



About 云

零基础学习 openstack (上) 【中级篇】

(第三期)

about 云为热爱云开发技术人员提供最全面的信息传播和服务平台，内容包括云技术文档，视频、云技术学习指导，解疑等。是大数据、云技术爱好者的学习分享基地。

about 云，本着活到老学到老的精神，为了广大云技术爱好者获取更多知识，在文章开头，都有几个问题，因此 about 云亦为学问社区。同样我们准备了每日一读，为了就是每天进步一点，每天能够学到新的内容。



QQ 群: 39327136、322273151、371358502、90371779

每日一读汇总腾讯认证空间：

<http://user.qqzone.qq.com/278595437/main>

接上一篇：

[零基础学习 openstack 【初级篇】](#)

本文链接

<http://www.aboutyun.com/thread-10124-1-1.html>

初级篇，我们主要是有这么一个概念，openstack 的组成
openstack 由哪些部分来组成：

- Identity(代号为"Keystone")
- Dashboard(代号为"Horizon")
- Image Service(代号为"Glance")
- Network(代号为"Quantum")
- Object Storage(代号为"Swift")
- Block Storage(代号为"Cinder")

及它们的初步认识，这篇，我们将深入这些概念，及对 openstack 的进一步的认识。

我们初级篇中，知道了如何部署集群，如何使用集群，但是遇到了很多的问题，

1. 什么是 *floating ip* ?
2. 什么是管理网络？
3. 为什么会获取不到 *ip*?
4. 为什么网络会不通？
5. 为什么虚拟机 *ping* 通，外网 *ping* 不同？
6. 出现问题了，会看日志了，可是还是找不出问题的原因？

这些都困扰着我们。

那么我们为什么会产生这些问题，并且遇到这些问题，还解决不了，到处求救，可是并非每次都那么幸运。

我们的学习方法，通常有两个极端：

- 1.先看书
- 2.不管三七二十一，先动手在说。

看书

看书是没有错的，但是切勿较真，因为很多书并不像《春秋》、《大学》、《易经》，那么值得推敲，现在的你看的更多的是一种框架，一本书籍，如果能够保证 98%以上都是正确的，就已经很不错了。但是个人认为看书总比不看书要好一些，毕竟花费了作者大量时间和精力去整理。

再回来，我们在看书的时候，有时候，并不能一次就能把书看透，使劲也是没用的，因为水平在那。所以建议看书的时候，先整体了解，有了初步概念和自己的理解，然后动手实践。

实践


有些同学，注重实践，因为实践才能获取知识，其实这个没有错，但是我们经常会遇到问题，并且不知该如何解决。比如在大数据、云技术中，搭建 **hadoop** 集群，**openstack** 集群，调试的信息，都在日志中了，我们也知道看日志，但是眼睁睁看日志，却还是不知道问题出现在什么问题。**这时候你该怎么做？**

有三种办法？

1.在 qq 群求助

这种效果一般不怎么好

2.在论坛社区发帖

论坛发帖一般也不怎么靠谱，当然除了 **about** 云以外 ，**about** 云对于发的帖子都会及时的回答。当我们遇到问题，解决问题的时候，我们就需要考虑，我们是否该看书了

3.回头看书

回头看书，这时候看书，你的收获很大，而且还会对以前的知识有一个重新理解和定义。

平时积累（爱好）

在看书和实践的过程中，其实还有更重要的一点，就是我们平时积累，这是任何学习方法都无法比拟的。没有比爱好更重要的。因为爱好所以琢磨，因为琢磨，所以有深度。

about 云也会为云技术、大数据爱好者，提供每日一读，网站和群（90371779、322273151）每天都会有相关内容

[about 云每日一读汇总](#)

以上方法是针对学习云技术、大数据，当然同样适合其它 IT 技术。

同时这里推荐一篇很不错的文章

[想学大数据、云技术、IT 人、大学生必读的一篇文章：如何快速掌握一门技术](#)

云计算、openstack 的理解

首先我们还是来说，什么是 openstack，什么是云计算。

引用百度：

云计算是一种通过网络以服务的方式提供动态可伸缩的虚拟化的资源的计算模式。

举个例子，你要做个网站，希望有一台独立的服务器，以前你可能得自行购买一台服务器并托管在 IDC 机房，不仅得花很多钱买服务器，而且每年要花很多钱托管（当然你也可以租一台服务器）。而现在，你可以在云计算服务商那里租一台同样由你掌握的“服务器”，你一样可以对它格式化，安装自己喜欢的操作系统和软件，但它并不是一台物理上的服务器，而且云计算平台上为你提供的一台虚拟机。

因此，云计算是由一系列可以动态升级和被虚拟化的资源组成，这些资源被所有云计算的用户共享并且可以方便地通过网络访问，用户无需掌握云计算的技术，只需要按照个人或者团体的需要租赁云计算的资源。

如果你真想了解，可以从虚拟机入手。简单讲，虚拟机是云计算的基础。

云计算的特点：

1. 云计算拥有以下特点：
2. · 虚拟化和自动化
3. · 服务器，存储介质，网络等资源都可以随时替换
4. · 所有的资源都由云端统一管理
5. · 高度的伸缩性以满足业务需求
6. · 集中于将服务传递给业务

引用：[云计算是什么](#)

上边便是云计算的解释，但是我们仍旧感觉很模糊。

那么我们从 **openstack** 的角度来理解，说到 **openstack** 我们必须说一下，**openstack** 的发展，**openstack** 的初期，**nova** 是主要的组件，但是由于不断的扩展，所以逐渐的从 **nova** 中分离出来。

比如：

网络组件 **nova-network**

发展如下：

```
nova network->Quantum->Neutron
```

Openstack 在 2010 年正式发布它的第一个版本 **Austin** 的时候，**nova-network** 作为它的核心组件被包含其中原先网络有 **nova network** 来承担，后来逐渐分离出来，改名为 **Quantum**。

Quantum 是随 Openstack 的 **Folsom** 版本正式发布的，其实它已经作为试用组件包含在之前的 **Essex** 版本中。在 **Grizzly** 里功能得到了增强。

为什么引入 **Quantum**？答案非常简单，**Quantum** 功能更强大，满足更多需求。

Neutron

因为商标侵权的原因，Openstack 在 **Havana** 版本上将 **Quantum** 更名为 **Neutron**，所以 **Neutron** 并不是什么新的东西。在 **Havana** 版里，**Neutron** 也只增加和增强了少数功能。

