简介： 联网是 IaaS 系统的一个重要部分；OpenStack 也不例外，它是 Rackspace Cloud 和 NASA 负责的一个开源基础架构即服务的云计算项目。在本文中，作者阐述了 OpenStack Cloud Compute-Nova 项目背后的 iptable 链和规则，它是使用 Python 进行编写并常用在许多外部库的云计算结构控制器（IaaS 系统的主要部分）中。作者详细介绍了 nova-network FlatDHCPManager 组件，以及其他的 OpenStack 组件。Iptable 是一个用户空间应用程序，允许系统管理员配置由 Linux® 内核防火墙提供的表。

本文阐述了 OpenStack 用 iptables、链和规则处理联网的，这跟其他系统非常类似。但是首先，让我们先来看看 iptables 的结构，作为本文内使用技术的前奏。

iptable 的结构

iptable 是一个用户空间应用程序，允许系统管理员配置由 Linux 内核防火墙提供的表；iptable 专指 IPv4 网络。

要设置一个 Linux 防火墙，就要使用规则，每个规则指定在包中与什么匹配，以及对包执行什么操作。链 是一个规则列表。

Iptable 的前身 ipchains 增加规则链的概念；iptable 则将概念扩展为表。所以 iptable 的结构是：iptables > tables > chains > rules。

iptable 具有四个内置表：

Filter 表：默认表，具有如下链：

INPUT 用于传到本地服务器的包。

OUTPUT 用于本地生成以及传出本地服务器的包。

FORWARD 用于通过本地服务器路由的包。

NAT 表（网络地址转换）：

PREROUTING：用于目的 NAT，它在路由前更改包 IP 地址。

POSTROUTING：用于源 NAT，它在路由前更改包 IP 地址。

OUTPUT：用于防火墙上本地生成包的 NAT。

Mangle 表：用于特定包的更改：

PREROUTING

OUTPUT

FORWARD

INPUT

POSTROUTING

Raw 表：用于配置免除：

PREROUTING

OUTPUT

OpenStack 内的 iptable

在 OpenStack 内，您会在 Compute-Nova 模块中发现 iptable 链和规则占主导作用，该模块是使用 Python 编写并使用在多数外部库的云计算结构控制器（IaaS 系统的主要部分）。本文详细介绍了 nova-network FlatDHCPManager 组件以及其他联网任务所需的 OpenStack 组件。

在开始时，OpenStack 定义了一些 OpenStack 链。这些链与 Linux 内置链形成了一个链结构。启动时的另一个任务是为固定的网络范围和元数据服务定义一些规则。创建并使用网络后，nova-network 就会设置一些规则。当创建一个实例（也称为一个服务器和 VM）后，nova-compute 就会创建一个特定于实例的链并设置此链下的规则以确保实例的连接性。就浮动 IP 而言，OpenStack 也可使用一些规则以正常运行起来。此外，OpenStack 的安全性组及其规则是由 iptables 规则体现。

初识 OpenStack

OpenStack 是一个由开发者和云计算技术人员的全球协作开发的面向公共和私有云的标准云操作系统，是在 Apache 许可条款下发布的免费开源软件。云服务提供者、企业和政府组织均可使用这个免费的 Apache 许可的软件来构建可大规模伸缩的云环境。

OpenStack 目前包含六个核心软件项目：

Cloud Compute-Nova

Cloud Storage-Swift

Image Service-Glance（交付和注册）

Identity Service-Keystone

Dashboard-Horizon

Network Connectivity-Quantum

这些项目以及充满活力的技术提供者和未来项目组成的生态系统带来了一个面向公共和私有云的可插入式的框架和操作系统。

Nova 项目内拥有 10 多个命令，其中的 3 个与 VM 连接性有关：

nova-api 为 VM 提供元数据服务。

nova-compute 为 VM 设置网络环境。

nova-network 为整个云生态系统设置网络环境，如 IP 配置和 DHCP 设置等任务。

虽然 Nova 要求并与很多本机系统组件集成，用于数据库、消息接送功能和虚拟化功能，但它的模块主要由一组 Python 守护进程组成。它使用了一个特殊的元数据服务来使虚拟机实例检索特定于实例的数据。实例访问 http://169.254.169.254 处的元数据服务。

