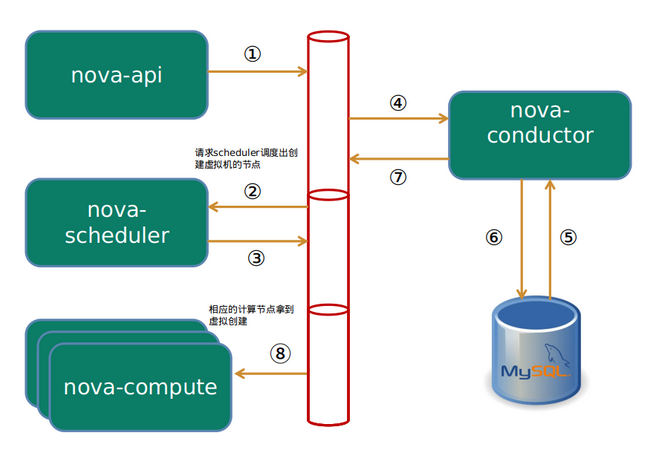
**RPC**

在openstack中使用两种通信方式，一种是Restful API，另一种是远程过程调用RPC。openstack中的RPC调用实现是自己的通用库oslo\_message，该库是对基于Rabbitmq实现的RPC的一个封装。



nova各个服务启动时就是开启了一个RPC的服务端，在调用其他服务时会创建一个客户端，通过客户端调用到相应的服务。以nova创建虚拟机为例，在上图的第三步中，nova-scheduler选择好计算节点，将创建虚拟机的信息通过RPC发送给计算节点。

openstack中RPC的实现封装在oslo\_messagin库当中。

**Neutron**

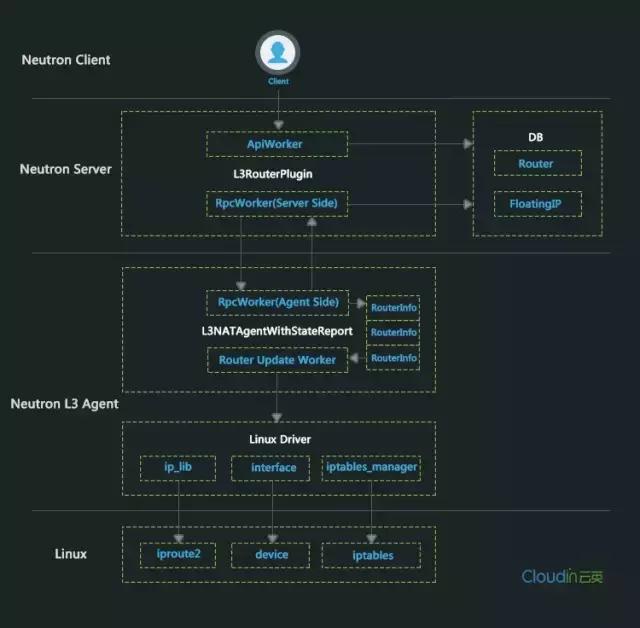
**Agent（完成的是how的问题，如何实现网络功能）：**

在 neutron 的架构中，各种 agent 运行在计算节点和网络节点上，接收来自 neutron-server 的 plugin 的指令，对所管理的网桥进行实际的操作，属于“直接干活”的部分。plugin 和 agent 之间进行双向交互，一般的，每个 plugin 会创建一个 RPC server 来监听 agent 的请求。

agent 可以大致分为 core agent、dhcp、l3 和其它（metadata等）。 本部分代码实现各种 agent 所需要的操作接口和库函数。

**Neutron-L3**

OpenStack Neutron中，Router提供虚拟三层服务，包括租户网络（Tenant Network）之间、租户网络与外部网络（External Network）之间的路由和网络地址转换，原生模式下路由器管理功能以Service Plugin的形式集成在Neutron Server中，路由器实体以Linux网络命名空间的形式存在，由Neutron L3 Agent管理。

