Tencent 腾讯 | 腾讯云

TStack 云计算管理平台

技术白皮书

前言

产品版本

本文档对应的软件以及版本为：

产品名称	产品版本
TStack 云计算管理平台	V3.0

读者对象

本书适合下列人员阅读：

- 网络工程师
- 技术推广人员

举例说明

- 本手册举例说明部分的端口类型同实际可能不符，实际操作中需要按照各产品所支持的端口类型进行操作。
- 本手册部分举例的显示信息中可能含有其它产品系列的内容（如产品型号、描述等），具体显示信息请以实际使用的设备信息为准。
- 本手册中涉及的路由器及路由器产品图标，代表了一般意义下的路由器，以及运行了路由协议的三层交换机。

修改记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

- 文档版本 01 (2019-02-28)
  第一次正式发布。

腾讯公司 版权所有

目录

腾讯公司 版权所有

腾讯公司 版权所有

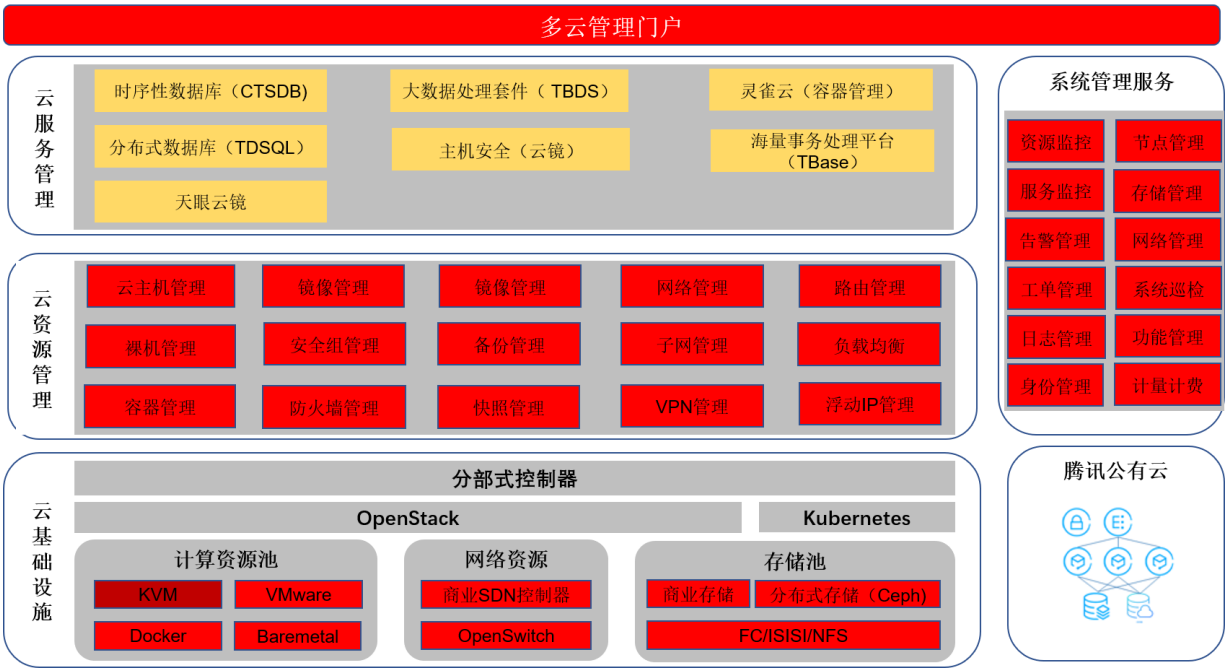
1 TStack 产品简介

TStack 是为企业用户提供的用于统一管理多种云资源的 SaaS 云计算管理平台。通过超融合、软件定义网络、容器、自动化运维等技术的综合应用，使企业能够以最小的初始成本快速实现 IT 基础设施的“云化”；同时，产品可以随着企业规模的扩大、自身业务的增长，实现“积木堆叠式”的弹性扩容，按需升级。以企业的视角，从数据中心、部门、项目等不同的维度对资源进行统一的规划、管理和计量。

整个平台基于 OpenStack 开发，提供标准的 OpenStack API 接口。

TStack 支持混合云管理模式，支持 KVM、VMware 等虚拟化管理，同时支持腾讯云等公有云的管理，由云计算管理平台提供统一管理。针对不同的业务场景提供两种不同的解决方案，一种采用的是优化的超融合一体机和基于互联网的 SaaS 服务模式的 TStack 平台统一交付，充分利用 TStack 的快速部署、零运维的优势，满足企业业务快速云化的需求；另一种是单独的软件解决方案，将 TStack 软件部署于企业本地 IT 基础设施上，充分利用企业既有 IT 物理设备，在企业业务云化过程中为企业节省大量 IT 建设成本，满足大型企业业务需求。

2 TStack 整体技术架构



整个 TStack 云计算管理平台包括云服务管理、云资源管理和云基础设施三部分。

云管平台分为计算资源管理、网络资源管理、存储资源管理、监控管理、运营管理、运维管理、用户管理、工单日志和安装部署等模块。

在基础设施层中，分为私有云部分和公有云两部分。

私有云由分布式控制平面统一管理负责计算、存储、网络资源调度的 OpenStack 和 Kubernetes 服务的生命周期两部分组成，在底层的计算部分可以同时管理 KVM/VMware 云主机、物理裸机和容器；网络部分兼容开源的 OpenvSwitch 和商业 SDN 控制器；在存储部分兼容市面上几乎所有的存储设备，包括 FC、ISCSI 集中存储、NFS 存储及 Ceph 分布式存储等。公有云支持无缝纳管腾讯云中的云服务器、数据库和网络资源，可以通过统一平台对腾讯云资源进行管理。

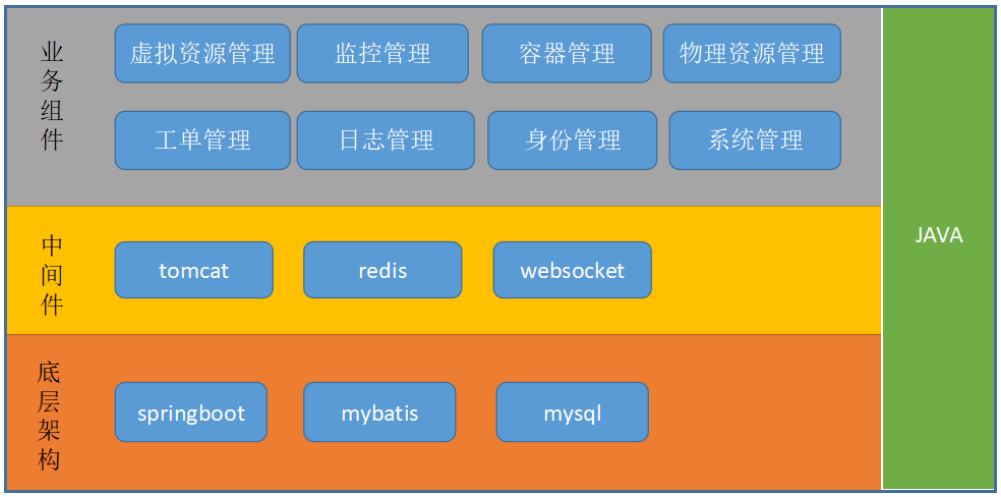
TStack 平台支持根据客户需求管理相关 PaaS 服务，使 TStack 平台具备提供 PaaS 相关服务的能力，便于用户能够快速使用各类云服务。

腾讯公司 版权所有

3 TStack 关键技术介绍

3.1 云管平台之 SaaS 云管

3.1.1 组件构成

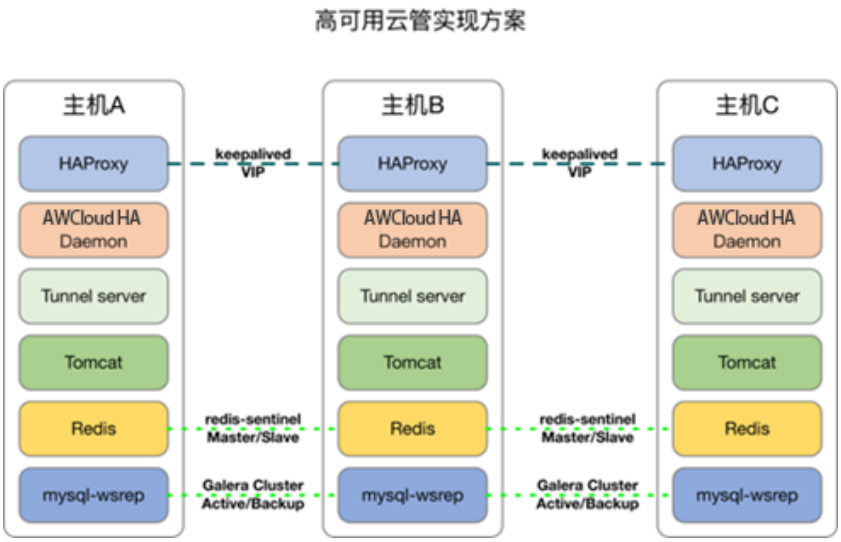


- 业务组件：实现 SaaS 云管的业务功能，主要包含虚拟资源管理，物理资源管理，容器管理，监控管理，工单管理，日志管理，身份管理，系统管理。
- 中间件：提供 SaaS 云管的服务引擎，消息缓存和消息订阅。
- 底层架构：构成整个 SaaS 云管的基础，统筹各个组件间的服务与通信。

3.1.2 高可用

高可用云管由三节点构成，每个节点均运行有 HAProxy、keepalived、tunnel server、tomcat、redis 以及 MySQL Galera 集群。

本方案使用 keepalived 提供外部访问的虚拟 IP（VIP），HAProxy 做负载均衡，tomcat 三节点为全为 Active 状态，虚拟 IP（VIP）位于哪个主机上，就访问对应的 tomcat。使用 Redis 作为缓存数据库，redis-sentinel 作 Redis 集群的监控，控制各个 Redis 节点的 Master 和 Slave 的转换。采用 MySQL Galera 集群作为云管的数据库。方案的架构实现如下图所示：



腾讯公司 版权所有

下沉版云管，云管服务运行在 Docker 容器内，通过 nomad 进行管理和调度，当云管容器不可用时，nomad 会重新启动一个容器，运行云管服务。

3.2 基础设施层之分布式控制平面

3.2.1 简介

3.2.1.1 形态

TStack 云计算管理平台采用 nomad+consul 的组合构建底层分布式控制平面，来运行 OpenStack 管理服务以及相关的监控日志服务。对无状态应用使用容器的方式来编排，有状态应用如 MySQL 使用虚拟机方式运行，nomad 可以统一进行调度编排，服务保活。通过 Consul 来实现服务发现和动态更新，Leader 选举，集群资产配置存储等功能。

3.2.1.2 容器编排

- OpenStack 管理服务的配置
  镜像继承自社区 Kolla 项目，并在此基础上进行修改。镜像里的服务配置一部分会通过主机名的方式固化，其余和环境相关的可以通过 consul-template 在镜像第一次 bootstrap 拉取。这个做法和社区的 kolla-mesos 做法相同，只不过社区使用的是 zookeeper，另外社区没有使用主机名的方式。通过主机名固化以后，服务的配置和环境相关的地方会非常少，甚至没有，但是会提供一个动态更新镜像的机制，方便运行时修改配置参数。可以把每个服务的配置模板以及配置参数都保存在 consul K/V 存储里，镜像提供是否更新配置的环境变量，如果需要更新，就从 consul 再拉取配置模板和配置来生成新的配置文件。

- Haproxy 的配置更新
  各服务之间也可以直接使用域名的方式做高可用和 load-balance，但是云管平台无法直接使用域名访问。所以还是使用 Haproxy 来做各服务之间的 load-balance，这样 Haproxy 需要动态更新配置文件，来保证新的后端服务可以加入。这个工作由 consul-template 来完成，在 Haproxy 镜像中运行 consul-template 这个 daemon，它会 watch consul 上注册的服务的变化，动态更新配置并重启 Haproxy 服务。

- 域名解析
  consul 可以对所有注册的服务做域名解析，而且在运行 consul agent 的机器上（我们的设计中，所有机器都会运行）可以直接通过 127.0.0.1 来完成域名解析，这样 DNS 服务器也不需要一个特殊的地址来发布，这是 consul 设计的巧妙之处。

3.2.1.3 服务调度

- nomad 可以实现容器，虚拟机，本地应用的统一资源调度。

- nomad 可以自动重启故障服务，或者创建新的实例来代替故障节点，也可以在不影响原有实例运行的情况下增加新的实例。

- nomad 可以实现不同服务之间的亲和力和反亲和力设置，实现同一服务的多个实例运行在不同节点上来保证高可用，而且依赖的服务运行在同一个节点上。

3.2.1.4 运维机器人 (Bob)

在每个节点完成自己的配置后，整个集群的初始化，以及后期的运维操作统一由运维机器人负责完成，名字 Bob。每个节点都运行这个服务，利用 Consul 做 leader 选举，成为 leader 的节点实际发生动作使用 Consul 获取实始化配置。包括的任务：

- 初始化，部署存储集群，向 nomad 提交作业运行所有管理服务，集群初始化

腾讯公司 版权所有

- 增加节点
- 维护节点
- 删除节点
- 节点晋升
- 增加服务实例数量

3.2.1.5 隧道

使用 SSH Tunnel 注册时，云管平台需要给该 OpenStack 环境指定一个 IP 地址和端口，客户数据中心服务器初始化时，在 Haproxy 所在节点，增加一个 task 建立隧道，把云管平台对指定地址和端口的请求转向数据中心的 OpenStack 集群，所有 OpenStack 对上使用同一个端口，这样就需要使用 Haproxy 或者 nginx 的七层代理功能，根据 URL 映射到不同的 OpenStack 服务中。在设置 public url 时，直接包括项目名字在 URL 里，这样不需要云端的 Haproxy，只需要数据中心的 Haproxy，根据不同的 url，映射到不同的后端。

使用一个 docker image 来跑 ssh tunnel，镜像参考 https://github.com/kingsquare/docker-tunnel 并注册服务检查 (http check 云端 tunnel 端点的 IP 和地址）。云端也可以使用 docker 来运行隧道端点 https://hub.docker.com/r/matttbe/docker-ssh-tunnel/。

3.2.1.6 高可用

所有被编排的服务都可能都通过分布式控制层实现高可用，所有节点都是对称的，只要有可用的资源，服务可以运行在任何节点上。nomad+consul 本身的可用性，都需要 3-5 个节点来组成集群，因为每个节点上运行的程序是一样的，只是启动的配置不同决定了是 server 还是 agent，可以通过让 Bob 检查 server 节点数量，来动态调整。另外在初始化部署时考虑物理服务器的拓扑关系，把 server 节点分散到不同故障域，在物理节点比较多的情况，可以部署 5 个 server 节点 Ceph monitor 高可用，可以增加数量，并且分布在不同的拓扑位置里。

3.2.2 优势

- 动态部署，管理服务和物理资源解耦，管理服务可以运行在所有节点，并且支持根据 API 请求压力的横向扩展。
- 统一调度，可以支持容器，虚拟机和物理机应用的统一调度运行。
- 发生节点故障，服务可以自动创建新的实例，保证完整可用性。
- 使用容器资源开销更小，服务启动快，便于实现滚动升级和回滚。
- 按微服务方式编排，使用域名来满足不同组件之间的服务发现。
- 可以方便部署新的管理应用，包括 K8S 集群的 controller 也可以通过这个控制面来管理部署。
- 资源隔离，每个服务都通过 cGroup 做资源隔离，保证不会因为这个服务程序的内存泄漏等问题影响客户虚拟机的运行。

3.3 计算资源管理

3.3.1 计算资源管理概述

TStack 计算资源管理模块依托于 OpenStack Nova 项目设计开发，实现对虚拟机、物理裸机的统一管理，此外为了方便云主机的快速创建在云平台中使用了基于 OpenStack Glance 模块开发的镜像管理功能。下

腾讯公司 版权所有

面将会对涉及到的管理平面（Nova、Glance 和裸机管理）和数据平面（KVM、QEMU）的技术原理进行逐一介绍。

3.3.2 OpenStack Nova

3.3.2.1 Nova 简介

Nova是OpenStack中的计算调度控制器。云主机生命周期的所有活动都由Nova处理。Nova自身并没有提供任何虚拟化能力，其使用libvirt API来与被支持的Hypervisors交互。

3.3.2.2 Nova 架构

- nova-api

  接收和响应用户的API请求，例如创建虚拟机操作。

- nova-compute

  执行创建和终止虚拟机的进程，该进程通过调用hypervisor的API（如XenAPI，Libvirt接口，VMwareAPI等等）

- nova-scheduler

  决策虚拟机创建在某一台物理主机（计算节点）上。

- nova-conductor

  nova-compute经常需要更新数据库，比如更新虚拟机的状态。出于安全性和伸缩性的考虑，nova-compute并不会直接访问数据库，而是将这个任务委托给nova-conductor。

- nova-consoleauth

  负责对访问虚拟机控制台请求提供Token认证。

- nova-novncproxy

  提供基于浏览器形式的VNC访问运行的实例。

3.3.2.3 Nova 相关概念

3.3.2.3.1 Aggregate Hosts

Aggregate Host 是管理员用来根据硬件资源的某一属性来对硬件进行划分的功能，只对管理员可见。

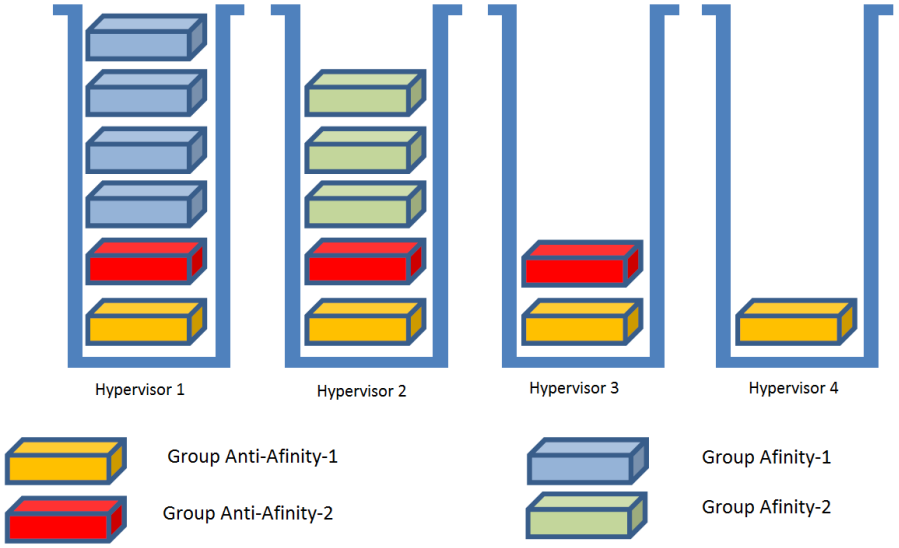
其主要功能就是实现根据某一属性来划分物理机，比如按照地理位置，使用固态硬盘的机器，内存超过 32G 的机器，根据这些指标来构成一个 Aggregate Host，例如创建一个 Aggregate Host 组，设置了属性为 ssd=true，创建 flavor 是也指定了 ssd=true，那么基于这个 flavor 创建出的虚拟机就会被 nova scheduler 调度到相应的 Aggregate Host。也可以将某个租户的使用资源绑定到 Aggregate Host。

3.3.2.3.2 Server Group

Server Group 是 Icehouse 版本新加入的功能，是一个虚拟机组，针对组可以设置不同策略，并配合 nova scheduler 调度达到相应目的，策略如下：

- Affinity：策略强制 nova 调度机制调度相同的组中的虚拟机到相同的物理宿主机。
- Anti-Affinity：策略强制 nova 调度机制调度相同的组中的虚拟机到不同的物理宿主机。

腾讯公司 版权所有



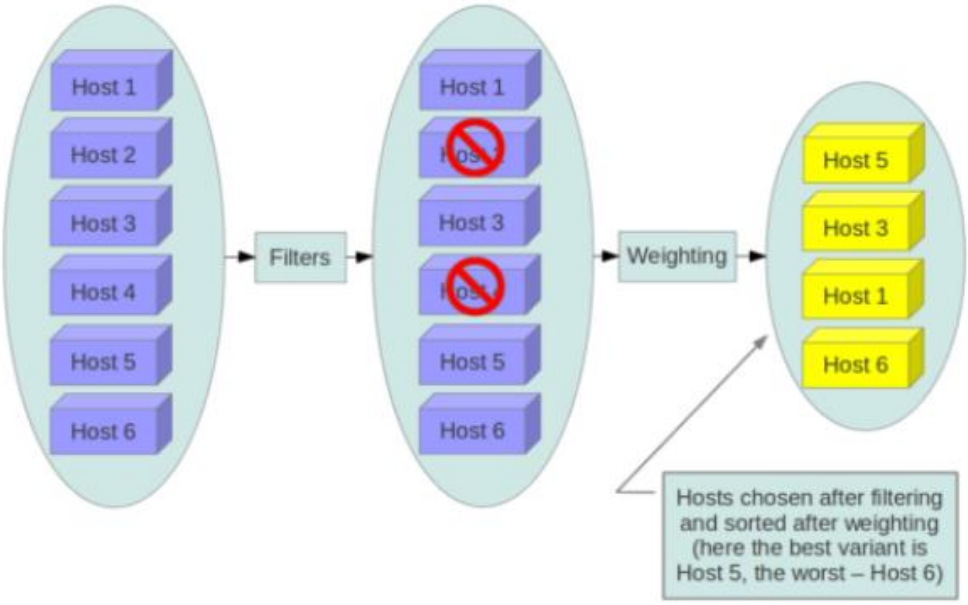
3.3.2.4 Nova 调度机制

Filter scheduler 是 nova-scheduler 默认的调度器，调度过程分为两步：

1. 通过过滤器（filter）选择满足条件的计算节点（运行 nova-compute）。

2. 通过权重计算（weighting）选择在最优（权重值最大）的计算节点上创建 Instance。

在/etc/nova/nova.conf 中，nova 通过 Driver，available_filters 和 enabled_filters 这三个参数来配置 nova-scheduler。Nova 允许使用第三方 scheduler，配置 Driver 即可，默认是 filter_scheduler。Scheduler 可以使用多个 filter 依次进行过滤，过滤之后的节点再通过计算权重选出最适合的节点。



上图是调度过程的一个示例：

最开始有 6 个计算节点 Host1-Host6，通过多个 filter 层层过滤，Host2 和 Host4 没有通过，被刷掉了。Host1，Host3，Host5，Host6 计算权重，结果 Host5 得分最高，最终入选。

腾讯公司 版权所有

3.3.3 OpenStack Glance

3.3.3.1 Glance 简介

Glance服务提供虚拟机镜像的集中式仓库，可以提供发现、注册、和下载镜像。通过Glance的RESTful API，可以查询镜像元数据或下载镜像。虚拟机的镜像可以很方便的存储在各种地方，从简单的文件系统到对象存储系统等等。

