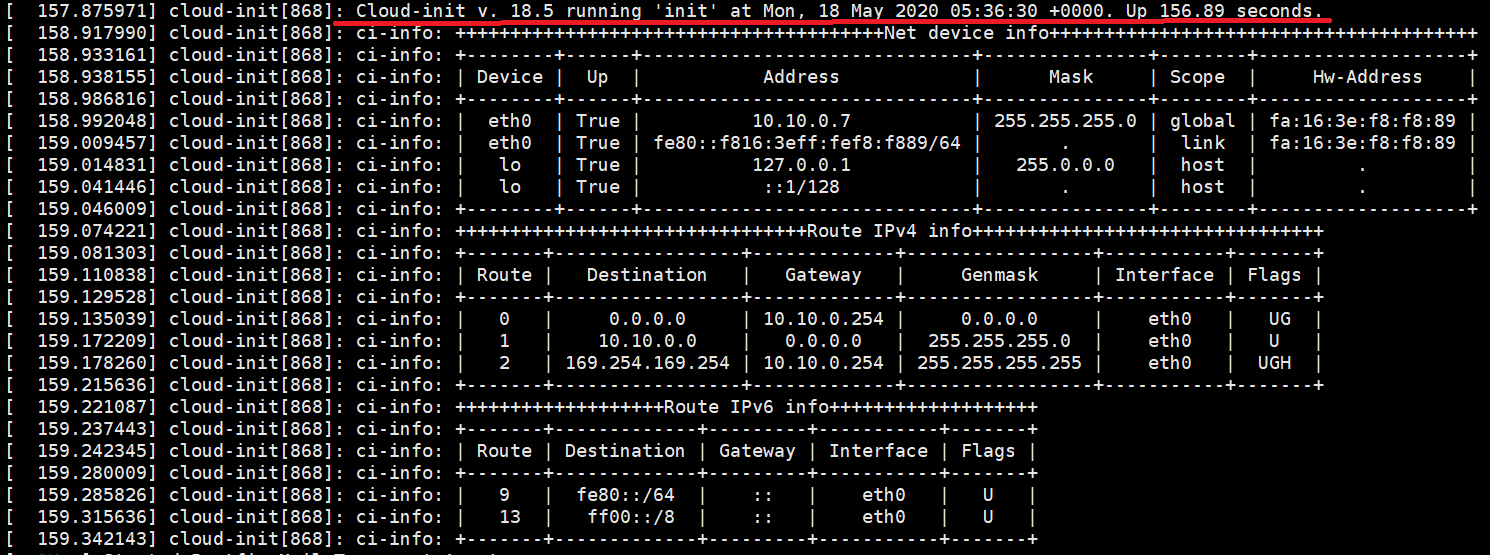
运行到该阶段时，虚拟机本身并不知道应该如何去配置网络。那么虚拟机根据数据源DataSource来进行网络配置。如果数据源采用ConfigDrive，那么cloud-init就会从ConfigDrive中获取配置信息，然后将网络相关的配置写入/etc/sysconfig/network-scripts/ifcfg-XXX中；如果使用的不是ConfigDrive方式，则将虚拟机所在网络设置成DHCP方式。只有在网络正确配置后，才能访问metadata访问，获取metadata数据。



