第10讲 云计算与虚拟资源

§10.1 云计算技术

§10.2 虚拟机管理平台OpenStack

§10.3 容器管理平台K8S

§10.1 云计算技术

Cloud Computing is an Evolution in IT

Grid Computing

- Solving large problems with parallel computing
- Made mainstream by Globus Alliance

Utility Computing

- Offering computing resources as a metered service
- Introduced in late 1990s

Software as a Service

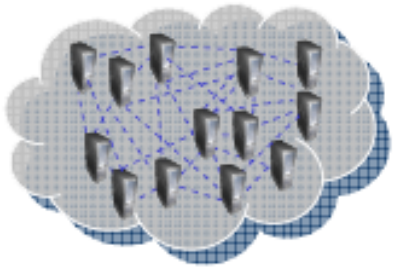
Network-based subscriptions to applications

Gained momentum in 2001

Cloud Computing

Next-Generation Internet computing

Next-Generation Data Centers



云计算诞生

- Late 1990s, Salesforce.com
- 2002, Amazon.com 启用 Amazon Web Services (AWS) 平台
- 2006, “云计算”这一术语才出现在商业领域
  Amazon 推出其弹性计算云(EC2)
- 2009, Google 应用引擎(GAE)
- 随后成为热点和主流

云计算定义

- Gartner report

…一种计算方式，能通过Internet技术将可扩展的和弹性的IT能力作为服务交付给外部用户。

- Forrester Research

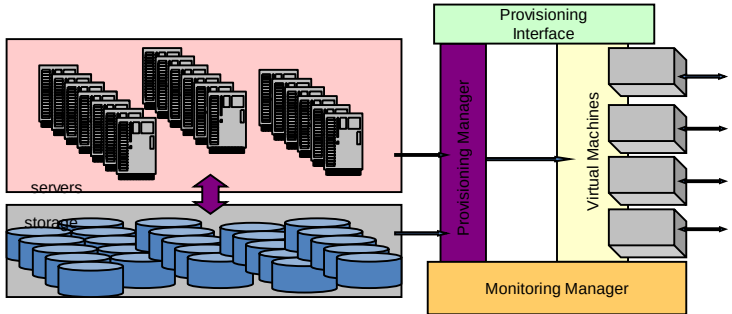
…一种标准化的IT性能（服务、软件或者基础设施），以按使用付费和自助服务方式，通过Internet技术进行交付。

- NIST

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility (like the electricity grid) over a network (typically the Internet). –from NIST

