ZStack 技术白皮书精选

ZStack--进程内的微服务架构

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。

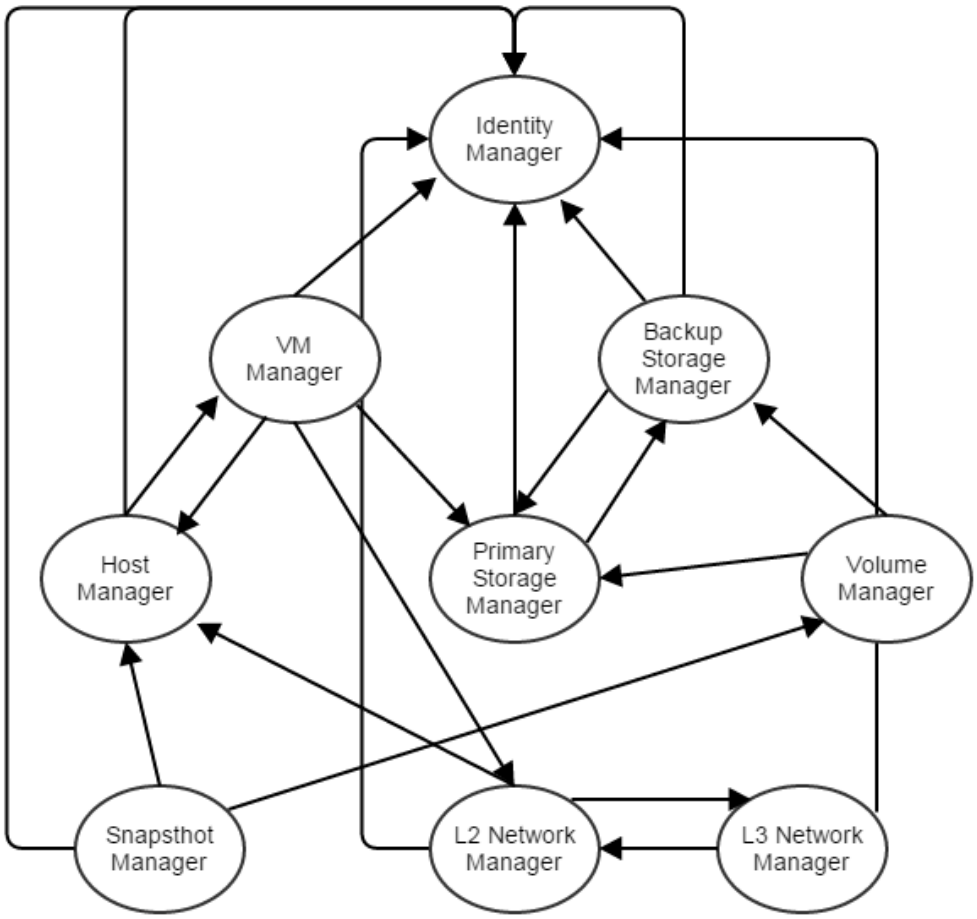
目前已有 1000 多家企业用户选择了 ZStack 云平台。

ZSTACK--进程内的微服务架构

为了应对诸如惊人的操作开销、重复的努力、可测试性等微服务通常面临的挑战，以及获得诸如代码解耦，易于横向扩展等微服务带来的好处，ZStack 将所有服务包含在单个进程中，称为管理节点，构建一个进程内的微服务架构。