Glance有以下几个组件：

- Glance-api：接收镜像发现，查询，和存储镜像的API调用。
- Glance-registry：存储，处理和查询镜像的元数据信息。
- Glance Store：适配各类型存储，比如文件系统，S3，RBD，Swift以及其它的HTTP镜像存储。

3.3.3.2 Glance 镜像格式

当上传镜像时，需要指定虚拟机镜像的格式，Glance 支持各种格式，如磁盘格式和容器格式。

3.3.3.2.1 磁盘格式

虚拟机镜像的磁盘格式是底层磁盘镜像的格式。虚拟设备供应商有不同的格式来放置虚拟机磁盘镜像中包含的信息。

磁盘格式支持如下：

Disk Format	Notes
Raw	非结构化磁盘格式
VHD	微软的虚拟机格式，kvm不支持这种格式
VMDK	Vmware的镜像格式
qcow2	Qemu镜像格式，KVM和QEMU的原生格式。支持更多的功能
VDI	起源于Oracle VM VirtualBox的虚拟镜像格式
ISO	光盘的存储格式
AMI, ARI, AKI	亚马逊的镜像格式，分别表示 machine, ramdisk, kernel

3.3.3.2.2 容器格式

容器格式指的是该文件格式包含虚拟机镜像文件和虚拟机元数据。

支持的容器格式如下：

腾讯公司 版权所有

Container Formats	Notes
bare	镜像没有容器和封装元数据
ovf	OVF格式
aki	Amazon kernel 镜像
ari	Amazon ramdisk 镜像
ami	Amazon machine镜像
ova	OVA tar文件
docker	container文件系统的tar包文件

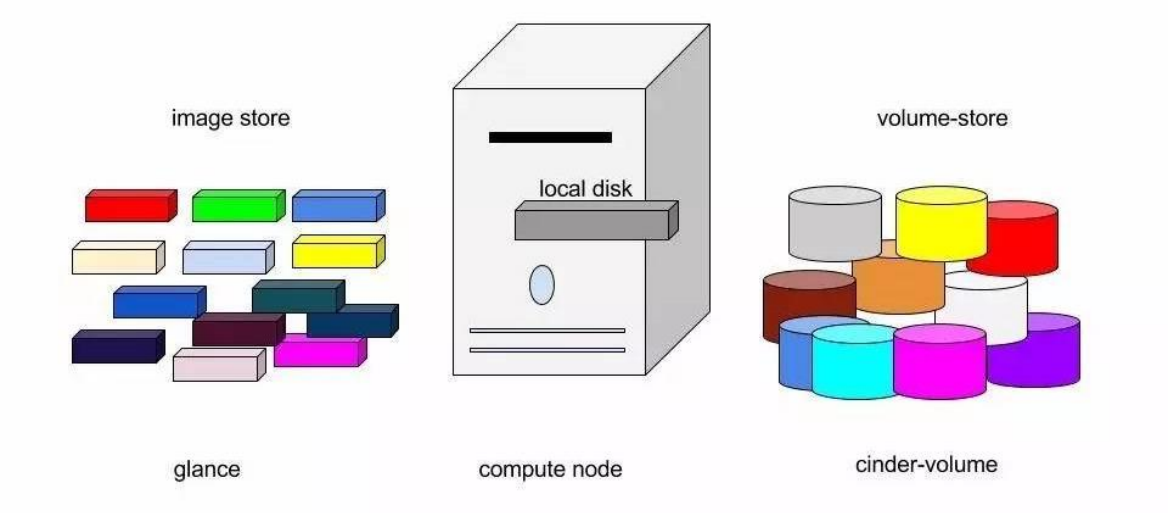
3.3.3.3 Glance 镜像访问权限

- public 公共的：可以被所有的用户使用。
- private 私有的/项目的：只能被 image owner 所在的 tenant 使用。
- protected 受保护的：设置为是，则任何人不能删除该镜像；如要删除只能管理员重新设置为否。

3.3.3.4 Glance 流程介绍

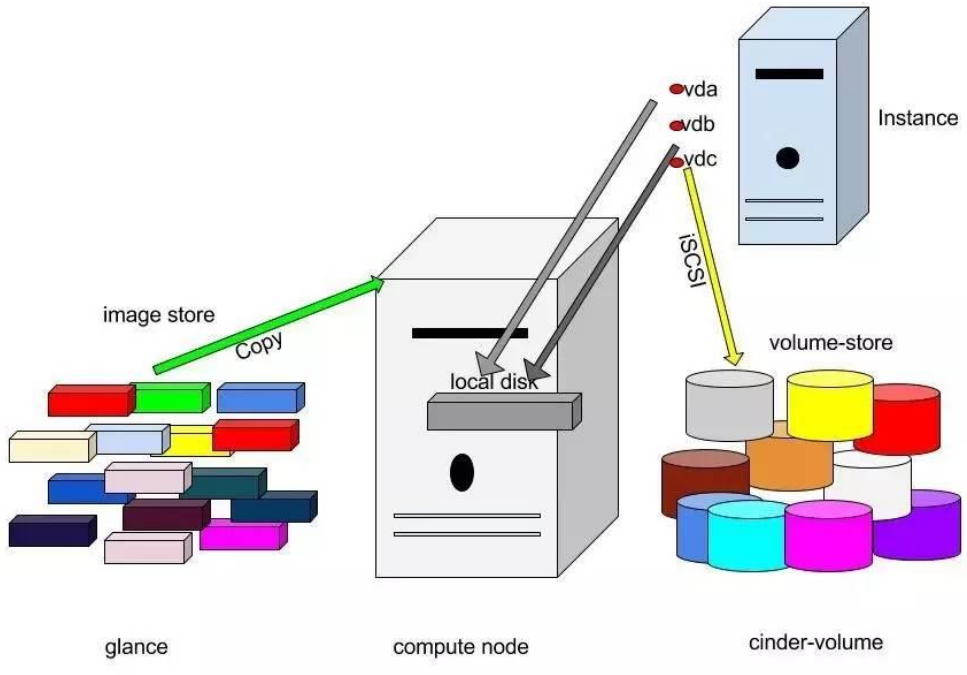
当启动一个实例时，需要指定 flavor。flavor 定义了实例可以有多少个虚拟 CPU，多大的 RAM 以及临时磁盘的大小。

下图显示了启动实例的系统状态。image store 包含一定数量的镜像，计算节点包含可用的 vcpu，内存和本地磁盘资源，Cinder-volume 包含一定数量的 volume。



启动实例之前，需要选择一个镜像，选定的镜像提供一个系统盘，标记为 vda，另外一个临时盘被标记为 vdb，Cinder-volume 提供的 volume 被映射到第三个虚拟磁盘并将其称为 vdc。

腾讯公司 版权所有



在该图中，基础镜像从 image store 复制到本地磁盘（如果后端使用 Ceph 就减少从控制节点下载到计算节点的过程，直接在 Ceph 后端复制）。vda 是实例的第一个磁盘，如果镜像文件越小，则通过网络复制的数据越少，实例启动就会更快。实例创建时会创建一块空的临时数据盘 vdb，当实例终止时，它将被删除。

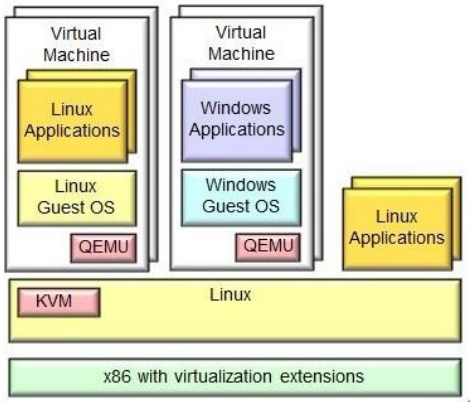
vdc 使用 iSCSI 连接到 Cinder-volume。随后计算节点提供 vCPU 和内存资源之后，该实例将从 vda.启动。该实例运行并更改磁盘上的数据。如果 volume store 位于单独的网络上，需要在计算节点 nova.conf 里指定的 my_block_storage_ip，将会通过存储网络直接通信。

实例被删除后，除 volume 之外的其它资源都会被回收。临时盘将会被清空，内存和 vCPU 资源被会被释放。在这个过程中镜像不会发生任何改变。

3.3.4 KVM

3.3.4.1 KVM 简介

KVM 全称 kernel-based virtual machine，由 Qumranet 公司发起，2008 年被 RedHat 收购。KVM 实现主要基于 Intel-V 或者 AMD-V 提供的虚拟化平台，KVM 的虚拟化需要硬件支持(如 Intel VT 技术或者 AMD V 技术)，是基于硬件的完全虚拟化。利用普通的 Linux 进程运行于虚拟态的指令集，模拟虚拟机监视器和 CPU。KVM 不提供硬件虚拟化操作，其 IO 操作等都借助 QEMU 来完成。KVM 是目前主流的虚拟化技术，自 Linux 2.6.20 之后集成在各主要 Linux 发行版本中。



腾讯公司 版权所有

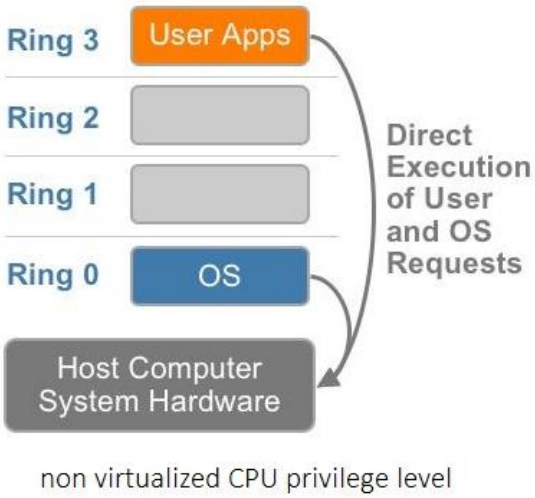
3.3.4.2 CPU 虚拟化

x86 操作系统是直接运行在裸硬件上的系统，因此它们自动认为它们完全占有计算机硬件。x86 架构提供四个特权级别给操作系统和应用程序来访问硬件。

Ring 是指 CPU 的运行级别：

- Ring 0
- Ring 1
- Ring 2
- Ring 3

就 x86 来说，操作系统（内核）需要直接访问硬件和内存，因此它的代码需要运行在最高运行级 Ring0 上，这样它可以使用特权指令，控制中断、修改页表、访问设备等等。应用程序的代码运行在最低运行级别上 Ring3 上，不能做受控操作。如果要做，比如要访问磁盘，写文件，那就要通过执行系统调用（函数），执行系统调用的时候，CPU 的运行级别会发生从 Ring3 到 Ring0 的切换，并跳转到系统调用对应的内核代码位置执行，这样内核就为你完成了设备访问，完成之后再从 Ring0 返回 Ring3。这个过程也称作用户态和内核态的切换。



但是 X86 虚拟化后就遇到难题，因为宿主操作系统是工作在 Ring0 的，客户操作系统就不能也在 Ring0 了，但是它不知道这一点，以前执行什么指令，现在还是执行什么指令，但是没有执行权限是会出错的。所以这时候虚拟机管理程序（VMM）需要避免这件事情发生。虚拟机如何通过 VMM 实现 Guest CPU 对硬件的访问，根据其原理不同有三种实现技术：

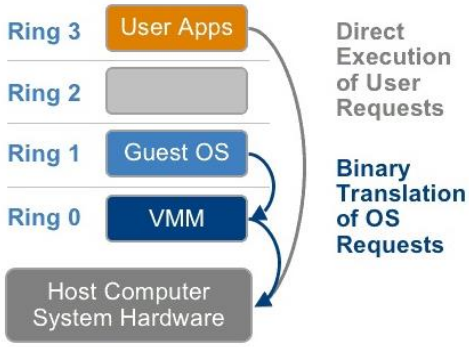
- 全虚拟化
- 半虚拟化
- 硬件辅助的虚拟化

3.3.4.2.1 CPU 虚拟化类型

- **二进制翻译的全虚拟化（Full Virtualization with Binary Translation）**

客户操作系统运行在 Ring 1，它在执行特权指令时，会触发异常（CPU 的机制，没权限的指令会触发异常），VMM 运行在 Ring 0，VMM 捕获这个异常，在异常里面做翻译，模拟，最后返回到客户操作系统内，客户操作系统认为自己的特权指令工作正常，继续运行。但是这个性能损耗，就非常的大，简单的一条指令，执行完结束，现在却要通过复杂的异常处理过程。

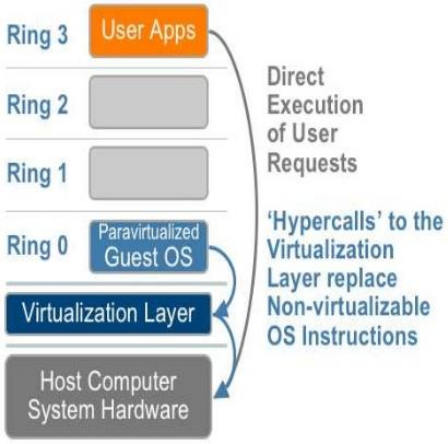
腾讯公司 版权所有



(Full Virtualization) Binary Translation

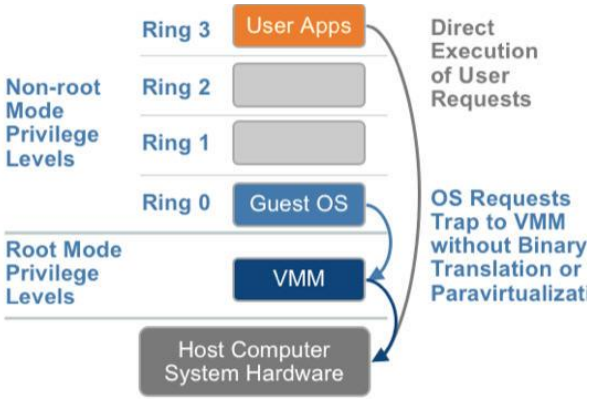
- **半虚拟化（Paravirtualization）**

半虚拟化的思想就是，修改操作系统内核，替换掉不能虚拟化的指令，通过超级调用（hypercall）直接和底层的虚拟化层 hypervisor 来通讯，hypervisor 同时也提供了超级调用接口来满足其他关键内核操作，比如内存管理、中断和时间保持。这种做法省去了全虚拟化中的捕获和模拟，大大提高了效率。所以像 XEN 这种半虚拟化技术，客户机操作系统都是有一个专门的定制内核版本，和 X86、mips、arm 这些内核版本等价。这样以来，就不会有捕获异常、翻译、模拟的过程了，性能损耗非常低。这就是 XEN 这种半虚拟化架构的优势。这也是为什么 XEN 只支持虚拟化 Linux，无法虚拟化 Windows 原因。



- **硬件辅助的全虚拟化**

2005 年后，CPU 厂商 Intel 和 AMD 开始支持虚拟化了。Intel 引入了 Intel-VT（Virtualization Technology）技术。这种 CPU，有 VMX root operation 和 VMX non-root operation 两种模式，两种模式都支持 Ring 0 ~ Ring 3 共 4 个运行级别。这样，VMM 可以运行在 VMX root operation 模式下，客户 OS 运行在 VMX non-root operation 模式下。硬件辅助的全虚拟化技术的性能逐渐逼近半虚拟化。



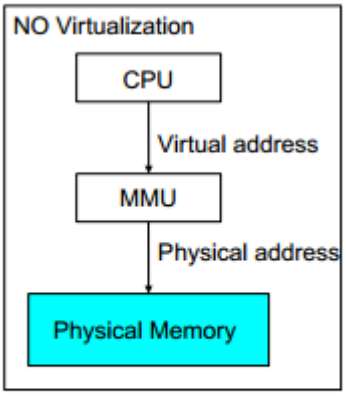
腾讯公司 版权所有

3.3.4.2.2 CPU 虚拟化技术对比

名称	利用二进制翻译的全虚拟化	硬件辅助虚拟化	操作系统协助/半虚拟化
实现技术	BT 和直接执行	遇到特权指令转到 root 模式执行	Hypercall
客户操作系统修改/兼容性	无需修改客户操作系统，最佳兼容性	无需修改客户操作系统，最佳兼容性	客户操作系统需要修改来支持 hypercall，因此它不能运行在物理硬件本身或其他的 hypervisor 上，兼容性差，不支持 Windows
性能	差	全虚拟化下，CPU 需要在两种模式之间切换，带来性能开销；但是，其性能在逐渐逼近半虚拟化。	好。半虚拟化下 CPU 性能开销几乎为 0，虚机的性能接近于物理机。
应用产品、技术	VMware Workstation/QEMU/Virtual PC	VMware ESXi/Microsoft Hyper-V/Xen 3.0/KVM	Xen

3.3.4.3 KVM 内存虚拟化

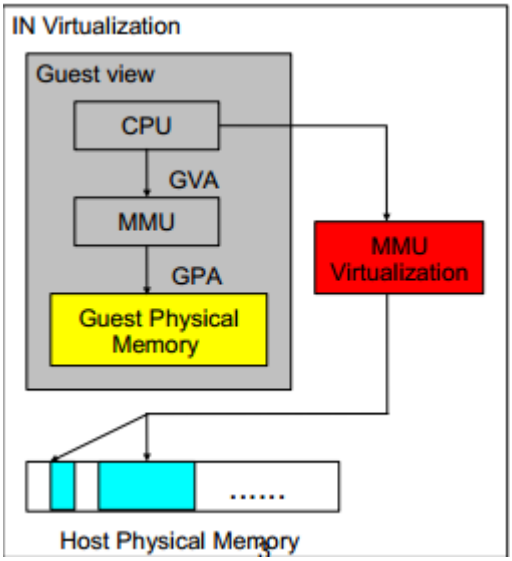
传统 OS 环境中，CPU 对内存的访问都必须通过 MMU(Memory Management Unit)将虚拟地址 VA 转换为物理地址 PA 从而得到真正的 Physical Memory Access，即：VA->MMU->PA，见下图：



GVA: guest virtual address
GPA: guest physical address

虚拟运行环境中由于 Guest OS 所使用的物理地址空间并不是真正的物理内存，而是由 VMM 供其所使用一层虚拟的物理地址空间，为使 MMU 能够正确的转换虚实地址，Guest 中的地址空间的转换和访问都必须借助 VMM 来实现，这就是内存虚拟化的主要任务，即：GVA->MMU Virtualation->HPA，见下图：

腾讯公司 版权所有

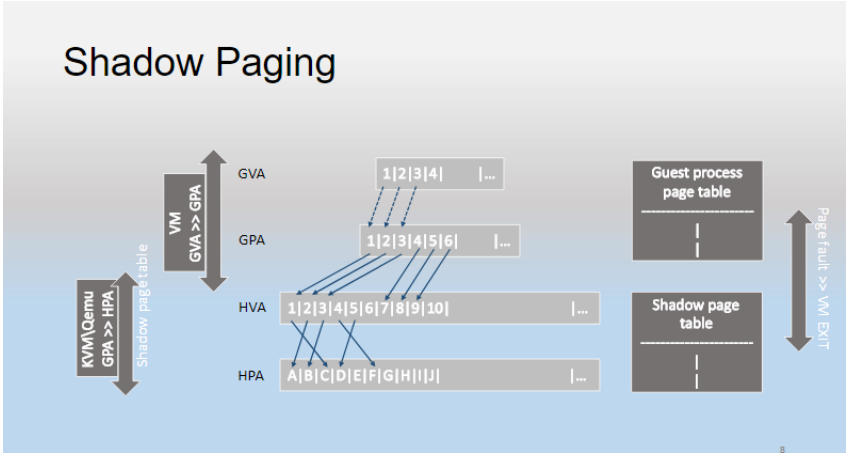


实现内存虚拟化（也称 MMU 虚拟化方案）最主要的是实现客户机虚拟地址 (Guest Virtual Address, GVA) 到宿主机物理地址(HPA)之间的转换，内存虚拟化的实现方式有如下两种：

- 软件方式：通过软件实现内存地址的翻译，比如 Shadow page table（影子页表）技术。
- 硬件实现：基于 CPU 的辅助虚拟化功能，比如 AMD 的 NPT 和 Intel 的 EPT 技术。

3.3.4.3.1 影子页表(Shadow Page Table)

影子页表是纯软件的 MMU 虚拟化方案，Guest OS 维护的页表负责 GVA 到 GPA 的转换，当 Guest OS 创建之后，VMM 创建其对应影子页表。刚开始影子页表是空的，此时任何客户 OS 的访存操作都会发生缺页中断，然后 VMM 捕获缺页异常，根据 VFN 在客户机页表中得到 PFN，然后在 P2M 表中得到 MFN，写入影子页表。这样逐步完成客户机所有虚拟地址到宿主机机器地址的映射。所以被加载到物理 MMU 中的页表是影子页表。
在多进程 Guest OS 中，每个进程有一套页表，进程切换时也需要切换页表，这个时候就需要清空整个 TLB，使所有影子页表的内容无效。但是某些影子页表的内容可能很快就会被再次用到，而重建影子页表是一项十分耗时的工作，因此又需要缓存影子页表。
缺点：实现复杂，会出现高频率的 VM Exit 还需要考虑影子页表的同步，缓存影子页表的内存开销大。

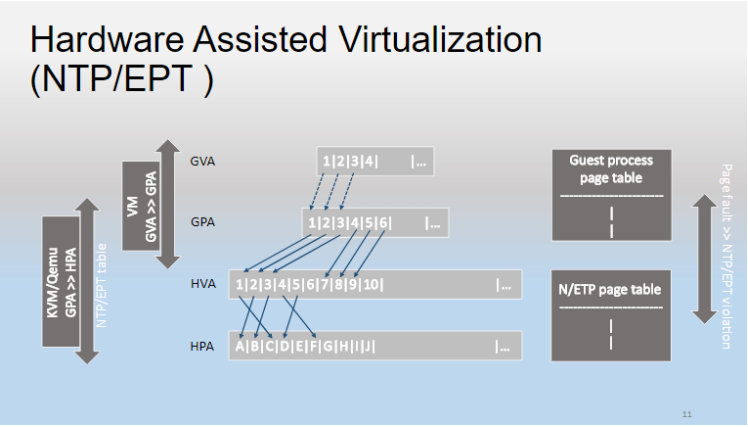


3.3.4.3.2 EPT(Extended Page Table)

为了解决影子页表的低效，VT-x(Intel 虚拟化技术方案)提供了 Extended Page Table(EPT)技术，直接在硬件上支持了 GVA->GPA->HPA 的两次地址转换，大大降低了内存虚拟化的难度，而且，由于客户机内部

腾讯公司 版权所有

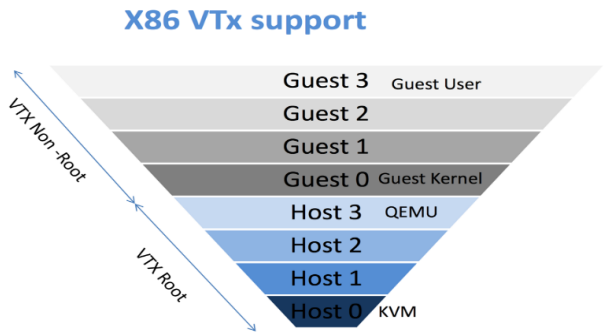
的缺页异常也不会致使客户机退出，因此提高了客户机运行的性能。此外，KVM 只需为每个客户机维护一套 EPT 页表，也大大减少了内存的额外开销。