元数据包括公共 SSH 密钥（当用户请求一个新的实例时，由密钥/对标识），用户数据（作为 API 调用中的user\_data 参数进行传递，或是由 Nova 启动命令中的 --user\_data 标记传递）。二进制的 nova-api 命令实现元数据服务。

OpenStack 是一套复杂组件集。要了解有关此系统的更多信息以及其他组件如何工作，可参考 参考资料 部分中给出的大量 OpenStack 资源。

在开始探讨 Nova 内的规则和链之前，让我们先来看看 IP 寻址的模式。

回页首

Nova 中的 IP 寻址

每个 VM 都会从可用的 nova-network 自动分配一个私有 IP 地址。这些 IP 地址称为固定 IP。您可以选择性地将公共 IP 地址分配给实例。OpenStack 使用术语浮动 IP 来指代可被动态添加到一个运行中虚拟实例的 IP 地址（通常都是公共的）。

有多个策略可用来实现固定 IP：

Flat 模式

Flat DHCP 模式

VLAN DHCP 模式

具有定额（quantum）模式的 nova-network

Flat 模式

Flat 模式是最简单的一种联网模式。每个实例接收一个来自池的固定 IP。所有实例均默认附加到相同的桥 (br100)。桥必须进行手动配置。联网配置在实例引导前插入到实例中。在这种模式中，没有浮动 IP 特性。

Flat DHCP 模式

与 flat 模式类似，所有实例也是都附加到同一个桥。在这种模式下，Nova 会进行更多的一些配置；它会试图桥接到一个 Ethernet 设备（默认为 eth0）。它还会运行 dnsmasq 作为 dhcpserver 来侦听桥。实例通过执行 dhcpdiscover接收它们的固定 IP。并且，还会提供浮动 IP 特性。

VLAN DHCP 模式

这是一种默认的联网模式，并且支持大多数特性。对于安装了多个机器的情况，它需要一个能支持主机托管的 VLAN 标签的开关。在这种模式下， Nova 将会创建一个 VLAN 并会为每个项目进行桥接（与一个租户一样）。项目会获得只能从 VLAN 内访问到的私有 IP 的范围。对于在其项目内访问实例的用户，需要创建一个特殊的 VPN 实例（代码命名为 “cloudpipe”）。Nova 会生成一个证书和密钥以供用户访问此 VPN，并自动启动此 VPN。

具有定额模式的 nova-network

在这个模式下，DHCP 服务器还可以自动启动，但不支持浮动 IP。在每个计算主机上运行一个定额代理以将虚拟实例连接到定额网络。网络拓扑可以是非常复杂的。

在 Nova 内，一个安全组就是网络访问规则的组合，比如防火墙策略。这些访问规则指定了哪些传入网络流量应该被发送到组内所有的 VM 实例；所有其他流量则被丢弃。用户可以随时修改组的规则。所有运行中的实例以及从那时起启动的实例将强制使用这些新的规则。可将安全组想象为安全配置文件或安全角色，比如 “webappserver”。

回页首

iptable 的规则和链

正如之前所讨论的，iptable 是一个用户空间应用程序，允许系统管理员配置由 Linux 内核防火墙提供的表以及它所存储的链和规则。

正如之前所提到的，可以定义几个不同的表；OpenStack 内使用表中有 filter 和 NAT。每个表都包含一些内置的链，也可能包含用户定义的链。每个链均是一个能与一组包相匹配的规则列表。每个规则指定对匹配的包执行什么操作，称之为一个目标，可以是跳到同一个表内的一个用户定义的链。

接下来，将讨论各种密钥链和规则。

nova-network 启动时创建的链

图 1. NetworkManager 类连接其他类

如图 1 所示，NetworkManager 是一个连接到很多其他类或模块的大类。linux\_net 就是这些类或模块中的一个。linux\_net 包含了一个 IptablesManager 对象，其 \_\_init\_\_() 方法则初始化 OpenStack 系统内定义的链。