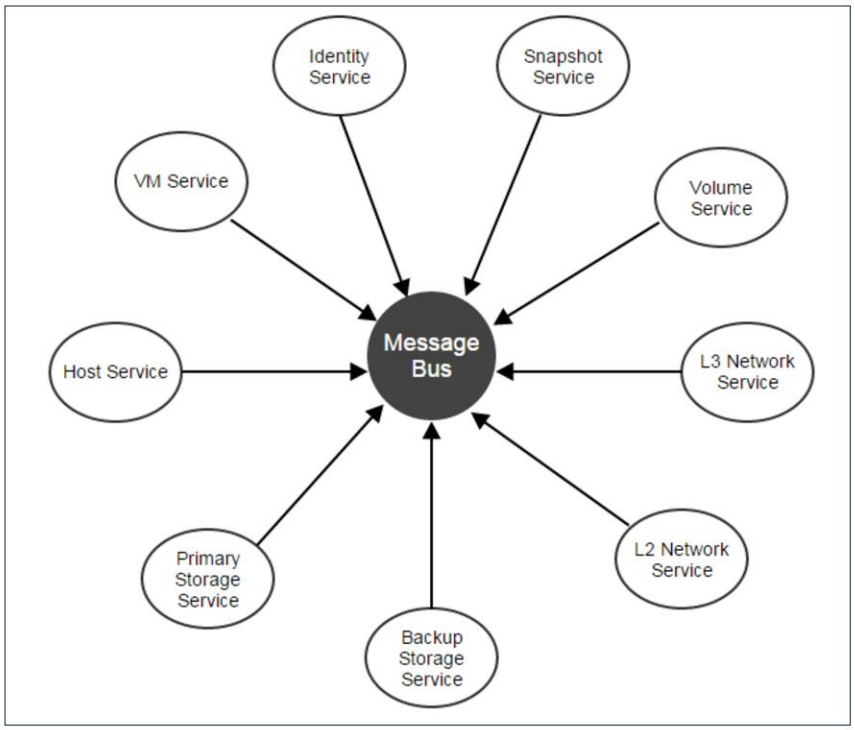
动机

构建一个 IaaS 软件是很难的，这是一个已经从市场上现存的 IaaS 软件获得的教训。作为一个集成软件，IaaS 软件通常需要去管理复杂的各种各样的子系统（如：虚拟机管理器 hypervisor，存储，网络，身份验证等）并且需要组织协调多个子系统间的交互。例如，创建虚拟机操作将涉及到虚拟机管理模块，存储模块，网络模块的合作。由于大多数 IaaS 软件通常对架构考虑不够全面就急于开始解决一个具体问题，它们的实现通常会演变成：



The organic growth of monolithic IaaS software

随着一个软件的不断成长，这个铁板一块的架构（monolithic architecture）将最终变为一团乱麻，以至于没有人可以修改这个系统的代码，除非把整个系统从头构建。这种铁板一块的编程问题是微服务可以介入的完美场合。通过划分整个系统的功能为一个个小的、专一的、独立的服务，并定义服务之间交互的规则，微服务可以帮助转换一个复杂笨重的软件，从紧耦合的、网状拓扑架构，变成一个松耦合的、星状拓扑的架构。



因为服务在微服务中是编译独立的，添加或者删除服务将不会影响整个系统的架构（当然，移除某些服务会导致功能的缺失）。

微服务远比我们已经讨论的内容更多：*微服务的确有很多引入注目的优点，尤其是在一个的开发运维 流程（DevOps process）中，当涉及到一个大机构的很多团队时。我们不打算讨论微服务的所有支持和反对意见，我们确定你可以在网上找到大量的相关文章，我们主要介绍一些我们认为对 IaaS 软件影响深远的特性。*

问题

虽然微服务可以解耦合架构，但这是有代价的。阅读 Microservices - Not A Free Lunch!和 Failing at Microservices 会对这句话有更深的理解。在这里，我们重点强调一些我们认为对 IaaS 软件影响重大的事情。

1. 难以定义服务的边界和重复做功

创建 Microservices 架构的挑战之一是决定应该把哪一部分的代码定义为服务，一些是非常明显的，比如说，处理主机部分的逻辑代码可以被定义为一个服务。然而，管理数据库交

互的代码非常难以决定应不应该被定义为服务。数据库服务可以使得整个架构更加清晰明了，但是这样会导致严重的性能下降。通常，类似于这样的代码可以被定义为库，库可以被各个服务调用。鉴于所有服务一般在互相隔离的目录下开发和维护，创建一个给不同的单一的软件提供接口的虚拟的库，要求开发者必须具有良好的和各个不同组的开发者沟通协调的能力。综上，服务很容易重复造轮子和导致不必要的重复做功。

2. 软件难以部署、升级和维护

服务，尤其是那些分散在不同进程和机器上的，是难以部署和升级的。用户通常必须去花费几天甚至几周去部署一个完整的可运行的系统，并害怕升级一个已经构建好的稳定的系统。尽管一些类似 puppet 的配置管理软件一定程度上缓解了这个问题，用户依旧需要克服陡峭的学习曲线去掌握这些配置工具，仅仅是为了部署或者升级一个软件。管理一个云是非常困难的，努力不应该被浪费在管理这些原本应该使生活更轻松的软件上。

服务的数量确实很重要：*IaaS 软件通常有许许多多的服务。拿著名的 openstack 举个例子，为了完成一个基础的安装你将需要：Nova, Cinder, Neutron, Horizon, Keystone, Glance。除了 nova 是在每台主机都需要部署的，如果你想要 4 个实例（instances），并且每个服务运行在不同机器上，你需要去操纵 20 台服务器。虽然这种人造的案例将不太可能真实地发生，它依旧揭示了管理相互隔离的服务的挑战。*

3. 零散的配置

运行在不同服务器上的服务，分别维护着它们散乱在系统各个角落的配置副本。在系统范围更新配置的操作通常由临时特定的脚本完成，这会导致由不一致的配置产生的令人费解的失败。

4. 额外的监控努力

为了跟踪系统的健康状况，用户必须付出额外的努力去监控每一个服务实例。这些监控软件，要么由第三方工具搭建，要么服务自身维护，仍然受到和微服务面临的问题所类似的问题的困扰，因为它们仍然是以分布式的方式工作的软件。