3.3.4.4 KVM 工作原理

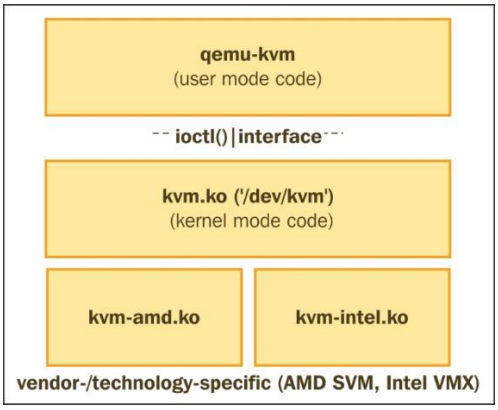
3.3.4.4.1 硬件支撑

KVM基于硬件虚拟化技术，在CPU虚拟化(Intel VT或者AMD-V)的支持下，Host运行在root级别下，Guest运行在非root级别下。root和非root级别都有4个特权级Ring0~Ring3，与传统的CPU兼容。Guest和Host的运行级别如下图所示：

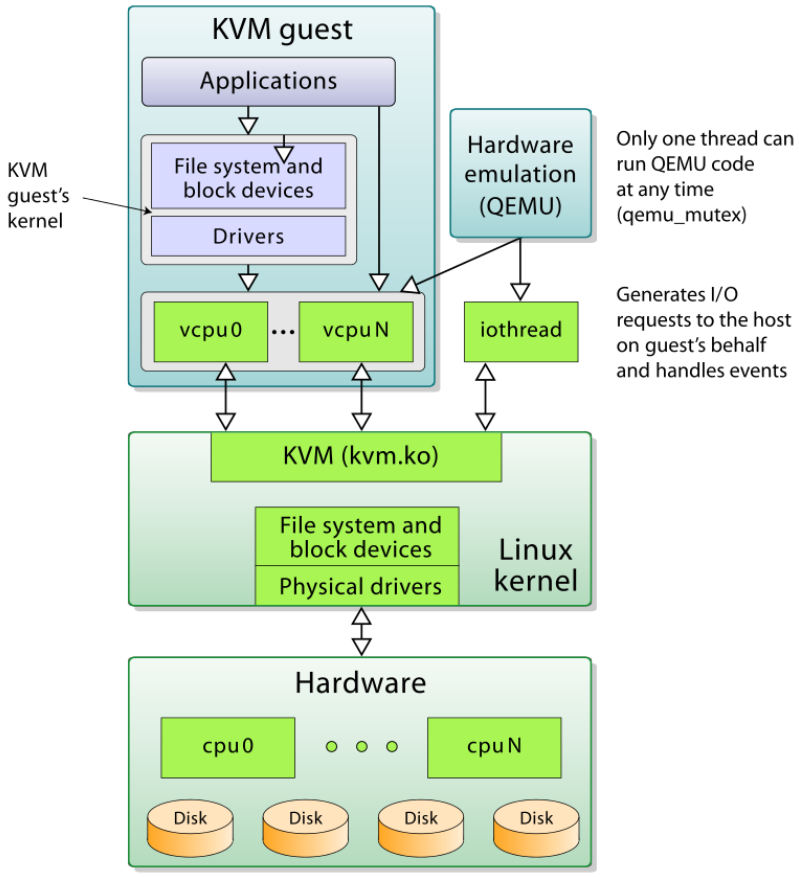


3.3.4.4.2 软件架构

KVM 包含一个内核模块 kvm.ko 用来实现核心虚拟化功能，以及一个和处理器相关的模块如 kvm-intel.ko 或 kvm-amd.ko。KVM 本身不实现任何模拟，仅仅是暴露了一个/dev/kvm 接口，这个接口可被宿主机用来主要负责 vCPU 的创建，虚拟内存的地址空间分配，vCPU 寄存器的读写以及 vCPU 的运行。有了 KVM 以后，Guest os 的 CPU 指令不用再经过 QEMU 来转译便可直接运行，大大提高了运行速度。但 KVM 的 kvm.ko 本身只提供了 CPU 和内存的虚拟化，所以它必须结合 QEMU 才能构成一个完整的虚拟化技术。



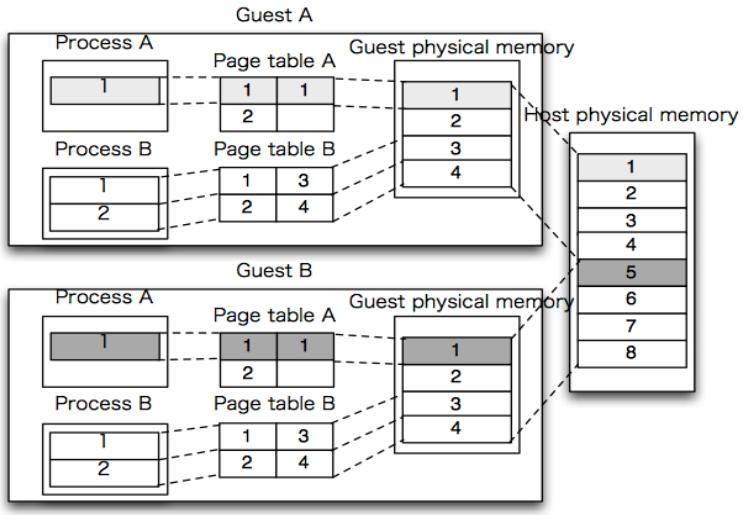
腾讯公司 版权所有



每个虚拟机实体，对应一个QEMU进程，虚拟机的VCPU实体，对应QEMU进程启动的VCPU线程。虚拟机访问资源（存储、网络以及其他外设访问请求），最终都会交给QEMU进程来处理。从原理上，QEMU保证了虚拟机只能访问到其有权访问的资源，保证了虚拟机资源的隔离，也保证了单个虚拟机崩溃并不会对Hypervisor以及其他虚拟机造成影响。

由于虚拟机的VCPU对应QEMU进程启动的VCPU线程，因此，虚拟机CPU的调度由Host Kernel(Linux)来完成，Linux Kernel保证了不同虚拟机VCPU线程的指令的隔离，互不影响。

内存方面，Guest OS自己维护内部进程的页表，Guest访问的内存，到Host上实际物理内存之间的映射关系由KVM来管理。映射关系如下图所示：



腾讯公司 版权所有

不同的Guest OS，其访问的内存，最终映射到不同的实体内存段之上，这种映射关系由影子页表或者在CPU的支持下由EPT来完成。这种映射关系从原理上保证了，不同虚拟机访问到的是独立的内存段，实现了内存隔离。

3.3.4.5 KVM 隔离机制

KVM 为开源的虚拟化技术，为用户提供可用的 KVM 虚拟机。由于物理服务器上的虚拟机有可能是不同用户的虚拟机，因此虚拟机和虚拟机之间的安全隔离就显得比较重要。为此，KVM 技术从以下几个方面对虚拟机进行了隔离处理，保证用户虚拟机的安全：

- 进程隔离
  由于 KVM 虚拟机在物理服务器上显示的都是一个一个独立的 qemu-kvm 进程，每一个进程有自己单独的命名空间，内存空间和地址空间，访问不同的内存。因为虚拟机实现为一个 Linux 进程，所以它利用标准的 Linux 安全模型来提供隔离和资源控制，利用 Linux 内核来添加强制访问控制、多级和多类别安全，以及处理策略的执行。

- 文件隔离
  每一个 KVM 虚拟机对应的磁盘，在物理服务器上对应的都是一个一个单独的（块）文件，每一个文件之间没有联系，每一个虚拟机独占一个（块）文件，保证了 KVM 虚拟机读写时的独占性，从而确保用户的数据隔离安全。如果有需要，您甚至可以对 KVM 的虚拟机磁盘进行加密。

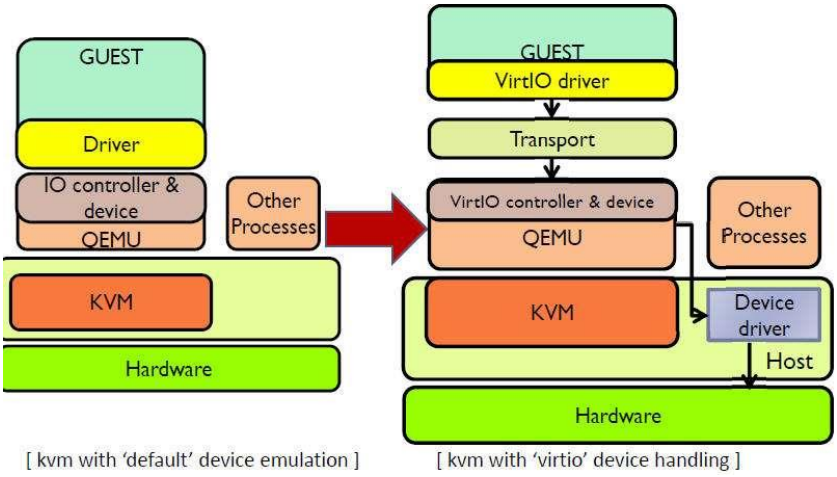
- 内存隔离
  KVM 从 Linux 继承了强大的内存管理功能。一个虚拟机的内存与任何其他 Linux 进程的内存一样进行存储，可以以大页面的形式进行交换以实现更高的性能，也可以磁盘文件的形式进行共享。NUMA 支持（非一致性内存访问，针对多处理器的内存设计）允许虚拟机有效地访问大量内存。KVM 支持来自 CUP 供应商的最新的内存虚拟化功能，支持 Intel 的扩展页面表(EPT)和 AMD 的快速虚拟化索引(RVI)，以实现更低的 CPU 利用率和更高的吞吐量。内存页面共享通过一项名为内核同页合并(Kernel Same-page Merging, KSM)的内核功能来支持。KSM 扫描每个虚拟机的内存，如果虚拟机拥有相同的内存页面，KSM 将这些页面合并到一个在虚拟机之间共享的页面，仅存储一个副本。如果一个虚拟机尝试更改这个共享页面，它将得到自己的专用副本。

3.3.4.6 VirtIO

KVM 是必须使用硬件虚拟化辅助技术（如 Intel VT-x、AMD-V）的 hypervisor，在 CPU 运行效率方面有硬件支持，其效率是比较高的；在有 Intel EPT 特性支持的平台上，内存虚拟化的效率也较高。QEMU/KVM 提供了全虚拟化环境，可以让客户机不经过任何修改就能运行在 KVM 环境中。不过，KVM 在 I/O 虚拟化方面，传统的方式是使用 QEMU 纯软件的方式来模拟 I/O 设备，其效率并不非常高。QEMU 模拟 I/O 设备的方式，其优点是可以通过软件模拟出各种各样的硬件设备，包括一些不常用的或者较老的经典的设备，而且它不用修改客户机操作系统，就可以实现模拟设备在客户机中正常工作。在 KVM 客户机中使用这种方式，对于解决手上没有足够设备的软件开发及调试有非常大的好处。而它的缺点是，每次 I/O 操作的路径比较长，也需要多次数据复制，所以它的性能较差。

在 KVM 中，可以在客户机中使用半虚拟化驱动（Paravirtualized Drivers，PV Drivers）来提高客户机的性能（特别是 I/O 性能）。目前，KVM 中实现半虚拟化驱动的方式是采用了 VirtIO 这个 Linux 上的设备驱动标准框架。VirtIO 是一种半虚拟化驱动，广泛用于在 XEN 平台和 KVM 虚拟化平台，用于提高客户机 IO 的效率，事实证明，VirtIO 极大的提高了 VM IO 效率，配备 VirtIO 前后端驱动的情况下，客户机 IO 效率基本达到和宿主机一样的水平。其基本原理可以总结为让客户机知道自己运行在虚拟化环境中，与 hypervisor 根据 VirtIO 标准协作，从而在客户机中达到更好的性能（特别是 I/O 性能）。

腾讯公司 版权所有



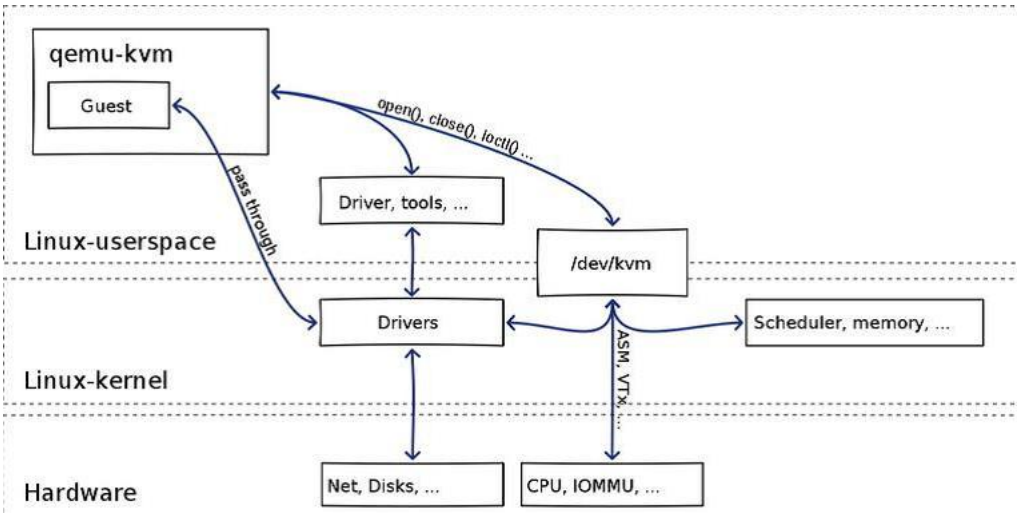
3.3.5 QEMU

QEMU 是纯软件实现的虚拟化模拟器，几乎可以模拟任何硬件设备，比如能够模拟一台能够独立运行操作系统的虚拟机，虚拟机认为自己和硬件打交道，但其实是和 QEMU 模拟出来的硬件打交道，QEMU 将这些指令转译给真正的硬件。正因为 QEMU 是纯软件实现的，所有的指令都要经 QEMU 过一手，性能非常低。在 KVM 出现之后，因为 KVM 只能负责 CPU 和内存虚拟化，不能模拟其它设备，如网卡，磁盘等，所以就由 QEMU 负责用户空间，KVM 负责内核空间，共同组成完整的硬件虚拟化方案，因为用到了上面两个东西，所以称之为 QEMU-KVM，其中 OpenStack 为了跨 VM 性，所以不会直接控制 QEMU-KVM，而是通过 libvit 的库去间接控制 QEMU-KVM 。

QEMU-KVM 具有两大作用：

- 提供对 CPU，内存（KVM 负责），IO 设备（QEMU 负责）的虚拟
- 对各种虚拟设备的创建，调用进行管理（QEMU 负责）

QEMU-KVM 中，KVM 运行在内核空间，QEMU 运行在用户空间，实际模拟创建、管理各种虚拟硬件，QEMU 将 KVM 整合了进来，通过 ioctl 调用 /dev/kvm，从而将 CPU 指令的部分交给内核模块来做，KVM 实现了 CPU 和内存的虚拟化，但 KVM 不能虚拟其他硬件设备，因此 QEMU 还有模拟 IO 设备（磁盘，网卡，显卡等）的作用，KVM 加上 QEMU 后就是完整意义上的虚拟化技术。



QEMU模拟其他的硬件，如Network，Disk，同样会影响这些设备的性能，于是又产生了pass through半虚拟化设备virtio_blk，virtio_net，提高设备性能，详见下章节。

腾讯公司 版权所有

3.3.6 裸机管理

当前越来越多的服务都放到了云上，但是针对高性能，CPU密集型计算服务，原有的虚拟机在很多场景下已经不再适用，尤其是虚拟机的性能受到QEMU以及物理机硬件本身虚拟化的各种限制，无法提供更高的计算性能。在这种情形下，直接使用物理服务器来满足用户的这些需求就非常有必要。

但是，在现实世界中，物理服务器从零到有的安装操作系统，部署服务是需要一定的人工操作的。如果直接使用人工的方式，给物理服务器安装操作系统、部署服务的确可以满足用户的需求，但是，带来的人工运维成本相对较高。

在这种现实需求和业务矛盾的基础上，裸金属服务器管理服务应运而生。裸金属服务器，顾名思义，就是没有任何操作系统的物理服务器，简称裸机；裸金属服务器管理服务，即专门负责管理这些裸机的服务，简称裸机管理。裸机管理需要至少满足以下几个要求：

- 裸机的电源管理，可以对裸机进行开关机
- 可以实现对裸机的操作系统灌装
- 可以实现对裸机回收和分配

在OpenStack当中，Ironic的定位就是为云环境下的用户提供裸机管理的功能。通过Ironic，可以支持在云环境，将用户需要的云服务直接部署在裸机（物理服务器）上，也便于用户直接管理物理服务器。

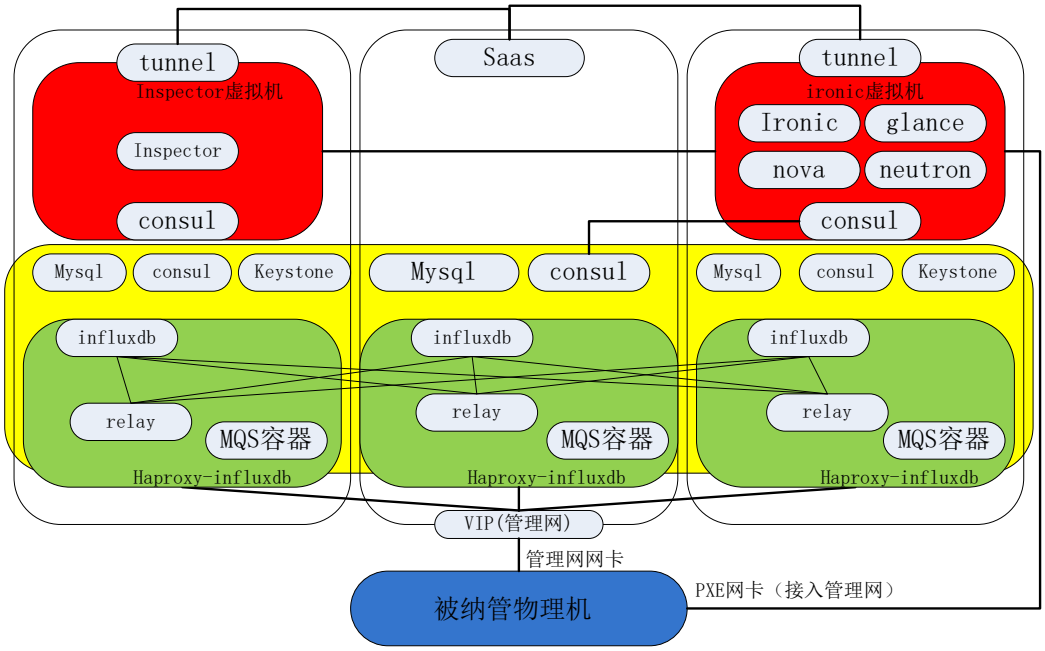
在管理裸机的时候，主要需要使用以下的几种技术：

- PXE
  PXE（preboot execute environment，预启动执行环境），工作于Client/Server的网络模式，支持工作站通过网络从远端服务器下载镜像，并由此支持通过网络启动操作系统，在启动过程中，终端要求服务器分配IP地址，再用TFTP（trivial file transfer protocol）或MTFTP（multicast trivial file transfer protocol）协议下载一个启动软件包到本机内存中执行，由这个启动软件包完成终端基本软件设置，从而引导预先安装在服务器中的终端操作系统。PXE可以引导多种操作系统。
- DHCP
  DHCP（Dynamic Host Configuration Protocol，动态主机配置协议）是一个局域网的网络协议，使用UDP协议工作，主要有两个用途：给内部网络或网络服务供应商自动分配IP地址，给用户或者内部网络管理员作为对所有计算机作中央管理的手段。在PXE中，BIOS使用DHCP来为网络接口申请IP，同时定位网络引导程序服务器。
- NBP
  NBP(Network Bootstrap Program，网络引导程序)是一种启动引导程序，功能与本地引导类似，NBP负责加载操作系统内核到内存中，保证系统能够通过网络引导启动。
- TFTP
  TFTP(Trivial File Transfer Protocol，简单文件传输协议)是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。端口号为69。
- IPMI
  智能平台管理接口(IPMI)是一种开放标准的硬件管理接口规格，定义了嵌入式管理子系统进行通信的特定方法。IPMI信息通过基板管理控制器(BMC)（位于IPMI规格的硬件组件上）进行交流。使用低级硬件智能管理而不使用操作系统进行管理，具有两个主要优点：首先，此配置允许进行带外服务器管理；其次，操作系统不必负担传输系统状态数据的任务。IPMI需要进入BIOS，进行设置IP地址。设置好了之后可以通过Web访问IPMI，实现对机器的如下操作：开机，关机，重启，查看机器当前的通电状态等。

通过整合以上的几类技术，OpenStack Ironic提供了一种比较通用的、智能化、自动化的裸机管理平台，省去大量的人工操作。

腾讯公司 版权所有

在TStack云计算管理平台产品里，Ironic服务采用虚拟机方式部署，启动2个虚拟机ironic-vm和inspector-vm，分别提供裸机API接口和裸机探测服务。ironic-vm虚拟机的集群网和管理网桥接到物理机；inspector-vm虚拟机网卡桥接到物理机的集群网上。



3.4 存储资源管理

3.4.1 存储资源管理概述

TStack存储资源管理模块依托于OpenStack Cinder项目设计开发，在控制平面上了对分布式存储Ceph在内多种分布式和集中式存储的管理和使用，可实现用户对存储资源的实际使用需求进行弹性分配、快照、备份、挂载和扩容等能力。下面将会对涉及到的控制平面（Cinder）和数据平面（Ceph）的技术原理进行介绍。

3.4.2 OpenStack Cinder

3.4.2.1 Cinder 简介

块存储服务（Cinder）为实例提供块存储。块存储的实际管理是由后端存储的驱动提供的，Cinder支持多种不同的后端存储，如：NAS、SAN、Ceph等。

3.4.2.2 Cinder 架构

Cinder 包含如下几个组件：

- Cinder-api
  Cinder-api提供对外接口，处理用户请求，调度Cinder-scheduler，Cinder-volume，Cinder-backup等服务。
- Cinder-volume
  管理volume的服务，与volume provider协同工作，管理volume的生命周期。运行Cinder-volume服务的节点被称作为存储节点。
- Cinder-scheduler
  scheduler通过调度算法选择最合适的存储节点创建volume。

腾讯公司 版权所有

- volume provider
  数据的存储设备，为volume提供物理存储空间。Cinder-volume支持多种存储后端，如：LVM,Ceph RBD等等，每种存储后端通过自己的Driver与Cinder-volume协同工作。