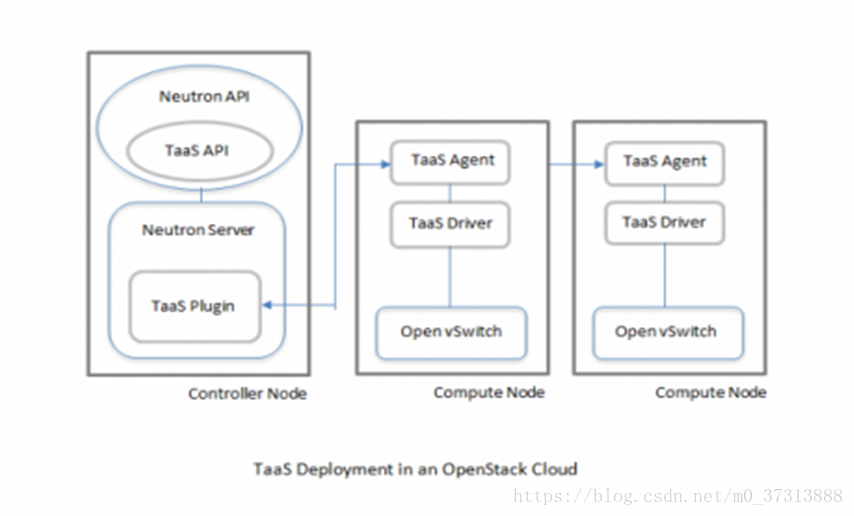


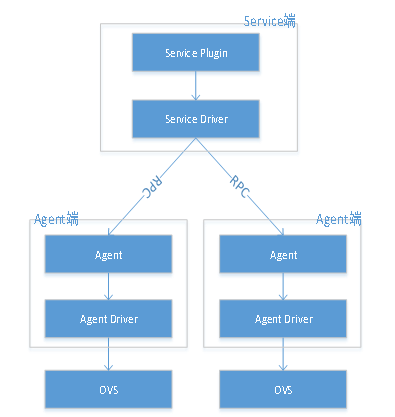
Neutron L3模块的功能主要由Server端的L3RouterPlugin和Agent端的L3NATAgentWithStateReport配合实现: Client通过HTTP向Server发起L3资源Router、Floating IP的管理命令，Server负责处理资源在数据库的增删改查，并通过RPC将资源的更新事件通知到Agent，Agent最终通过Linux下的网络工具栈来实现具体的L3功能。

**Taas(Neutron的一个延申)**

**将流量镜像到特定的、运行有流量分析软件的虚拟机中，以实现租户流量的可视化。**

OpenStack中云端流量捕获插件, 策略采集。作为OpenStack的Neutron的一项插件，其核心是为跨节点的vm以及por0t提供一个统一的流量监听以及捕获的方案。





TaaS总体框架如上图所示，主要分为两大部分，即：Service端和Agent端。通常Service端部署在控制节点上，而Agent端部署在计算节点上，二者之间通过RPC通道进行交互信息。图中的Service Driver和Agent Driver都是设计成可插式的，可以替换成其他Driver。

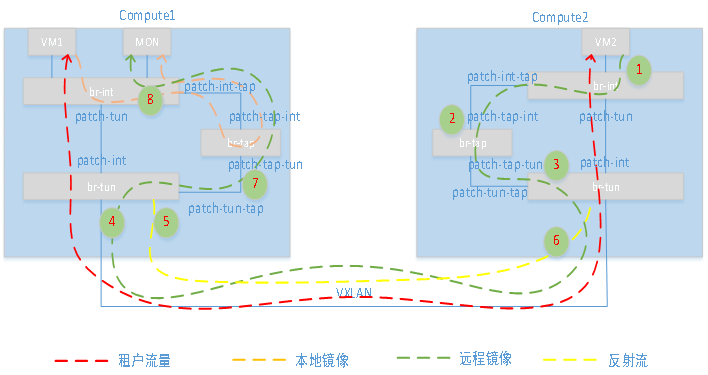
taas在这里是作为OpenvSwitch的一个extension集成到neutron-openvswitch-agent中,并且在neutron-openvswitch-agent启动时进行加载与初始化。

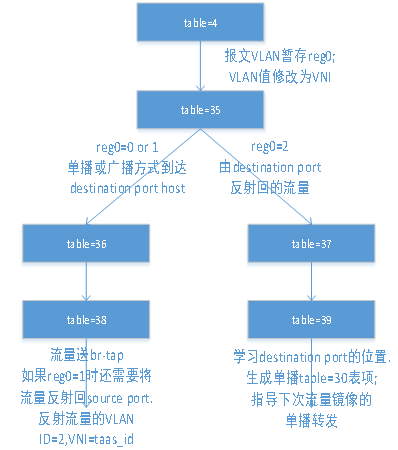
tap-as--a-service的主要设计理念是为不同节点上的虚拟机提供统一的流量捕获服务，这样的话方案中流量必须能够方便的进行跨主机迁移。如果采用vxlan隧道的话就很容易建立新的流量隧道实现流量的跨节点迁移。换句话说，流量既能够定向到相同宿主机上的另一台虚拟机，也能重定向到另一台宿主机上的另一台虚拟机！

TAAS工作流始于tap-service实例的建立，并将有Neutron 端口为端口镜像会话目的端口。监控虚拟机通常连接到这个端口，以消耗镜像流量。接着，一个或多个抽头流可以添加到tap-service实例。

def create\_tap\_service(self, context, tap\_service):

