ZStack 技术白皮书精选

ZStack 可拓展性的秘密武器 2：

无状态的服务

扫一扫二维码，获取更多技术干货吧

版权声明

摘要

大道至简·极速部署，ZStack 致力于产品化私有云和混合云。

ZStack 是新一代创新开源的云计算 IaaS 软件，由英特尔、微软、CloudStack 等世界上最早一批虚拟化工程师创建，拥有 KVM、Xen、Hyper-V 等成熟的技术背景。

ZStack 创新提出了云计算 4S 理念，即 Simple （简单）、Strong （健壮）、Smart （智能）、Scalable （弹性），通过全异步架构，无状态服务架构，无锁架构等核心技术，完美解决云计算执行效率低，系统不稳定，不能支撑高并发等问题，实现 HA 和轻量化管理。

ZStack 发起并维护着国内最大的自主开源 IaaS 社区——zstack.io，吸引了 6000 多名社区用户，对外公开的 API 超过 1000 个。基于这 1000 多个 API，用户可以自由组装出自己的私有云、混合云，甚至利用 ZStack 搭建公有云对外提供服务。

ZStack 拥有充足的知识产权储备，积极申报多项软著和专利，参与业内标准、白皮书的撰写，入选云计算行业方案目录，还通过了工信部云服务能力认证和信通院可信云认证。ZStack 面向企业用户提供基于 IaaS 的私有云和混合云，是业内唯一一家实现产品化，并领先业内首家推出同时打通数据面和控制面无缝混合云的云服务商。选择 ZStack，用户可以官网直接下载、1 台 PC 也可上云、30 分钟完成从裸机的安装部署。

目前已有 1000 多家企业用户选择了 ZStack 云平台。

ZSTACK—可拓展性秘密武器 2：无状态的服务

每一个 ZStack 服务都是无状态的，简单的开启一个富余的服务实例然后使之负载均衡，就能实现服务的高可用和可横向拓展；此外，ZStack 把所有服务封装进一个称为管理节点的进程中，使得部署和管理服务变得尤其简单。

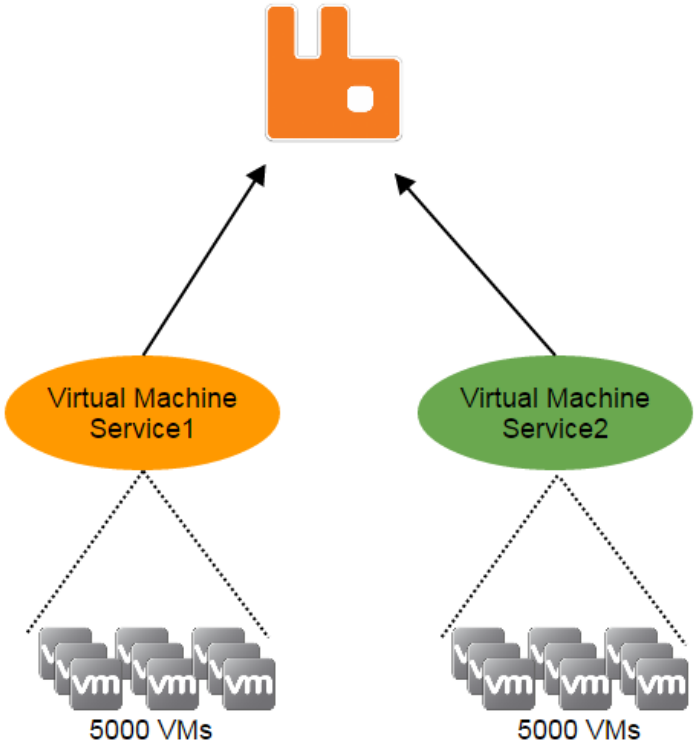
动机

在“ZStack 的可拓展性秘密武器 1：全异步架构”中，我们论述了异步的架构使得单一的 ZStack 管理节点足以承担大多数云的负载量；然而当用户想要去创建一个高可用的生产环境或处理非常大的并发工作负载，一个管理节点是不够的。解决方案是建立一个负载均衡的分布式系统，这种通过添加新节点来拓展整个系统的能力的方法被称为横向拓展。

