云计算底层技术–使用openvswitch

Posted on January 23, 2017 by opengers in openstack

Open vSwitch介绍

在过去，数据中心的服务器是直接连在硬件交换机上，后来VMware实现了服务器虚拟化技术，使虚拟服务器(VMs)能够连接在虚拟交换机上，借助这个虚拟交换机，可以为服务器上运行的VMs或容器提供逻辑的虚拟的以太网接口，这些逻辑接口都连接到虚拟交换机上，有三种比较流行的虚拟交换机: VMware virtual switch, Cisco Nexus 1000V,和Open vSwitch

Open vSwitch(OVS)是运行在虚拟化平台上的虚拟交换机，其支持OpenFlow协议，也支持gre/vxlan/IPsec等隧道技术。在OVS之前，基于Linux的虚拟化平台比如KVM或Xen上，缺少一个功能丰富的虚拟交换机，因此OVS迅速崛起并开始在Xen/KVM中流行起来，并且应用于越来越多的开源项目，比如openstack neutron中的网络解决方案

在虚拟交换机的Flow控制器或管理工具方面，一些商业产品都集成有控制器或管理工具，比如Cisco 1000V的`Virtual Supervisor Manager(VSM)`，VMware的分布式交换机中的`vCenter`。而OVS则需要借助第三方控制器或管理工具实现复杂的转发策略。例如OVS支持OpenFlow 协议，我们就可以使用任何支持OpenFlow协议的控制器来对OVS进行远程管理。OpenStack Neutron中的ML2插件也能够实现对OVS的管理。但这并不意味着OVS必须要有一个控

制器才能工作。在不连接外部控制器情况下，OVS自身可以依靠MAC地址学习实现二层数据包转发功能，就像`Linux Bridge`

在基于Linux内核的系统上，应用最广泛的还是系统自带的虚拟交换机`Linux Bridge`，它是一个单纯的基于MAC地址学习的二层交换机，简单高效，但同时缺乏一些高级特性，比如OpenFlow,VLAN tag,QOS,ACL,Flow等，而且在隧道协议支持上，Linux Bridge只支持vxlan，OVS支持gre/vxlan/IPsec等，这也决定了OVS更适用于实现SDN技术

OVS支持以下features

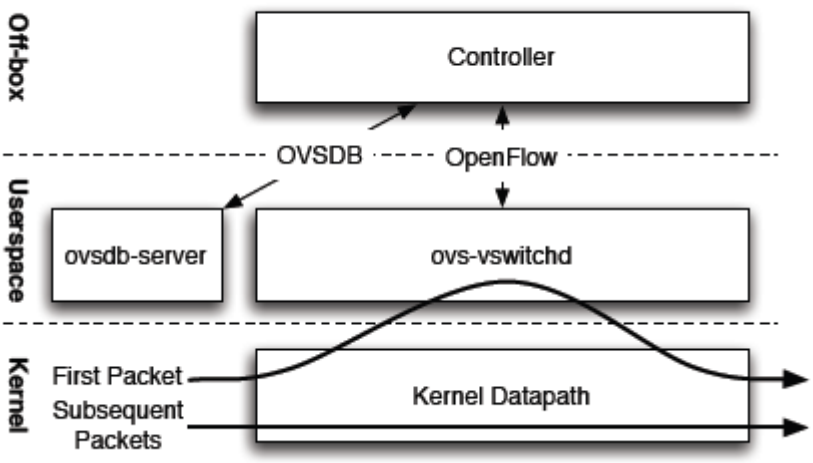
- 支持NetFlow, IPFIX, sFlow, SPAN/RSPAN等流量监控协议
- 精细的ACL和QoS策略
- 可以使用OpenFlow和OVSDB协议进行集中控制
- Port bonding，LACP，tunneling(vxlan/gre/Ipsec)
- 适用于Xen，KVM，VirtualBox等hypervisors
- 支持标准的802.1Q VLAN协议
- 基于VM interface的流量管理策略
- 支持组播功能
- flow-caching engine(datapath模块)