1. Init阶段：网络network

systemd服务：cloud-init.service。

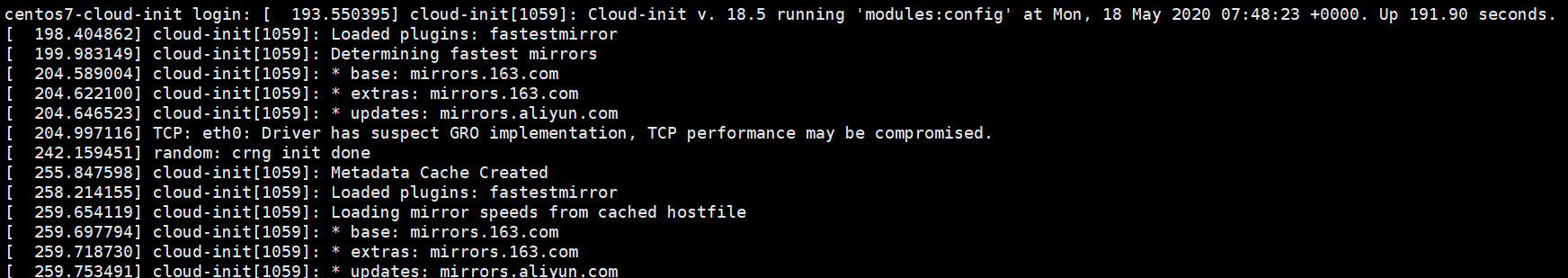
在local阶段和所配置的网络服务生效之后，尽可能多地阻止剩余的启动任务。根据/etc/cloud/cloud.cfg中配置的cloud\_init\_modules模块组配置模块，如运行disk-setup和mounts模块，对磁盘进行分区、格式化和挂载。



1. 配置config

systemd服务：cloud-config.service。

该服务运行在network阶段之后，根据/etc/cloud/cloud.cfg中配置的cloud\_config\_modules模块组配置模块。

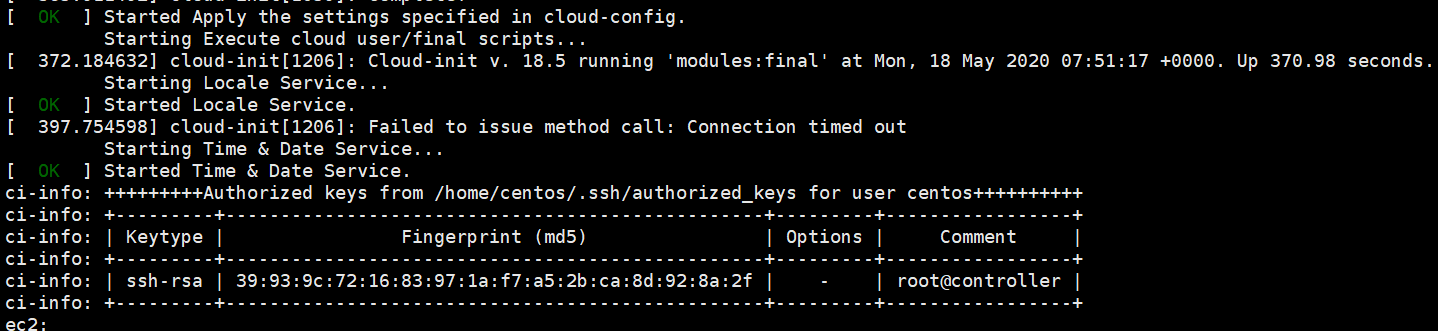


1. 完成Final

systemd服务：cloud-final.service

该阶段运行的启动的最后阶段（传统的rc.local）。根据/etc/cloud/cloud.cfg中配置的cloud\_final\_modules模块组配置模块。

本阶段在系统启动的过程中尽可能晚运行。用户脚本应该在此处运行，如软件包安装、配置管理插件、其它用户脚本等。



正常情况下，当执行到init、config和final这三个阶段前，虚拟机的网络已经配置好了，并且能够正常获取metadata数据。虚拟机中完成定制化的配置是由这三个模块中的脚本实现的。例如在cloud\_config\_modules阶段开启了set\_hostnamegognneng ,那么cloud-init首先会尝试从/etc/cloud/cloud.cfg读取hostname的值，如果找不到，则会从metadata的hostname参数中读取这个变量的值，然后设置虚拟机主机名。

1. Cloud-init应用

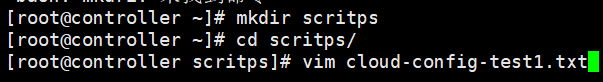
实现虚拟机实例定制一个可选的方法是利用OpenStack已制作好且完全符合要求的镜像，在创建实例时注入编写好的客户化脚本，然后由cloud-init来完成实例的定制化工作。在nova boot和openstack server create命令行相关的选项有以下几个：