更详细信息参考

[OpenStack 网络组件 Neutron 的研究](#)

nova-volume 组件

Essex 将 **nove** 的卷管理 **api** 独立化后，**Folsom** 终于将卷管理服务抽离成了 **Cinder**；**Cinder** 管理所有的块存储设备，块设备可以挂接在虚机的实例中，然后虚机里的 **guest** 系统可以像操作本地卷一样操作块存储设备；

Cinder 需要处理的主要问题应该是接入各种块设备，如本地磁盘、**LVM** 或各大厂商提供的设备如 **EMC**、**NetApp**、**HP**、**HuaWei**，还有如 **Vmware** 提供的虚拟块设备等。

从上面我们认识，**nova** 为 **openstck** 的重要组件，而 **nova** 中 **nova-compute** 则可以创建虚拟机。它也是云计算的核心。

所谓的云计算，从技术角度来讲，**其实就是能够灵活的创建和删除虚拟机。**

你或许会有很多的疑问或则不相信，为什么创建和删除个虚拟机就被称之为云计算，我们创建和删除虚拟机这不是很平常的一件事情吗？openstack 为什么会如此的火热。

到这里，让我们在来看看什么是云计算，或许有更进一步的认识。

引用百度：

云计算是一种通过网络以服务的方式提供动态可伸缩的虚拟化的资源的计算模式。

举个例子，你要做个网站，希望有一台独立的服务器，以前你可能得自行购买一台服务器并托管在 IDC 机房，不仅得花很多钱买服务器，而且每年要花很多钱托管（当然你也可以租一台服务器）。而现在，你可以在云计算服务商那里租一台同样由你掌握的“服务器”，你一样可以对它格式化，安装自己喜欢的操作系统和软件，但它并不是一台物理上的服务器，而且云计算平台上为你提供的一台虚拟机。

因此，云计算是由一系列可以动态升级和被虚拟化的资源组成，这些资源被所有云计算的用户共享并且可以方便地通过网络访问，用户无需掌握云计算的技术，只需要按照个人或者团体的需要租赁云计算的资源。

如果你真想了解，可以从虚拟机入手。**简单讲，虚拟机是云计算的基础。**

云计算的特点：

1. 云计算拥有以下特点：
2. 虚拟化和自动化
3. 服务器，存储介质，网络等资源都可以随时替换
4. 所有的资源都由云端统一管理
5. 高度的伸缩性以满足业务需求
6. 集中于将服务传递给业务

openstack 个组件部署

当我们想学习大数据、云技术的时候，部署往往是我们的第一步，这样能够对 openstack 有一个直观的认识，比如那个文件需要修改，部署完成后该如何使用，详细参考：

[openstack 零基础入门：OpenStack Grizzly 安装指导（1）](#)

[openstack 零基础入门：OpenStack Grizzly 安装指导（2）](#)

更多内容参考

[零基础学习 openstack【初级篇】](#)

当然上面部署是一套 openstack 的部署，如果想单个部署，网上资料也还是不少的。

openstack 开发

1.环境搭建

对于 openstack 开发，开发环境还是比较重要的，有了开发环境，我们可以阅读源码，同样可以修改里面的环境。那么我们该如何搭建开发环境。

比较可靠的方法，可以参考

[about 云课程 5：配置 Linux 中的 eclipse 环境，导入 openstack keystone 源码](#)

当然网上流行了比较多就是使用 dev 来搭建开发环境，由于各种原因，使用 dev 搭建过程中，可能会遇到比较多的问题，参考

[使用 DevStack 安装和配置 OpenStack 开发环境](#)

[基于 DevStack 的 Openstack folsom 版开发环境搭建（1-2）](#)

[基于 davstack 搭建 openstack folsom 开发环境（3-4）](#)

[OpenStack Nova 开发与测试环境搭建](#)

[准备 OpenStack 开发环境遇到的问题及解决办法](#)

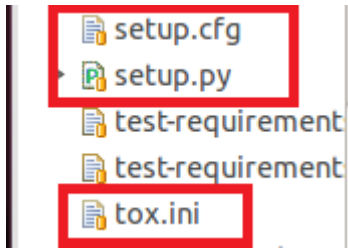
[建立 openstack quantum 开发环境](#)

2. 源码架构

当我们拿到源码的时候，我们如果直接每个文件查看源码，这个难度是相当大的，首先我们需要搞清楚源码

python 工程

一般有

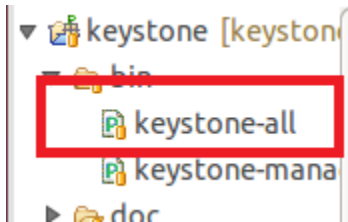


setup 文件

setup.cfg 配置文件

tox.ini 测试文件

bin 目录



keystone-all

keystone-manager

当项目启动时，keystone-all 为项目总入口

openstack 各组件

我们了解了云计算，在我们技术人眼里，其实根本不是什么云计算，就是创建个虚拟机，而 openstack 就能完成这件事情。云计算之所以这么称呼当然有它的道理，当然咱们只关心技术，所以咱们需要对 openstack 有一个深度的认识。

对于英语比较好的同学，学习可以直接访问官网，[详细参考](#)：

[新手指导：OpenStack 官网介绍](#)

官方地址：<https://github.com/openstack>

孵化项目：<https://github.com/stackforge>

首先我们需要对各个组件有一个认识，包括从原理、内部结构、部署、源码等角度。

了解认识 Nova

这个是最核心的，Nova 最开始的时候，可以说是一套虚拟化管理程序，还可以管理网络和存储。不过从 Essex 版本后，Nova 开始做减法，和网络相关的内容，包括安全组，交给 **Neutron** 负责，存储相关的交给 **Cinder** 负责。调度有关的内容，会交给新的项目 [Marconi](#)。

以前还有一个 nova common，这其实是各个组件都使用相同的东西，现在也专门成立一个项目：oslo，已经是核心项目。

未来 Nova 对各种 Hyperv 的支持是有差异的，KVM 和 XEN，基本是最好的。微软的 Hyper-V 算是很不错，微软投入再研发。计算节点，不直接查询数据库，而是通过 rpc 的方式，据说这是一大进步。

Nova 的稳定性，其实取决于 libvirt, qemu，希望未来可以能更加稳定。功能现在其实已经不是大问题。

那么我们再来看一下 nova。

nova 是一个很复杂的组件，而且内容很多。

认识 nova

nova 可以说是一套虚拟化管理程序，为什么这么说，因为 nova 可以创建、删除虚拟机、重启虚拟机等，openstack 之所以能够搭建云平台，也是因为它能够创建虚拟机，其它的组件，比如 Neutron 则是为了让虚拟机之间、虚拟机与外网之间能够互通，Cinder 则是为了增加虚拟机的存储空间。可见 nova 在 openstack 中作用是非常大的。

