

**API 前端服务**

每个 OpenStack 组件可能包含若干子服务，其中必定有一个 API 服务负责接收客户请求。

以 Nova 为例，nova-api 作为 Nova 组件对外的唯一窗口，向客户暴露 Nova 能够提供的功能。 当客户需要执行虚机相关的操作，能且只能向 nova-api 发送 REST 请求。 这里的客户包括终端用户、命令行和 OpenStack 其他组件。

设计 API 前端服务的好处在于： 1. 对外提供统一接口，隐藏实现细节 2. API 提供 REST 标准调用服务，便于与第三方系统集成 3. 可以通过运行多个 API 服务实例轻松实现 API 的高可用，比如运行多个 nova-api 进程

**Scheduler 调度服务**

对于某项操作，如果有多个实体都能够完成任务，那么通常会有一个 scheduler 负责从这些实体中挑选出一个最合适的来执行操作。

在前面的例子中，Nova 有多个计算节点。 当需要创建虚机时，nova-scheduler 会根据计算节点当时的资源使用情况选择一个最合适的计算节点来运行虚机。

调度服务就好比是一个开发团队中的项目经理，当接到新的开发任务时，项目经理会评估任务的难度，考察团队成员目前的工作负荷和技能水平，然后将任务分配给最合适的开发人员。

除了 Nova，块服务组件 Cinder 也有 scheduler 子服务，后面我们会详细讨论。

**Worker 工作服务**

调度服务只管分配任务，真正执行任务的是 Worker 工作服务。

在 Nova 中，这个 Worker 就是 nova-compute 了。 将 Scheduler 和 Worker 从职能上进行划分使得 OpenStack 非常容易扩展：

1. 当计算资源不够了无法创建虚机时，可以增加计算节点（增加 Worker）
2. 当客户的请求量太大调度不过来时，可以增加 Scheduler

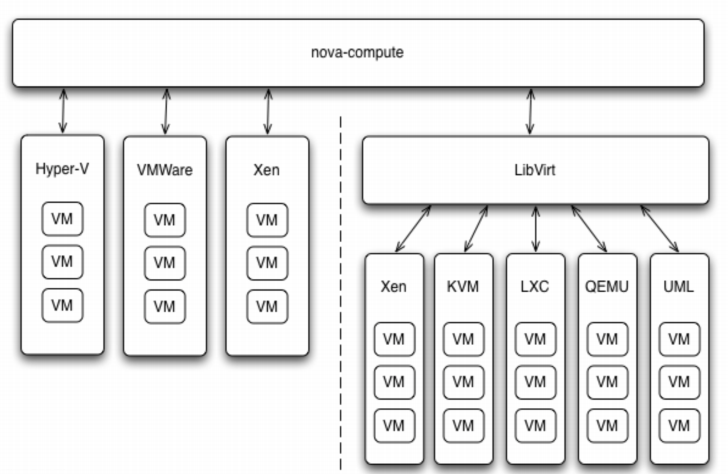
**Driver 框架**

OpenStack 作为开放的 Infrastracture as a Service 云操作系统，支持业界各种优秀的技术。 这些技术可能是开源免费的，也可能是商业收费的。 这种开放的架构使得 OpenStack 能够在技术上保持先进性，具有很强的竞争力，同时又不会造成厂商锁定（Lock-in）。

那 OpenStack 的这种开放性体现在哪里呢？ 一个重要的方面就是采用基于 Driver 的框架。

以 Nova 为例，OpenStack 的计算节点支持多种 Hypervisor。 包括 KVM, Hyper-V, VMWare, Xen, Docker, LXC 等。

Nova-compute 为这些 Hypervisor 定义了统一的接口，hypervisor 只需要实现这些接口，就可以 driver 的形式即插即用到 OpenStack 中。 下面是 nova driver 的架构示意图