* --user-data提供用户脚本
* --file 提交要注入的文件
* --config-drive设置是否启用配置驱动器。

在Dashboard界面可以进行cloud-init的应用。

1. 例：cloud-init虚拟机定制实例：定制要求1)定制用户初始密码；2)设置域名解析；3)设置时区为Asia/Shanghai；4)设置系统支持中文；5)安装软件包

在控制节点的/root目录下创建一个scripts目录，并创建一个实例的自定义脚本文件用来编写cloud-init配置内容。实例的自定义脚本，在OpenStack里就是用户数据(User Data)，使用YAML，在实例创建后第一次引导时由cloud-init解释执行。





创建实例的“配置选项”将上述脚本代码复制到“自定义脚本”中，或者将其保存文件，选择“从文件载入脚本”后选择启动实例。



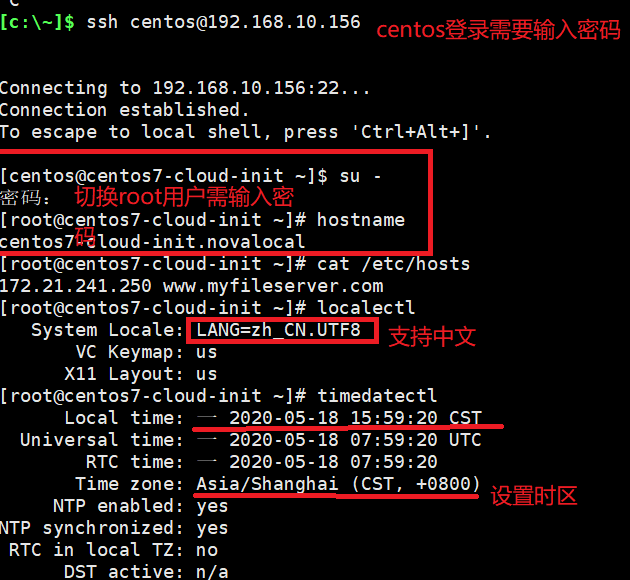
注意：自定义脚本有大小限制，不能超过16KB，但一般都能满足要求。

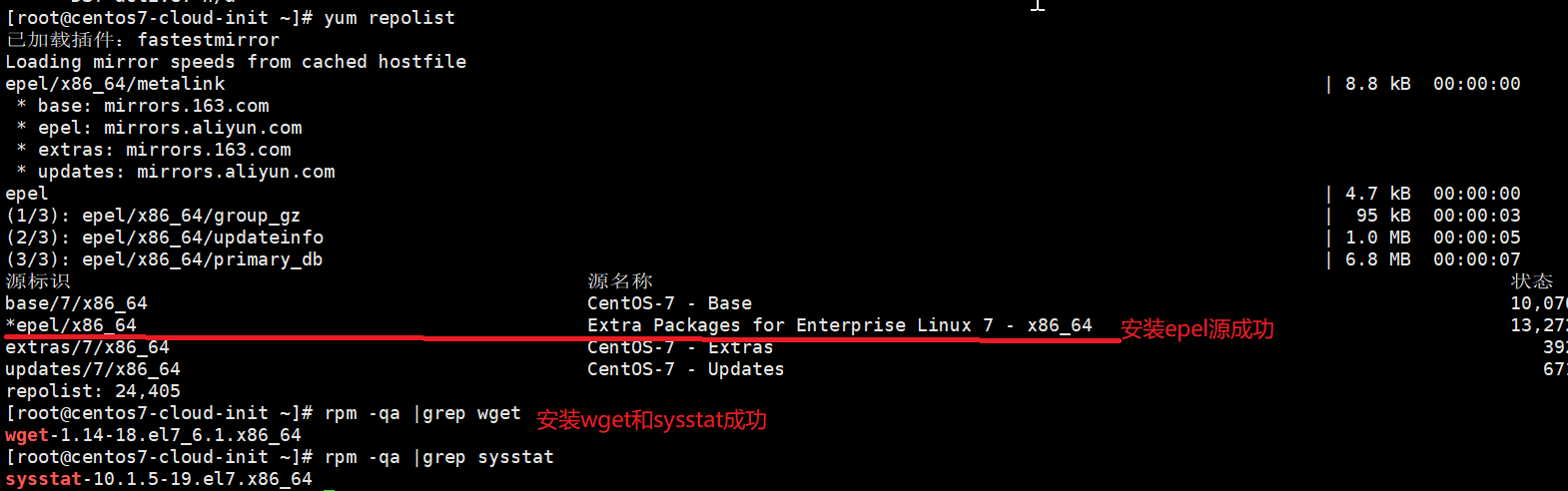
也可以使用openstack server create 命令行中使用--user-data传入该脚本文件。