5. 插件杀手

插件这个词在微服务的世界中很少被听到，因为每个服务都是运行在不同进程中一个很小的功能单元（function unit）；传统的插件模式（参考 The Versatile Plugin System）目标是把不同的功能单元相互挂在一起，这在微服务看来是不可能的，甚至是反设计模式的。然而，对于一些很自然的，要在功能单元间强加紧密依赖的业务逻辑，微服务可能会让事情变得非常糟糕，因为缺乏插件支持，修改业务逻辑可能引发一连串服务的修改。

所有的服务都在一个进程

意识到上述的所有问题，以及这么一个事实，即一个可以正常工作的 IaaS 软件必须和所有的编排服务一起运行之后，ZStack 把所有服务封装在单一进程中，称之为管理节点。除去一些微服务已经带来的如解耦架构的优点外，进程内的微服务还给了我们很多额外的好处：

1. 简洁的依赖

因为所有服务都运行在同一进程内，软件只需要一份支持软件（如：database library, message library）的拷贝；升级或改变支持库跟我们对一个单独的二进制应用程序所做的一样简单。

2. 高可用，负载均衡和监控

服务可以专注于它们的业务逻辑，而不受各种来自于高可用、负载均衡、监控的干扰，这一切只由管理节点关心；更进一步，状态可以从服务中分离以创建无状态服务，详见 ZStack's Scalability Secrets Part 2: Stateless Services。

3. 中心化的配置

由于在一个进程中，所有的服务共享一份配置文件——zstack.properties；用户不需要去管理各种各样的分散在不同机器上的配置文件。

4. 易于部署、升级、维护和横向扩展

部署，升级或者维护一个单一的管理节点跟部署升级一个单一的应用程序一样容易。横向扩展服务只需要简单的增加管理节点。

5. 允许插件

因为运行在一个单一的进程中，插件可以很容易地被创建，和给传统的单进程应用程序添加插件一样。

进程内的微服务并不是一个新发明：*早在 90 年代，微软在 COM（Component Object Model）中把 server 定义为远程、本地和进程内三种。这些进程内的 server 是一些 DLLs，被应用程序在同一进程空间内加载，属于进程内的微服务。Peter Kriens 在四年前就声称已经定义了一种总是在同一进程内通信的服务，OSGi µservices。*

服务样例

在微服务中，一个服务通常是一个可重复的业务活动的逻辑表示，是无关联的、松耦合的、自包含的，而且对服务的消费者而言是一个“黑盒子”。简单来说，一个传统的微服务通常只关心特定的业务逻辑，有自己的 API 和配置方法，并能像一个独立的应用程序一样运行。尽管 ZStack 的服务共享同一块进程空间，它们拥有这些特点中的绝大多数。ZStack 很大程度上是一个使用强类型语言 java 编写的项目，但是在各个编排服务之间没有编译依赖性，例如：计算服务（包含 VM 服务、主机服务、区域服务、集群服务）并不依赖于存储服务（包含磁盘服务、基础存储服务、备份存储服务、磁盘快照服务等），虽然这些服务在业务流程中是紧密耦合的。

在源代码中，一个 ZStack 的服务并不比一个作为一个独立的 jar 文件构建的 maven 模块多任何东西。每一个服务可以定义自己的 APIs、错误码、全局配置，全局属性和系统标签。例如 KVM 的主机服务拥有自己的 APIs（如下所示）和各种各样的允许用户自己定义配置的方式。

```
?xml version="1.0" encoding="UTF-8"?>
```

```
<service xmlns="http://zstack.org/schema/zstack">
    <id>host</id>
    <message>
        <name>org.zstack.kvm.APIAddKVMHostMsg</name>
        <interceptor>HostApiInterceptor</interceptor>
        <interceptor>KVMApiInterceptor</interceptor>
    </message>
</service>
```

通过全局配置来配置

备注：*这里只简单展示一小部分，用户可以使用 API 去更新/获取全局配置，在这里展示一下全局配置的视图。*

```
<?xml version="1.0" encoding="UTF-8"?>
      <globalConfig xmlns="http://zstack.org/schema/zstack">
   <config>
       <category>kvm</category>
       <name>vm.migrationQuantity</name>
       <description>A value that defines how many vm can be migrated in parallel when putting a KVM
host into maintenance mode.（当一个 KVM 主机变成维护模式的时候，这里的值定义了可以被并发迁移的虚拟机的数量）
</description>
       <defaultValue>2</defaultValue>
       <type>java.lang.Integer</type>
```

```
    </config>



    <config>

        <category>kvm</category>

        <name>reservedMemory</name>

        <description>The memory capacity reserved on all KVM hosts. ZStack KVM agent is a python web server that needs some memory capacity to run. this value reserves a portion of memory for the agent as well as other host applications. The value can be overridden by system tag on individual host, cluster and zone level（所有的 KVM 主机预留的内存容量。ZStack 中的 KVM 代理运行时是一个需要一部分内存容量去运行的 python 的 web 服务器，这个值为代理和其他主机应用程序预留了一部分内存，在单一主机上的、集群上的、区域上的系统标签可以覆盖这个值）</description>

        <defaultValue>512M</defaultValue>

    </config>

      </globalConfig>
```