文章使用环境

```
centos7
openvswitch 2.5
OpenFlow 1.4
```

OVS架构

先看下OVS整体架构，用户空间主要组件有数据库服务ovsdb-server和守护进程ovs-vswitchd。kernel中是datapath内核模块。最上面的Controller表示OpenFlow控制器，控制器与OVS是通过OpenFlow协议进行连接，控制器不一定位于OVS主机上，下面分别介绍图中各组件



ovs-vswitchd

`ovs-vswitchd`守护进程是OVS的核心部件，它和`datapath`内核模块一起实现OVS基于流的数据交换。作为核心组件，它使用openflow协议与上层OpenFlow控制器通信，使用OVSDB协议与`ovsdb-server`通信，使用`netlink`和`datapath`内核模块通信。`ovs-vswitchd`在启动时会读取`ovsdb-server`中配置信息，然后配置内核中的`datapaths`和所有OVS switches，当ovsdb中的配置信息改变时(例如使用ovs-vsctl工具)，`ovs-vswitchd`也会自动更新其配置以保持与数据库同步

```
# ps -ef |grep ovs-vs
root     22176 22175  0 Jan17 ?        00:16:56 ovs-vswitchd unix:/var/run/op
```

`ovs-vswitchd`需要加载`datapath`内核模块才能正常运行。它会自动配置`datapath` flows，因此我们不必再使用`ovs-dpctl`去手动操作`datapath`，但`ovs-dpctl`仍可用于调试场合

在OVS中，`ovs-vswitchd`从OpenFlow控制器获取流表规则，然后把从`datapath`中收到的数据包在流表中进行匹配，找到匹配的flows并把所需应用的actions返回给`datapath`，同时作为处理的一部分，`ovs-vswitchd`会在`datapath`中设置一条datapath flows用于后续相同类型的数据包可以直接在内核中执行动作，此datapath flows相当于OpenFlow flows的缓存。对于`datapath`来说，其并不知道用户空间OpenFlow的存在，datapath内核模块信息如下

```
# modinfo openvswitch
filename:       /lib/modules/3.10.0-327.el7.x86_64/kernel/net/openvswitch/ope
license:        GPL
description:    Open vSwitch switching datapath
rhelversion:    7.2
srcversion:     F75F2B83324DCC665887FD5
depends:        libcrc32c
intree:         Y
...
```

ovsdb-server

`ovsdb-server`是OVS轻量级的数据库服务，用于整个OVS的配置信息，包括接口/交换内容/VLAN等，OVS主进程`ovs-vswitchd`根据数据库中的配置信息工作，下面是`ovsdb-server`进程详细信息

```
ps -ef |grep ovsdb-server
root     22166 22165  0 Jan17 ?        00:02:32 ovsdb-server /etc/openvswitch
```

`/etc/openvswitch/conf.db`是数据库文件存放位置，文件形式存储保证了服务器重启不会影响其配置信息，`ovsdb-server`需要文件才能启动，可以使用`ovsdb-tool create`命令创建并初始化此数据库文件
`--remote=punix:/var/run/openvswitch/db.sock`实现了一个Unix sockets连接，OVS主进程`ovs-vswitchd`或其它命令工具(ovsdb-client)通过此socket连接管理ovsdb
`/var/log/openvswitch/ovsdb-server.log`是日志记录

OpenFlow

OpenFlow是开源的用于管理交换机流表的协议，OpenFlow在OVS中的地位可以参考上面架构图，它是Controller和ovs-vswitched间的通信协议。需要注意的是，OpenFlow是一个独立的完整的流表协议，不依赖于OVS，OVS只是支持OpenFlow协议，有了支持，我们可以使用OpenFlow控制器来管理OVS中的流表，OpenFlow不仅仅支持虚拟交换机，某些硬件交换机也支持OpenFlow协议

OVS常用作SDN交换机(OpenFlow交换机)，其中控制数据转发策略的就是OpenFlow flow。OpenStack Neutron中实现了一个OpenFlow控制器用于向OVS下发OpenFlow flows控制虚拟机间的访问或隔离。本文讨论的默认是作为SDN交换机场景下

OpenFlow flow的流表项存放于用户空间主进程`ovs-vswitchd`中，OVS除了连接OpenFlow控制器获取这种flow，文章后面会提到的命令行工具`ovs-ofctl`工具也可以手动管理OVS中的OpenFlow flow，可以查看`man ovs-ofctl`了解