1. 测试验证

以上实例绑定浮动IP192.168.10.156





以上说明虚拟机定制完成。

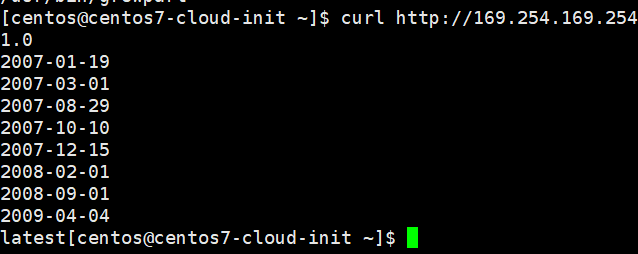
创建虚拟机实例时传入的Userdata自定义脚本，可以使用如下命令查看:



### 13.4元数据服务（Metadata Service）

元数据服务为虚拟机提供Metadata和Userdata，这两类数据只是代表了不同的信息类型，实质上都是提供配置信息的数据源，使用了相同的信息注入机制。在虚拟机中访问http://169.254.169.254可以获得这些信息。

#curl <http://169.254.169.254>



#curl <http://169.254.169.254/latest>



虚拟机的元数据实际保存在Nova数据库中，真正提供元数据服务的是nova-api。而169.254.169.254是一个不跨路由的本地地址，neutron中提供了一组服务将虚拟机访问这个地址的请求转发到nova-api上。Nova和neutron共同提供了元数据服务。

#### 13.4.1 元数据注入

元数据是结构化的结构，以键值对（Key/Value）形式注入实例，包括实例自身的一些常用属性，如主机名，网络配置信息（IP地址和安全组）、SSH密钥等。用户数据是非结构化的数据，通过文件或脚本方式进行注入，支持多种文件格式，如gzip压缩文件、shell脚本、cloud-init配置文件等，主要包括一些命令、脚本等，如提供shell脚本，设置root密码。

OpenStack将元数据和用户数据的配置信息注入机制分为两种：

* 配置驱动器（Config Driver）
* 元数据服务（Metadata Service）机制

cloud-init工具和配置驱动器机制或者元数据服务机制一起实现了虚拟机实例的个性化定制。

#### **13.4.2 配置驱动器**

1. 实现机制

OpenStack将元数据信息写入实例的一个特殊的配置设备中，即在配置驱动器中存储元数据，然后在启动实例时，自动挂载该设备，并由cloud-init读取其中的元数据信息，从而实现配置信息注入。任何可以挂载ISO 9660或者VFAT文件系统的客户操作系统，都可使用配置驱动器。