- Message Queue
  Cinder各个子服务通过消息队列实现进程间通信和相互协作。因为有了消息队列，子服务之间实现了解耦，这种松散的结构也是分布式系统的重要特征。

- Database Cinder
  有一些数据需要存放到数据库中，一般使用MySQL。数据库是安装在控制节点上的，比如在我们的实验环境中，可以访问名称为“Cinder”的数据库。

3.4.2.3 Cinder 工作流程介绍

以下介绍一种简单 Cinder 工作流程（实际场景不同，工作流程也会不同）：

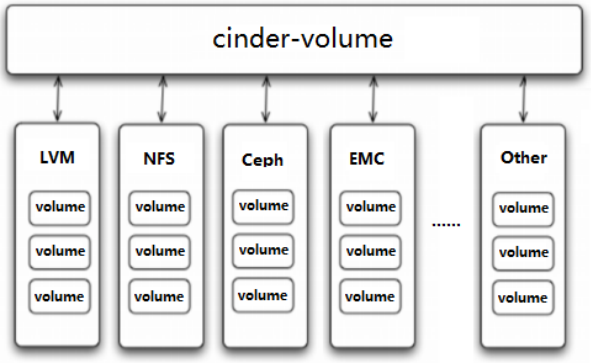
1. 客户（可以是 OpenStack 最终用户，也可以是其他程序）向 API（Cinder-API）发送请求：“帮我创建一个 volume”。
2. API 对请求做一些必要处理后，向 Messaging（RabbitMQ）发送了一条消息：“让 Scheduler 创建一个 volume”。
3. Scheduler（Cinder-scheduler）从 Messaging 获取到 API 发给它的消息，然后执行调度算法，从若干存储节点中选出节点 A。
4. Scheduler 向 Messaging 发送了一条消息：“让存储节点 A 创建这个 volume”。
5. 存储节点A的Volume（Cinder-volume）从Messaging中获取到scheduler发给它的消息，然后通过Driver在volume provider上创建volume。

3.4.2.4 Cinder 异构存储管理

TStack异构存储统一管理组件基于OpenStack Cinder-Volume组件，实现了对后端存储资源的统一管理，来提供业务驱动、自动化的数据服务。异构存储统一管理组件将应用与底层存储解耦，解除设备厂商的绑定，打破异构存储设备之间的壁垒。将不同架构的存储封装起来，无论是传统的集中式存储还是分布式存储都进行统一管理并向上提供服务。

Cinder-volume在存储节点上运行，OpenStack对Volume的操作，最后都是交给Cinder-volume来完成的。Cinder-volume自身并不管理真正的存储设备，存储设备是由volume provider管理的。Cinder-volume与volume provider一起实现volume生命周期的管理。

Cinder-volume为这些volume provider定义了统一的接口，volume provider只需要实现这些接口，就可以Driver的形式即插即用到OpenStack系统中。下面是Cinder Driver的架构示意图：



异构存储兼容性列表：https://wiki.OpenStack.org/wiki/CinderSupportMatrix

腾讯公司 版权所有

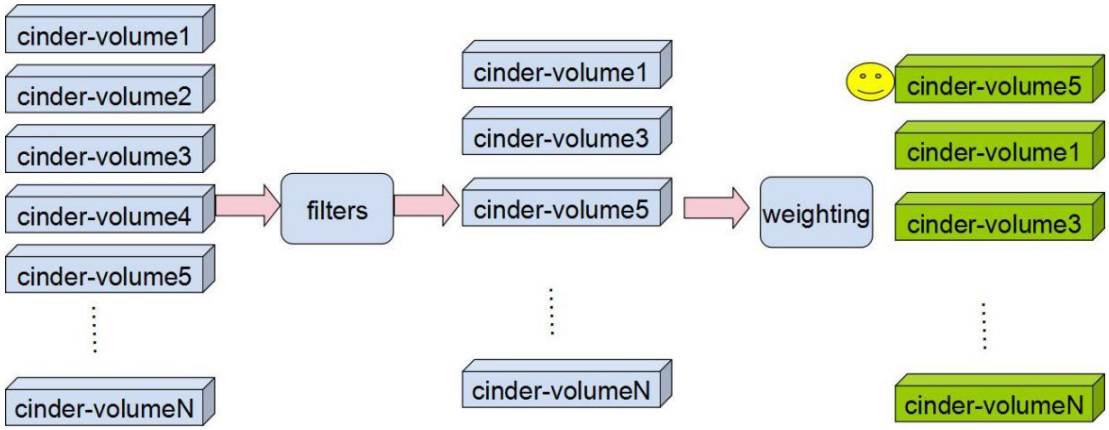
3.4.2.5 Cinder 调度机制

Cinder 创建 Volume 时，Cinder-scheduler 会基于容量、Volume Type 等条件选择出最合适的存储节点，然后让其创建 Volume，下面介绍 Cinder-scheduler 是如何实现这个调度工作的。

在/etc/Cinder/Cinder.conf 中，Cinder 通过 scheduler_Driver，scheduler_default_filters 和 scheduler_default_weighers 这三个参数来配置 Cinder-scheduler。scheduler_Driver 默认参数为 Cinder.scheduler，filter_scheduler，FilterScheduler，Cinder 也允许使用第三方 scheduler，配置 scheduler_Driver 即可。scheduler_default_filters 默认参数为 AvailabilityZoneFilter，CapabilitiesFilter，scheduler_default_weighers 默认参数为 CapacityWeigher。

scheduler 调度过程如下：

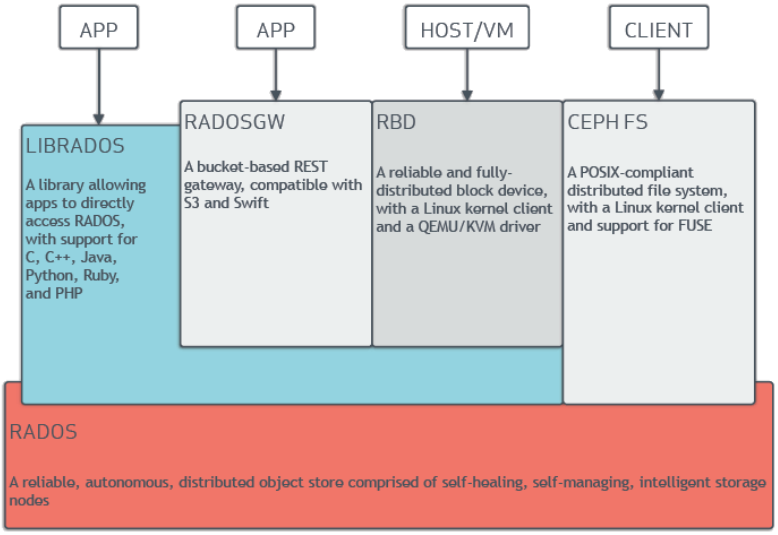
1. 通过过滤器（filter）选择满足条件的存储节点（运行 Cinder-volume）。
2. 通过权重计算（weighting）选择最优（权重值最大）的存储节点。



3.4.3 Ceph

3.4.3.1 Ceph 介绍

Ceph提供了一种统一的、软件定义的分布式存储系统解决方案，其具有优异的性能、可靠性、可扩展性并且没有单点故障。Ceph的底层是RADOS(可靠、自动、分布式对象存储)，可以通过LIBRADOS直接访问到RADOS的对象存储系统。RBD(块设备接口)、RADOS Gateway(对象存储接口)、Ceph File System(POSIX接口)都是基于RADOS的，下图简要说明了Ceph的基本架构。



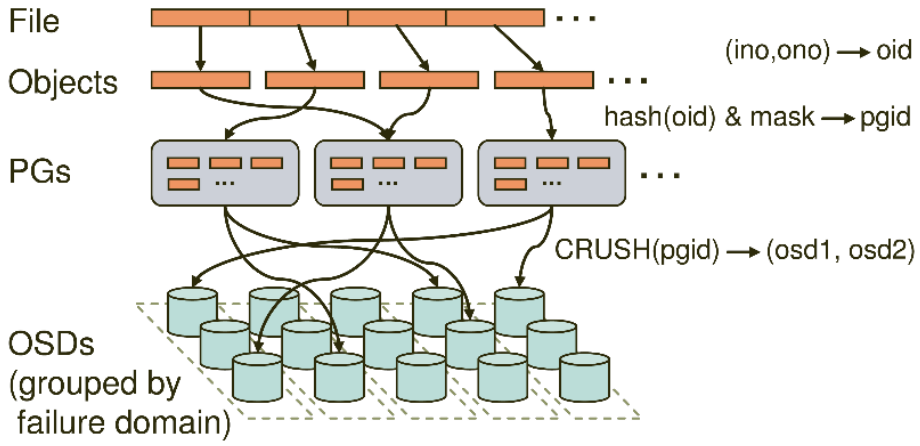
腾讯公司 版权所有

Ceph集群包含多个组件，不同的组件保持相互独立并且提供不同的功能。这些组件包括RADOS、OSD、MON、RADOS gateway以及MDS。

- RADOS（Reliable Autonomic Distributed Object Store）是整个Ceph集群的基石。在Ceph集群中，所有类型的数据都以object对象的形式进行存储，RADOS通过数据复制、故障检测、数据恢复、数据迁移保证数据均衡地分布在整个集群中，同时也保证了数据的一致性和可靠性。
- OSD是Ceph集群中唯一的负责将用户数据写入磁盘以及读取的组件。当上层应用将数据写入Ceph集群中，最终是由OSD将这些数据写入磁盘；当上层应用从Ceph集群读取数据时，同样是由OSD从磁盘读取数据。通常而言，一个OSD就对应一个物理磁盘，也就是说，有多少个物理磁盘，就有多少个OSD。
- Mon通过一组maps来维护整个集群的健康状态，这组maps包含了OSD、MON、PG以及CRUSH的信息。集群中所有的组件都要向Mon汇报并且分享它们的状态信息。需要说明的是，Mons并不会存储应用的任何数据。
- RADOS gateway（RGW）提供了RESTful形式的API接口，它兼容Amazon S3以及OpenStack对象存储Swift。同时，RGW支持多租户，并且可以使用OpenStack的Keystone作为其认证系统。
- MDS（Ceph Metadata Server）主要用来存储CephFS的元数据，也就是说只有使用CephFS的时候才需要部署MDS。
- RBD（RADOS block device）是指Ceph集群提供的块存储。Ceph块存储具备了很多商业存储的特性，比如精简配置和快照功能。用户可以挂载Ceph块存储，并通过格式化、创建文件系统来进行使用。
- CephFS（Ceph File System）是Ceph集群提供的兼容POSIX的分布式文件系统，它依赖于Ceph MDS存储文件系统的元数据。

3.4.3.2 Ceph 工作原理

在 Ceph 存储系统中，数据存储分三个映射过程，首先要将用户要操作的 file，映射为 RADOS 能够处理的 object。就是简单的按照 object 的 size 对 file 进行切分，相当于 RAID 中的条带化过程。接着把 Object 映射到 PG，在 file 被映射为一个或多个 object 之后，就需要将每个 object 独立地映射到一个 PG 中去。第三次映射就是将作为 object 的逻辑组织单元的 PG 映射到数据的实际存储单元 OSD。

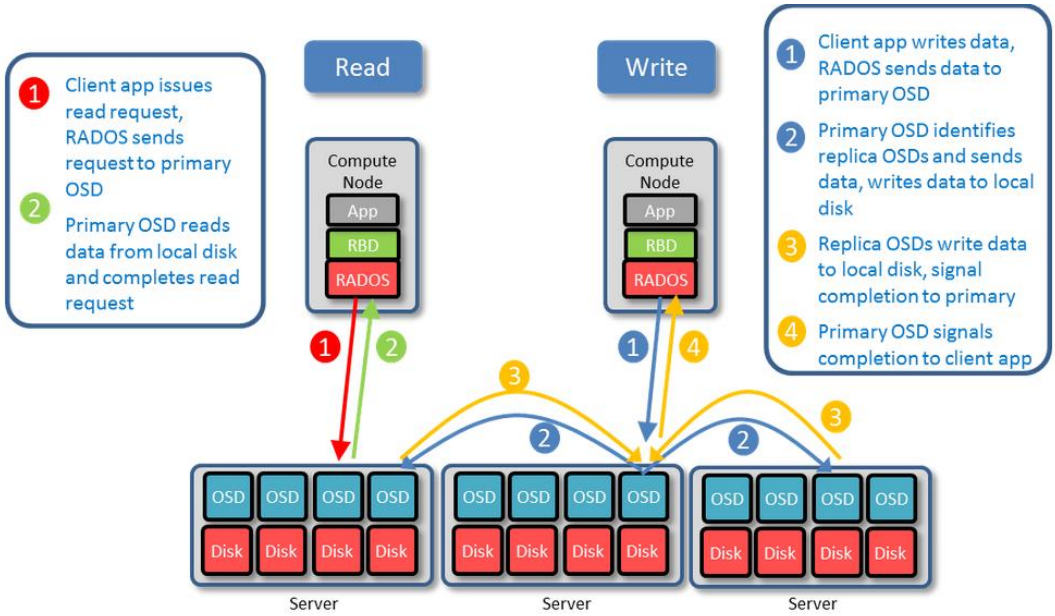


文件存入时，首先把 File 切分为 RADOS 层面的 Object，每个 Object 一般为 2MB 或 4MB(大小可设置)。每个 Object 通过哈希算法映射到唯一的 PG。每个 PG 通过 Crush 算法映射到实际存储单元 OSD，PG 和 OSD 间是多对多的映射关系。OSD 在物理上可划分到多个故障域中，故障域可以跨机柜和服务器，通过策略配置使 PG 的不同副本位于不同的故障域中。

腾讯公司 版权所有

3.4.3.3 Ceph journal

正常OSD读取和写入流程如图所示：



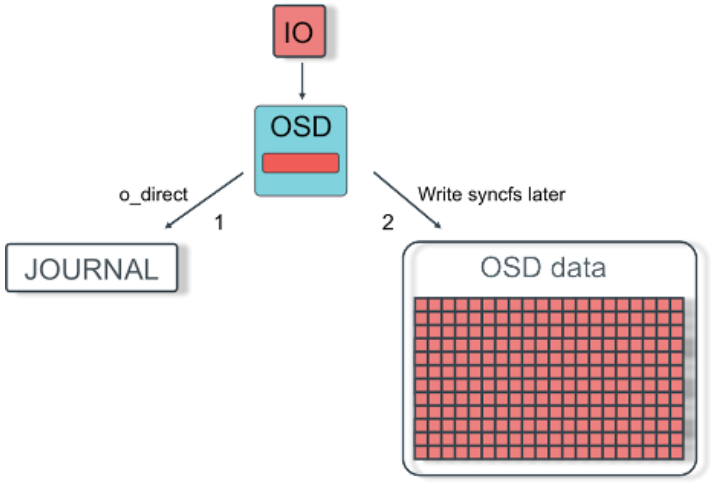
读取：

- 客户机应用发出读取请求，RADOS向主OSD发送请求。
- 主OSD从本地磁盘读取数据，并完成读取请求。

写入：

- 客户机应用写入数据，RADOS向主OSD发送数据。
- 主OSD确认副本OSD并发送数据，将数据写入本地磁盘。
- 副本OSD将数据写入本地磁盘，向主机发送完成信号。
- 主OSD向客户机应用发送完成信号。

根据上图原理每个写入OSD操作，会产生2-3个磁盘操作，使用SSD做日志盘，写入SSD日志盘，就返回写入成功，可以减少访问时间，降低延迟，大幅提升吞吐量。



腾讯公司 版权所有

3.4.3.4 Ceph 与 OpenStack 集成

- Glance集成

  Glance是OpenStack中的镜像服务。默认情况下，镜像存储在本地控制器，然后在被请求时复制到计算主机。计算主机缓存镜像，但每次更新镜像时，都需要再次复制。

  Ceph为Glance提供了后端，允许镜像存储在Ceph中，而不是本地存储在控制器和计算节点上。这大大减少了抓取镜像的网络流量，提高了性能，因为Ceph可以克隆镜像而不是复制镜像。此外，它使得在OpenStack部署或诸如多站点OpenStack之类的概念的迁移变得更简单。

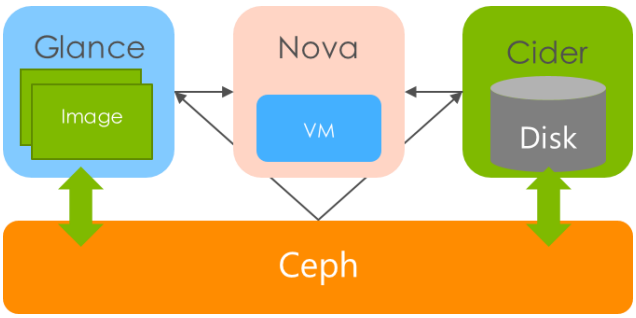
- Nova集成

  Nova存储与默认的运行虚拟机相关联的虚拟磁盘镜像，在/var/lib/nova/instances下的Hypervisor上。在虚拟磁盘镜像的计算节点上使用本地存储有一些缺点：

  - 镜像存储在根文件系统下。太多镜像会导致文件系统被填满，从而可能导致计算节点崩溃。
  - 计算节点上的磁盘崩溃可能导致虚拟磁盘丢失，因此无法进行虚拟机恢复。

- Cinder集成

  Cinder提供了关于块存储的抽象，并允许供应商通过提供驱动程序进行集成。在Ceph中，每个存储池可以映射到不同的Cinder后端。这允许创建诸如金、银或铜的存储服务。用户可以决定例如金应该是复制三次的快速SSD磁盘，银应该是复制两次，铜应该是使用较慢的擦除编码的磁盘。



3.5 网络资源管理

3.5.1 网络资源管理概述

TStack网络资源管理模块依托于OpenStack Neutron项目设计开发，在控制平面上了实现了对虚拟网络创建、分配和隔离等能力的管理，并且提供虚拟路由器、虚拟负载均衡、虚拟VPN等高级网络特征，并且可支持管理多种开源和商业的SDN控制器，云平台默认使用了OpenvSwitch作为虚拟二层网络能力的提供者。下面将会对涉及到的控制平面（Neutron）和数据平面（OpenvSwitch）的技术原理进行介绍。

3.5.2 OpenStack Neutron

3.5.2.1 Neutron 简介

OpenStack网络服务，是OpenStack核心项目之一，允许创建、插入接口设备，这些设备由其他的OpenStack服务管理。插件式的实现可以容纳不同的网络设备和软件，为OpenStack架构与部署提供了灵活性。

腾讯公司 版权所有

3.5.2.2 Neutron 相关概念

3.5.2.2.1 网络

Neutron网络目的是划分物理网络，在多租户环境下提供给每个租户独立的网络环境。另外，Neutron提供API来实现这种目标。Neutron中“网络”是一个可以被用户创建的对象，如果要和物理环境下的概念映射的话，这个对象相当于一个巨大的交换机，可以拥有很多个动态可创建和销毁的虚拟端口。

3.5.2.2.2 端口

在物理网络环境中，端口是用于连接设备进入网络的地方。Neutron中的端口起着类似的功能，它是路由器和虚拟机挂接网络的附着点。

3.5.2.2.3 路由器

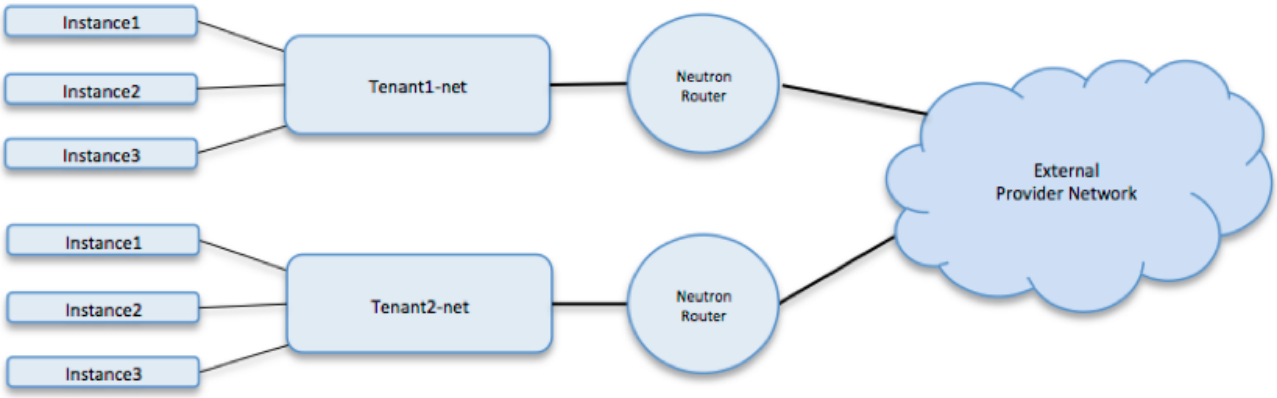
和物理环境下的路由器类似，Neutron中的路由器也是一个路由选择和转发部件。只不过在Neutron中，它是可以创建和销毁的软部件。

3.5.2.2.4 子网

简单地说，子网是由一组IP地址组成的地址池。不同子网间的通信需要路由器的支持，这个Neutron和物理网络下是一致的。Neutron中子网隶属于网络。

3.5.2.2.5 VPC

VPC(Virtual Private Cloud)，是AWS提出的一个概念，是一个与传统网络极其相似的虚拟网络，不同私有网络间完全逻辑隔离，可以通过软件定义网络的方式管理私有网络VPC，实现IP地址、子网、路由表等功能的配置管理。在OpenStack中与VPC对应的概念是Tenant Network，当在OpenStack中使用Tenant Network时，用户可以自定义路由器，网络，子网等，从而实现不同网络之间的完全逻辑隔离。



3.5.2.2.6 DVR

DVR，即分布式路由功能。按照传统集中式路由的设计，需要专门的网络节点来负责路由和 NAT 功能，实现不同网络下的虚机之间访问以及虚机同外网的访问。

大部分的东西向流量和南北向流量都在网络节点上进行，这就意味着网络节点需要承载大量的流量，这会导致网络节点成为瓶颈。

业务流量类型	业务场景	是否通过网络节点
东西向流量	虚机之间（同一子网）	否
	虚机之间（不同子网）	是
南北向流量	公网 IP（通过公网 IP 访问虚机）	是
	SNAT（虚机借助路由器访问外网）	是

腾讯公司 版权所有

为了降低网络节点的负载，同时提高可扩展性，引入了分布式路由机制。来让计算节点自己来处理原先的大量东西向流量和非 SNAT 南北向流量（有公网 IP 的虚拟机跟外网的通信）。

这样网络节点只需要处理占到一部分的 SNAT （无公网 IP 的虚拟机跟外网的通信）流量，大大降低了负载和整个系统对网络节点的依赖。

业务流量类型	业务场景	是否通过网络节点
东西向流量	虚拟机之间（同一子网）	否
	虚拟机之间（不同子网）	否
南北向流量	公网 IP（通过公网 IP 访问虚机）	否
	SNAT（虚拟机借助路由器访问外网）	是