在OVS中，OpenFlow flow是最重要的一种flow, 然而还有其它几种flows存在，文章下面OVS概念部分会提到

Controller

Controller指OpenFlow控制器。OpenFlow控制器可以通过OpenFlow协议连接到任何支持OpenFlow的交换机，比如OVS。控制器通过向交换机下发流表规则来控制数据流向。除了可以通过OpenFlow控制器配置OVS中flows，也可以使用OVS提供的`ovs-ofctl`命令通过OpenFlow协议去连接OVS，从而配置flows，命令也能够对OVS的运行状况进行动态监控。

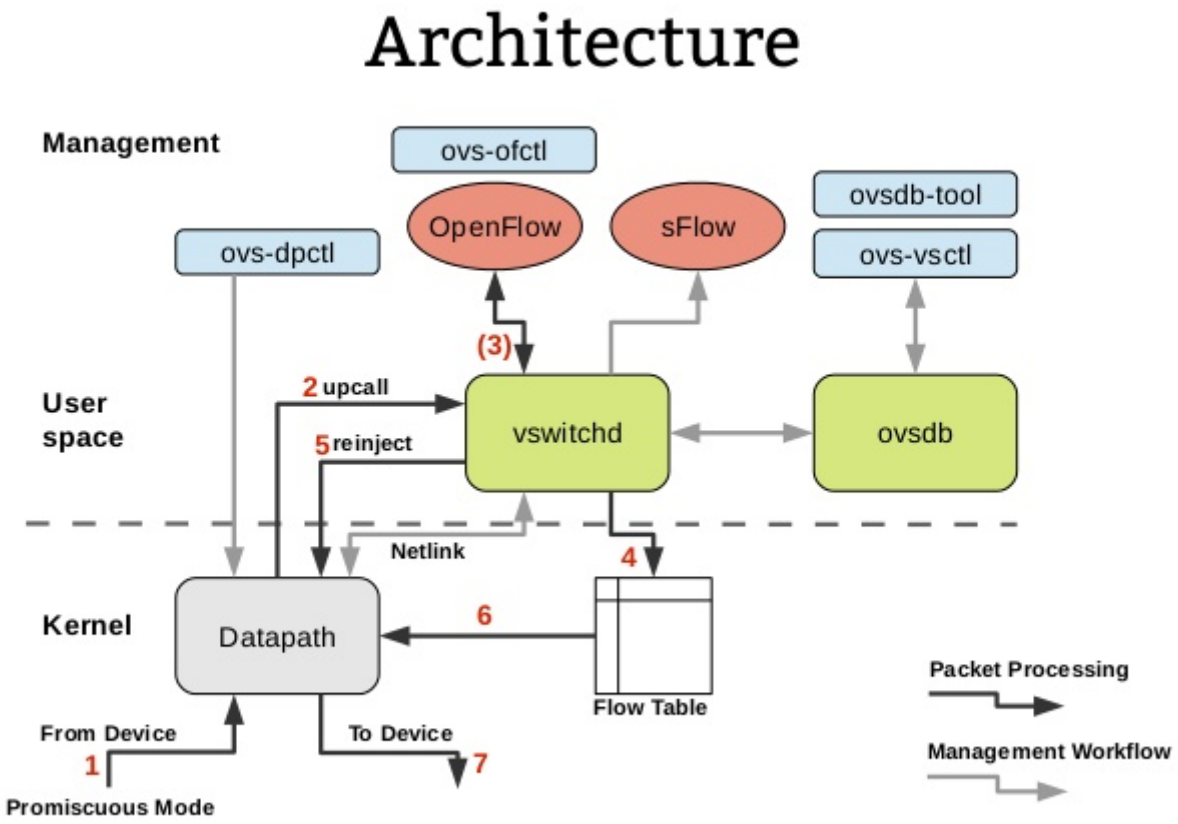
Kernel Datapath

下面讨论场景是OVS作为一个OpenFlow交换机

datapath是一个Linux内核模块，它负责执行数据交换。关于datapath，The Design and Implementation of Open vSwitch中有描述