图 2. 初始化链

如之前所述，Linux 内核中的 IPv4 包过滤器规则表拥有一些内置的链：PREROUTING、INPUT、FORWARD、OUTPUT 和 POSTROUTING。总体上，抵达 Linux 主机的包会转向 PREROUTING 链。在传递之后，内核就会做出一个路由决定。如果包的目标是本 Linux 主机，它就会转向 INPUT 链，如果被接受，则会转向目标进程。如果包不是用于本 Linux 主机的，则会转向 FORWARD 链，接着再转向 POSTROUTING 链，然后离开主机。由本地进程生成的包首先会转向 OUTPUT 链，如果被接受，就会转向 POSTROUTING 链。

除了这些内置链外，OpenStack 系统会创建其他一些可钩挂到链系统的链。OpenStack 链包含两种类型：未包装（unwrapped）链和已包装（wrapped）链。

nova-filter-top 和 nova-postrouting-bottom 是两种未包装链。nova-filter-top 被添加到 FORWARD 和 OUTPUT 链顶部。由于起未包装，所以它可在各种 Nova 工作进程间进行共享。它针对的是 FORWARD 和 OUTPUT 链顶部存在的规则。在表中的 IPv4 和 IPv6 集合均均存在 nova-filter-top。

已包装链的例子则包括绿色链（green colored chains）。这些链的名称会有进程名作为后缀。比如，nova-network 会创建 nova-network-PREROUTING 链。

对于 IPv4 和 IPv6，内置的 INPUT、OUTPUT 和 FORWARD 过滤器链都是已包装的，这意味着这个“真正”的 INPUT 链具有一个规则，可跳至已包装的 INPUT 链等。此外，还有一种已包装链，名为 local，它从 nova-filter-top 开始跳转。

对于 IPv4，内置的 PREROUTING、OUTPUT 和 POSTROUTING NAT 链的包装方式与内置的过滤器链一样。此外，在 POSTROUTING 链之后，还会应用一个 sNAT 链和一个 float-sNAT 链。

nova-network 启动时的规则

图 3 显示了 FlatDHCPManager 的部分启动进程。之前介绍过的 \_\_init\_\_() 方法，它可构成 IptableManager 对象来创建很多 OpenStack 链。所以，我们将着重介绍 init\_host()。init\_host() 方法调用 LinuxNet3 的 initialize() 方法，它调用 linux\_net 的方法来设置 NAT 表内的一些 iptables 规则。

图 3. FlatDHCPManager 的启动进程

（图 3 的放大图。）

让我们来看看这些规则。

元数据主机的规则

在由 linux\_net 的 init\_host() 创建的这些规则之中，其中的一个规则就是允许 FLAGS.fixed\_range 下的 IP 访问 metadata\_host（在本例中为 192.168.1.90）。

-A nova-network-POSTROUTING -s 10.0.0.0/8 -d 192.168.1.90/32 -j ACCEPT

ensure\_metadata\_ip() 将 169.254.169.254/32 添加到 lo 设备，使用命令如下：

# ip addr add 169.254.169.254/32 scope link dev lo

然后，metadata\_forward() 将一个 dNAT 规则添加到 route 包，从 169.254.169.254/32 到 metadata\_host：

-A nova-network-PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j DNAT

--to-destination 192.168.1.90:8775

如果此 nova-network 和 nova-api 并未在相同的主机上运行，那么您必须定义 nova-network 上的 metadata\_host 来指向此 nova-api 主机。

访问 dmz 的规则

FLAGS.dmz\_cidr 定义了一个 dmz 列表（边界联网）CIDRs（无类别域间路由）。在默认的情况下，它是一个空列表。在本例中为 10.128.0.0/24。所以，此规则为：

-A nova-network-POSTROUTING -s 10.0.0.0/8 -d 10.128.0.0/24 -j ACCEPT

VM 之间互相连接的规则

另一个规则是确保具有两个规定 IP 的 VM 能够互相会话：

-A nova-network-POSTROUTING -s 10.0.0.0/8 -d 10.0.0.0/8 -m conntrack ! --ctstate DNAT

-j ACCEPT

固定子集之外的访问所需规则

add\_snat\_rule() 方法将一个 sNAT 规则添加到此 NAT 表的已包装链 sNAT 内。在 图 3 中可以看到 ip\_range 和FLAGS.routing\_source\_ip。ip\_range 的值是由 FLAGS.fixed\_range 定义。在本例中为 10.0.0.0/8。FLAGS.routing\_source\_ip默认为 FLAGS.my\_ip，由函数 flags.\_get\_my\_ip() 默认。在本例中，FLAGS.my\_ip 为 192.168.1.90。在这之后，您会获得一个规则，类似于：