3.5.2.3 Neutron 架构

Neutron 由如下组件构成：

- Neutron Server
  对外提供OpenStack网络API，接收请求，并调用Plugin处理请求。
- Plugin
  处理Neutron Server发来的请求，维护OpenStack逻辑网络状态，并调用Agent处理请求。
- Agent
  处理Plugin的请求，负责在network provider上真正实现各种网络功能。
- network provider
  提供网络服务的虚拟或物理网络设备，例如Linux Bridge，OpenvSwitch或者其他支持Neutron的物理交换机。
- Queue
  Neutron Server，Plugin和Agent之间通过Messaging Queue通信和调用。
- Database
  存放OpenStack的网络状态信息，包括Network,Subnet,Port,Router等。

Neutron Server主要分为API Router和Neutron Plugins。

- API Router负责将不同的RESTFUL API分发到不同的neutron-plugin上。
- Neutron Plugins接收到API的请求后，调用对应的网络功能实现，之后与各种Agent通过MQ进行交互。

Neutron Agents可以看作是plugin在设备上的代理，它接收plugin的消息并完成最终的设备级别指令操作。当设备本地操作出现问题时，Neutron agent会将结果通知到plugin，plugin会根据结果进行异常处理及数据库信息更新。

在此基础上，目前支持以下几个Agent：

- OpenvSwitch Agent
  二层网络相关操作的处理，主要负责二层设备的操作，如ovs网桥、二层端口设备、流表规则处理。
- L3 Agent
  三层路由设备操作，主要负责虚拟路由的Namespace、路由中端口设备、网关接口、网关IP、路由表等信息的维护和处理。
- DHCP Agent
  为云主机提供DHCP服务，通过该服务云主机可以获取到平台指定的IP地址。所以该服务主要处理DHCP请求。

腾讯公司 版权所有

- Loadbalancer Agent
  提供负载均衡服务，利用Haproxy和Namespace实现云主机中业务L4-L7负载均衡。

- Metering Agent
  该服务主要负责获取云主机的上下行流量信息，如速率、当前流量等。并将数据上报给监控服务。

- Metadata Agent
  该服务提供元数据服务，云主机密码注入时通过网络获取metadata最终就是访问的该服务。该服务会与Nova进行交互获取云主机的元数据信息。

由上面的介绍可以看出，Neutron Server其实是网络服务的微内核，指导Plugin和Agent来完成网络功能的虚拟化。这种可插拔的设计模型具有很好的可扩展性，各个厂商或者开发者都可以很方便的根据自己的需求添加Plugin和Agent。

3.5.2.3.1 二层网络

二层网络主要支持VLAN和VXLAN两种协议。VLAN又称作虚拟局域网，通过在交换机上配置VLAN模式，可使具有相同VLAN tag的数据包互相通信，从而达到隔离广播域的效果。由于VLAN需要在数据包头部打tag，该字段的长度为12位，所以同一局域网中VLAN最大支持4096个二层网络。这在大型数据中心或者公有云中很难满足日益增长的客户需求，为了避免该问题，可以使用VXLAN协议。VXLAN是基于UDP报文封装，头部使用24位存储VNI标签，所以VXLAN理论支持2^24个二层网络。

3.5.2.3.2 虚拟交换机

与物理网络离不开物理交换机类似，虚拟网络肯定也离不开虚拟交换机。虚拟交换机是指利用程序和Linux内核实现的具有网络二层交换功能的软件。网络服务中使用到的虚拟交换机主要包括OpenvSwitch（简称ovs）和Linux Bridge。

3.5.2.3.3 VLAN 模型

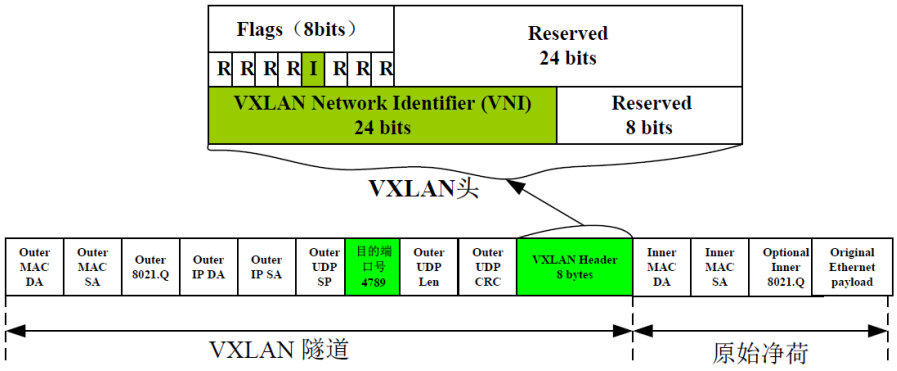
在每个节点都有一个集成网桥br-int，在计算节点上该网桥可以理解为云主机的接入交换机，在这个网桥上虚拟网卡设备都有一个内部VLAN标识。该VLAN标识只是用作集成网桥内部的网络隔离，避免二层广播风暴。无论Neutron Type Driver使用VLAN还是VXLAN模式，这个网络结构都是相同的。

在Type Driver使用VLAN模式的情况下，流表会控制集成网桥中的内部VLAN与物理网络VLAN进行映射，确保从计算节点出去的流量中的VLAN号为外部物理网络VLAN号。该模式需要保证租户网络的交换机配置TRUNK模式。

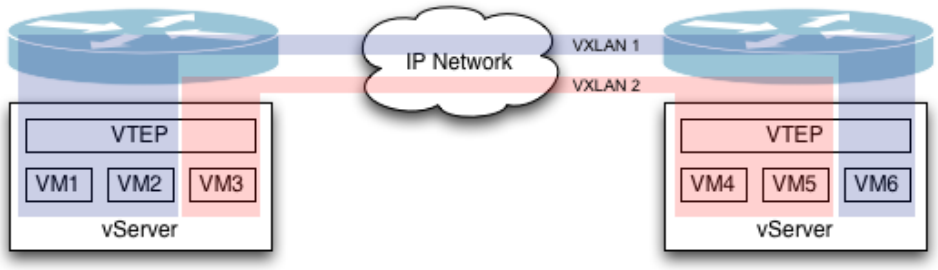
3.5.2.3.4 VXLAN 模型

VXLAN技术是一种大二层的虚拟网络技术，主要的技术原理就是引入一个UDP格式的外层隧道，作为数据的链路层，而原有数据报文内容作为隧道净荷来传输。由于外层采用了UDP作为传输手段，就可以让净荷数据轻而易举的在二三层网络中传送，为了能够支持原有VLAN广播域寻址能力，VXLAN还引入了三层IP组播来代替以太网的广播，让BUM（Broadcast：广播，unknown unicast：未知单播， multicast：多播）报文通过组播方式在虚拟网络中传送。

腾讯公司 版权所有



VXLAN引入了一个VXLAN ID（VNI - VXLAN Network Identifier)来作为虚拟网络的标识，为24bits，与原有的VLAN 12bits 长度相比，VXLAN ID长度增加了一倍，可以支持2^24个虚拟网络。



在VXLAN标准中还引入了一个VTEP(VXLAN Tunnel End Point)的逻辑体，来完成VXLAN报文的封装和解封装。VTEP与物理网络相连，分配有物理网络的IP地址，该地址与虚拟网络无关。一般VTEP设备都是物理设备，而在我们的平台中，VTEP由虚拟交换机OVS提供，所以是一种软件定义的VTEP。

3.5.2.3.5 三层网络

三层网络服务主要是负责路由转发和提供NAT功能，在做三层转发时，最核心的设备就是虚拟路由器。当云主机和外网进行通信，或者不同网络之间进行通信时，需要三层路由转发功能或者NAT功能的都会经过虚拟路由器。

无论二层网络采用VLAN或者VXLAN模型，都可以使用网络服务中的虚拟路由功能。在这种情况下，用户创建子网网段时必须选择租户私有网络，因为通过虚拟路由时会进行NAT转换，所以必须是一个内部网段。另外，值得说明的是，在二层网络模型采用VLAN时，可以不经过网络服务中的虚拟路由，直接使用物理的三层路由，而虚拟网络服务只提供二层网络相关服务。这种情况下，云主机的网络使用物理网络，通信性能更好，但是不适用于大规模的数据中心，毕竟VLAN是有限的，而且很多三层虚拟网络功能都无法使用。

3.5.2.3.6 虚拟路由

虚拟路由功能由L3 Agent管理，底层利用Linux Namespace网络隔离技术实现。每一个虚拟路由器对应一个Namespace，Namespace之间以及与主机之间都是相互隔离的，它们都拥有独立的网络协议栈，包括arp表，路由表，iptables规则，网卡设备等等。如果Namespace之间需要通信，则可以使用桥接或者veth pair等技术。

3.5.2.3.7 公网 IP

在用户创建公网IP并绑定到云主机的端口之后，L3 Agent会在虚拟路由中的端口设备上设置公网IP，并且根据映射关系在iptables中的nat表中加入DNAT和SNAT的规则。SNAT是指将数据包源地址进行改写，把内网IP改写为公网IP，这是在云主机流量出外网时需要，如果不绑定公网IP，则使用路由网关进行改写，

腾讯公司 版权所有

使用公网IP之后，会使用公网IP地址进行改写。DNAT是指目的地址的NAT，即把数据包的目的地址进行替换，将公网IP替换为租户私网IP，一般是外部流量访问云主机时需要做DNAT，而公网IP也主要是为了解决该问题而存在的。

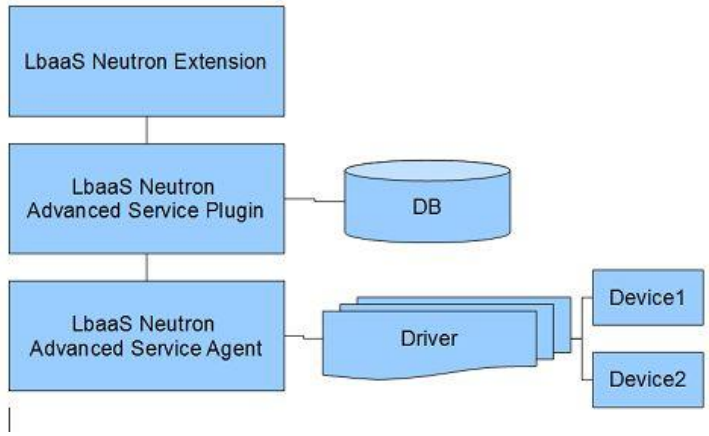
3.5.2.3.8 端口映射

端口映射功能其实是基于端口的NAT服务，它可以将虚拟路由器的某个端口映射到云主机的某个端口，这样可以减少对浮动IP地址数量的需求，满足简单的应用环境。

3.5.2.3.9 负载均衡

负载均衡（Load Balance）是 OpenStack Neutron 支持的高级功能之一。负载均衡能够将网络请求分发到多个实际处理请求的虚机上，这样能有效处理高流量的网络请求。

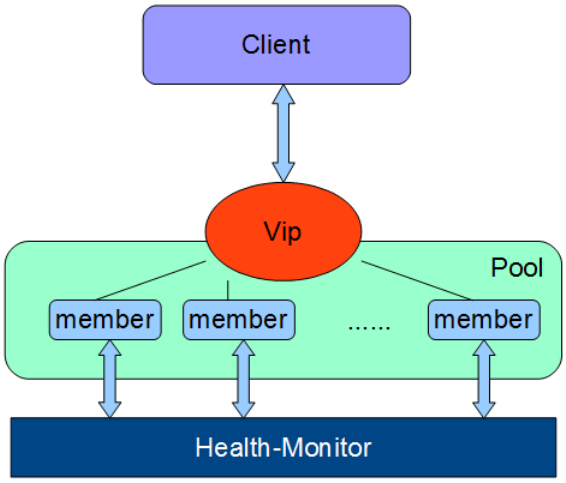
Neutron LBaaS 是基于 Neutron 实现的，pythone-neutronclient 中包含了 LBaaS 相关的命令，LBaaS 的数据也是存储在 Neutron 的数据库中。LBaaS 的程序架构也遵循着 Neutron 对 service plugin 的要求。



从图中可以看出，代码主要由三部分组成：

- Extension：负责处理 REST API 请求，并将 REST API 请求转发到 Plugin。
- Plugin：处理核心逻辑，管理 DB 的读写。
- Agent：处理并响应由 Plugin 发来的请求，管理 Driver。默认采用 HAProxy 作为 Driver。

负载均衡工作流程如下图所示：



腾讯公司 版权所有

- VIP

VIP(Virturl IP address)就是负载均衡对外提供服务的地址。VIP 有自己的 IP 地址，而且一般都能通过公网进行访问。在 Neutron 中，VIP 对应着二层虚拟设备 br-int 上的一个 port。当负载均衡 Pool 里面至少有一个 Member 时，VIP 才有存在的意义，因为这时至少有一个实际的 OpenStack 实例在处理网络请求。VIP 负责将网络流量分发到各个 Member，Neutron LBaaS 中支持以下三种网络流量的分发方式。

  - Round robin：平均的将网络流量分发到多个Member。
  - Source IP：从某一个特定 IP 发来的网络请求，总是发到特定的Member。
  - Least connections：将收到的网络请求，发给当前负载均衡池中连接数最少的Member。如果所有的Member 连接数一样，则遵循Round robin的方式。

Neutron 中，VIP 还支持设置最大连接数(connection-limit)，这样在网络流量大时，可以保护负载均衡池。最大连接数可以设置成-1，这时不限制连接数。

- Pool

Pool 是 LBaaS V1 中的 root resource。所有的其他资源都是基于 pool 来创建的。Neutron LBaaS 默认以 HAProxy 为 Driver 实现。在默认情况下，一个 pool 对应着一个独立的 HAProxy 进程，一个独立的 namespace。目前一个 pool 只能有一个 VIP，在 LBaaS V2 里面，可以允许一个 pool 对应多个 VIP。

- Member

Member 对应的是 pool 里面处理网络请求的 OpenStack 实例。在 OpenStack Neutron 中，Member 是一个逻辑关系，表示着实例与 pool 的对应关系。这也就是说，一个 OpenStack 实例，可以对应不同的 pool，在 Neutron 的 LBaaS 里面创建多个 Member。在创建 Member 时所选择的实例，必须与 pool 处于同一个 subnet，否则将不能工作。

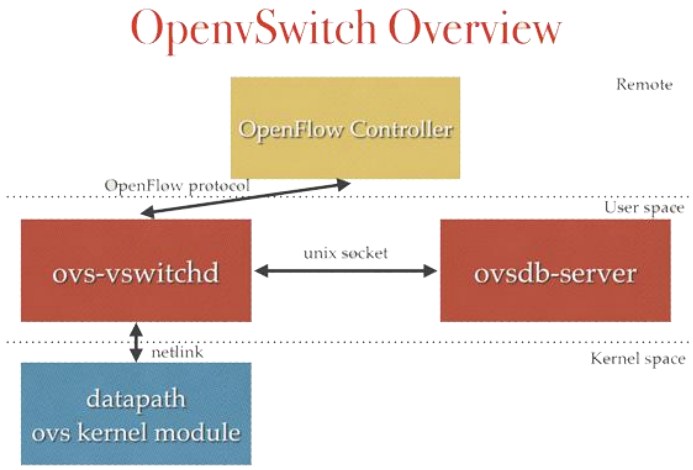
- Health monitor

Health monitor 只有在关联 pool 时才有意义。它用来监测 pool 里面 Member 的状态。它会以轮询的方式，去查询各个 Member，如果 Member 未作出响应，它会更新 Member 的状态至 INACTIVE，这样在 VIP 分发网络请求时，就不会考虑这个 Member 了。如果 Member 恢复了响应，它会更新 Member 的状态至 ACTIVE。这时，Member 会重新出现在 VIP 的分发列表中。

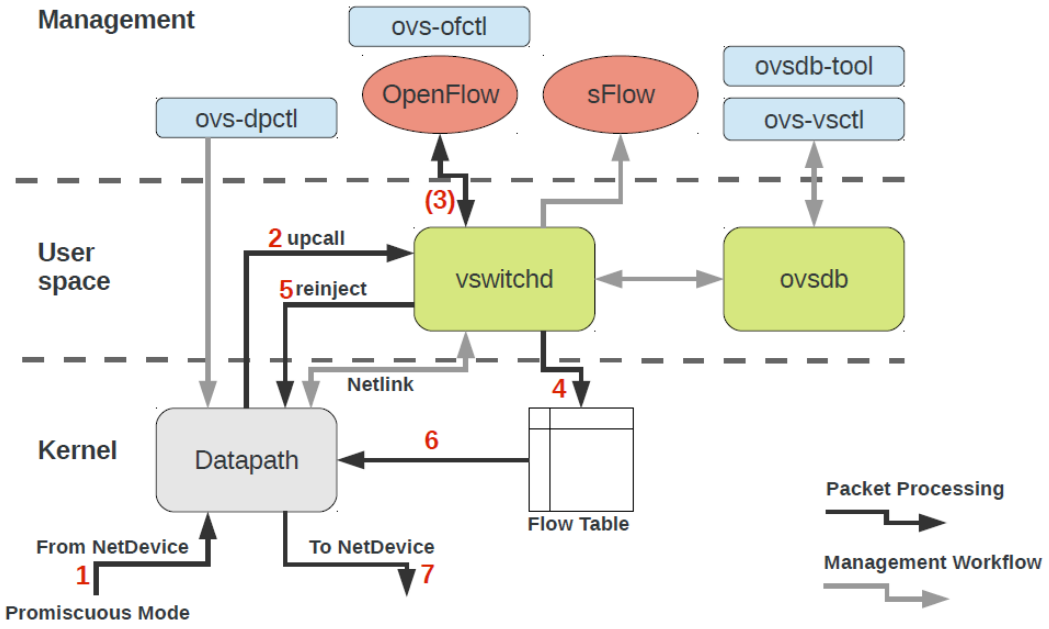
与其他的概念不同，Health monitor 在 Neutron 的负载均衡中不是必须的。也就是说，没有 Health monitor，也能组成一个负载均衡池。但是，如果没有 Health monitor，pool 会一直认为所有的 Member 都是 ACTIVE 状态，这样所有的 Member 会一直出现在 VIP 的分发列表中，哪怕 Member 对应的实例不能响应网络请求。这在实际应用中会造成负载均衡的响应异常。

腾讯公司 版权所有

3.5.3 OpenvSwitch



OpenvSwitch的官方定义：OpenvSwitch是一个具有工业级质量的多层虚拟交换机。通过可编程扩展，可以实现大规模网络的自动化（配置、管理、维护）。它支持现有标准管理接口和协议（比如netFlow，sFlow，SPAN，RSPAN，CLI，LACP，802.1ag等，熟悉物理网络维护的管理员可以毫不费力地通过OpenvSwitch转向虚拟网络管理）。



OVS主要包括以下模块和特性：

- ovs-vswitchd：核心模块，实现交换功能的守护程序（daemon），和Linux内核模块一起，实现基于流的交换；
- ovsdb-server：提供轻量级数据库查询服务。其保存了整个OVS的配置信息，包括接口，流表，VLAN等。ovs-vswitchd从其查询配置信息；
- ovsdb-tool：不通过ovs-server就能直接操控数据库；
- ovsdb-client：直接通过ovs-server数据库操作；
- ovs-dpctl：dapapath control用来配置switch内核模块，可以控制转发规则；

腾讯公司 版权所有

- ovs-vsctl：获取或者更改ovs-vswitchd的配置信息，此工具操作的时候会更新ovsdb-server数据库；
- ovs-appctl：openvswitch apply control，发送命令来运行相关 daemon(很少使用)；

ovsdbmonitor GUI 工具，用于显示OVS数据库中的相关数据。

此外，OVS也提供了支持OpenFlow的特性实现，包括：

- ovs-openflowd：一个简单的OpenFlow交换机；
- ovs-controller：一个简单的OpenFlow控制器；
- ovs-ofctl：OpenvSwitch openflow control，用来控制OVS作为OpenFlow交换机工作时的流表内容；

ovs-vswitchd是OpenvSwitch的核心模块。vswitchd模块主要包括 bridge、ofproto等模块。作为主模块，负责解析和执行其他各个ovs命令。vswitchd的主要功能就是不断检测并调用所有bridge上的ofproto，执行其上的处理函数。

OpenvSwitch中许多网络上的概念与平时接触到的不同，OpenvSwitch中用到的一些名词及概念如下：

- Packet（数据包）
  网络转发的最小数据单元，每个包都来自某个端口，最终会被发往一个或多个目标端口，也可能被直接丢弃，转发数据包的过程就是网络的唯一功能。
- Bridge（网桥）
  Open vSwitch中的一个网桥就是一台以太网交换机，可以创建一个或者多个 Bridge 设备。其功能是根据一定流规则，把从端口收到的数据包转发到另一个或多个端口或丢弃数据包等。
- Port（端口）
  端口是收发数据包的单元，和物理以太网交换机的端口概念类似。OpenvSwitch中，每个端口都属于一个特定的网桥。端口收到的数据包会经过流规则的处理，发往其他端口；也会把其他端口来的数据包发送出去。OpenvSwitch支持的端口有以下几种：
  - Normal Port：用户可以把操作系统中的网卡绑定到OpenvSwitch上，OpenvSwitch会生成一个普通端口处理这块网卡进出的数据包。
  - Internal Port：当设置端口类型为internal，OpenvSwitch会创建一块虚拟网卡，此端口收到的所有数据包都会交给这块网卡，网卡发出的包会通过这个端口交给OpenvSwitch。

  当OpenvSwitch创建一个新网桥时，默认会创建一个与网桥同名的Internal Port
  - Patch Port：当机器中有多个OpenvSwitch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，在两个网桥之间交换数据。
  - Tunnel Port：隧道端口是一种虚拟端口，支持使用gre或VXLAN等隧道技术与位于网络上其他位置的远程端口通讯。
- Interface（iface/接口）
  它是连接到Port的网络接口设备。OpenvSwitch通过Interface与外部交换数据包。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才是一对多的关系。一个接口就是操作系统的一块网卡，这块网卡可能是OpenvSwitch生成的虚拟网卡，也可能是物理网卡挂载在OpenvSwitch上，也可能是操作系统的虚拟网卡（TUN/TAP）挂载在OpenvSwitch上。
- Flow（流）
  流定义了端口之间数据包的交换规则。每条流分为匹配和动作两部分，匹配部分选择哪些数据包需要可以通过这条流处理，动作决定这些匹配到的数据包如何转发。流描述了一个网桥上，端口到端口的转发规则。比如我可以定义这样一条流：当数据包来自端口A，则发往端口B，来自端口A 就是匹配部

腾讯公司 版权所有

分，发往端口B 就是动作部分。流的定义非常复杂，比如：当数据包来自端口A，并且其源MAC是aa:aa:aa:aa:aa:aa，并且其拥有VLAN tag为a，并且其源IP是a.a.a.a，并且其协议是TCP，其TCP源端口号为a，则修改其源IP为b.b.b.b，发往端口B。