> The datapath module in the kernel receives the packets first, from a physical NIC or a VM's virtual NIC. Either ovs-vswitchd has instructed the datapath how to handle packets of this type, or it has not. In the former case, the datapath module simply follows the instructions, called actions, given by ovs-vswitchd, which list physical ports or tunnels on which to transmit the packet. Actions may also specify packet modifications, packet sampling, or instructions to drop the packet. In the other case, where the datapath has not been told what to do with the packet, it delivers it to ovs-vswitchd. In userspace, ovs-vswitchd determines how the packet should be handled, then it passes the packet back to the datapath with the desired handling. Usually, ovs-vswitchd also tells the datapath to cache the actions, for handling similar future packets.

为了说明datapath，来看一张更详细的架构图，图中的大部分组件上面都有提到



用户空间`ovs-vswitchd`和内核模块`datapath`决定了数据包的转发，首先，`datapath`内核模块收到进入数据包(物理网卡或虚拟网卡)，然后查找其缓存(datapath flows)，当有一个匹配的flow时它执行对应的操作，否则`datapath`会把该数据包送入用户空间由`ovs-vswitchd`负责在其OpenFlow flows中查询(图1中的First Packet)，`ovs-vswitchd`查询后把匹配的actions返回给`datapath`并设置一条datapath flows到`datapath`中，这样后续进入的同类型的数据包(图1中的Subsequent Packets)因为缓存匹配会被`datapath`直接处理，不用再次进入用户空间。

`datapath`专注于数据交换，它不需要知道OpenFlow的存在。与OpenFlow打交道的是`ovs-vswitchd`，`ovs-vswitchd`存储所有Flow规则供`datapath`查询或缓存.

虽然有`ovs-dpctl`管理工具的存在，但我们没必要去手动管理`datapath`，这是用户空间`ovs-vswitchd`的工作

OVS概念

这部分说下OVS中的重要概念，使用OpenStack neutron+vxlan部署模式下网络节点OVS网桥作为例子

```
# ovs-vsctl show
e44abab7-2f65-4efd-ab52-36e92d9f0200
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-ext
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
```

```
        fail_mode: secure
        Port br-ext
            Interface br-ext
                type: internal
        Port "eth1"
            Interface "eth1"
        Port phy-br-ext
            Interface phy-br-ext
                type: patch
                options: {peer=int-br-ext}
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port "vxlan-080058ca"
            Interface "vxlan-080058ca"
                type: vxlan
                options: {df_default="true", in_key=flow, local_ip="8.0.88.20
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "qr-11591618-c4"
            tag: 3
            Interface "qr-11591618-c4"
                type: internal
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port int-br-ext
            Interface int-br-ext
                type: patch
                options: {peer=phy-br-ext}
```

Bridge

Bridge代表一个以太网交换机(Switch)，一个主机中可以创建一个或者多个Bridge。Bridge的功能是根据一定规则，把从端口收到的数据包转发到另一个或多个端口，上面例子中有三个Bridge，`br-tun`，`br-int`，`br-ext`

添加一个网桥`br0`

```
ovs-vsctl add-br br0
```

Port

端口Port与物理交换机的端口概念类似，Port是OVS Bridge上创建的一个虚拟端口，每个Port都隶属于一个Bridge。Port有以下几种类型

- **Normal**

可以把操作系统中已有的网卡(物理网卡em1/eth0,或虚拟机的虚拟网卡tapxxx)挂载到ovs上，ovs会生成一个同名Port处理这块网卡进出的数据包。此时端口类型为Normal。

如下，主机中有一块物理网卡`eth1`，把其挂载到OVS网桥`br-ext`上，OVS会自动创建同名Port`eth1`。

```
ovs-vsctl add-port br-ext eth1
#Bridge br-ext中出现Port "eth1"
```

有一点要注意的是，挂载到OVS上的网卡设备不支持分配IP地址，因此若之前`eth1`配置有IP地址，挂载到OVS之后IP地址将不可访问。这里的网卡设备不只包括物理网卡，也包括主机上创建的虚拟网卡

- **Internal**

Internal类型是OVS内部创建的虚拟网卡接口，每创建一个Port，OVS会自动创建一个同名接口(Interface)挂载到新创建的Port上。接口的概念下面会提到。