更多内容，可以参考下面内容。

[大家谈 OpenStack-Nova 组件理解](#)

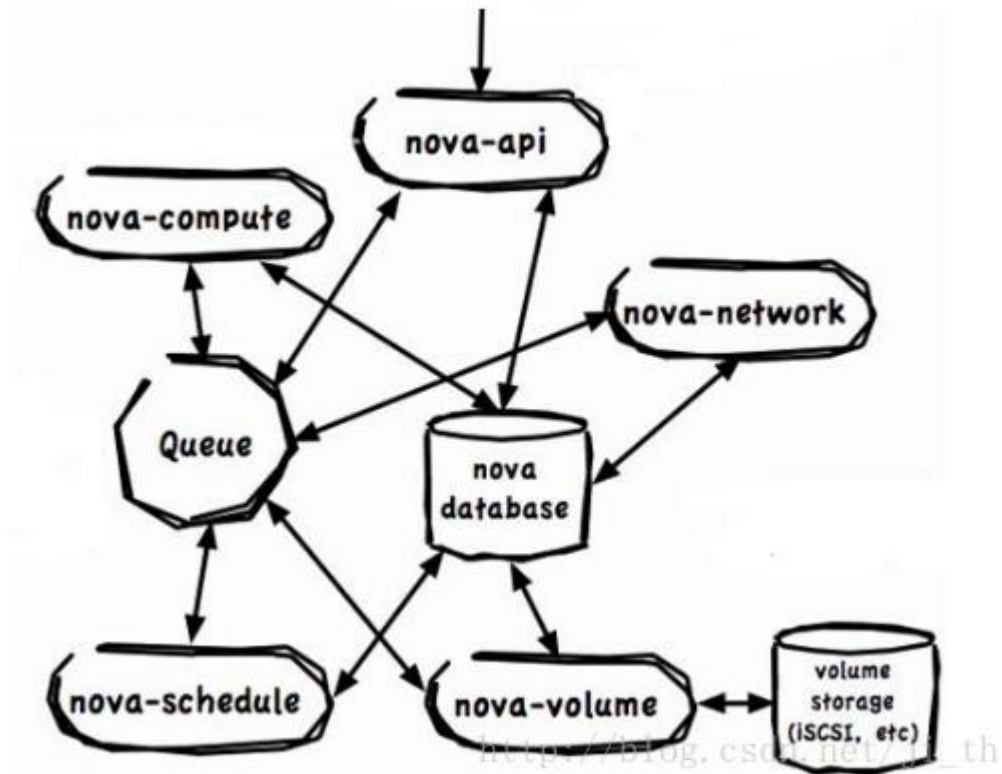
[关于 OpenStack 中 Nova 的几个基本概念](#)

[Openstack 核心，nova 详细介绍](#)

[OpenStack Compute\(Nova\)功能分析](#)

nova 结构

nova 是云主机控制器。它包含了很多组件, API 服务器 (nova-api), 计算服务器 (nova-compute), 网络控制器 (nova-network), 调度器 (nova-schedule), 卷控制器 (nova-volume), 消息队列以及 DashBoard。



至于 nova 的发展, 上文我们已经介绍。

对于 nova 各个组件的作用, 及它们之间是如何通信的, 详细可以参考：

Nova 各个组件介绍以及功能分析(逻辑架构, 运行架构, 开发架构以及数据库)

<http://www.aboutyun.com/thread-10069-1-1.html>

nova 命令行：

我们认识 nova 和了解了 nova, 那么 nova 具体该如何使用, 如何删除虚拟机、添加虚拟机、启动虚拟机等, 参考下面帖子。

openstack nova 用户管理

<http://www.aboutyun.com/thread-8717-1-1.html>

openstack nova 命令行指令大全

<http://www.aboutyun.com/thread-6373-1-1.html>

nova 源码及开发

对于 nova 有了一定的认识，如果我们对 nova 二次开发

- 1.我们首先搭建开发环境
- 2.然后阅读源码修改源码

至于开发环境的搭建参考上文 [openstack 开发](#)，源码的阅读，这里有一些帖子供大家参考.对于内容，有的是对源码的整体认识，及虚拟机启动源码分析，更多的内容，相信能从标题可以看到。

OpenStack Nova 源码结构解析

<http://www.aboutyun.com/thread-10105-1-1.html>

- 1.处理虚拟机磁盘镜像由哪个文件来完成？
- 2.调度器中的主机权重在哪个文件中？
- 3./nova/scheduler/host_manager.py 文件的作用是什么？

OpenStack 基于 Libvirt 的虚拟化平台调度实现----Nova 虚拟机动态迁移源码分析

<http://www.aboutyun.com/thread-10108-1-1.html>

- 1.实现虚拟机动态迁移主要实现的语句是什么？
- 2.方法_update 实现了哪方面的内容？
- 3.live_migration 方法的作用是什么？

NOVA 源码分析——NOVA 中的 RabbitMQ 解析

<http://www.aboutyun.com/thread-10107-1-1.html>

- 1.终端用户 (DevOps、Developers 和其他 OpenStack 组件) 如何与 openstack 系统互动？
- 2.Nova 守护进程之间如何执行 API 请求？
- 3.RabbitMQ 是什么？
- 4.构成 AMQP 的三个关键要素，那么它们之间是如何工作的呢？

OpenStack Nova-cell 服务的源码解析 (1)

<http://www.aboutyun.com/thread-10104-1-1.html>

1. **nova-cell** 服务的具体实现包含哪些流程？
2. 哪个类定义了当路由信息到特定的 **cell** 上时，需要调用的方法？
3. **schedule_run_instance** 实现了什么？

OpenStack Nova-cell 服务的源码解析 (2)

<http://www.aboutyun.com/thread-10103-1-1.html>

OpenStack Nova-scheduler 组件的源码解析 (1)

<http://www.aboutyun.com/thread-10102-1-1.html>

1. 哪个文件实现了基于随即选取主机节点的方式的调度器？
2. **/nova/scheduler/manager.py** 文件作用是什么？
3. **/nova/scheduler/filters/affinity_filter.py** 定义了那四个过滤器？

OpenStack Nova-scheduler 组件的源码解析 (2)