通过全局属性配置

备注：*以下代码对应 zstack.properties 文件夹中相应的属性*

```
@GlobalPropertyDefinition

public class KVMGlobalProperty {

    @GlobalProperty(name="KvmAgent.agentPackageName", defaultValue = "kvmagent-0.6.tar.gz")

    public static String AGENT_PACKAGE_NAME;

    @GlobalProperty(name="KvmAgent.agentUrlRootPath", defaultValue = "")

    public static String AGENT_URL_ROOT_PATH;
```

```
    @GlobalProperty(name="KvmAgent.agentUrlScheme", defaultValue = "http")

    public static String AGENT_URL_SCHEME;

}
```

通过系统标签配置

备注：*以下代码对应数据库中相应的系统标签。*

```
      @TagDefinition

      public class KVMSystemTags {

    public static final String QEMU_IMG_VERSION_TOKEN = "version";

    public static PatternedSystemTag QEMU_IMG_VERSION = new PatternedSystemTag(String.format("qemu-
img::version::%s", QEMU_IMG_VERSION_TOKEN), HostVO.class);



    public static final String LIBVIRT_VERSION_TOKEN = "version";

    public static PatternedSystemTag LIBVIRT_VERSION = new
PatternedSystemTag(String.format("libvirt::version::%s", LIBVIRT_VERSION_TOKEN), HostVO.class);



    public static final String HVM_CPU_FLAG_TOKEN = "flag";

    public static PatternedSystemTag HVM_CPU_FLAG = new PatternedSystemTag(String.format("hvm::%s",
HVM_CPU_FLAG_TOKEN), HostVO.class);

}
```

载入服务

服务在 Spring 的 bean 的 xml 文件中声明自身，例如，kvm 的部分声明类似于：

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"

    xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:zstack="http://zstack.org/schema/zstack"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

    http://www.springframework.org/schema/aop

    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

    http://www.springframework.org/schema/tx

    http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

    http://zstack.org/schema/zstack

    http://zstack.org/schema/zstack/plugin.xsd"

    default-init-method="init" default-destroy-method="destroy">



    <bean id="KvmHostReserveExtension" class="org.zstack.kvm.KvmHostReserveExtension">

        <zstack:plugin>

            <zstack:extension interface="org.zstack.header.Component" />

            <zstack:extension
interface="org.zstack.header.allocator.HostReservedCapacityExtensionPoint" />

        </zstack:plugin>
```

```
    </bean>



    <bean id="KVMHostFactory" class="org.zstack.kvm.KVMHostFactory">
        <zstack:plugin>
            <zstack:extension interface="org.zstack.header.host.HypervisorFactory" />
            <zstack:extension interface="org.zstack.header.Component" />
            <zstack:extension
interface="org.zstack.header.managementnode.ManagementNodeChangeListener" />
            <zstack:extension interface="org.zstack.header.volume.MaxDataVolumeNumberExtensionPoint"
/>
        </zstack:plugin>
    </bean>



    <bean id="KVMSecurityGroupBackend" class="org.zstack.kvm.KVMSecurityGroupBackend">
        <zstack:plugin>
            <zstack:extension
interface="org.zstack.network.securitygroup.SecurityGroupHypervisorBackend" />
            <zstack:extension interface="org.zstack.kvm.KVMHostConnectExtensionPoint" />
        </zstack:plugin>
    </bean>



    <bean id="KVMConsoleHypervisorBackend" class="org.zstack.kvm.KVMConsoleHypervisorBackend">
```

```
    <zstack:plugin>
        <zstack:extension interface="org.zstack.header.console.ConsoleHypervisorBackend"/>
    </zstack:plugin>
</bean>

<bean id="KVMApiInterceptor" class="org.zstack.kvm.KVMApiInterceptor">
    <zstack:plugin>
        <zstack:extension interface="org.zstack.header.apimediator.ApiMessageInterceptor"/>
    </zstack:plugin>
</bean>
  </beans>
```

管理节点，作为所有服务的容器，将在启动阶段读取它们的 XML 配置文件，载入每一个服务。

总结

在这篇文章中，我们演示了 ZStack 的进程内微服务架构。通过使用它，ZStack 拥有一个非常干净的，松耦合的代码结构，这是创建一个强壮 IaaS 软件的基础。