下面创建一个网桥br0，并创建一个Internal类型的Port `p0`

```
ovs-vsctl add-br br0
ovs-vsctl add-port br0 p0 -- set Interface p0 type=internal

#查看网桥br0
ovs-vsctl show br0
    Bridge "br0"
        fail_mode: secure
        Port "p0"
            Interface "p0"
                type: internal
        Port "br0"
            Interface "br0"
                type: internal
```

可以看到有两个Port。当ovs创建一个新网桥时，默认会创建一个与网桥同名的Internal Port。在OVS中，只有"internal"类型的设备才支持配置IP地址信息，因此我们可以为`br0`接口配置一个IP地址，当然`p0`也可以配置IP地址

```
ip addr add 192.168.10.11/24 dev br0
ip link set br0 up
```

```
#添加默认路由
ip route add default via 192.168.10.1 dev br0
```

上面两种Port类型区别在于，Internal类型会自动创建接口(Interface)，而Normal类型是把主机中已有的网卡接口添加到OVS中

- **Patch**

当主机中有多个ovs网桥时，可以使用Patch Port把两个网桥连起来。Patch Port总是成对出现，分别连接在两个网桥上，从一个Patch Port收到的数据包会被转发到另一个Patch Port，类似于Linux系统中的`veth`。使用Patch连接的两个网桥跟一个网桥没什么区别，OpenStack Neutron中使用到了Patch Port。上面网桥`br-ext`中的Port `phy-br-ext`与`br-int`中的Port `int-br-ext`是一对Patch Port

可以使用`ovs-vsctl`创建patch设备，如下创建两个网桥`br0,br1`，然后使用一对`Patch Port`连接它们

```
ovs-vsctl add-br br0
ovs-vsctl add-br br1
ovs-vsctl \
-- add-port br0 patch0 -- set interface patch0 type=patch options:peer=patch1 \
-- add-port br1 patch1 -- set interface patch1 type=patch options:peer=patch0

#结果如下
#ovs-vsctl show
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "patch0"
            Interface "patch0"
                type: patch
                options: {peer="patch1"}
    Bridge "br1"
        Port "br1"
            Interface "br1"
                type: internal
        Port "patch1"
            Interface "patch1"
                type: patch
                options: {peer="patch0"}
```

连接两个网桥不止上面一种方法，linux中支持创建`Veth`设备对，我们可以首先创建一对`Veth`设备对，然后把这两个`Veth`分别添加到两个网桥上，其效果跟OVS中创建Patch Port一样，只是性能会有差别

```
#创建veth设备对veth-a,veth-b
ip link add veth-a type veth peer name veth-b
#使用Veth连接两个网桥
```

```
ovs-vsctl add-port br0 veth-a
ovs-vsctl add-port br1 veth-b
```

- **Tunnel**

OVS中支持添加隧道(Tunnel)端口，常见隧道技术有两种`gre`或`vxlan`。隧道技术是在现有的物理网络之上构建一层虚拟网络，上层应用只与虚拟网络相关，以此实现的虚拟网络比物理网络配置更加灵活，并能够实现跨主机的L2通信以及必要的租户隔离。不同隧道技术其大体思路均是将以太网报文使用隧道协议封装，然后使用底层IP网络转发封装后的数据包，其差异性在于选择和构造隧道的协议不同。Tunnel在OpenStack中用作实现大二层网络以及租户隔离，以应对公有云大规模，多租户的复杂网络环境。

OpenStack是多节点结构，同一子网的虚拟机可能被调度到不同计算节点上，因此需要有隧道技术来保证这些同子网不同节点上的虚拟机能够二层互通，就像他们连接在同一个交换机上，同时也要保证能与其它子网隔离。

OVS在计算和网络节点上建立隧道Port来连接各节点上的网桥`br-int`，这样所有网络和计算节点上的`br-int`互联形成了一个大的虚拟的跨所有节点的逻辑网桥(内部靠tunnel id或VNI隔离不同子网)，这个逻辑网桥对虚拟机和qrouter是透明的，它们觉得自己连接到了一个大的`br-int`上。从某个计算节点虚拟机发出的数据包会被封装进隧道通过底层网络传输到目的主机然后解封装。