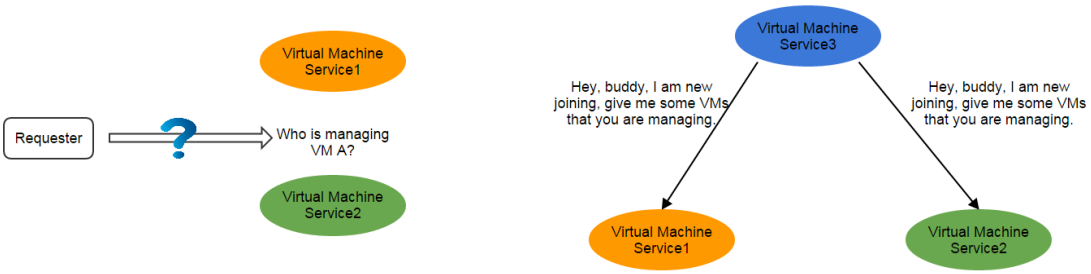
问题

设计一个分布式系统不是一件简单的事情。一个分布式系统，特别是一个有状态的系统，必须处理一致性（consistency）、可用性（availability）和分区容忍性（partition tolerance）（CAP 理论）的问题，每一个问题都是非常复杂的。与之相反，一个无状态的分布式系统一定程度上降低了复杂度。第一，因为节点不用分享状态，整个系统的一致性是可以保证的。第二，因为节点都是相似的，对于分区问题系统通常是可以容忍的。因此，通常把一个分布式系统设计为无状态的而不是有状态的。但是设计一个无状态的分布式系统通常比设计一个有状态的分布式系统难得多。利用消息代理和数据库的优点，ZStack 建立了一个包含各种无状态服务的无状态分布式系统。

使整个系统无状态的基础是无状态的服务，在讨论什么是无状态的服务之前，我们首先理解什么是“状态”。在 ZStack 中，主机、虚拟机、镜像和用户等资源是被一个个服务管理的。当整个系统的服务实例不止一个的时候，资源会被分发到不同的服务实例中。假设有 10000 台虚拟机和两个虚拟机服务实例，理想状态下每个实例会管理 5000 台虚拟机。



因为有两个服务实例，在向一个虚拟机发出请求前，请求者必须知道哪个实例管理哪个虚拟机，否则，他将不知道向哪个实例发出请求。类似“**哪个服务实例管理哪个资源**”的信息就是我们所说的状态。如果一个服务是有状态的，每个服务维护自己的状态。请求者必须可以获取到当前的状态信息。当服务实例的数量改变的时候，服务需要去改变状态。