- 云计算教材《云计算：概念、技术与架构》

云计算是分布式计算的一种特殊形式，它引入效用模型来远程供给可扩展和可测量的资源。



基本要点

- 数据存储在云端，应用也存储在云端，这些对用户透明，通常由第三方提供;
- 云计算强调服务，用户按需使用服务，根据使用多少付费。

1：提供“资源”——包括计算、存储及网络资源。

  - 规模巨大 的全球化的数据库及存储中心，能够实现“海量”的存储、出色的安全性和高度的隐私性和可靠性
  - 此外，它还应是高效的、低价的、节省能源的。

2：提供动态的数据服务。

  - 数据包括原始数据、半结构化数据和经过处理的结构化数据。

3：提供云计算平台 ——包括软件开发API、环境和工具。

云计算服务

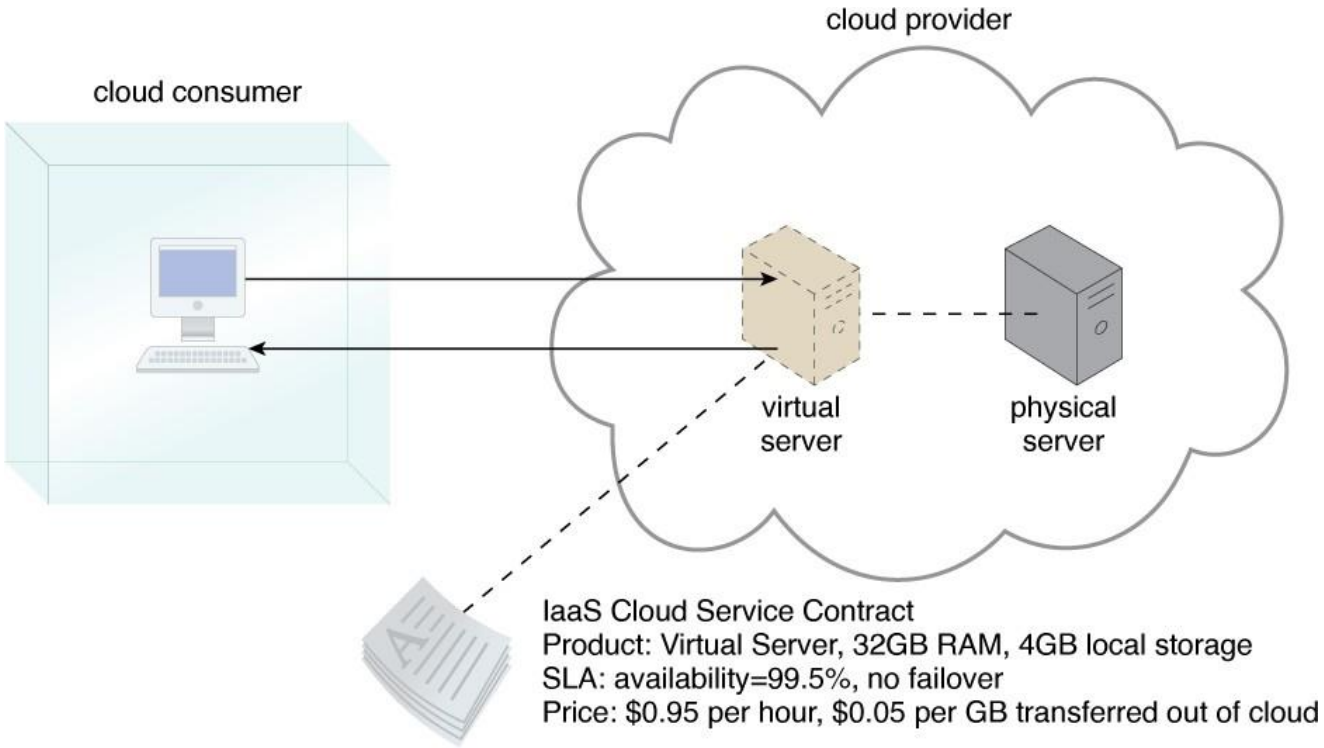
云计算 =数据*（软件+平台+基础设施）* 服务

- 数据（Data）
  - 爆炸增长（传感器、物联网）：1.2ZB = 10^{21}B
  - 各个领域各个层面
- 软件（Software）
  - 检索、发现、关联、处理和创造数据
- 平台（Platform）：
  - “云计算”时代也会诞生自己的通用平台
- 基础设施（Infrastructure）
  - 存储资源、计算资源等
- 服务（Service）
  - IT服务化：产品→服务 XaaS

云交付模型-IaaS

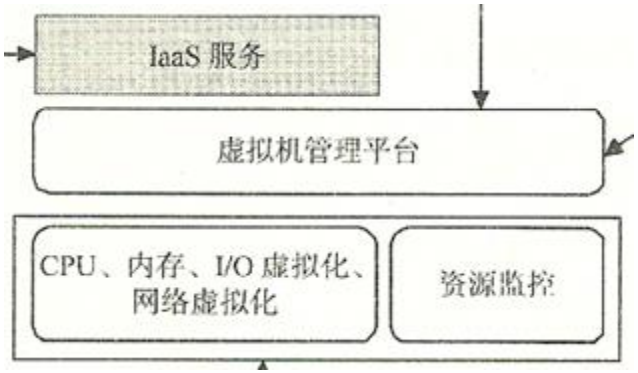
- 基础设施作为服务Infrastructure-as-a-Service (IaaS)
- IaaS交付模型是一种自我包含的IT环境，由以基础设施为中心的IT资源组成，可以通过基于云服务的借口和工具访问和管理这些资源。
- IaaS 环境包括硬件、网络、连通性、操作系统以及其他一些原始的IT资源。
- 在 IaaS中, IT 资源通常是虚拟化的并打包成包。