- Datapath
  由于流非常复杂，对每个进来的数据包都去尝试匹配所有流，效率会非常低，所以有了datapath。Datapath对流进行缓存，把流的执行结果保存起来，当下次遇到匹配到同一条流的数据包，直接通过datapath处理。考虑到转发效率，datapath完全是在内核态实现的，并且默认的超时时间非常短。Datapath也可以视作一个交换机，或者说一个桥的实体模块， 因为传统交换机缓存MAC地址并根据MAC地址进行数据包的转发，而Datapath缓存Flow（流）并根据流进行数据包的转发。

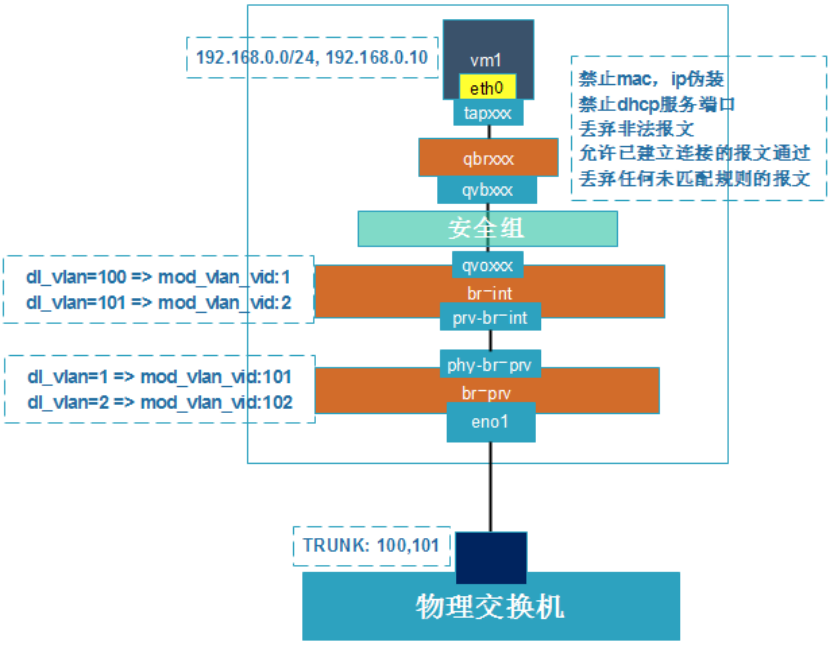
- Controller
  OpenFlow控制器。OVS可以同时接受一个或者多个基于OpenFlow的控制器的管理。

- Flow table
  每个datapath都和一个“flow table”关联，当datapath接收到数据之后，OVS会在flow table中查找可以匹配的flow，执行对应的操作，例如转发数据到另外的端口。

3.5.4 虚拟网络安全

3.5.4.1 安全组

安全组是端口级别的安全防护，这一点跟防火墙功能是有区别的，防火墙是子网间的流量安全防护。安全组可作用于二层和三层，同一个安全组的所有端口可以互相安全访问。安全组默认有两条规则，出方向默认可以出去，入方向默认同安全组的可以进来。安全组是利用计算节点 iptables 实现，由 L2 Agent 来管理规则，也就是说 L2 Agent 会将安全组规则设置到 iptables 中，该规则会在所有具有该安全组的端口所在计算节点设置。



上图是安全组功能网络拓扑，安全组是在计算节点上生效的，具体作用点是 qvb 与 qvo 之间，当流量从 qvb 出去到达 qvo 的过程中，会进行安全组规则的过滤。安全组具有几个系统级别默认规则，这几个规则在端口创建之后就会在计算节点的 Iptables 上生成，这些规则是隐形存在的，无法进行管理。这些规则是：

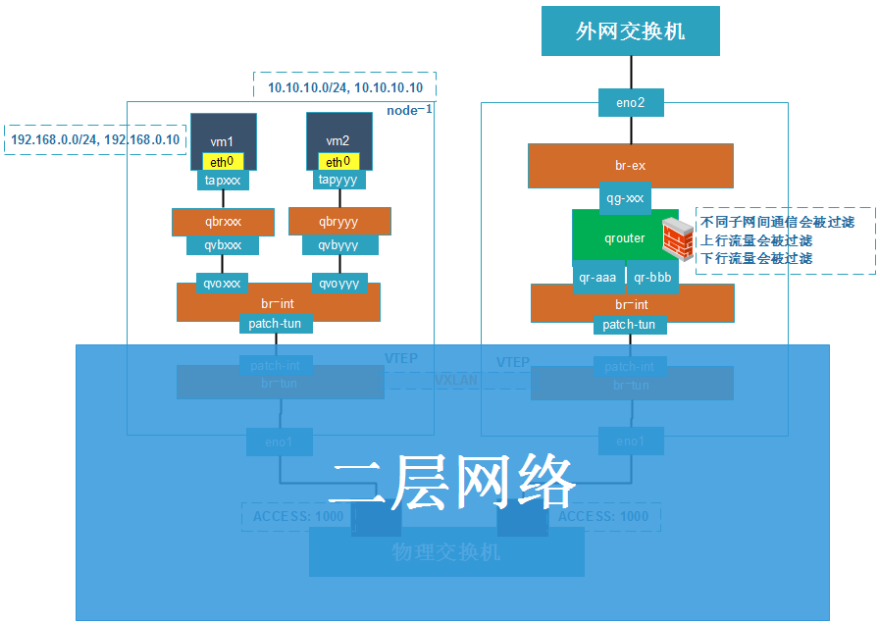
腾讯公司 版权所有

- 禁止 mac，ip 伪装。即平台中只会允许该云主机使用网络服务分配给它的 mac 和 ip 和外界通信，如果用户私自改动了 ip，即使该 ip 在云平台中的网络中可用，也无法正常通信，该规则会进行拦截。
- 禁止 dhcp 服务的端口。平台中云主机只能向外部发送 dhcp 请求，而不能发送 dhcp 应答。也就是云主机不能作为 dhcp server 存在。这条规则是通过 DROP 源端口为 67，目标端口为 68 的 UDP 报文实现的。相反的对于源端口是 68，目的端口是 67 的 UDP 报文是允许的。
- 丢弃非法报文。通过报文状态进行判断，如果状态是 INVALID 则丢弃。
- 允许已建立连接的报文通过。判断连接状态是 RELATED，ESTABLISHED 则允许通过。
- 丢弃任何未匹配到规则的报文。这条就保证了如果报文没有匹配到任何规则，则该报文禁止通过。

平台会为每个租户默认创建一个安全组，该安全组默认规则是允许所有流出流量、允许相同安全组的虚拟机流入流量，并支持用户自定义安全组，以及添加和删除安全组规则。

3.5.4.2 防火墙

防火墙服务（Firewall as a Service）提供网络边界安全防护功能，它与安全组的防护边界是不同的，通过在 namespace 里的 iptables 添加 filter 表规则来实现。目前防火墙作用范围是整个租户的路由器，即防火墙会影响该租户里所有路由器的通信。防火墙由 L3 Agent 控制管理，用户创建防火墙之后，并为它添加规则，此时 L3 Agent 会根据规则去设置租户下所有路由器的 iptables filter 规则。另外需要注意的是，如果该租户没有路由器，则防火墙创建之后是不会正常工作的。



上图是防火墙的网络拓扑，当不同子网之间互相通信时，流量会经过防火墙中的规则进行过滤，不符合规则的数据报文会被丢弃。其他的就是流量进出整个云平台时也会被防火墙过滤。

3.6 身份管理

3.6.1 身份管理概述

TStack 身份管理模块依托于 OpenStack Keystone 项目设计开发，实现了将云平台用户划分为超级管理机、部门管理员、项目管理员、普通用户四种角色，并对登录的用户进行鉴权和服务列表的访问控制。下面将会对涉及到的 keystone 项目的技术原理进行介绍。

腾讯公司 版权所有

3.6.2 Keystone

3.6.2.1 Keystone 简介

OpenStack Identity Service (Keystone)是 OpenStack 的组件之一，用于为OpenStack 其它组件成员提供统一的认证服务，包括身份验证、令牌的发放和校验、服务列表、用户权限的定义等等。用户访问资源需要验证用户的身份与权限，服务执行操作也需要进行权限检测，这些都需要通过Keystone来处理。Keystone类似一个服务总线，或者说是整个OpenStack框架的注册表，其他服务通过Keystone来注册其服务的Endpoint（服务访问的URL），任何服务之间相互的调用，需要经过Keystone的身份验证，来获得目标服务的Endpoint。

3.6.2.2 Keystone 相关概念

- User
  User 指代任何使用 OpenStack 的实体，表示可以正常访问 OpenStack 服务的实体。

- Group
  Group 是 Keystone V3 API 中引入的概念，通过引入 Group 的概念，Keystone V3 实现了对用户组的管理，达到了同时管理一组用户权限的目的。

- Credentials
  Credentials 是 User 用于标识自己身份的信息，可以是用户名/密码、Token、API Key、其他高级方式。

- Authentication
  Authentication 是 Keystone 验证 User 身份的过程。User 访问 OpenStack 时向 Keystone 提交用户的 Credentials，Keystone 验证通过后会给 User 签发一个 Token 作为后续访问的令牌。

- Token
  Token 是由数字和字母组成的字符串，User 成功 Authentication 后由 Keystone 分配给 User。Token 用做访问 Service 的令牌，Service 会通过 Keystone 验证 Token 的有效性,Token 的有效期默认是 24 小时。

- Project
  Project 用于将 OpenStack 的资源（计算、存储和网络）进行分组和隔离。根据 OpenStack 服务的对象不同，Project 可以是一个客户（公有云，也叫租户）、部门或者项目组（私有云）。

- Catalog
  Service Catalog（服务目录）是 Keystone 为 OpenStack 提供的一个 REST API 端点列表，OpenStack 服务需要注册到 Catalog 中，用户和服务可以使用 Catalog 列表定位到其他的服务，Catalog 是 OpenStack 相关服务的集合，每个服务都有一个或者多个 endpoint（即可以访问的 url 地址），即 Catalog=services+endpoint，作为一个简单的服务发现功能。

- Service
  OpenStack 的 Service 包括 Compute(Nova)、Block Storage(Cinder)、Object Storage(Swift)、Image Service(Glance)、Networking Service(Neutron)等，用户也可以新建自定义服务。每个 Service 都会提供若干个 Endpoint，User 通过 Endpoint 访问资源和执行操作。

- Endpoint
  Endpoint 是一个网络上可访问的地址，通常是一个 URL， Service 通过 Endpoint 暴露自己的 API，使其它组件可以定位到相应 service 的地址，如：当 Nova 需要访问 Glance 服务去获取 image 时，Nova 通过访问 Keystone 拿到 Glance 的 endpoint，然后通过访问该 endpoint 去获取 Glance 服务 Keystone 负责管理和维护每个 Service 的 Endpoint。不同的 region 有不同的 endpoint（可以通过 endpoint 的 region 属性去定义多个 region）。
  Endpoint 分为三类：

腾讯公司 版权所有

- public API：对整个互联网可见，这样客户就可以方便的管理自己的云了。
- admin API：应该严格限定只有管理云基础设施的组织内的运营商，才能使用该 API。
- internel API：应该被限定只有那些安装有 OpenStack 服务的主机，才能使用该 API。

- Domain
  Domain 是 Keystone V3 API 中引入的概念，是一个单位，它包含用户、组和项目，在公有云或者私有云中常常表示一个集团客户。

- Regions
  Regions 是一个地理上的概念，每个 region 有自己独立的 endpoint，regions 之间完全隔离，但是多个 regions 之间共享同一个 Keystone 和 dashboard。所以除了提供隔离的功能，region 的设计更多侧重地理位置的概念，用户可以选择离自己更近的 region 来部署自己的服务，一般云平台中对应到的概念是数据中心。

3.6.2.3 Keystone 功能简介

Keystone 主要有两个功能：

- 用户管理：包括管理用户和验证用户身份信息合法性，以及控制用户的权限。
- 服务目录管理：提供各个服务目录的（Service Catalog:包括 service 和 endpoint）服务，无论任何服务或者客户访问 OpenStack 都要访问 Keystone 获取服务列表，以及每个服务的 endpoint。

3.6.3 TStack 身份管理

TStack 认证功能分为如下几个模块：

- 部门管理模块

  TStack 中的部门管理模块对应 OpenStack 中 Domain 概念。

- 项目管理模块

  TStack 中的部门管理模块对应 OpenStack 中 Project 概念。

- 用户管理模块

  TStack 中的部门管理模块对应 OpenStack 中 User 概念。

- 角色管理模块

  TStack 中的部门管理模块对应 OpenStack 中 Role 概念，并允许管理自定义显示云平台各个功能模块。

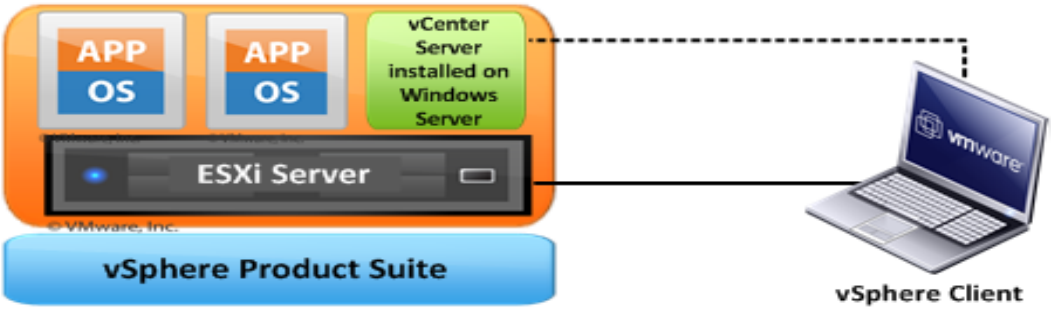
3.7 VMware 纳管

3.7.1 VMware 概念

- vSphere/vCenter Server/ESXi

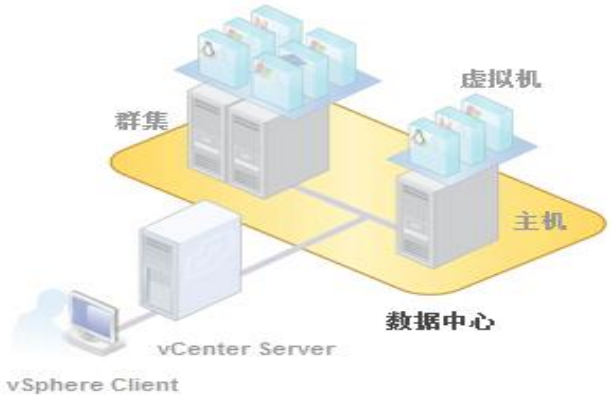
  vSphere产品套件是多个产品与特征的集合，包含vCenter Server，ESXi，vSphere client。ESXi是vSphere套件中的虚拟机管理程序，和KVM, Hyper-V类似。vCenter Server为所有ESXi主机及其虚拟机提供集中化的管理配置，一些VMware中的高级特征，也是由它来提供，如动态迁移，分布式资源调度，vSphere高可用。

腾讯公司 版权所有



- 数据中心/集群/主机

  数据中心是包含集群，ESXi主机和虚拟机的集合。集群由一组ESXi主机组成，管理所包含的所有ESXi主机的资源。



3.7.2 TStack VMware 管理

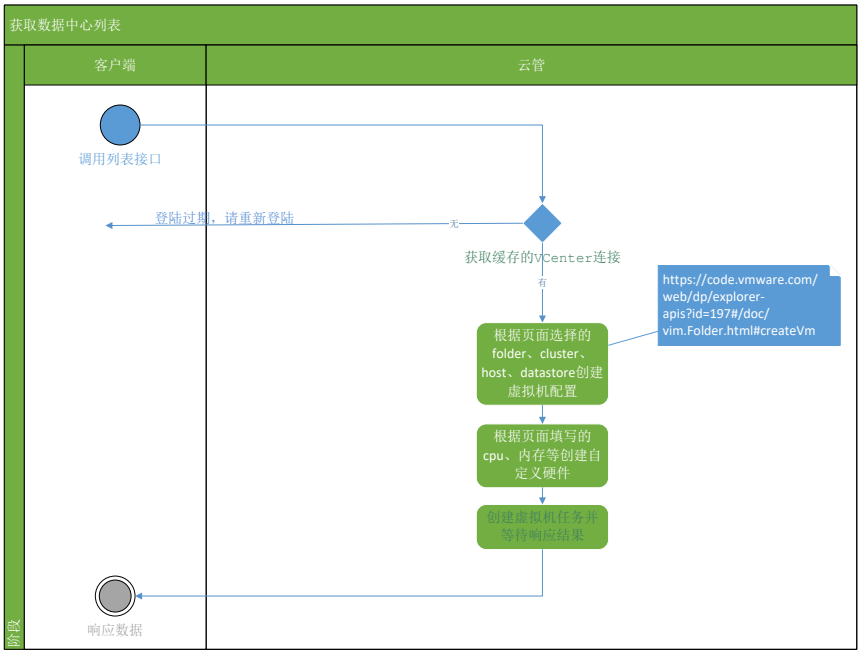
TStack产品中使用vSphere WebService SDK的方式，完成对VMware数据中心、集群、主机、虚拟机数据信息收集。

3.7.2.1 数据中心管理

数据中心是主机、虚拟机等各种实体对象的容器，可以添加实体对象到数据中心，比如，你可以添加主机、集群到一个数据中心。

腾讯公司 版权所有

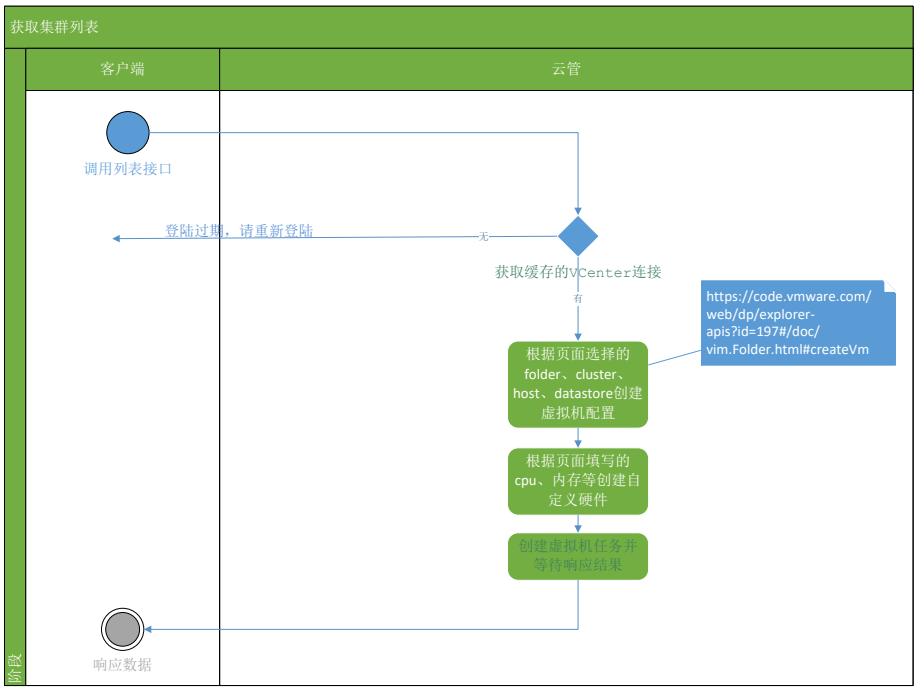
TStack VMware的数据中心管理目前主要提供数据中心列表查询及搜索功能。点击数据中心列表的任一数据中心的名称，将进入此数据中心中下面的集群、主机、虚拟机列表页面。



3.7.2.2 集群管理

集群是主机的集合，当添加一个主机到一个集群时，主机资源变为集群资源的一部分，集群管理集群内的所有主机的资源。

TStack VMware 的集群管理目前主要提供集群列表查询及搜索功能。点击集群列表的任一集群的名称，将进入此集群的主机和虚拟机列表页面。

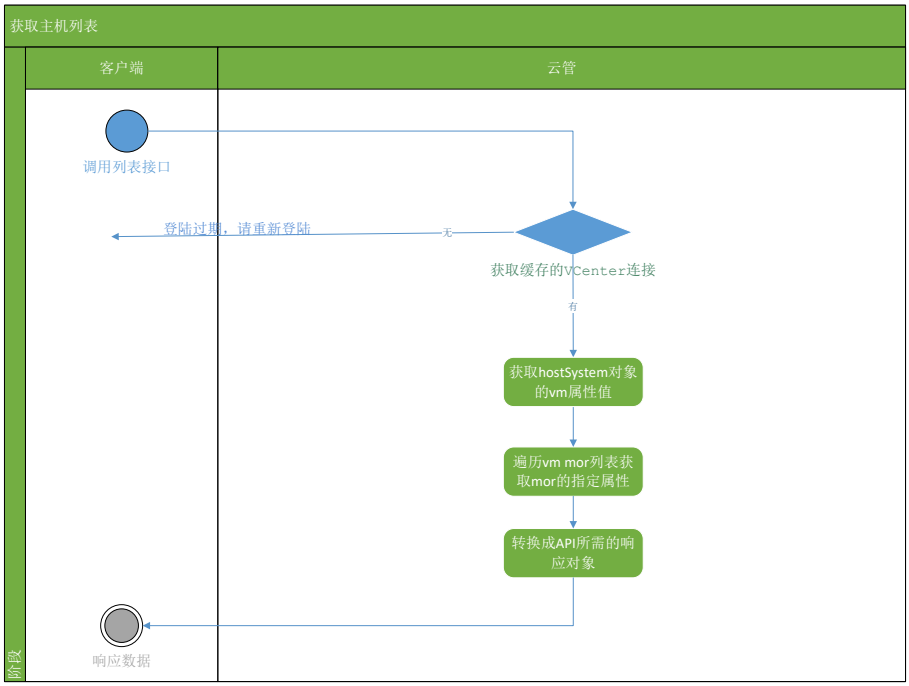


3.7.2.3 主机管理

主机是使用 ESX、ESXI 等虚拟化软件的用来运行虚拟机的计算机，主机提供虚拟机使用的 CPU 和内存资源，并让虚拟机能够访问存储和网络。

腾讯公司 版权所有

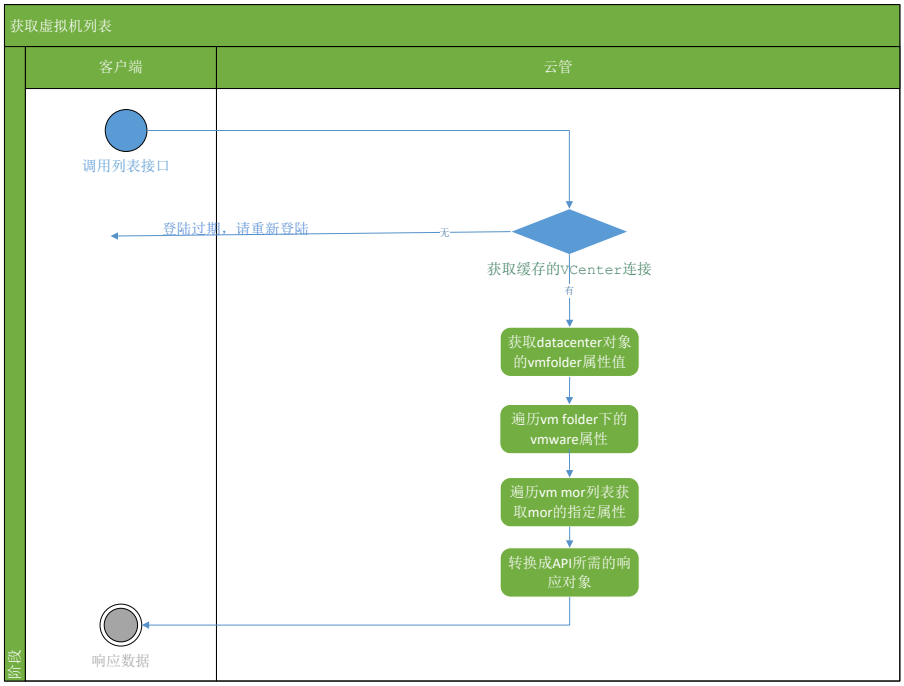
TStack VMware 的主机管理目前主要提供主机列表查询及搜索功能。点击主机列表的任一主机名称，可以进入此主机的虚拟机列表页面。