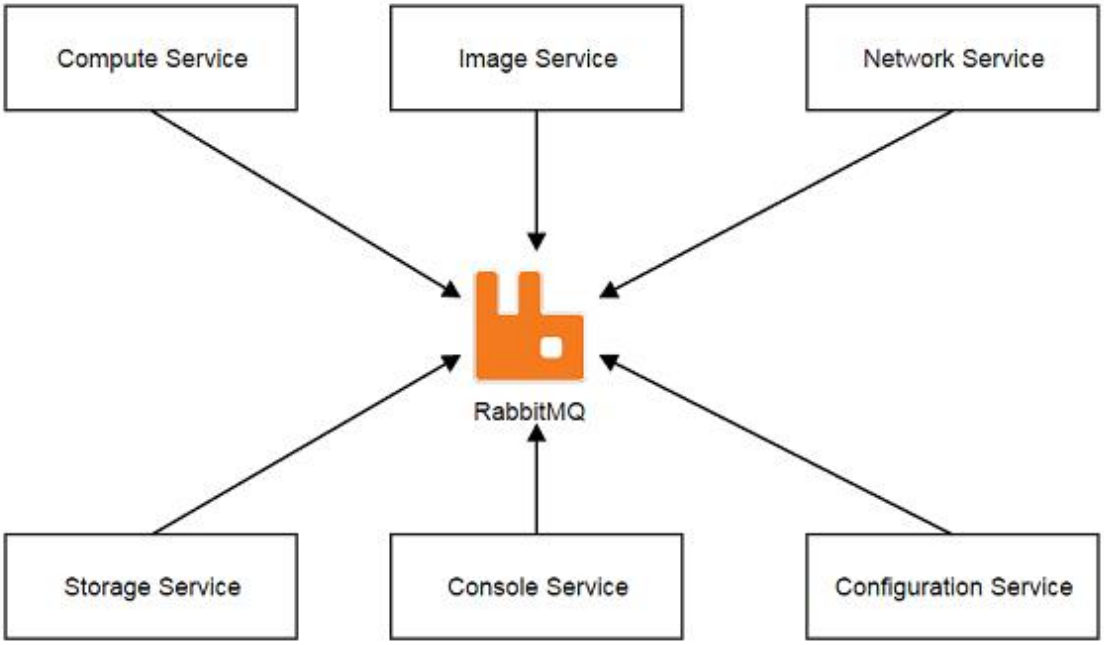


状态的改变是危险且易错的，这通常限制了整个系统的可拓展性。为了使整个系统的可靠性和横向拓展性增强，把状态和服务分离开，使得服务无状态是比较理想的解决办法（参考 Service Statelessness Principle）。**无状态的服务使请求者不需要询问向哪里发送请求，当新添一个新的服务实例或者删除一个旧的服务实例的时候，服务之间也不用交换状态。**

备注：在以下文档中，为了简便，“服务”和“服务实例”是可以互换的。

服务和管理节点

通过中心消息代理-- RabbitMQ 彼此通信的服务，在 ZStack 中是一等公民。



与典型的 microservice 架构不同，典型的 microservice 架构中每个服务通常运行在不同的进程或者不同的机器上，ZStack 则把所有服务封装在一个被称为管理节点的进程中。文章“进程内的 microservice 架构”解释了我们这么做的原因。

每个管理节点都是一个功能齐全的 ZStack 软件。因为包含的服务是无状态的，管理节点不共享任何状态，但仍然需要维护和其他节点的心跳记录，和一个一致性哈希环，我们接下来将详细介绍哈希环。心跳用来监视管理节点是否正常运行，一旦一个管理节点停止更新自己的心跳一段时间后，其他的管理节点将会驱逐它然后接管他所管理的资源。

无状态的服务

针对 ZStack 的业务逻辑，实现无状态服务的核心技术，是一致性哈希算法。当系统启动的时候，每一个管理节点将被分配一个 version 4 UUID（管理节点 UUID），这个 UUID 将和服