<http://www.aboutyun.com/thread-10121-1-1.html>

1. **host_state.update_from_compute_node(compute)** 这条语句实现了什么功能？
2. 哪一个函数循环实现了为每一个实例获取合适的主机后，返回选择的主机列表？
3. **_schedule** 实现有哪三步？

OpenStack 基于 Libvirt 的虚拟化平台调度实现----Nova 虚拟机启动源码实现 (1)

<http://www.aboutyun.com/thread-10100-1-1.html>

1. **Nova-Compute** 中 **Libvirt** 默认调用的底层虚拟化平台是什么？

2.Libvirt 是什么？

3.Libvirt 哪些底层虚拟化平台？

4.一台虚拟机随着用户需求的改变可能会经历哪些状态？

5.哪个方法实现了确定来宾系统的磁盘映射信息？

OpenStack 基于 Libvirt 的虚拟化平台调度实现----Nova 虚拟机启动源码实现 (2)

<http://www.aboutyun.com/thread-10111-1-1.html>

1.类 Image 下的方法 cache 实现了什么功能？

2.哪个方法实现下载镜像文件？

3.方法 download 由那两部分组成？

OpenStack 基于 Libvirt 的虚拟化平台调度实现----Nova 虚拟机动态迁移源码分析

<http://www.aboutyun.com/thread-10108-1-1.html>

OpenStack 基于 Libvirt 的虚拟化平台调度实现----Nova 虚拟机启动源码实现 (3)

<http://www.aboutyun.com/thread-10110-1-1.html>

1.哪个方法实现了获取元数据？

2.对文件注入代码了解多少？

3.哪个方法实现向磁盘镜像注入不同的文件信息？

OpenStack 基于 Libvirt 的虚拟化平台调度实现----Nova 虚拟机启动源码实现 (4)

<http://www.aboutyun.com/thread-10109-1-1.html>

1._create_domain_and_network 你认为完成了什么？

2.inst_path = libvirt_utils.get_instance_path(instance)语句的作用是什么？

3.domain.createWithFlags(launch_flags)实现什么功能？

openstack nova 源码分析 1-setup 脚本

<http://www.aboutyun.com/thread-10090-1-1.html>

openstack nova 源码分析 2 之 nova-api,nova-compute

<http://www.aboutyun.com/thread-10091-1-1.html>

openstack nova 源码分析 3-nova 目录下的 service.py、driver.py

<http://www.aboutyun.com/thread-10092-1-1.html>

1.nova 下的 **service.py** 的源码主要完成什么任务？

2.driver.py 位于哪个目录下？

openstack nova 源码分析 4-1 -nova/virt/libvirt 目录下的 connection.py

<http://www.aboutyun.com/thread-10094-1-1.html>

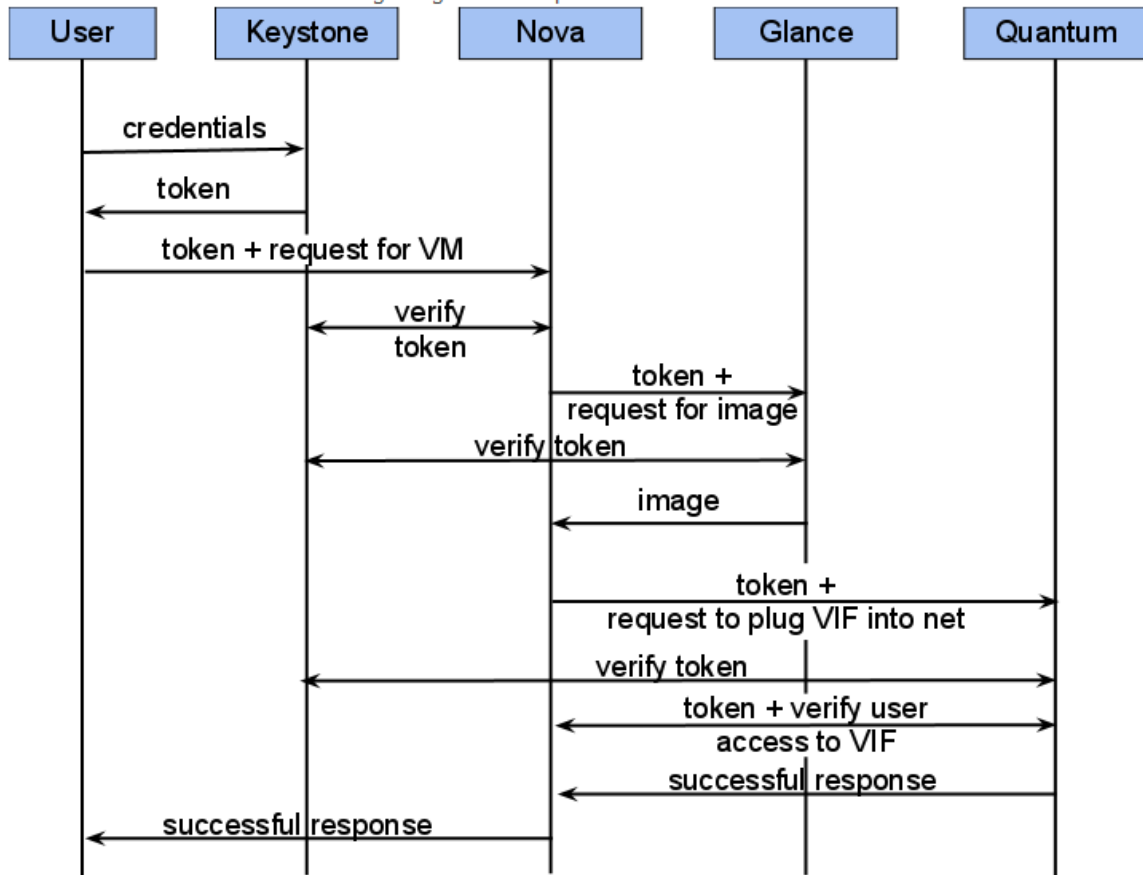
openstack nova 源码分析 4-2 -nova/virt/libvirt 目录下的 connection.py