-A nova-network-snat -s 10.0.0.0/8 -j SNAT --to-source 192.168.1.90

为了固定 IP 能够访问外部，必须创建一个是 FLAGS.fixed\_range 子集的网络。这样一来，随着默认网关指向 nova-network 机器上的 br100 的一个 IP ，具有此子集 IP 的 VM 就将能够访问一个外部网络。

每个网络的规则

要设置一个网络，nova-network 会在此 filter 表内创建一些规则。要触发此 nova-network 在给定网络上的操作，可运行如下命令：

创建一个网络并设置主机：

# ./bin/nova-manage network create mynet 10.10.10.0/24

引导一个服务器：

nova boot --image a3fb743d-42df-49ba-b9c4-8042ebbd344e --flavor 1 myserver

执行这些命令后，如下规则适用：

支持转发流量传递桥，这样 br100 上的 IP 就可以充当一个网关：

-A nova-network-FORWARD -i br100 -j ACCEPT

-A nova-network-FORWARD -o br100 -j ACCEPT

支持 DHCP 和 DNS 流量进入到本地 dnsmasq 过程：

-A nova-network-INPUT -i br100 -p udp -m udp --dport 67 -j ACCEPT

-A nova-network-INPUT -i br100 -p tcp -m tcp --dport 67 -j ACCEPT

-A nova-network-INPUT -i br100 -p udp -m udp --dport 53 -j ACCEPT

-A nova-network-INPUT -i br100 -p tcp -m tcp --dport 53 -j ACCEPT

备注：67 为 DHCP 端口，而 53 为 DNS 端口。

nova-compute 主机上的规则

nova-compute 模块还会创建已包装的链。在这些链上，它会在此 filter 表内创建一些规则。让我们一起来看看。

支持转发流量传递桥的规则

nova-compute 主机上的这些规则允许 VM 与 nova-network 主机以及其他计算主机上的 VM 相连接。

-A nova-compute-FORWARD -i br100 -j ACCEPT

-A nova-compute-FORWARD -o br100 -j ACCEPT

每个实例的链和规则

对于每个实例，它都会在此 filter 表内创建一个链及一些规则（参见图 4）。

图 4. 每个实例的链和规则

可以看到：

nova-compute 为每个实例创建一个链。在图 4 中，此链的名称是 nova-compute-inst-1。

所有针对此实例的流量，无论是转发而来的还是从本地过程生成的，都会进入此实例的特定链。

所有来自该实例所在的子集的 IP 的流量都是允许的。

所有来自特定 DHCP 服务器的 DHCP 流量都是允许的。

所有其他流量均放弃。

nova-api

一开始，nova-api 会在 filter 表内创建一个规则以允许他人能够访问 nova-api 服务。

-A nova-api-INPUT -d 192.168.1.90/32 -p tcp -m tcp --dport 8775 -j ACCEPT

浮动 IP

要了解浮动 IP 是如何实现的，首先将一个浮动 IP 关联到此实例的固定 IP。之前所创建的这个实例的固定 IP 是 10.10.10.2。

在默认池内创建一个浮动 IP

要在默认池内创建一个浮动 IP，请运行一下代码：

# nova-manage floating create --ip\_range=192.168.1.232/30

要从这个池分派一个浮动 IP，请运行一下代码：

# Nova floating-ip-create

这样，就拥有一个 IP 192.168.1.233。现在将它分配给 ID 为 8f773639-c04f-4885-9349-ac7d6a799843 的实例：

# nova add-floating-ip 8f773639-c04f-4885-9349-ac7d6a799843 192.168.1.233

将浮动 IP 绑定到公共接口

FLAGS.public\_interface 被用来绑定浮动 IP。在运行了 nova add-floating-ip 命令后，可以看到 public\_interface 下就具有了如下的浮动 IP：