务名称拼在一起在消息代理上注册一个服务队列。例如，一个管理节点可能有类似下面的服务队列：

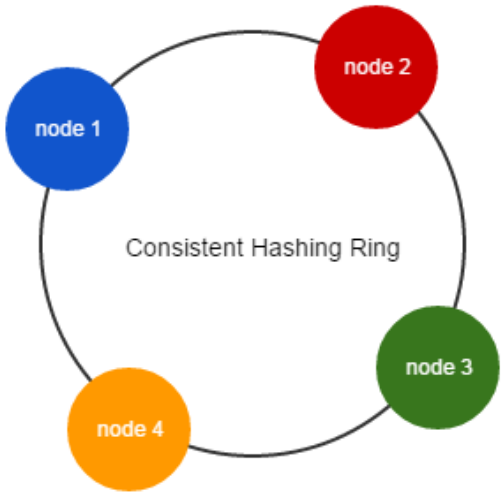
```
zstack.message.ansible.3694776ab31a45709259254a018913ca

zstack.message.api.portal

zstack.message.applianceVm.3694776ab31a45709259254a018913ca

zstack.message.cloudbus.3694776ab31a45709259254a018913ca

zstack.message.cluster.3694776ab31a45709259254a018913ca

zstack.message.configuration.3694776ab31a45709259254a018913ca

zstack.message.console.3694776ab31a45709259254a018913ca

zstack.message.eip.3694776ab31a45709259254a018913ca

zstack.message.globalConfig.3694776ab31a45709259254a018913ca

zstack.message.host.3694776ab31a45709259254a018913ca

zstack.message.host.allocator.3694776ab31a45709259254a018913ca

zstack.message.identity.3694776ab31a45709259254a018913ca

zstack.message.image.3694776ab31a45709259254a018913ca

zstack.message.managementNode.3694776ab31a45709259254a018913ca

zstack.message.network.l2.3694776ab31a45709259254a018913ca

zstack.message.network.l2.vlan.3694776ab31a45709259254a018913ca

zstack.message.network.l3.3694776ab31a45709259254a018913ca

zstack.message.network.service.3694776ab31a45709259254a018913ca

zstack.message.portForwarding.3694776ab31a45709259254a018913ca

zstack.message.query.3694776ab31a45709259254a018913ca
```

```
zstack.message.securityGroup.3694776ab31a45709259254a018913ca

zstack.message.snapshot.volume.3694776ab31a45709259254a018913ca

zstack.message.storage.backup.3694776ab31a45709259254a018913ca
```

备注：你应该已经注意到所有的队列都是以一个相同的管理节点的 UUID 结尾的。

主机，磁盘，虚拟机等资源也有特定的 UUID。和资源相关的消息通常在服务之间传递，在发送一个消息之前，发送者必须基于资源的 UUID 选择一个接收服务，一致性哈希算法这时候就发挥作用了。

一致性哈希是一种较特别的哈希，当一个哈希表的大小发生变化时，只有一部分键需要被重新映射。深入了解一致性哈希，请阅读 http://www.tom-e-white.com/2007/11/consistent-hashing.html，在 ZStack 中，管理节点组成了一个一致性哈希如下：

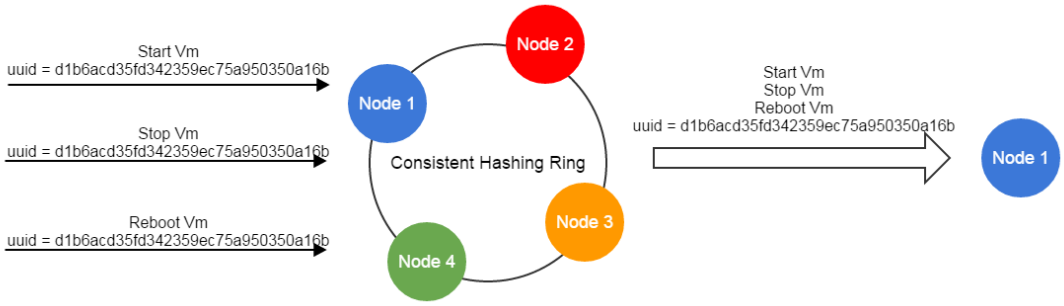


每一个管理节点维护了一份包含系统中所有管理节点的 UUID 的环拷贝，当一个管理节点添加或删除的时候，一个生命周期事件将通过消息代理广播到其他的管理节点，这将导致这些节点拓展或者收缩环去描述当前系统的状态。当发送一条消息时，发送者将使用资源的 UUID 哈希得出目标管理节点的 UUID。例如，当 VM 的 UUID 是 932763162d054c04adaab6ab498c9139 时发送一个 StartVmInstanceMsg，伪代码如下所示：

```
msg = new StartVmInstanceMsg();
```

```
destinationManagementNodeUUID =

consistent_hashing_algorithm("932763162d054c04adaab6ab498c9139");

msg.setServiceId("vmInstance." + destinationManagementNodeUUID);

cloudBus.send(msg)
```