<http://www.aboutyun.com/thread-10095-1-1.html>

了解认识 keystone

这是提供身份认证和授权的组件。任何系统，身份认证和授权，其实都比较复杂。尤其 Openstack 那么庞大的项目，每个组件都需要使用统一认证和授权。

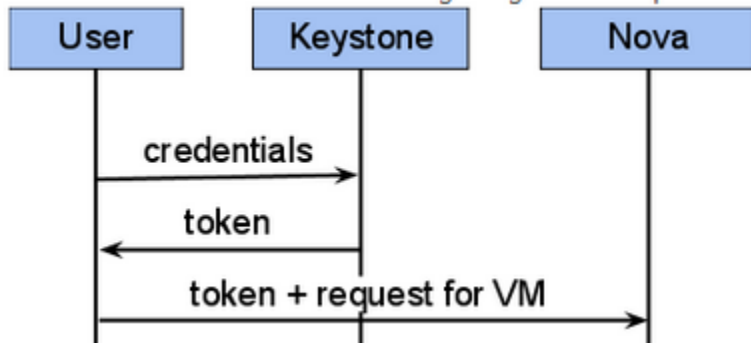
认识 keystone



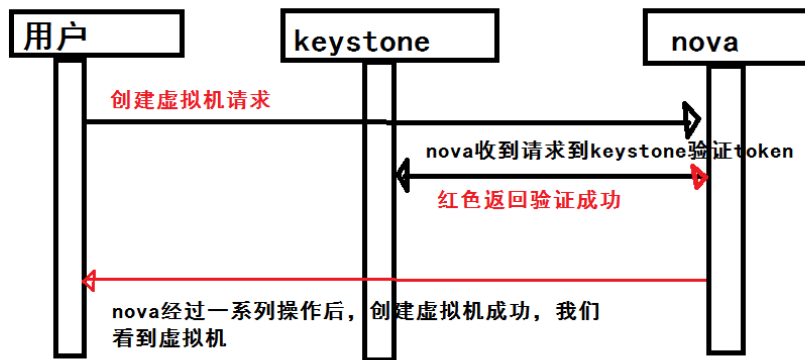
我们或许知道 **keystone** 是用来身份验证的，但是它是如何身份验证的，很多人或许不太清楚。上面的图示，当用户进行操作的时候，用户首先发送用户名和密码给 **Keystone**，（这里的用户名和密码，则是我们配置的环境变量，）然后获取 **token**，**token** 是什么？我们称之为令牌，有了这个令牌在请求资源，那么就畅通无阻了，我们为什么需要令牌，用户名和密码不也一样吗？如果作为一名程序员，我们都知道两个字段的对比与一个字段的对比在性能上是不一样的，何况是云计算组件之间通信是非常频繁的，所以个人认为了身份验证方便，所以产生了令牌（**token**）。

令牌的作用是什么，为什么需要令牌，我们就需要仔细看上图了。
比如

下图用户请求创建虚拟机，截图如下，然后 **nova** 最后经过上图中操作，最后操作成功



我们将上图简化



用户带着 token 到 Nova 去请求虚拟机，nova 这时候需要验证这个 token 是否有效，自己无法判断，所以必须去 keystone 去验证，由于 keystone 记录了由它产生的 token，所以对照一下，就能知道是否有效，如果有效，返回 nova 验证成功。这时候 nova 经过一系列的操作，创建虚拟机，最后创建成功。

keystone 包含的概念

1. User

User 即用户，他们代表可以通过 keystone 进行访问的人或程序。Users 通过认证信息（credentials，如密码、API Keys 等）进行验证。

2. Tenant

Tenant 即租户，它是各个服务中的一些可以访问的资源集合。例如，在 **Nova** 中一个 **tenant** 可以是一些机器，在 **Swift** 和 **Glance** 中一个 **tenant** 可以是一些镜像存储，在 **Quantum** 中一个 **tenant** 可以是一些网络资源。**Users** 默认的总是绑定到某些 **tenant** 上。

3. Role

Role 即角色，**Roles** 代表一组用户可以访问的资源权限，例如 **Nova** 中的虚拟机、**Glance** 中的镜像。**Users** 可以被添加到任意一个全局的 或 租户内的角色中。在全局的 **role** 中，用户的 **role** 权限作用于所有的租户，即可以对所有的租户执行 **role** 规定的权限；在租户内的 **role** 中，用户仅能在当前租户内执行 **role** 规定的权限。

4. Service

Service 即服务，如 **Nova**、**Glance**、**Swift**。根据前三个概念（**User**，**Tenant** 和 **Role**）一个服务可以确认当前用户是否具有访问其资源的权限。但是当 **user** 尝试着访问其租户内的 **service** 时，他必须知道这个 **service** 是否存在以及如何访问这个 **service**，这里通常使用一些不同的名称表示不同的服务。在上文中谈到的 **Role**，实际上也是可以绑定到某个 **service** 的。例如，当 **swift** 需要一个管理员权限的访问进行对象创建时，对于相同的 **role** 我们并不一定也需要对 **nova** 进行管理员权限的访问。为了实现这个目标，我们应该创建两个独立的管理员 **role**，一个绑定到 **swift**，另一个绑定到 **nova**，从而实现对 **swift** 进行管理员权限访问不会影响到 **Nova** 或其他服务。

5. Endpoint

Endpoint，翻译为“端点”，我们可以理解它是一个服务暴露出来的访问点，如果需要访问一个服务，则必须知道他的 **endpoint**。因此，在 **keystone** 中包含一个 **endpoint** 模板（**endpoint template**，在安装 **keystone** 的时候我们可以在 **conf** 文件夹下看到这个文件），这个模板提供了所有存在的服务 **endpoints** 信息。一个 **endpoint template** 包含一个 **URLs** 列表，列表中的每个 **URL** 都对应一个服务实例的访问地址，并且具有 **public**、**private** 和 **admin** 这三种权限。**public url** 可以被全局访问（如 <http://compute.example.com>），**private url** 只能被局域网访问（如 <http://compute.example.local>），**admin url** 被从常规的访问中分离。

很多人觉得比较难以，更多可以参考 [openstack 中 tenant 的作用到底是什么](#)

keystone 命令

keystone 都有哪些操作，**keystone** 可以创建租户、角色、用户，详细参考 [openstack 之 role 篇](#)

[openstack 之 user 篇](#)

[openstack 之 tenant 篇](#)

keystone 源码

源码的阅读，如果感兴趣，可以自己有一个理解，然后与作者进行对比，这样才会对自己的思想有所提高，也算是与作者的思想交流。当然作者也会有错的，所以通过彼此的角，这样达到提高的目的