…



第1步：VM2发出流量到达br-int，匹配taas相关流表，给报文打上VLAN ID=taas\_id，从patch-int-tap口送出；  
第2步：br-tap收到VM2的镜像流量，匹配流表，发现destination port不在本地(没有动态生成的table=1流表项，因此判断不在本地)，于是将流量从patch-tap-tun送出；  
第3步：br-tun收到VM2的镜像流量，匹配流表，送到table=30的流表项继续匹配。table=30表项是通过学习动态生成的单播表项，如果没有学习到任何的单播表项，则会转到table=31(泛洪表项)进行广播泛洪。这里假设没有学习到任何的单播table=30流表项，因此选择广播泛洪，封装报文为VXLAN，VNI为报文的VLAN ID值，而报文原有的VLAN ID值修改成1，接着从所有的VXLAN端口中泛洪出去；  
第4步：br-tun收到镜像流量后，走到table=4匹配，将报文的VLAN值暂存到寄存器reg0中(有分类表table=35作进一步分类)，将报文的VLAN值修改为VNI值，送入table=35作进一步分类。这一步流程详细可以参考下图的Pipeline(只给出与taas相关的匹配表)。  
  
第5步：将流量反射给source port，具体参考上图的Pipeline；  
第6步：source port所在host的br-tun接收到反射流后，学习单播表项，具体参考上图的pipeline；  
第7步：br-tap接收到VM2的镜像流后，从patch-tap-int端口送出；  
第8步：br-int接收到VM2的镜像流后，修改报文的VLAN值为MON虚拟机关联的内部VLAN值，最后送入MON虚拟机。

**Ceilometer**

ceilometer项目是openstack中用来做计量计费功能的一个组件，后来又逐步发展增加了部分监控采集、告警的功能。

meter：

针对一个资源的某些测量值，比如，一个虚拟机可以有多个meters：虚拟机在一段时间内cpu的使用时间、磁盘的请求次数等。statistics 一般是统计学上某个周期内，meter对应的值(平均值之类)。在ceilometer中针对这些meter定义了三种类型：

* Cumulative(累积型): 随着时间会不断增长(eg. disk I/O)
* Gauge(测量型):离散型的值(eg. floating IPs)和浮动的值(eg. swift对象的数量)
* Delta(变化量)：一段时间内某个采集值的变化量(eg. 带宽变化量)

sample：

针对一个特定的meter的具体数据结构

event:

在某个特定时间，发生的一个动作，比如：19:09:08 创建了一个虚拟机

resource：

资源，比如instance(虚拟机)、disk(磁盘）都是资源。

**ceilometer收集数据的方式**

OpenStack原始数据的收集方式有两种：

一种是通过ceilometer polling agent主动轮询方式，调用相应插件获取性能数据；

另一种是各openstack核心组件主动向消息队列上报自身的性能数据。这两种收集方式都将原始数据发送到消息队列。

ceilometer notification agent监听该消息队列，并将这些原始数据按照一定规则转换成sample和event，再发布到配置的目的端。可配的发布方式包括direct、notifier、udp、kafka和file。

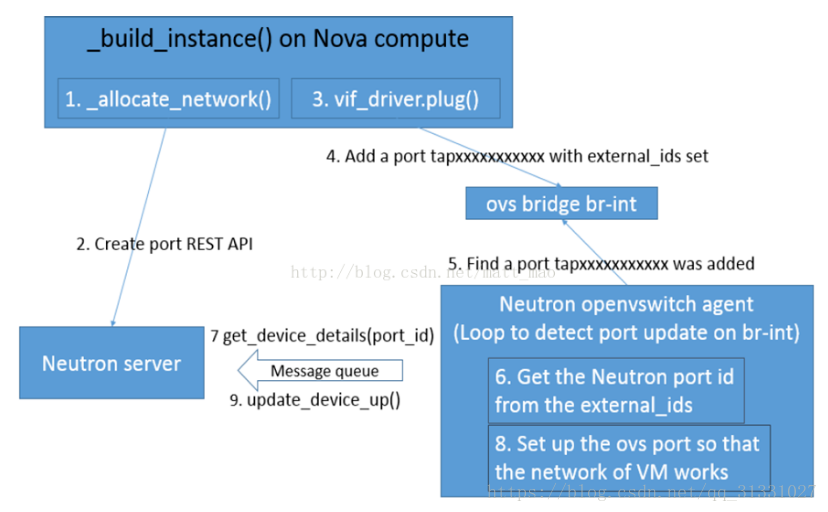
ceilometer collector是可选服务，它从notification消息队列中消费sample和event消息，然后将数据dispatch到配置的目的端：database、file、http、gnocchi，该目的端可以同时配置多项。

**最近思考的问题：**

开发计划/neutron与sdn的集成/neutron的agent怎么对服务做出响应

neutron-openvswitch-agent服务：

openvswitch正在底层实际提供二层通信服务，agent对ovs：增加删除虚拟端口、创建vlan、决定转发策略等等



7，9关键

启动ovs\_agent时：设置plugin\_rpc（rpc通信）、state\_rpc（agent状态信息上报）、connection（接收neutron-server的信息）、br-int（ovs网桥）、设置bridge\_mapping对应的网桥，周期检测br-int

上的端口变化，调用process\_networks\_ports处理添加/删除端口。