3.7.2.4 虚拟机管理

虚拟机是通过软件模拟的计算机，就像物理机一样可以运行操作系统和应用软件，每一个虚拟机是一个隔离的完整计算机系统，同一主机可以运行多个虚拟机。

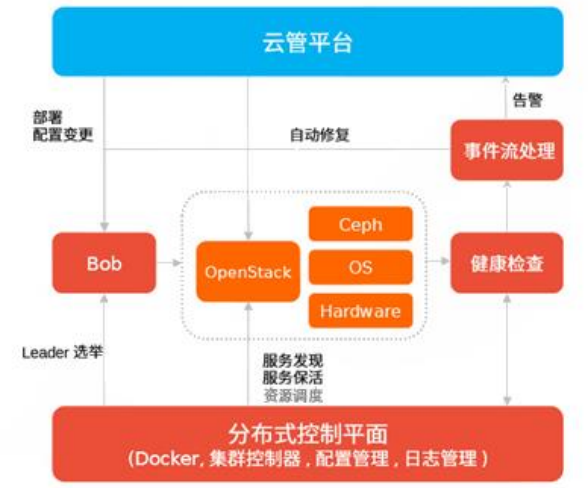
TStack VMware 的主机管理目前提供主机列表查询及搜索、虚拟机的新建、开机、关机、控制台、删除、重启、迁移、挂起、生成快照、管理快照、恢复为最新快照等功能。



腾讯公司 版权所有

3.8 安装部署

3.8.1 安装部署架构



云管平台采用容器化部署，主要使用 kolla 来部署 OpenStack 组件。

3.8.2 容器化安装部署特点

- 配置通过数据卷从宿主映射进容器
- 容器与宿主机耦合：服务发现机制
- 部署架构简单
- OpenStack 配置最小化，支持额外定制
- 支持版本升级
- 镜像和容器用 label 标记版本

3.9 云平台高可用

3.9.1 计算节点高可用

- Fence 不用关计算节点

  目前的计算节点高可用有一个比较大的问题。fence 操作只是简单的关机，而计算节点同时又是 osd 节点，会引起 Ceph 对象的副本数减少，进而引起副本的重新同步，在存储网上造成压力。如果关的是控制、计算、存储融合节点，对控制面还有额外的冲击，可能引起切虚 IP 等比较大的切换操作。我们的还发现如果频繁通过 IPMI 直接关机，可能会触发 mysql 的 bug 导致其数据库文件损坏。所有节点的 computeha 服务都原地监控自己的管理、存储、租户网。consul 的 server 节点运行 computeha 的 3 个 leader 实例，互相热备。拿到 leader 锁的 computeha 服务同样需要监控其他节点的三个网。仍然按原来的决策矩阵来决定是否需要 fence 节点。只是 fence、unfence 操作实际执行的动作变成只关闭虚拟机，不关计算节点，只有在操作系统卡住无法关闭虚拟机时才重置计算节点。为此，需要在所有计算节点上激活本机的 IPMI Watchdog，由计算节点高可用的监控服务来负责发 Watchdog 心跳。

- 自动 Unfence

腾讯公司 版权所有

在 leader 检查到 victim 的所有网都恢复后，leader 自动重新启用 victim（nova service-force-down unset）。实现了基于 Gossip 消息的多 Fence 绑定后，不再需要关闭 Consul Agent，同时也达到了 leader、victim 做 fence 后一定能让对方知道且归结成同一状态的要求，能实现自动 unfence 功能了。值得注意的是，有多个节点掉线时我们并不会做 Evacuate，在这种情况下 KV 中的 Host Evacuate 应视为已结束。

- 多 Fence 网绑定

  现在的 ComputeHA 方案中，存储网是 Fence 网，计算节点 Agent 会自己检测存储网的在线状态，如果存储网掉线，便自己关机，以期 ComputeHA 的 Master 能将其上虚拟机重建。只有节点实现这种自 Fence 才能将断网、断电归结成关机（原来的 Fenced 状态，现在是不关机只关虚拟机），这样 Master 才能在从外部探测到某节点三个网都不通后，安全的执行虚拟机的重建。由于存储网掉线一定会触发虚拟机重建，因此目前把存储网当成 Fence 网。

  如果分别有两台计算/存储混合节点的存储网网卡掉线，两者都把自己 Fence 掉，Master 节点其实未必能完全成功重建虚拟机。根据当前资源利用率的情况，Master 可能只能重建一部分虚拟机，剩下的虚拟机被关机却没得到重建，还不如不关机。问题是 Victim 节点在发现故障时，并不知道 Master 的判决和动作，盲目做自 Fence。

  优化的目标是减少做自 Fence 的场景。要减少自 Fence，就要减少 Fence 网彻底断线的概率。通常存储、租户、上连网使用不同的网卡，且上连网和存储/租户可能使用不同的交换机。所以在 Fence 时用三个网依次 failover 的方式来执行，实现一种多网绑定成一个逻辑 Fence 网的机制。

- 使用 Consul Event 实现 Fence 消息的投递

  目前的 Fecne 消息的投递是存储网 Consul Event 实现的，在 TStack 云计算管理平台项目开始时我们决定修改成 K/V 实现。再次调研过 Consul Event 和看过 Consul 代码后发现 Event 是几乎不会丢失的。Consul/Serf 的 Agent 维护了一个 512 个消息的循环缓冲区，且 Consul 和 Serf 除了用 Gossip（UDP）来群发消息之外，节点和节点之间还会周期性的用 tcp 做完整的状态同步，同步时会互相交换消息缓冲区里的所有消息。所以一旦发送方把消息广播到 Gossip 集群里，其他节点就会缓存这个消息，等目标节点如果当时离线也不怕，再上线后，会重新收到这个消息。可以把整个 Gossip 集群看成是分布式的消息循环缓冲区。所以 Fence 消息还是可以通过 Event 来发，而且每个 Event 都自带一个 Lamport Logical Clock，比自己通过 K/V 来实现带 Timestamp 的消息机制要有效。

  使用多 Fence 网+Consul 的 Event 机制来投递 Fence、Self Fenced 和 Fenced 消息，能保证 Victim 和 Leader 都能把断网和故障都归结成同一个状态。比如 Victim 检查到三个 Fence 网都不通时，自己成功关闭虚拟机后，通过 Consul Event 在多 Fence 网上发一条 Self Fenced 消息，即使 Leader 和 Victim 在检测网络时出现相位差，Leader 检查时又通了，由于 Consul Event 会持续重发消息，因此 Leader 能收到 Self Fenced 消息，得知虚拟机已关闭，可以做 Evacuate。如果多 Fence 网一直不通，Leader 在等待一段时间后，也保证要么 Victim 成功关闭了虚拟机，要么 Watchdog 重置了 Victim，也是导致关闭了虚拟机，因此可以做 Evacuate。使用这种方式，把 nova 的 resume_Guests_state_on_host_boot 设置成 True，仍然是安全的。nova 的 compute 在启动时，会询问 nova API，找到被 evacuate 走的虚拟机，从本地节点删除掉。

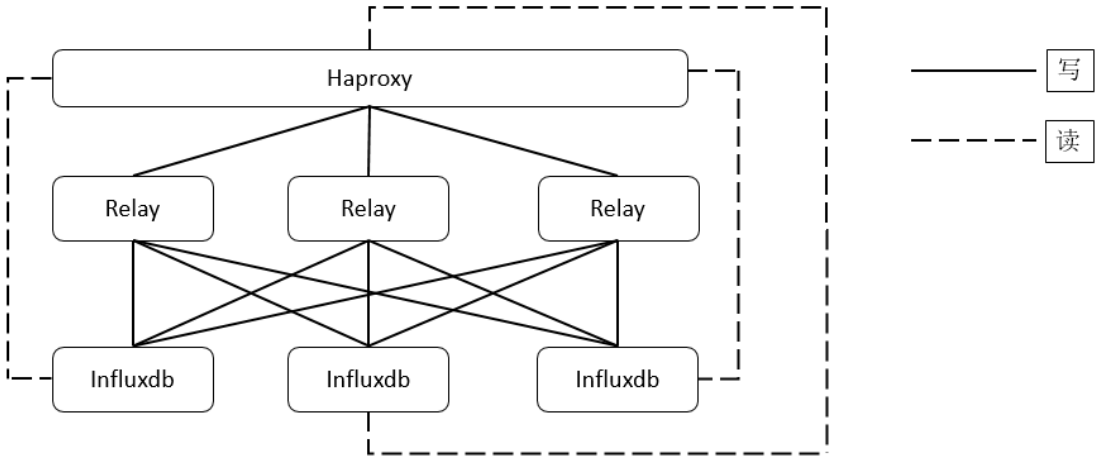
  使用 Consul 的 Event 机制来实现多 Fence 网绑定还有个好处，Fence 操作不用关 Consul Agent 了。

3.9.2 服务高可用

3.9.2.1 Influxdb 数据库高可用

influxdb-relay 采用缓存保存需要写入到 influxdb 中的数据，如果 influxdb 出现故障则 influxdb-relay 会把数据保存在缓存中，在 influxdb 下次启动的时候写入。架构如下：

腾讯公司 版权所有



正如图中所示，每个 relay 都会把所有 influxdb 实例配置成后端，relay 会作为一个服务注册到 consul 并通过 nomad 调度。

集群自动恢复策略采用每次启动判断距离最后运行时间是否超过 2 小时来进行不同操作实现。如果超过两小时，则会主动尝试向集群拉取全库数据并还原到本地，如果小于两小时则直接启动。由于 influxdb-relay 的缓存数据存在上限，我们将 influxdb 作为系统服务直接和 consul-server 进行了绑定，作为本地服务运行。这样就可以防止 nomad 调度引起的服务跳转到其他节点而引起的频繁的数据拉取和恢复。

需要注意的是，由于 influxdb-relay 不支持读操作，在 haproxy 的配置当中，读和写必须要分开，实际操作的时候通过云管访问的 url 进行区分。

3.9.2.2 Galera 集群 Bootstrap 方案

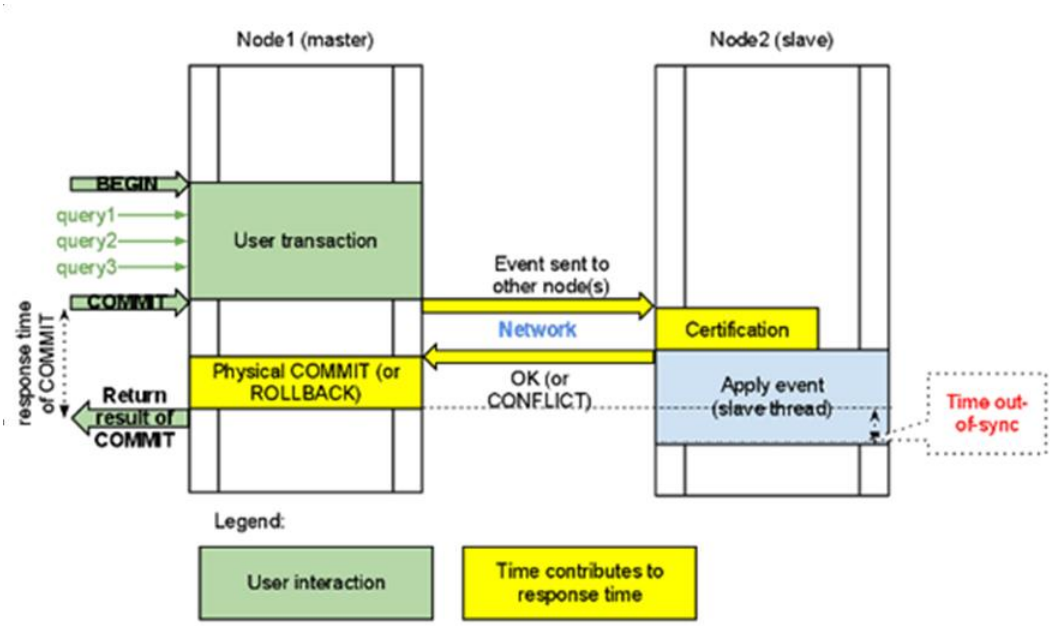
在运行 Galera 的节点上运行 galera_bootstrap 这个服务，这个服务的执行流程如下：

1. 等待可以连接 haproxy，可以通过访问 haproxy 代理的数据库服务判断当前集群的状态，因为如果所有节点都不是 Synced 的，那么这个服务无法访问。
2. 如果能够访问，表明当前集群已经 bootstrap 过，可能是只有这个节点重启了，直接加入集群就可以，不需要 Bootstrap。
3. 如果不能访问，判断本节点是不是之前的 master node。master node 的判断方法是，解析 haproxy 的 mysql 日志，然后通过 heka 写入到 consul。第一次部署时，指定一个节点主机名写入 consul。
4. 如果该节点不是 master，那就等待可以访问，如果超时，上传事件到云管平台，人工恢复。

数据库集群的数据不同步，在通常情况下，都是由于在错误的节点上进行写操作引起的。数据同步是非常重要的企业级需求，并且在近多年内成为数据库集群的核心功能。

通过 Percona MySQL Galera 集群，你可以随时对任意节点进行写操作，整个集群的机制保证了写一致性，也就是说，单一的写操作会在所有节点上生效，或者在所有节点上都不生效。简单来说，下面的图片描述了整个逻辑。

腾讯公司 版权所有



所有的数据库查询都会在指定的节点执行。但有一个例外，就是数据库事务。事务会通过认证机制与所有的节点通信，如果认证失败，该事务会返回 ERROR 给应用程序，同时执行事务回滚。如果认证成功，事务仅会在指定的节点真正执行，并且返回给应用程序，然后其它节点会在后台同步该事务。

关于部署多主同步复制架构，有以下最重要的两个配置：

首先，我们可以为同步事件配置多个事件处理线程，完全并发去执行。这个并发是利用计算机多核处理能力真正的并发，每个节点可以拥有多个并发的线程，需要在配置文件中配置如下选项：wsrep_slave_threads。

其次，在多主同步复制框架内，很有可能某个节点由于延迟会与其它节点不同步，这个是正常的，这是因为当一个节点处理事务能力比其他节点效率高时，就会发生。但在这个时刻，如果你去请求了那个延迟的节点，也就意味着你获得了过期的数据。如果你无法认可这种模式，那么你可以在配置文件中配置：wsrep_causal_reads=ON。这样的话，延迟节点上的读操作，会等待事务复制完成后再进行，保证数据一致性，这个叫做虚拟同步复制，并不是真正的同步复制，当然，这样读操作就可以会有一段忙等。

在 Percona MySQL Galera 集群中，还有一个重要的算法，乐观锁。

当你同时执行两个写事务到不同的节点时，集群会使用乐观锁来同步状态。这就意味着，事务是不会严格地检查每个请求的锁状态，而直接执行事务（COMMIT）。这样，你有可能会在执行事务时得到 ERROR 返回值。值得注意的是，这个和传统的 InnoDB 引擎的处理模式不同。

传统的 InnoDB 引擎，会在每个请求中单独返回 DEADLOCK 或 LOCK TIMEOUT 的错误，而不会在事务执行（COMMIT）时抛出。如果你正在使用多主同步复制，那么你需要在应用程序中确保每次事务执行（COMMIT）时有正确的异常处理机制。

3.10 监控管理

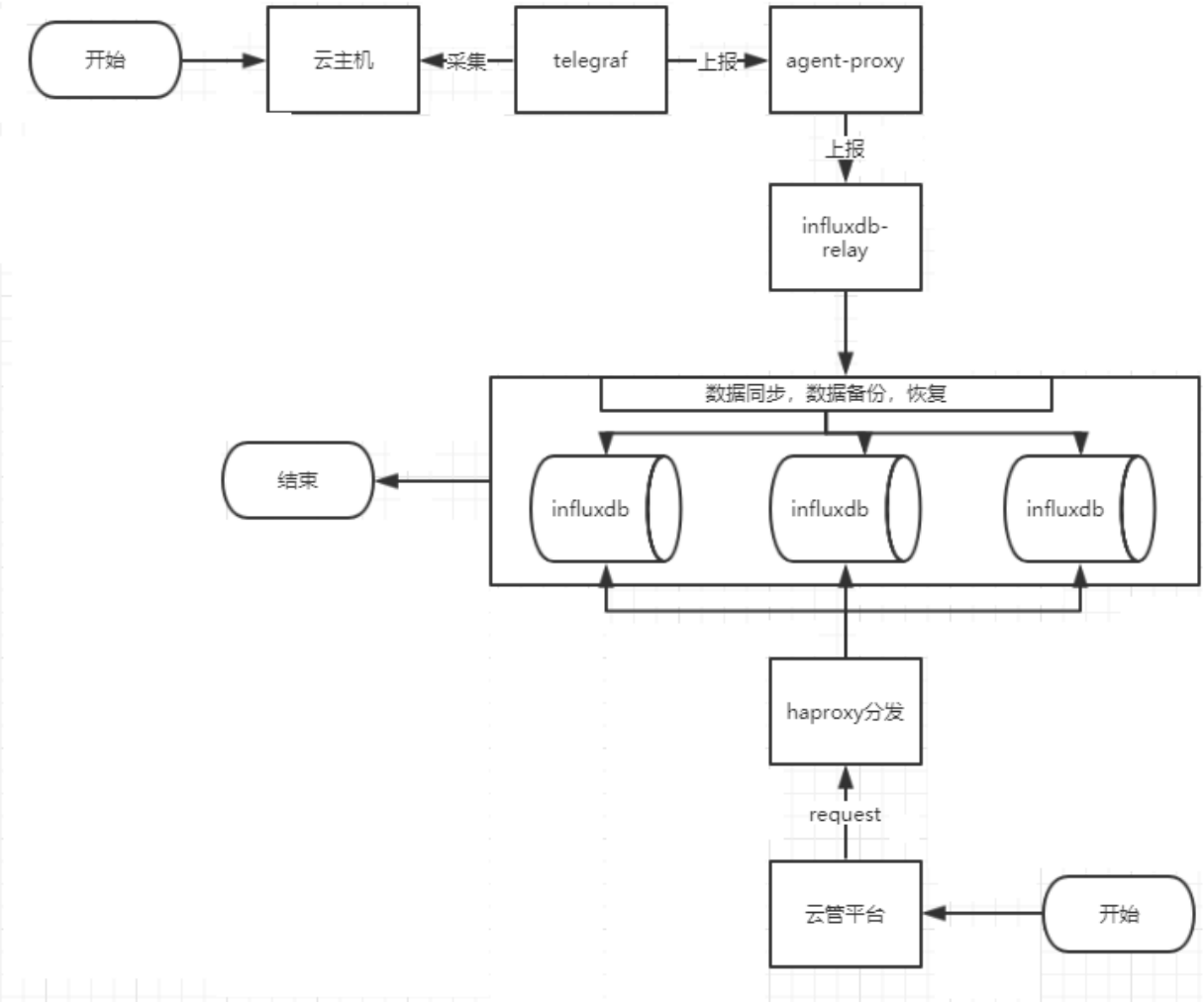
3.10.1 云主机监控和告警

- 监控

第三方件 telegraf 负责监控数据的采集，由于云主机网络（租户网）与物理机网络（集群网）不是同属一个网络下，所以需要自定义项目 agent-proxy 来收集云主机上的采集数据。

腾讯公司 版权所有

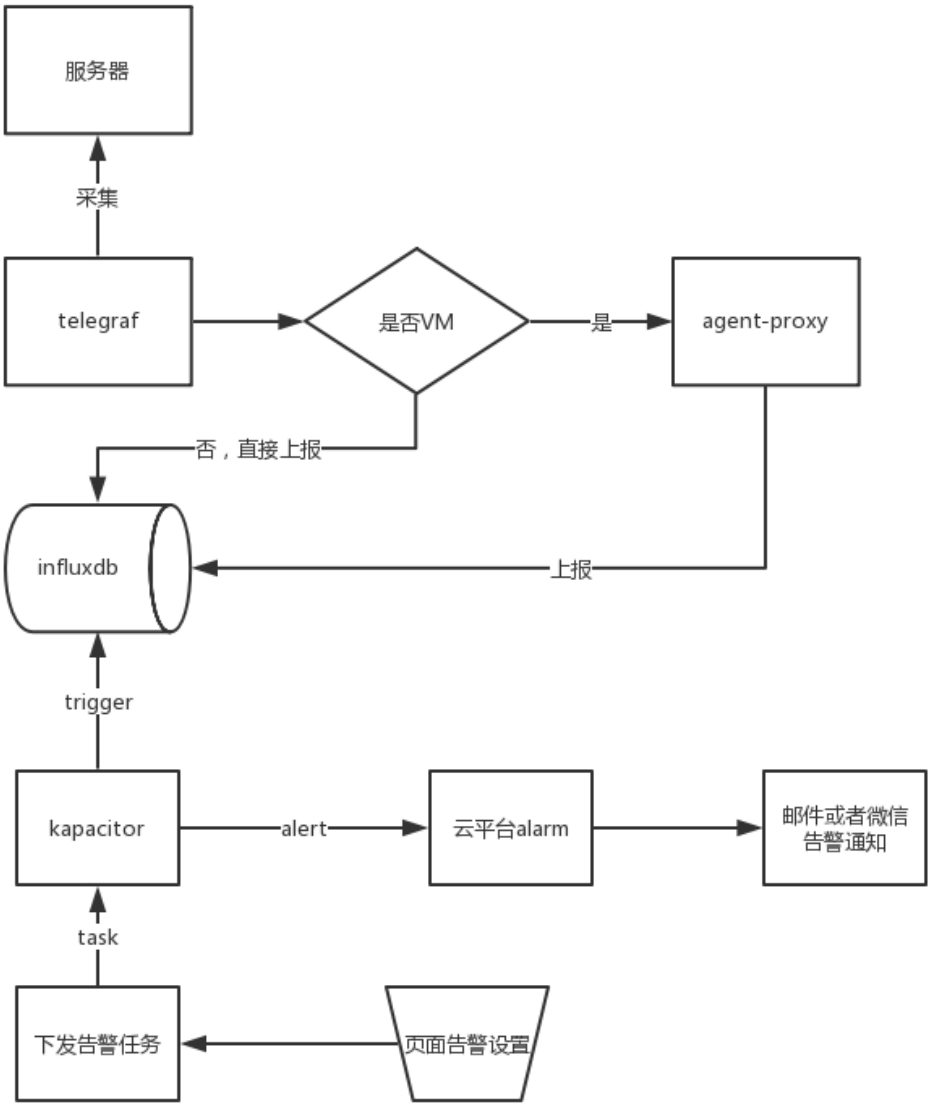
Agent-proxy 项目通过 libvirt 模块和 socket 文件来实现连接，在连接上建立监听，即可实时获取 telegraf 的上报数据。然后 agent-proxy 解析并处理采集数据，上报给 influxdb-relay，relay 写入到 influxdb 数据库。最后云管平台通过读取 influxdb 数据库中的数据，通过图形或报表展示。