[openstack][G 版]keystone 源码记录

<http://www.aboutyun.com/thread-10136-1-1.html>

- 1.在 G 版中密码和 token 两个验证方法由哪个文件来实现？
- 2.WSGI middleware 在 keystone 应用中的作用是什么？

Openstack 之 keystone 源代码分析 1--WSGI 接口流程分析

<http://www.aboutyun.com/thread-10137-1-1.html>

- 1.keystone 是怎么通过 WSGI 接口访问其中的服务的？
- 2.你认为 add_routes 作用是什么？

Openstack 之 keystone 源代码分析 2--Controller->Manager->Driver

<http://www.aboutyun.com/thread-10138-1-1.html>

- 1.Driver 在那个配置文件中可配置？
- 2.Manager 怎么调用 conf 下面配置的或者默认的 driver 的？

Openstack 源代码分析之 keystone 服务 (keystone-all)

<http://www.aboutyun.com/thread-10139-1-1.html>

- 1.keystone-all.py 的作用是什么？
- 2.Python 应用程序使用 WSGI (Web Server Gateway Interface) 协议来做什么？

了解认识 Neutron

Neutron 是 OpenStack 核心项目之一，提供云计算环境下的虚拟网络功能。Neutron 的功能日益强大，并在 Horizon 面板中已经集成该模块。作为 Neutron 的核心开发者之一，个人觉得 Neutron 完全代替 Nova Network 模块作为云计算网络管理中心是必然趋势。要使用好 OpenStack, 了解 Neutron 概念及其相应操作就显得格外重要。

Neutron 对于开发人员为什么难以理解：

初学者很难理解 Neutron，这是因为网络不在是实实在在的网线、路由等，都是通过命令来实现的。

例如使用 openvswitch 创建网桥

1. `brctl addbr qbr02`

在比如添加路由 router01

1. `ip netns add router01`

这些都是虚拟化的，也就是说在虚拟机之间，也就是在云中，网络都是虚拟化，所以我们才会觉得难以理解。

同样对于一些概念也比较模糊，比如

固定 IP

私有 IP 地址，用于租户实例间通信

浮动 IP

公共 IP 地址，用于实例与外部或 Internet 的通信

特别是浮动 IP 很多不太理解，一个网卡如果赋予的 ip 能够与外部 Internet 通信，那么它就是浮动 ip。

公共 IP 地址不一定是 Internet 上可路由的地址，也可以是站点内部或局域网的地址

私有地址和公共地址的关系以及必要的路由由 nova-network 来处理，实例不必考虑此问题。

在我们有了一定的理解，在回头看进行一些，我们又会 openstack 有一个新的认识。

下面我们从基础开始认识 Neutron

Neutron 基本概念

网络

在普通人的眼里，网络就是网线和供网线插入的端口，一个盒子会提供这些端口。对于网络工程师来说，网络的盒子指的是交换机和路由器。所以在物理世界中，网络可以简单地被认为包括网线，交换机和路由器。当然，除了物理设备，我们还有软的物件：IP 地址，交换机和路由器的配置和管理软件以及各种网络协议。要管理好一个物理网络需要非常深的网络专业知识和经验。

Neutron 网络目的是（为 **OpenStack** 云更灵活地）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，可以拥有无限多个动态可创建和销毁的虚拟端口。

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着附点。

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。

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。

为什么引入 Quantum？

答案非常简单，**Quantum** 功能更强大，满足更多需求。下面列几条主要功能。

- 提供面向租户的 **API**，以便控制 2 层网络和管理 IP 地址
- 支持插件式网络组件，像 **Open vSwitch**，**Cisco**，**Linux Bridge**，**Nicira NVP** 等等
- 支持位于不同的 2 层网络的 IP 地址重叠
- 支持基本的 3 层转发和多路由器
- 支持隧道技术（**Tunneling**）
- 支持 3 层代理和 **DHCP** 代理的多节点部署，增强了扩展性和可靠性
- 提供负载均衡 **API** （试用版本）

Neutron 主要有以下几部分组成。

Neutron Server：这一部分包含守护进程 **neutron-server** 和各种插件 **neutron-*-plugin**，它们既可以安装在控制节点也可以安装在网络节点。**neutron-server** 提供 **API** 接口，并把对 **API** 的调用请求传给已经配置好的插件进行后续处理。插件需要访问数据库来维护各种配置数据和对对应关系，例如路由器、网络、子网、端口、浮动 **IP**、安全组等等。

插件代理 (Plugin Agent)：虚拟网络上的数据包的处理则是由这些插件代理来完成的。名字为 **neutron-*-agent**。在每个计算节点和网络节点上运行。一般来说你选择了什么插件，就需要选择相应的代理。代理与 **Neutron Server** 及其插件的交互就通过消息队列来支持。

DHCP 代理 (DHCP Agent)：名字为 **neutron-dhcp-agent**，为各个租户网络提供 **DHCP** 服务，部署在网络节点上，各个插件也是使用这一个代理。

3 层代理 (L3 Agent)：名字为 **neutron-l3-agent**，为客户机访问外部网络提供 3 层转发服务。也部署在网络节点上。

下面这张图取自官网，很好的反映了 **Neutron** 内部各部分之间的关系。（**SDN** 服务在这里是额外的外部功能，可以暂时略过。）

上面简单的介绍，下面内容可以参考：

[openstack---Neutron 网络入门](#)

[OpenStack Neutron 解析](#)

[OpenStack 网络组件 Neutron 的研究](#)

[OpenStack Neutron 运行机制解析概要](#)

[OpenStack: 网络组件 Neutron](#)

[Openstack 之 neutron 入门一](#)

[Openstack 之 neutron 入门二](#)

[Openstack 之 neutron 入门三](#)

[openstack 网络,外部网络、内部网络、管理网络作用介绍](#)

[openstack neutron 创建多个外网](#)

[开发人员必读 openstack 网络基础 1:什么是 L2、L3](#)

[开发人员必读 openstack 网络基础 2:交换机、路由器、DHCP](#)

[开发人员必读 openstack 网络基础 3: iptables 详解](#)

[开发人员必读 openstack 网络基础 4:Dnsmasq、网络混杂模式](#)

[开发人员必读 openstack 网络基础 5:网络叠加模式 VLAN、VxLAN、GRE](#)

[开发人员必读 openstack 网络基础 6:什么是 Tap/Tun、网桥](#)

[开发人员必读 openstack 网络基础 7:到底什么是 Open vSwitch](#)

源码分析参考

[Neutron 分析（1）——neutron-server 启动过程分析](#)

[openstack Neutron 分析（2）—— neutron-l3-agent](#)

[openstack Neutron 分析（3）—— neutron-dhcp-agent 源码分析](#)

[openstack Neutron 分析（4）—— neutron-l3-agent 中的 iptables](#)

[openstack Neutron 分析（5）-- neutron openvswitch agent](#)

[OpenStack Neutron DVR L2 Agent 的初步解析（一）](#)