上面网桥`br-tun`中`Port "vxlan-080058ca"`就是一个`vxlan`类型tunnel端口。下面使用两台主机测试创建vxlan隧道

```
#主机192.168.7.21上
ovs-vsctl add-br br-vxlan
#主机192.168.7.23上
ovs-vsctl add-br br-vxlan
#主机192.168.7.21上添加连接到7.23的Tunnel Port
ovs-vsctl add-port br-vxlan tun0 -- set Interface tun0 type=vxlan options:rem
#主机192.168.7.23上添加连接到7.21的Tunnel Port
ovs-vsctl add-port br-vxlan tun0 -- set Interface tun0 type=vxlan options:rem
```

然后，两个主机上桥接到`br-vxlan`的虚拟机就像连接到同一个交换机一样，可以实现跨主机的L2连接，同时又完全与物理网络隔离。

Interface

Interface是连接到Port的网络接口设备，是OVS与外部交换数据包的组件，在通常情况下，Port和Interface是一对一的关系，只有在配置Port为 bond模式后，Port和Interface是一对多的关系。这个网络接口设备可能是创建`Internal`类型Port时OVS自动生成的虚拟网卡，也可能是系统的物理网卡或虚拟网卡(TUN/TAP)挂载在ovs上。 OVS中只有”Internal”类型的网卡接口才支持配置IP地址

`Interface`是一块网络接口设备，负责接收或发送数据包，Port是OVS网桥上建立的一个虚拟端口，`Interface`挂载在Port上。

Controller

OpenFlow控制器。OVS可以同时接受一个或者多个OpenFlow控制器的管理。主要作用是下发流表(Flow Tables)到OVS，控制OVS数据包转发规则。控制器与OVS通过网络连接，不一定要在同一主

机上

可以看到上面实例中三个网桥`br-int`,`br-ext`,`br-tun`都连接到控制器`Controller "tcp:127.0.0.1:6633`上

datapath

OVS内核模块，负责执行数据交换。其内部有作为缓存使用的flows，上面已经介绍过

OVS中的各种流(flows)

flows是OVS进行数据转发策略控制的核心数据结构，区别于Linux Bridge是个单纯基于MAC地址学习的二层交换机，flows的存在使OVS作为一款SDN交换机成为云平台网络虚拟机化主要组件

OVS中有多种flows存在，用于不同目的，但最主要的还是OpenFlow flows这种，文中未明确说明的flows都是指OpenFlow flows

OpenFlow flows

OVS中最重要的一种flows，Controller控制器下发的就是这种flows，OVS架构部分已经简单介绍过，关于openflow的具体使用，会在另一篇文章中说明

“hidden” flows

OVS在使用OpenFlow flow时，需要与OpenFlow控制器建立TCP连接，若此TCP连接不依赖OVS，即没有OVS依然可以建立连接，此时就是`out-of-band control`模式，这种模式下不需要”hidden” flows

但是在`in-band control`模式下，TCP连接的建立依赖OVS控制的网络，但此时OVS依赖OpenFLow控制器下发的flows才能正常工作，没法建立TCP连接也就无法下发flows，这就产生矛盾了，因此需要存在一些”hidden” flows，这些”hidden” flows保证了TCP连接能够正常建立。关于`in-band control`详细介绍，参考OVS官方文档Design Decisions In Open vSwitch 中**In-Band Control**部分

“hidden” flows优先级高于OpenFlow flows，它们不需要手动设置。可以使用`ovs-appctl`查看这些flows，下面命令输出内容包括`OpenFlow flows`,`"hidden" flows`

```
ovs-appctl bridge/dump-flows <br>
```

datapath flows

datapath flows是`datapath`内核模块维护的flows，由内核模块维护意味着我们并不需要去修改管理它。与OpenFlow flows不同的是，它不支持优先级，并且只有一个表，这些特点使它非常适合做缓存。与OpenFlow一样的是它支持通配符，也支持指令集(多个action)

datapath flows可以来自用户空间`ovs-vswitchd`缓存，也可以是datapath内核模块进行MAC地址学习到的flows，这取决与OVS是作为SDN交换机，还是像Linux Bridge那样只是一个简单基于MAC地址学习的二层交换机