1. 应用场合

如果实例无法通过DHCP正确获取网络信息，则可以使用配置驱动器来实现配置信息注入。配置驱动器主要用于配置实例的网络信息，包括IP、子网掩码、网关等。

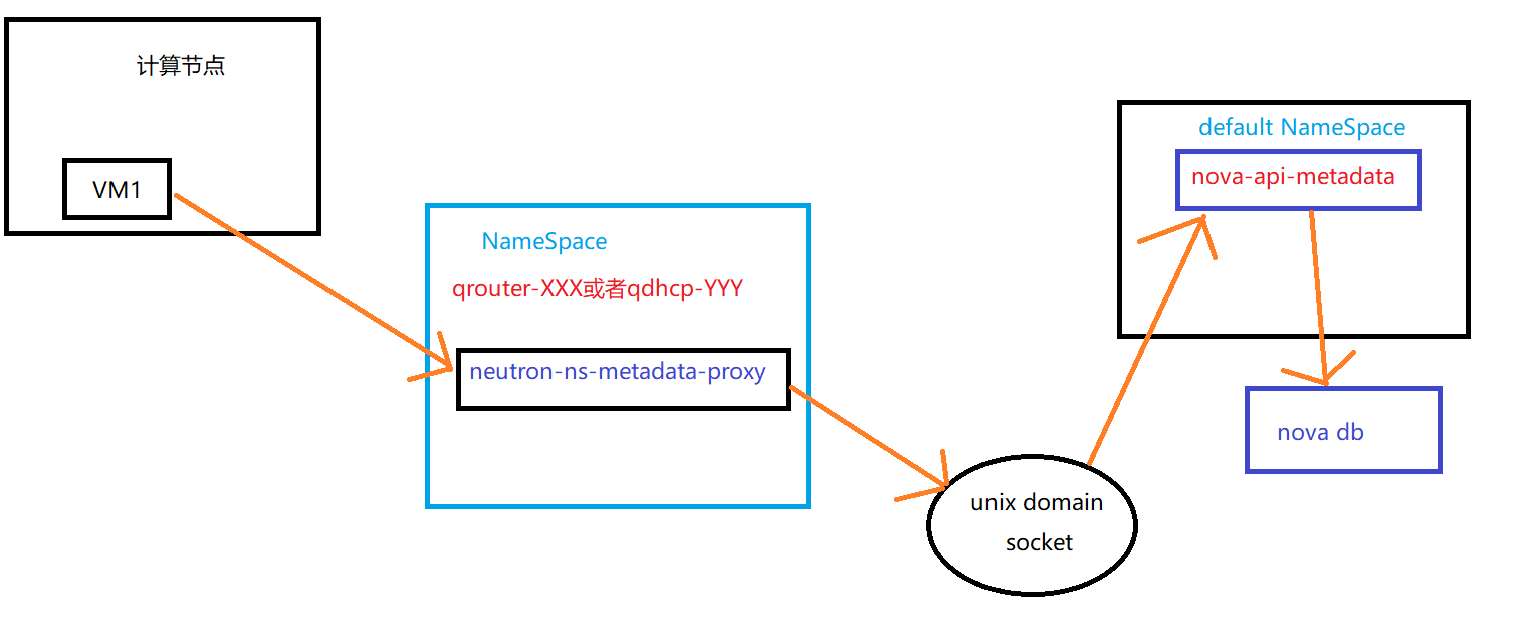
1. 启用配置驱动器

在/etc/nova/nova.conf中设置：

force\_config\_drive = true

#### **13.4.3 元数据服务**

如果实例能够正确配置网络，则可以通过元数据服务的方式获取元数据信息。OpenStack提供RESTful接口让虚拟机实例可以通过REST API来获取元数据。元数据服务的基本架构如图所示：



* nova-api-metadata:nova-api-metadata 启动了 RESTful 服务，负责处理虚拟机发送来的 REST API 请求。从请求的 HTTP 头部中取出相应的信息，获得虚拟机的 ID，继而从数据库中读取虚拟机的 metadata 信息，最后将结果返回。
* neutron-metadata-agent:负责将接收到的获取 metadata 的请求转发给 nova-api-metadata。Neutron-metadata-agent 会获取虚拟机和租户的 ID，添加到请求的 HTTP 头部中。nova-api-metadata 会根据这些信息获取 metadata。
* neturon-ns-metadata-proxy:为了解决网络节点的网段和租户的虚拟网段重复的问题，OpenStack 引入了网络命名空间。Neutron 中的路由和 DHCP 服务器都在各自独立的命名空间中。由于虚拟机获取 metadata 的请求都是以路由和 DHCP 服务器作为网络出口，所以需要通过 neutron-ns-metadata-proxy 联通不同的网络命名空间，将请求在网络命名空间之间转发。Neutron-ns-metadata-proxy 利用在 unix domain socket 之上的 HTTP 技术，实现了不同网络命名空间之间的 HTTP 请求转发。并在请求头中添加’X-Neutron-Router-ID’和’X-Neutron-Network-ID’信息，以便 Neutron-metadata-agent 来辨别发送请求的虚拟机，获取虚拟机的 ID。