[OpenStack J 版 Neutron-server 服务加载与启动源码分析（一）](#)

[OpenStack J 版 Neutron-server 服务加载与启动源码分析（二）](#)

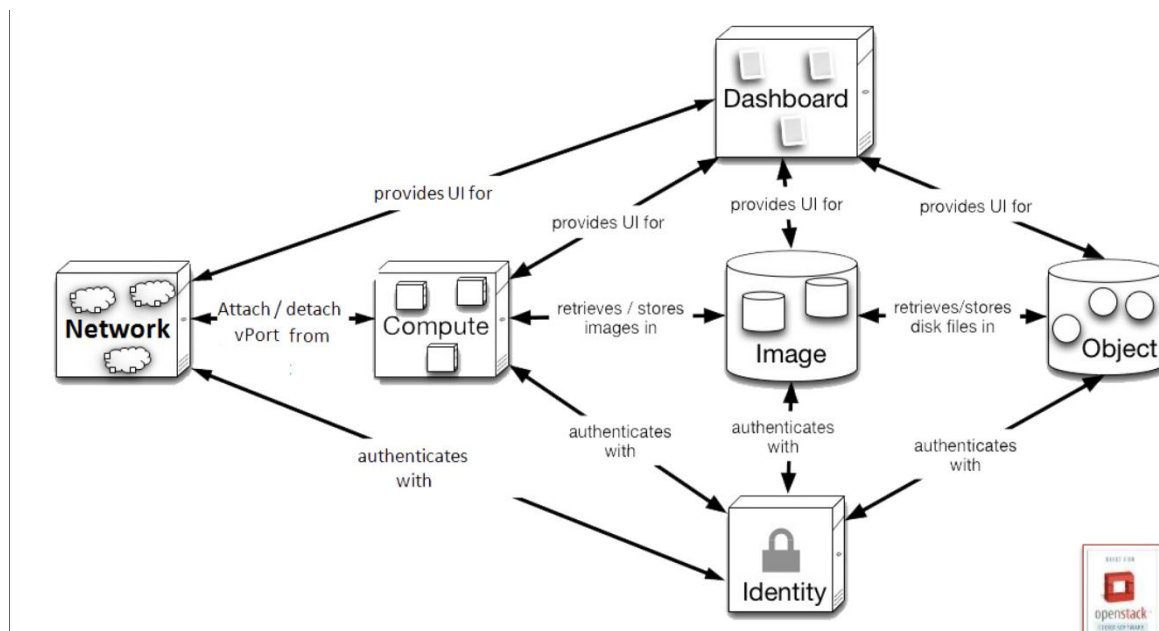
[Openstack Neutron-server 服务加载与启动源码分析（三）](#)

Swift

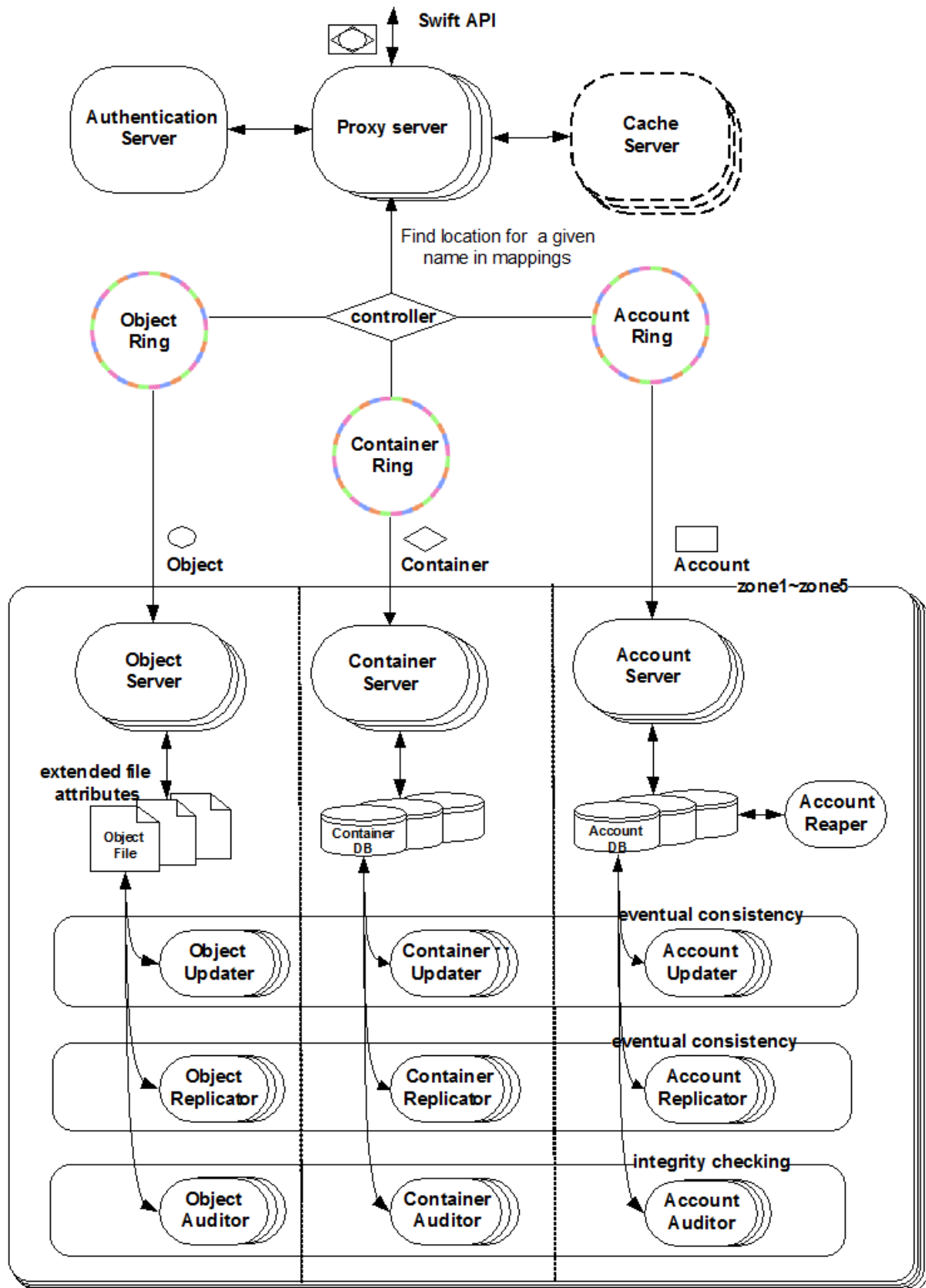
这是对象存储的组件。对于大部分用户来说，**swift** 不是必须的。你只有存储数量到一定级别，而且是非结构化数据才有这样的需求。很多人都问一个相同的问题：是否可以把虚拟机的存储放在 **swift** 上。简单回答：不行。你需要搞明白对象存储是干啥，擅长那些地方，那些是不行的。**swift** 是 Openstack 所有组件了最成熟的,可以在线升级版本,各种版本可以混合在一起,也就是说,1.75 版本的 **swift** 可以和 1.48 的在一个群集里.这个是很难得的.