*管理*flows*的命令行工具*

我们可以修改和配置的是OpenFlow flows。datapath flow和"hidden" flows由OVS自身管理，我们不必去修改它。当然，调试场景下还是可以使用工具修改的

介绍下上面三种flows管理工具，不具体说明，具体使用可以查看相关man手册

- `ovs-ofctl dump-flows <br>` 打印指定网桥内的所有OpenFlow flows，可以存在多个流表(flow tables)，按表顺序显示。不包括"hidden" flows。这是最常用的查看flows命令，当然这条命令对所有OpenFlow交换机都有效，不单单是OVS
- `ovs-appctl bridge/dump-flows <br>` 打印指定网桥内所有OpenFlow flows，包括"hidden" flows，`in-band control`模式下排错可以用到
- `ovs-dpctl dump-flows [dp]` 打印内核模块中datapath flows，`[dp]`可以省略，默认主机中只有一个datapath `system@ovs-system`
  man手册可以找到非常详细的用法说明，注意`ovs-ofctl`管理的是OpenFlow flows

ovs-*工具的使用及区别

上面介绍了OVS用户空间进程以及控制器和OpenFlow协议，这里说下相关的命令行工具的使用及区别

ovs-vsctl

`ovs-vsctl`是一个管理或配置`ovs-vswitchd`的高级命令行工具，高级是说其操作对用户友好，封装了对数据库的操作细节。它是管理OVS最常用的命令，除了配置flows之外，其它大部分操作比如Bridge/Port/Interface/Controller/Database/Vlan等都可以完成

```
#添加网桥br0
ovs-vsctl add-br br0
#列出所有网桥
ovs-vsctl list-br
#添加一个Port p1到网桥br0
ovs-vsctl add-port br0 p1
#查看网桥br0上所有Port
ovs-vsctl list-ports br0
#获取br0网桥的OpenFlow控制器地址，没有控制器则返回空
ovs-vsctl get-controller br0
#设置OpenFlow控制器,控制器地址为192.168.1.10，端口为6633
ovs-vsctl set-controller br0 tcp:192.168.1.10:6633
#移除controller
ovs-vsctl del-controller br0
#删除网桥br0
ovs-vsctl del-br br0
#设置端口p1的vlan tag为100
ovs-vsctl set Port p1 tag=100
#设置Port p0类型为internal
ovs-vsctl set Interface p0 type=internal
#添加vlan10端口，并设置vlan tag为10，Port类型为Internal
ovs-vsctl add-port br0 vlan10 tag=10 -- set Interface vlan10 type=internal
```

```
#添加隧道端口gre0，类型为gre，远端IP为1.2.3.4
ovs-vsctl add-port br0 gre0 -- set Interface gre0 type=gre options:remote_ip=
```

ovsdb-tool

`ovsdb-tool`是一个专门管理OVS数据库文件的工具，不常用，它不直接与`ovsdb-server`进程通信

```
#可以使用此工具创建并初始化database文件
ovsdb-tool create [db] [schema]
#可以使用ovsdb-client get-schema [database]获取某个数据库的schema(json格式)
#可以查看数据库更改记录，具体到操作命令，这个比较有用
ovsdb-tool show-log -m
record 48: 2017-01-07 03:34:15.147 "ovs-vsctl: ovs-vsctl --timeout=5 -- --if-
        table Interface row "tapcea211ae-10" (151f66b6):
                delete row
        table Port row "tapcea211ae-10" (cc9898cd):
                delete row
        table Bridge row "br-int" (fddd5e27):
        table Open_vSwitch row a9fc1666 (a9fc1666):

record 49: 2017-01-07 04:18:23.671 "ovs-vsctl: ovs-vsctl --timeout=5 -- --if-
        table Port insert row "tap5b4345ea-d5" (4befd532):
        table Interface insert row "tap5b4345ea-d5" (b8a5e830):
        table Bridge row "br-int" (fddd5e27):
        table Open_vSwitch row a9fc1666 (a9fc1666):
...
```