虚拟机实例获取元数据的流程：

1. 实例通过项目网络将元数据请求发送到neutron-ns-metadata-proxy，此时会在请求中添加route-id和network-id。
2. neutron-ns-metadata-proxy通过unix domain socket将请求发送给neutron-metadata-agent。此时根据请求中的route-id和network-id和IP获取端口信息，从而获得instance-id和project-id（tenant-id）并加入请求中。
3. neutron-metadata-agent通过内部管理网络将请求转发给nova-api-metadata。此时利用instance-id和project-id（tenant-id）获取实例的元数据。
4. 将获取的元数据原路放回给发送请求的实例。

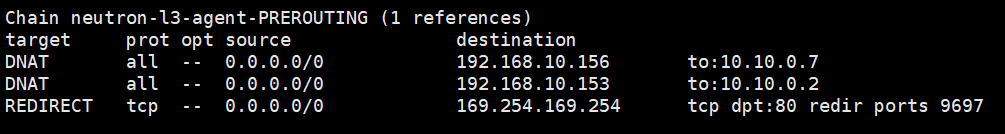
#### **13.4.4 虚拟机实例访问元数据服务**

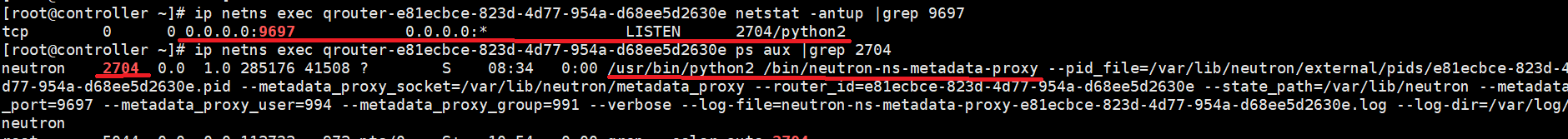
元数据最早是由amazon提出的，当时规定元数据服务的地址为169.254.169.254:80，为了兼容EC2，OpenStack沿用了这一规定，因此实例通过http://169.254.169.254访问元数据服务。那么这一请求是如何从虚拟机中发送出来的呢？目前 Neutron 有两种方式来解决这个问题：通过 router 发送请求和通过 DHCP 发送请求。

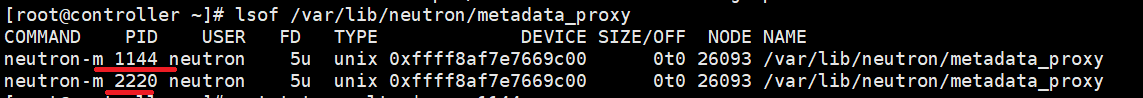
1. 通过 router 发送请求

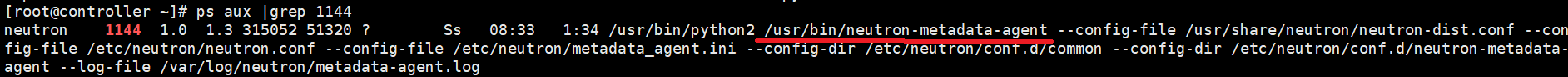
如果虚拟机所在 subnet 连接在了 router 上，那么发向 169.254.169.254 的报文会被发至 router。Neutron 通过在 router 所在网络命名空间添加 iptables 规则，将该报文转发至 9697 端口，而 neutron-ns-metadata-proxy 监听着该端口，所以报文被 neutron-ns-metadata-proxy 获取，进入上述后续处理和转发流程。