swift 是什么及在 openstack 的作用

OpenStack Object Storage (Swift) 是开源的，用来创建可扩展的、冗余的、对象存储（引擎）。swift 使用标准化的服务器存储 PB 级可用数据。但它并不是文件系统 (file system)，实时的数据存储系统(real-time data storage system)。swift 看起来更像是一个长期的存储系统 (long term storage system)，为了获得、调用、更新一些静态的永久性的数据。比如说，适合存储一些类型的数据：虚拟机镜像，图片存储，邮件存储，文档的备份。没有“单点”或者主控结点 (master point of control)，swift 看起来具有更强的扩展性、冗余和持久性。



swift 结构



Swift 组件包括：

- 代理服务（Proxy Server）：对外提供对象服务 API，会根据环的信息来查找服务地址并转发用户请求至相应的账户、容器或者对象服务；由于采用无状态的 REST 请求协议，可以进行横向扩展来均衡负载。
- 认证服务（Authentication Server）：验证访问用户的身份信息，并获得一个对象访问令牌（Token），在一定的时间内会一直有效；验证访问令牌的有效性并缓存下来直至过期时间。
- 缓存服务（Cache Server）：缓存的内容包括对象服务令牌，账户和容器的存在信息，但不会缓存对象本身的数据；缓存服务可采用 Memcached 集群，Swift 会使用一致性散列算法来分配缓存地址。
- 账户服务（Account Server）：提供账户元数据和统计信息，并维护所含容器列表的服务，每个账户的信息被存储在一个 SQLite 数据库中。
- 容器服务（Container Server）：提供容器元数据和统计信息，并维护所含对象列表的服务，每个容器的信息也存储在一个 SQLite 数据库中。
- 对象服务（Object Server）：提供对象元数据和内容服务，每个对象的内容会以文件的形式存储在文件系统中，元数据会作为文件属性来存储，建议采用支持扩展属性的 XFS 文件系统。
- 复制服务（Replicator）：会检测本地分区副本和远程副本是否一致，具体是通过对比散列文件和高级水印来完成，发现不一致时会采用推式（Push）更新远程副本，例如对象复制服务会使用远程文件拷贝工具 rsync 来同步；另外一个任务是确保被标记删除的对象从文件系统中移除。
- 更新服务（Updater）：当对象由于高负载的原因而无法立即更新时，任务将会被序列化到在本地文件系统中进行排队，以便服务恢复后进行异步更新；例如成功创建对象后容器服务器没有及时更新对象列表，这个时候容器的更新操作就会进入排队中，更新服务会在系统恢复正常后扫描队列并进行相应的更新处理。
- 审计服务（Auditor）：检查对象，容器和账户的完整性，如果发现比特级的错误，文件将被隔离，并复制其他的副本以覆盖本地损坏的副本；其他类型的错误会被记录到日志中。
- 账户清理服务（Account Reaper）：移除被标记为删除的账户，删除其所包含的所有容器和对象。

上面只是简单的介绍，或许并不能让你真正明白什么是 swift，如果感兴趣可以了解更多内容

[Openstack Swift 原理、架构与 API 介绍](#)

[openstack 入门之 swift 基础一：什么是对象存储](#)

[openstack 入门之 swift 基础二：三种存储类型比较-文件、块、对象存储](#)

[openstack 入门之 swift 基础三：swift 能干什么，不能干什么及相关概念](#)

[对象存储系统 Swift 技术详解：综述与概念（上）](#)

[对象存储系统 Swift 技术详解：综述与概念（下）](#)

[swift 初见](#)

[Ubuntu 12.04 OpenStack Swift 单节点部署指导](#)

[Object Storage \(Swift\)和 Block Storage \(Cinder\)有什么区别？](#)

[hadoop 中 HDFS 与 opesntack 的 swift 有何不同](#)

单独部署

[Swift 能单独使用吗？如何单独部署？](#)

Swift 源码想开发和了解原理的途径之一

Swift 源码分析

[Swift 源码分析----swift-object-auditor（1）](#)

[Swift 源码分析----swift-object-auditor\(2\)](#)

[Swift 源码分析----swift-container-auditor](#)

[Swift 源码分析----swift-account-auditor](#)

[Swift 源码分析----swift-account-audit\(1\)](#)

[Swift 源码分析----swift-account-audit\(2\)](#)

[OpenStack Swift 源码分析（1） ----swift 服务启动源码分析之一](#)

[OpenStack Swift 源码分析（2） ----swift 服务启动源码分析之二](#)

[OpenStack Swift 源码分析（3） ----swift 服务启动源码分析之三](#)

[OpenStack Swift 源码分析（4） ----swift-ring-builder 源代码解析之一](#)

[OpenStack Swift 源码分析 \(5\) ----swift-ring-builder 源代码解析之二](#)

[Swift 源码分析----swift-object-updater](#)

[Swift 源码分析----swift-object-info](#)

[Swift 源码分析----swift-object-replicator\(1\)](#)

[Swift 源码分析----swift-object-replicator\(2\)](#)

[Swift 源码分析----swift-proxy 实现请求 req 的转发](#)

[Swift 源码分析----swift-container-info](#)

[Swift 源码分析----swift-proxy 与 swift-account](#)

[Swift 源码分析----swift-account-reaper\(1\)](#)

[Swift 源码分析----swift-account-reaper\(2\)](#)

[Swift 源码分析----swift-proxy 与 swift-container](#)

[Swift 源码分析----swift-account-replicator](#)

[Swift 源码分析----swift-container-replicator](#)

[Swift 源码分析----swift-proxy 与 swift-object](#)

[Swift 源码分析----swift-container-updater](#)

本文链接

<http://www.aboutyun.com/thread-10124-1-1.html>

由于时间有限，后面将会继续出完整篇，可以随时关注 about 云（www.aboutyun.com）

注释：资源来自网络，参考整理，外加个人观点，共享给网友