在 nova-compute 的配置文件 /etc/nova/nova.conf 中由 compute\_driver 配置项指定该计算节点使用哪种 Hypervisor 的 driver

http://7xo6kd.com1.z0.glb.clouddn.com/upload-ueditor-image-20160424-1461498587779038837.jpg

在我们的环境中因为是 KVM，所以配置的是 Libvirt 的 driver。

不知大家是否还记得我们在学习 Glance 时谈到：

OpenStack 支持多种 backend 来存放 image。

可以是本地文件系统，Cinder，Ceph，Swift 等。

其实这也是一个 driver 架构。

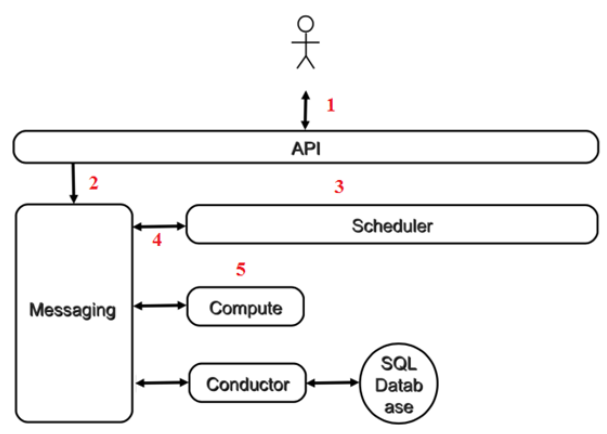
只要符合 Glance 定义的规范，新的存储方式可以很方便的加入到 backend 支持列表中。

再后面 Cinder 和 Neutron 中我们还会看到 driver 框架的应用。

**Messaging 服务**

在前面创建虚机的流程示意图中，我们看到 nova-\* 子服务之间的调用严重依赖 Messaging。

Messaging 是 nova-\* 子服务交互的中枢。



以前没接触过分布式系统的同学可能会不太理解为什么不让 API 直接调用Scheduler，或是让Scheuler 直接调用 Compute，而是非要通过 Messaging 进行中转。

这里做一些解释。

程序之间的调用通常分两种：同步调用和异步调用。

**同步调用**

API 直接调用 Scheduler 的接口就是同步调用。

其特点是 API 发出请求后需要一直等待，直到 Scheduler 完成对 Compute 的调度，将结果返回给 API 后 API 才能够继续做后面的工作。

**异步调用**

API 通过 Messaging 间接调用 Scheduler 就是异步调用。

其特点是 API 发出请求后不需要等待，直接返回，继续做后面的工作。

Scheduler 从 Messaging 接收到请求后执行调度操作，完成后将结果也通过 Messaging 发送给 API。

在 OpenStack 这类分布式系统中，通常采用异步调用的方式，其好处是：

1. 解耦各子服务
2. 子服务不需要知道其他服务在哪里运行，只需要发送消息给 Messaging 就能完成调用。
3. 提高性能
4. 异步调用使得调用者无需等待结果返回。这样可以继续执行更多的工作，提高系统总的吞吐量。
5. 提高伸缩性
6. 子服务可以根据需要进行扩展，启动更多的实例处理更多的请求，在提高可用性的同时也提高了整个系统的伸缩性。而且这种变化不会影响到其他子服务，也就是说变化对别人是透明的。

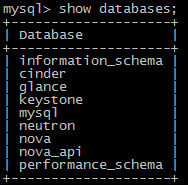
在后面各章节，我们都能看到 Messaging 的应用。

**Database**

OpenStack 各组件都需要维护自己的状态信息。

比如 Nova 中有虚机的规格、状态，这些信息都是在数据库中维护的。

每个 OpenStack 组件在 MySQL 中有自己的数据库。



**小结**

Nova 是 OpenStack 中最重要的组件，也是很典型的组件。

Nova 充分体现了 OpenStack 的设计思路。

理解了这种思路，再来学习 OpenStack 的其他组件就能够举一反三，清晰容易很多。

我们在后面 Cinder 和 Neutron 的学习中还会回顾这些设计思路。