ovsdb-client

`ovsdb-client`是`ovsdb-server`进程的命令行工具，主要是从正在运行的`ovsdb-server`中查询信息，操作的是数据库相关

```
#列出主机上的所有databases，默认只有一个库Open_vSwitch
ovsdb-client list-dbs
#获取指定数据库的schema信息
ovsdb-client get-schema [DATABASE]
#列出指定数据库的所有表
ovsdb-client list-tables [DATABASE]
#dump指定数据库所有数据,默认dump所有table数据，如果指定table，只dump指定table数据
ovsdb-client dump [DATABASE] [TABLE]
#监控指定数据库中的指定表记录改变
ovsdb-client monitor DATABASE TABLE
```

ovs-ofctl

`ovs-ofctl`是专门管理配置OpenFlow交换机的命令行工具，我们可以用它手动配置OVS中的OpenFlow flows，注意其不能操作datapath flows和”hidden” flows

```
#查看br-tun中OpenFlow flows
ovs-ofctl dump-flows br-tun
#查看br-tun端口信息
ovs-ofctl show br-tun
#添加新的flow：对于从端口p0进入交换机的数据包，如果它不包含任何VLAN tag，则自动为它添加VL
ovs-ofctl add-flow br0 "priority=3,in_port=100,dl_vlan=0xffff,actions=mod_vla
#对于从端口3进入的数据包，若其vlan tag为100，去掉其vlan tag，并从端口1发出
ovs-ofctl add-flow br0 in_port=3,dl_vlan=101,actions=strip_vlan,output:1
#添加新的flow: 修改从端口p1收到的数据包的源地址为9.181.137.1,show 查看p1端口ID为100
ovs-ofctl add-flow br0 "priority=1 idle_timeout=0,in_port=100,actions=mod_nw_
#添加新的flow: 重定向所有的ICMP数据包到端口 p2
ovs-ofctl add-flow br0 idle_timeout=0,dl_type=0x0800,nw_proto=1,actions=outpu
#删除编号为 100 的端口上的所有流表项
ovs-ofctl del-flows br0 "in_port=100"
```

`ovs-vsctl`是一个综合的配置管理工具，`ovsdb-client`倾向于从数据库中查询某些信息，而`ovsdb-tool`是维护数据库文件工具

文章地址https://opengers.github.io/openstack/openstack-base-use-openvswitch/

参考文章

> https://www.sdxcentral.com/cloud/open-source/definitions/what-is-open-vswitch/
>
> http://openvswitch.org/features/
>
> https://www.ibm.com/developerworks/cn/cloud/library/1401_zhaoyi_openswitch/
>
> http://openvswitch.org/slides/OpenStack-131107.pdf
>
> http://horms.net/projects/openvswitch/2010-10/openvswitch.en.pdf
>
> http://benpfaff.org/papers/ovs.pdf
>
> https://networkheresy.com/category/open-vswitch/

默认使用ovs情况下，neutron实现了一个简单的flow控制器

回复 · 分享 ›

jython.li 管理员 ➔ 谢小天 · 2年前

你用的什么l2 agent？ "每个桥都没有连接控制器" 这个是怎么确定的?

回复 · 分享 ›

郭小明 · 2年前

“有一点要注意的是，挂载到OVS上的网卡设备不支持分配IP地址，因此若之前eth1配置有IP地址，挂载到OVS之后IP地址将不可访问。”

这个有问题。如果创建网桥的时候datapath_type指定为netdev，是没有影响的。

回复 · 分享 ›

Danster Huang · 2年前

好文，对OVS的介绍简单易懂，特别在OVS实现原理以及OpenFlow部分说明得非常透彻！

回复 · 分享 ›

在 **OPENGERS** 上还有

openstack底层技术-虚拟网络设备(Bridge,VLAN)

1条评论 • 1年前

Qunli Zhang — 写的很好，这段也在看这方面馆的知识，学习了，谢谢分享！

openstack底层技术-openflow在OVS中的应用

1条评论 • 1年前

大大薇 — 赞，感谢分享！

About Me | opengers

1条评论 • 1年前

罗曙晖 — 能支持rss 吗?

virt-install工具安装基于rbd磁盘的虚拟机

5条评论 • 1年前

WanLi Chen — 我以为pool已经向nfs那样在每个节点上define了,在任意节点的更新都会在其他节点立即得到更新...为什么还需要