# ip addr list dev wlan0

3: wlan0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP qlen 1000

link/ether 08:11:96:75:91:54 brd ff:ff:ff:ff:ff:ff

inet 192.168.1.90/16 brd 192.168.255.255 scope global wlan0

inet 192.168.1.233/32 scope global wlan0

inet6 fe80::a11:96ff:fe75:9154/64 scope link

valid\_lft forever preferred\_lft forever

NAT 表内针对浮动 IP 的规则

在此实例获得了 nova-network 主机上的一个浮动 IP 后，这些规则适用：

-A nova-network-OUTPUT -d 192.168.1.233/32 -j DNAT --to-destination 10.10.10.2

-A nova-network-PREROUTING -d 192.168.1.233/32 -j DNAT --to-destination 10.10.10.2

-A nova-network-float-snat -s 10.10.10.2/32 -j SNAT --to-source 192.168.1.233

您会看到该 dNAT 规则被用来将此浮动 IP 转换成此实例的固定 IP。如果一个包到达 nova-network 主机并以浮动 IP 为目标 IP，则此目标 IP 就会被转换。然后，就会有另一项 sNAT 规则会将来自此实例的固定 IP 的流量转换到此浮动 IP。由于所有从 VM 到固定网络之外的流量均被指向网关（是由 nova-network 的 dnsmasq 进行设置），有了 sNAT 规则，出 VM 的流量就可以成功标志成来自此浮动 IP。此外，已包装的 OUTPUT 链内还有一个 dNAT 规则，允许 nova-network 上的本地过程访问具有浮动 IP 的 VM。

使用浮动 IP 的 Ping VM

要将 VM 与浮动 IP Ping 在一起，也需要一些规则。请记住，在此 nova-compute 主机上，针对每个实例必须有一个特定链；其内的规则只允许来自固定子集内的 IP 的流量。如果想要 ping 一个浮动 IP，流量就会被这些规则丢弃，因为 ping 包的源 IP 并不在此固定子集内。显然，需要一个规则以允许 icmp 流量。

为了添加一个允许 ping 的规则，可以使用 OpenStack 的安全组规则的概念：

# nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0

之后，就可以看到在此实例的特定链下多创建了一个规则：

-A nova-compute-inst-1 -p icmp -j ACCEPT

同样的方式，可以对具有浮动 IP 的 VM 启用 SSH。

回页首

结束语

来自 iptable 的规则被 OpenStack 网络广泛使用。其安全组以及浮动 IP 的概念只是使用 iptable 规则的开始。请深入探索随后给出的参考资料以了解有关 OpenStack 的 IaaS 环境的更多相关信息。

参考资料

学习

借助来自开发者社区的资源探索 OpenStack 系统：

Cloud Compute-Nova 模块：

OpenStack 网站

GitHub 的 Nova 项目开发者网站

OpenStack 计算管理手册

OpenStack 的 Network Connectivity-Quantum

Image Services-Glance

项目文档

OpenStack 网站

Identity Services-Keystone

OpenStack 网站

项目文档

OpenStack 的 Cloud Storage-Swift

OpenStack 的 Dashboard-Horizon

OpenStack 手册：

面向开发人员

API

管理指南

常见问题的答案

本文作者介绍了如何使用 OpenStack nova-scheduler 来帮助决定哪个计算节点主机应该启动一个图像实例。有关本文作者对 OpenStack 的更多见解，请参阅 博客。

developerWorks 拥有一个 OpenStack 维基。

探索 developerWorks 云计算，获得有价值的社区讨论并了解与云相关的新技术资源。

随时关注 developerWorks 技术活动和网络广播，了解各种 IBM 产品和 IT 行业话题。 关注 Twitter 上的 developerWorks。

developerWorks 演示中心：观看演示，从为初学者准备的产品安装，到为经验丰富的开发人员准备的高级功能。

获得产品和技术以最适合您的方式 评估 IBM 产品：下载产品试用版，在线试用产品，在云环境下试用产品，或者在 SOA 沙盒 中花费几个小时来学习如何高效实现面向服务的架构。

讨论加入 developerWorks 社区，查看开发人员推动的博客、论坛、组和维基，并与其他 developerWorks 用户进行交流。

关于作者

龚永生，IBM 资深软件工程师，热衷于开源软件，具有多年的 Linux，Java 和 JavaEE 经验。目前是 OpenStack 项目的积极贡献者。