有了哈希环，资源 UUID 相同的消息将被映射到特定管理节点的相同服务中，这点是 ZStack 的无锁架构的基础（参考 ZStack 的可拓展性秘密武器 3：无锁架构）。



当环收缩或者拓展的时候，因为哈希环的固有特性，仅有小部分节点将被影响。

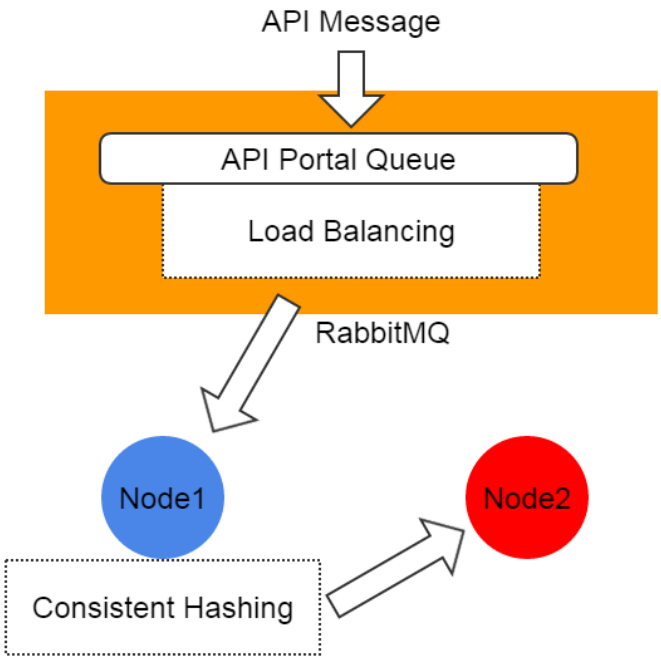
因为使用一致性哈希环，发送者不需要知道哪个服务实例将处理这条消息，因为服务实例将被哈希计算出来。服务也不用维护、交换他们管理的资源信息，并且因为选择正确的服务实例可以由哈希环完成，服务只需要单纯的处理消息。因此，服务变得极其简单且无状态。

除了包含资源 UUID 的消息（例如 StartVmInstanceMsg, DownloadImageMsg）以外，有一种不包含资源 UUID 的消息，这种消息通常是创造性的消息（例如 CreateVolumeMsg）和不进行资源操作的消息（例如 AllocateHostMsg），因为这些消息可以被发送到任意管理节点的服务中，他们就被发送到本地的管理节点，因为发送者和接收者在同一个节点上，接收者在发送者发送消息时一定是可用的。

对于 API 消息（如 APIStartVmInstanceMsg），有一个特别的处理方法是他们经常和一个重要的服务 ID api.portal 绑在一起发送。在消息代理中，一个称为 zstack.message.api.portal 的全局的队列被所有管理节点的 API 服务共享，带有 api.portal 的消息将通过一致性哈希环把消息映射到正确的服务中，从而实现负载均衡。通过上面这种方式，ZStack 隐藏了 API 客户端消息选路的实现，减少了 ZStack API 客户端代码。

```
msg = new APICreateVmInstanceMsg()
```

```
msg.setServiceId("api.portal")

cloudBus.send(msg)
```



总结

本文演示了 ZStack 是如何通过构建一个无状态的分布式系统来进行横向拓展的。因为管理节点共享的信息非常少，建立一个庞大的拥有几十或上百个管理节点的集群是非常容易的。然而，在现实中，对于私有云，两个管理节点足够满足高可用性和可拓展性的需求。对于公有云，管理者可以依据负载量大量创建管理节点。因为异步架构和无状态架构，ZStack 能够处理现有的 IaaS 软件处理不了的非常大的并发任务。