- 告警

第三方件 kapacitor 实现告警。云管平台通过 kapacitor 的 API，实现告警任务的建立和启用。Kapacitor 内部运行告警任务，kapacitor 对接 influxdb，使得任务中的阈值巡检可以在 influxdb 运行查询语句。依据查询语句的返回值，依据告警策略，判定是否推送告警信息给云管平台。

腾讯公司 版权所有

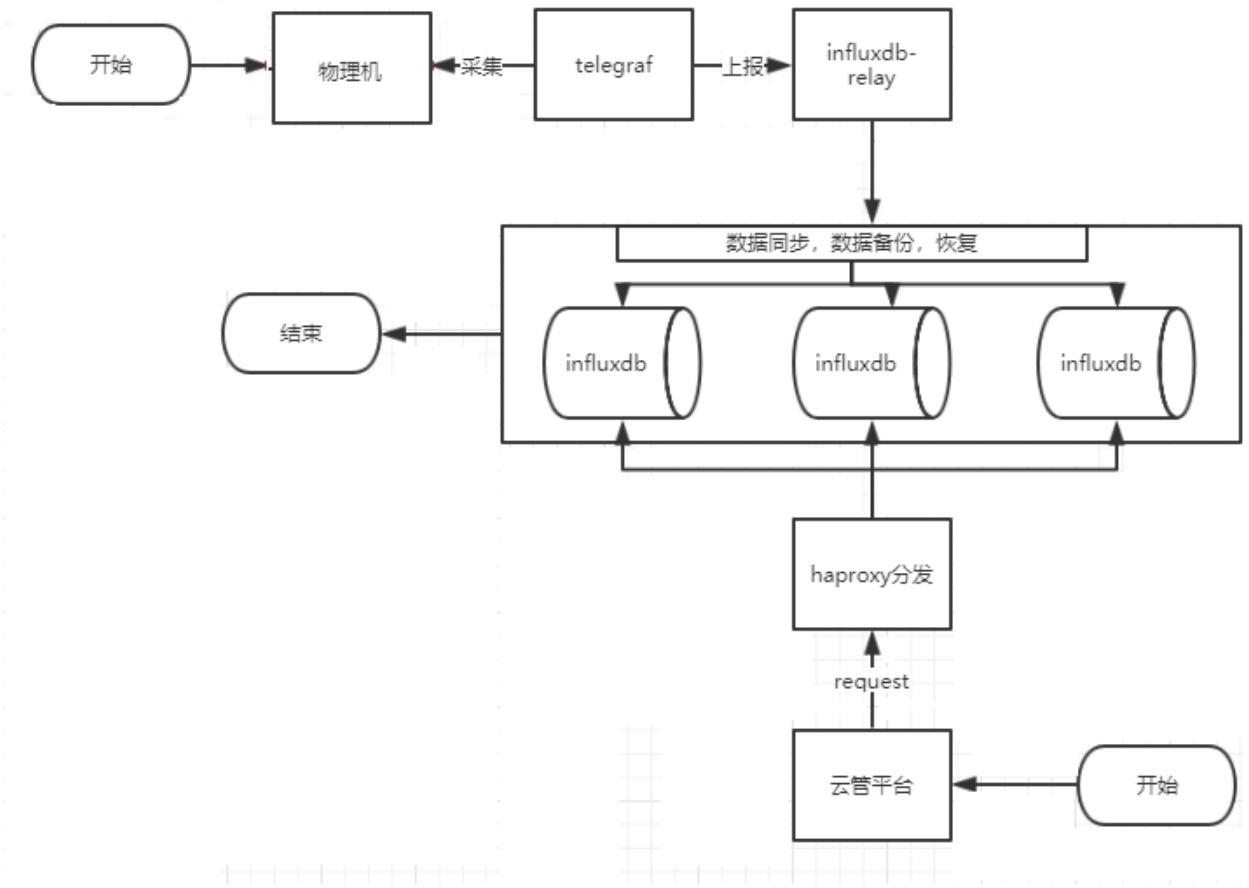


3.10.2 宿主机监控和告警

- 监控

第三方件 telegraf 负责监控数据的采集，然后直接上报给 influxdb-relay，relay 写入到 influxdb 数据库。最后云管平台通过读取 influxdb 数据库中的数据，进行图形或报表展示。

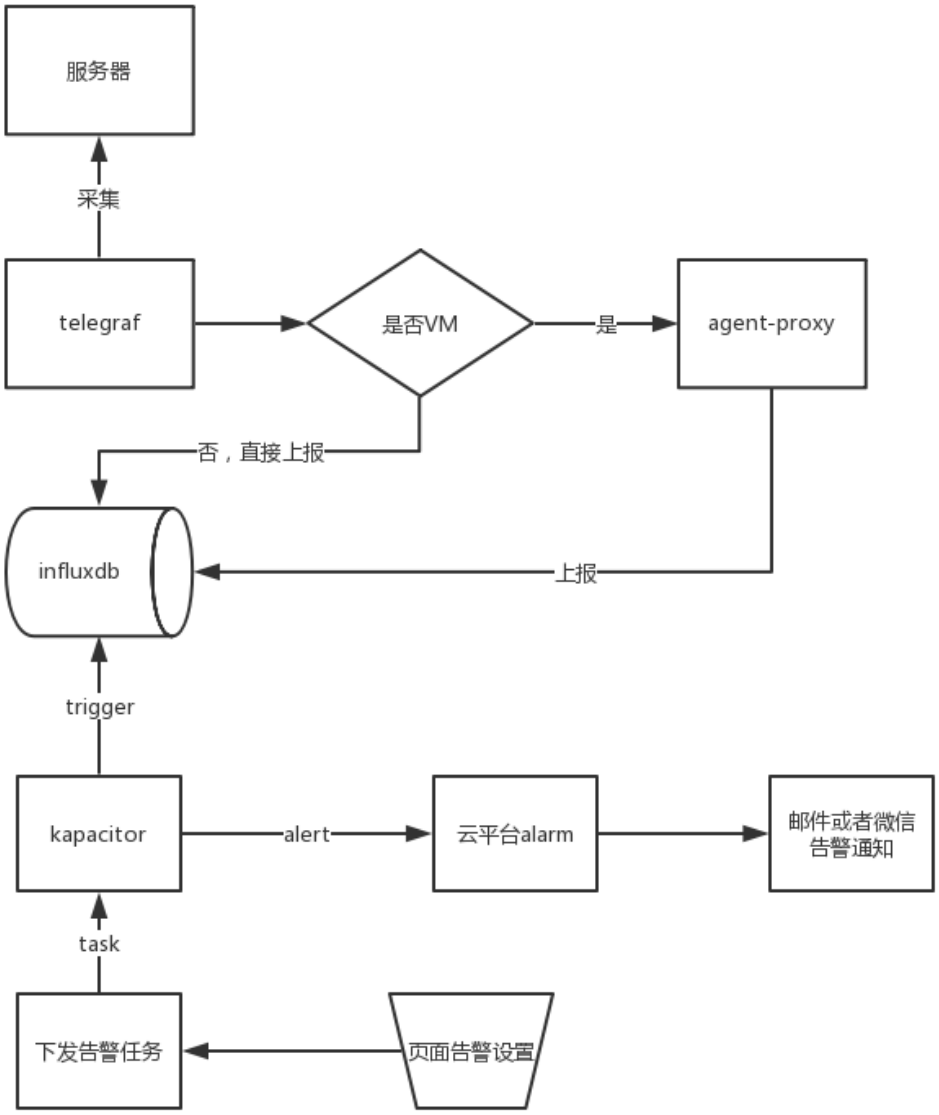
腾讯公司 版权所有



- 告警

第三方件 kapacitor 实现告警。云管平台通过 kapacitor 的 API，实现告警任务的建立和启用。Kapacitor 内部运行告警任务，kapacitor 对接 influxdb，使得任务中的阈值巡检可以在 influxdb 运行查询语句。依据查询语句的返回值，依据告警策略，判定是否推送告警信息给云管平台。

腾讯公司 版权所有



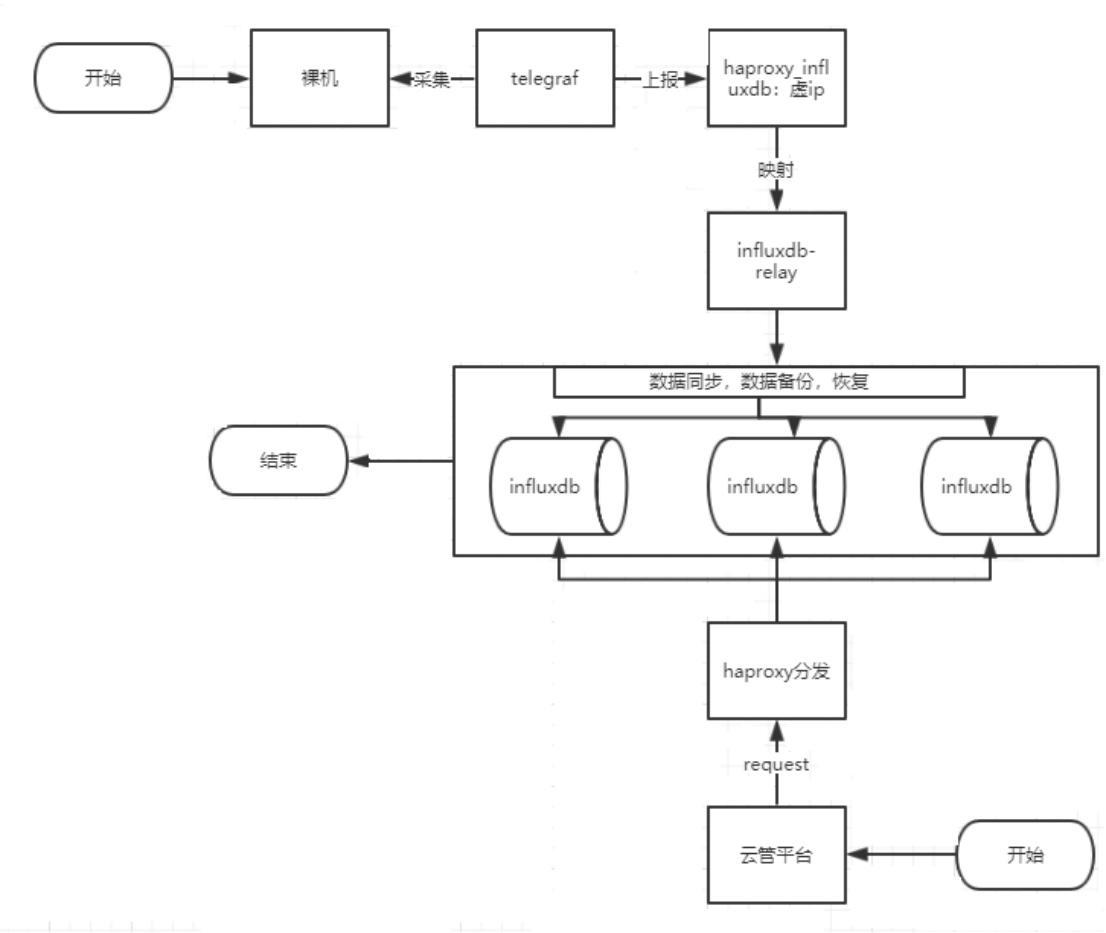
3.10.3 裸机监控和告警

- 监控

在裸机上安装适配的 telegraf 采集插件（安装方式，可参考用户操作手册）。telegraf 负责监控数据的采集，然后上报通过映射的虚 ip 转发给 influxdb-relay，relay 写入到 influxdb 数据库。最后云管平台通过读取 influxdb 数据库中的数据，进行图形或报表展示。

说明：
虚 IP 是为保证网络的连通性而专门设置的。在 ironic_vm 上。有个 haproxy_influxdb 服务，该服务负责虚 IP 的请求转发，端口为 8086。

腾讯公司 版权所有



- 告警

第三方件 kapacitor 实现告警。云管平台通过 kapacitor 的 API，实现告警任务的建立和启用。Kapacitor 内部运行告警任务，kapacitor 对接 influxdb，使得任务中的阈值巡检可以在 influxdb 运行查询语句。依据查询语句的返回值，依据告警策略，判定是否推送告警信息给云管平台。

3.10.4 Ceph 存储监控

对第三方件 telegraf 进行自定义扩展，增加对 ceph 数据的采集。Ceph 数据的采集通过调用 Ceph API 来实现，主要采集的监控数据有：ceph 集群状态，ceph 存储池基本信息（name,avail,total,used,rBandwidth,wBandwidth,IOPS）。另外 ceph 的 PG 和 OSD 是直接通过云管调用 ceph API 来实时获取的。

3.10.5 网络监控（含物理网络监控）

- 集群网络监控：

集群网络的监控是通过自定义项目 awstack-healthcheck 来实现的，healthcheck 服务运行在节点上。

集群网络的监控实现：检查系统持续运行时间>10 分钟，consul members 中的 3 个 server 节点全部在线，且 nomad 调度的服务全部 Ok。然后确认集群中所有节点都在线。

- 网卡监控：

配置文件中配置需要监控的网卡名称，遍历各个网卡状态：

 # /usr/sbin/ethtool %s|grep 'Link detected'

腾讯公司 版权所有

3.10.6 云平台自身监控

健康检查：

集群网络的监控是通过自定义项目 awstack-healthcheck 来实现的，healthcheck 服务运行在节点上。

检查项主要分两块：

集群相关：

- ceph 存储
- nomad 调度服务
- openstack 服务

节点相关：

- 网络
- 磁盘
- 网卡

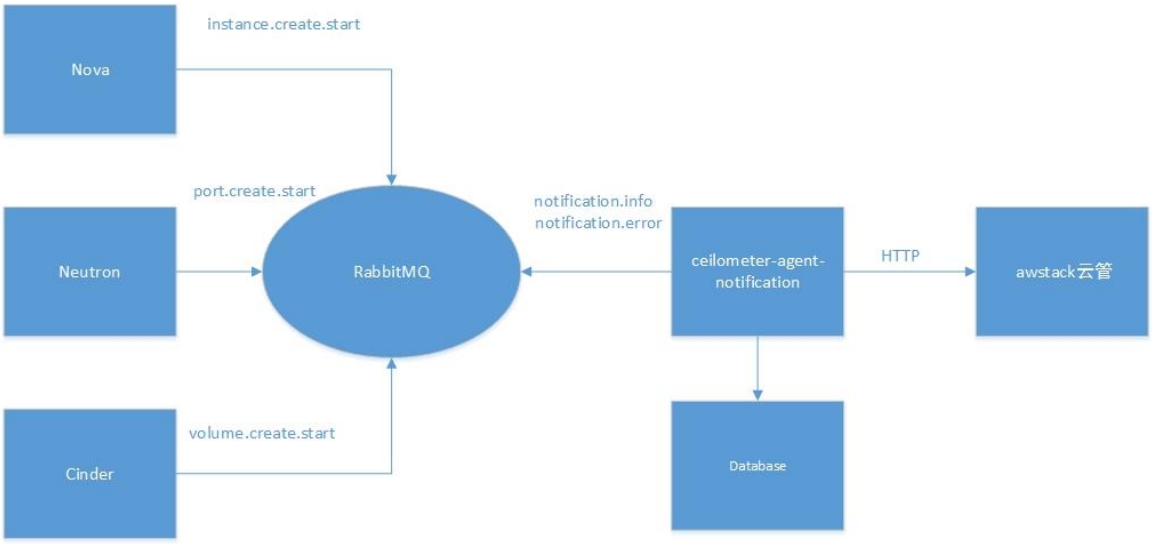
对于服务：

- 集群相关的服务名称为 clustercheck。只有一个。
- 节点基础信息的服务名称为 nodecheck。每个节点上都有一个。

3.11 计量计费

3.11.1 资源计量

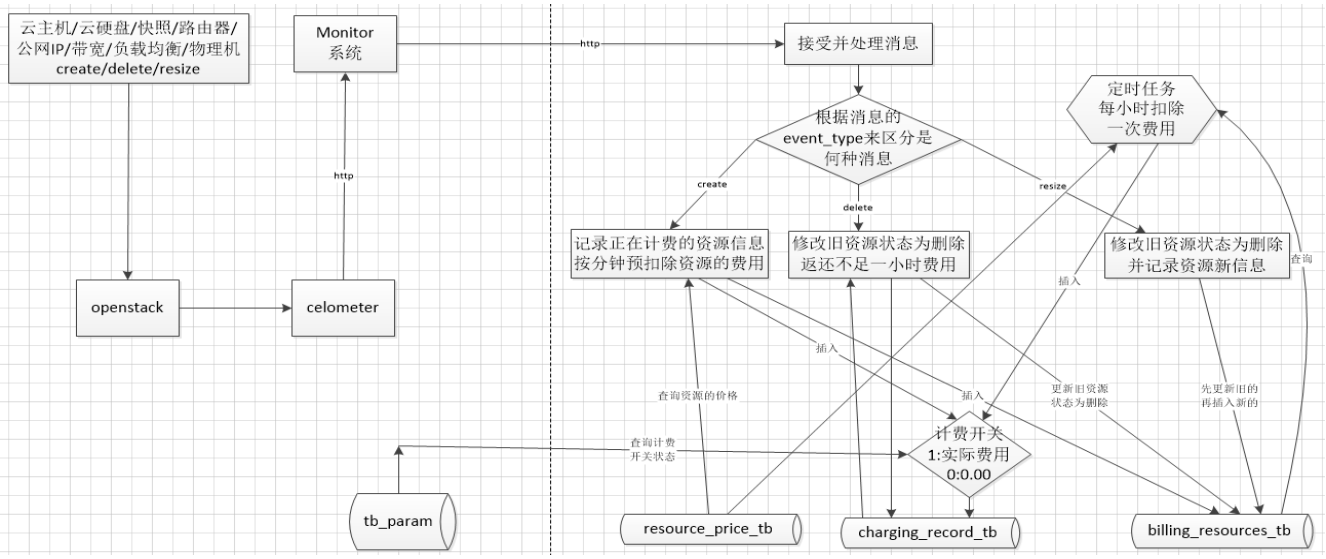
OpenStack 通过 Ceilometer 实现资源计量。在我们产品里，启动 ceilometer-agent-notification 服务，OpenStack 服务（如 Nova、Neutron、Cinder）配置 oslo_messaging_notifications 的驱动为 messaging，即事件通知（创建虚机、创建网络、创建硬盘等操作）发送到消息队列，ceilometer-agent-notification 通过访问消息队列，消费相关事件通知消息，转换 event 数据格式后，存放在数据库中进行备份，同时通过 HTTP publisher 发送 HTTP 请求至 SAAS 云管，云管解析消息内容，推送消息到前端页面。



腾讯公司 版权所有

3.11.2 资源计费

在云管平台系统设置中设置相应的计费开关，当打开计费开关时，云管平台的资源可以进行相应的计费。计费的前置条件需要云管计费管理中的价格管理，对相应资源录入计费标准，云管通过对计量的资源按小时计费，云管目前采用预计费模式，对于资源不足一小时的情况通过返还金额方式补偿差额。



4 TStack 云计算管理平台核心特性

4.1 智运维

- TStack 平台系统支持监控管理、日志管理、系统巡检和配置管理，能对物理和虚拟资源、应用及服务的状态进行实时监控，并定期对环境进行健康巡检。
- TStack 平台有丰富的告警策略，支持大屏查看实时监控，并通过微信或邮件的方式通知到管理员。
- 通过策略自动调整业务资源的服务，按需调整资源。在业务增长时能够实现应用系统扩容，以满足业务需求；业务下降时能够实现应用系统减容，减少资源浪费。
- TStack 平台支持 Ceph 超融合存储管理，采用多数据副本保护机制，当安装超融合 Ceph 时，平台会对 Ceph 进行智能运维，比如 Ceph 集群中的一块硬盘故障后，在操作页面中将硬盘删除，并更换新硬盘后，可以将新硬盘加载到集群中，设置定时时间后，会自动恢复数据。
- TStack 平台支持将挂载在物理机上的 USB 或 GPU 等设备透传给云主机，云主机可以使用该透传设备，且被加载的设备会一直属于该云主机。
- TStack 平台采用去中心化、全对称的分布式架构，最大限度利用物理资源，消除单点故障，且系统具备数据副本自动恢复的机制，可以保障数据安全可靠。当集群中某个节点或某功能出现故障时，可以自动化恢复功能，同时通知负载均衡器将请求分发到功能正常的节点上，保障业务系统的持续可用性。

4.2 智运营

- TStack 平台可以统计并展示出各种类型资源的使用量（当前使用量和累计使用量）及其排行，有利于管理员进行资源整合分析，避免僵尸资源和资源浪费资源的情况产生，以达到资源最大化利用。

腾讯公司 版权所有

- 在云管平台中设置计费开关，当打开计费开关时，云管平台的资源可以进行相应的计费。消费统计可以显示各数据中心费用及占比，各类资源的费用及占比，并且支持查看历史费用信息。帮助运营管理员更清晰的掌握当前平台用户所使用的资源情况。

4.3 智部署

- TStack 平台可根据安装部署的物理主机规模，自动设置系统服务的负载数，可以达到让管理平台正常运行的目的，同时尽量减少系统开销，将有限的计算资源留给业务使用，并根据后续集群规模的扩展，细粒度扩展管理服务的负载能力。
- TStack 平台支持最少 3 节点的安装部署，用户仅需通过注册、选择主机、网络配置、配置下发等几步操作，分钟级实现资源池的部署和构建，大幅降低实施的人力成本和时间成本。

4.4 智使用

- TStack 平台可以自定义组织结构，针对不同的角色分配不同的权限，灵活应对企业多级组织架构，从用户、项目和部门不同粒度管理资源针对不同维度进行资源统计计量。
- TStack 平台提供完整的虚拟机生命周期管理，支持虚拟机的创建、修改、启动、重启、关机、迁移、快照等常用功能，同时支持通过管理界面的控制台远程连接到虚拟机。
- 云平台管理员可以根据实际需求对审批级别和流程进行自定义，严格且高效的执行规范流程。

腾讯公司 版权所有