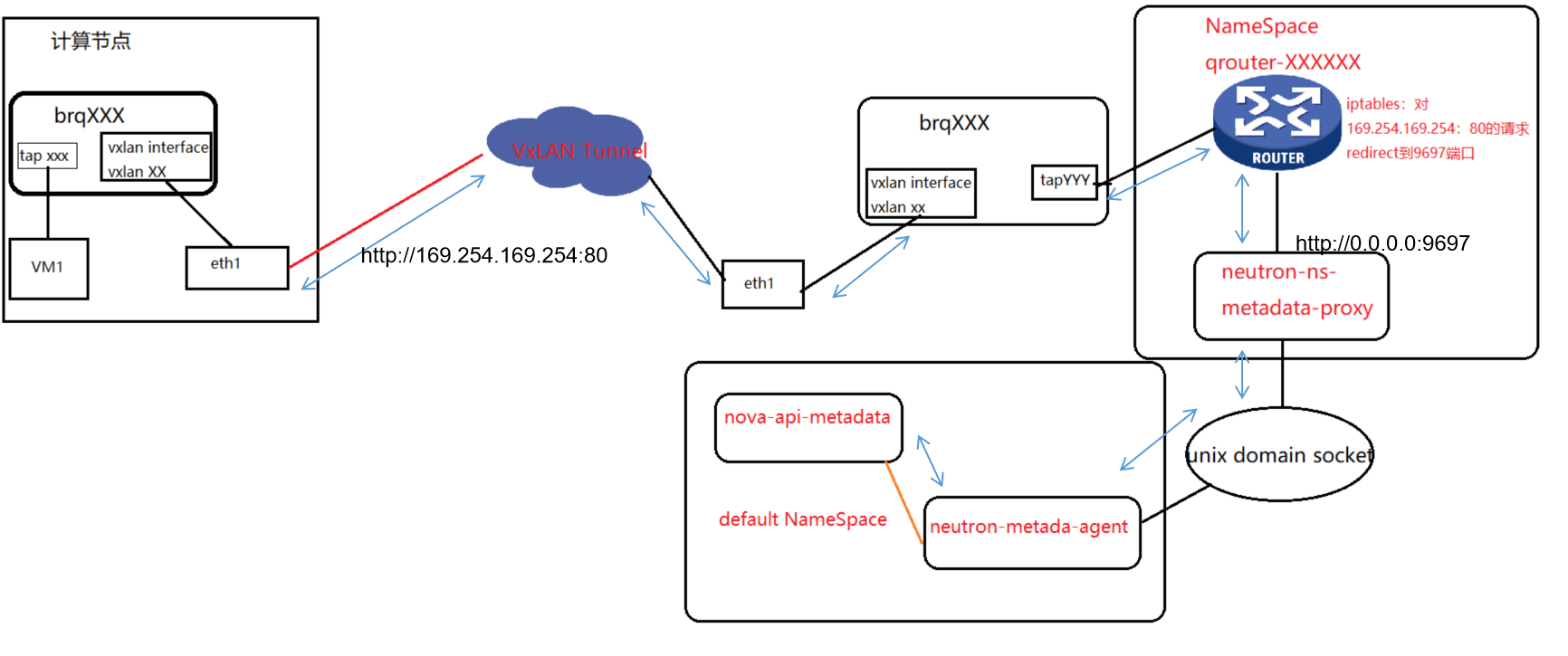








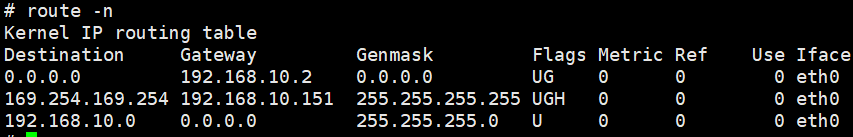
请求流程图如下：



1. 通过 DHCP 发送请求

如果虚拟机所在 subnet 没有连接在任何 router 上，那么请求则无法通过 router 转发。此时 Neutron 通过 DHCP 服务器来转发 metadata 请求。DHCP 服务通过 DHCP 协议的选项 121 来为虚拟机设置静态路由。

虚拟机中的静态路由表：



上图中 192.168.10.151 为 DHCP 服务器的 IP 地址。通过查看虚拟机的静态路由表，可以发现发送至 169.254.169.254 的报文被发送到了192.168.10.151，即 DHCP 服务器。

另外再查看 DHCP 服务器的 IP 配置信息，发现 DHCP 服务器配置了两个 IP，其中一个就是 169.254.169.254。与 router 类似的，Neutron 在 DHCP 网络命名空间中启动了监听 80 端口的 neutron-ns-metadata-proxy 服务，从而进入处理和转发请求的流程。