云交付模型- IaaS



IaaS

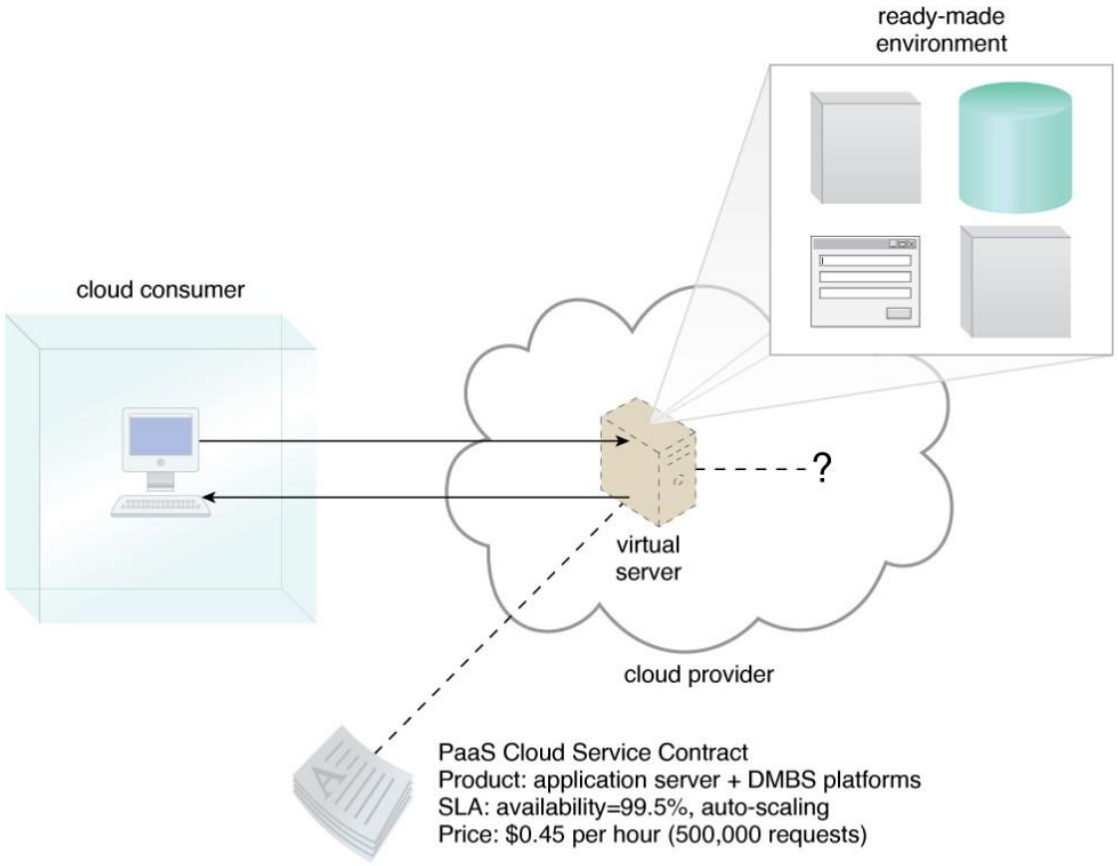
- Infrastructure as a Service
- 硬件资源作为服务提供给用户
- 主要技术
  - 虚拟化技术
  - 资源动态管理与调度技术
- 典型产品
  - EC2 from Amazon
  - ECS from Aliyun



云交付模型-PaaS

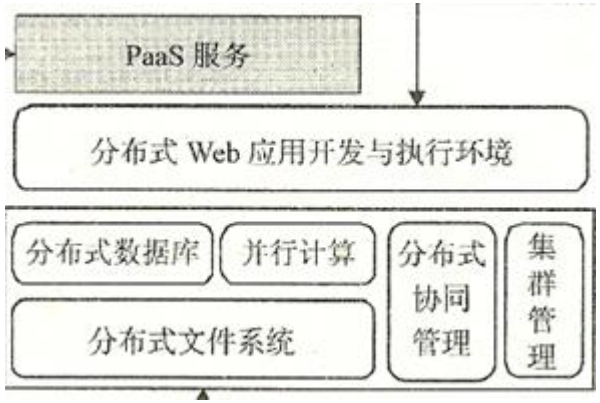
- 平台作为服务Platform-as-a-Service (PaaS)
- Paas交付模型是预先定义好的“就绪可用”的环境，一般由已经部署好和配置好的IT资源组成
- PaaS依赖于使用已就绪环境，设立好一套预先打包好的产品和用来支持定制化应用的整个交付生命周期的工具
- 例如，GAE提供了基于Java和Python的环境

PaaS



PaaS

- Platform as a Service
- 提供应用软件的开发、测试、部署和运行环境
  - 运行平台，如应用服务器
  - 辅助系统软件，如数据库
- 关键技术
  - 分布式文件系统
  - 分布式数据库
  - 并行计算技术
- 典型产品
  - GAE – Google
  - SAE – Sina
  - ACE – Aliyun
  - Windows Azure



Google App Engine

阿里云 aliyun.com ACE

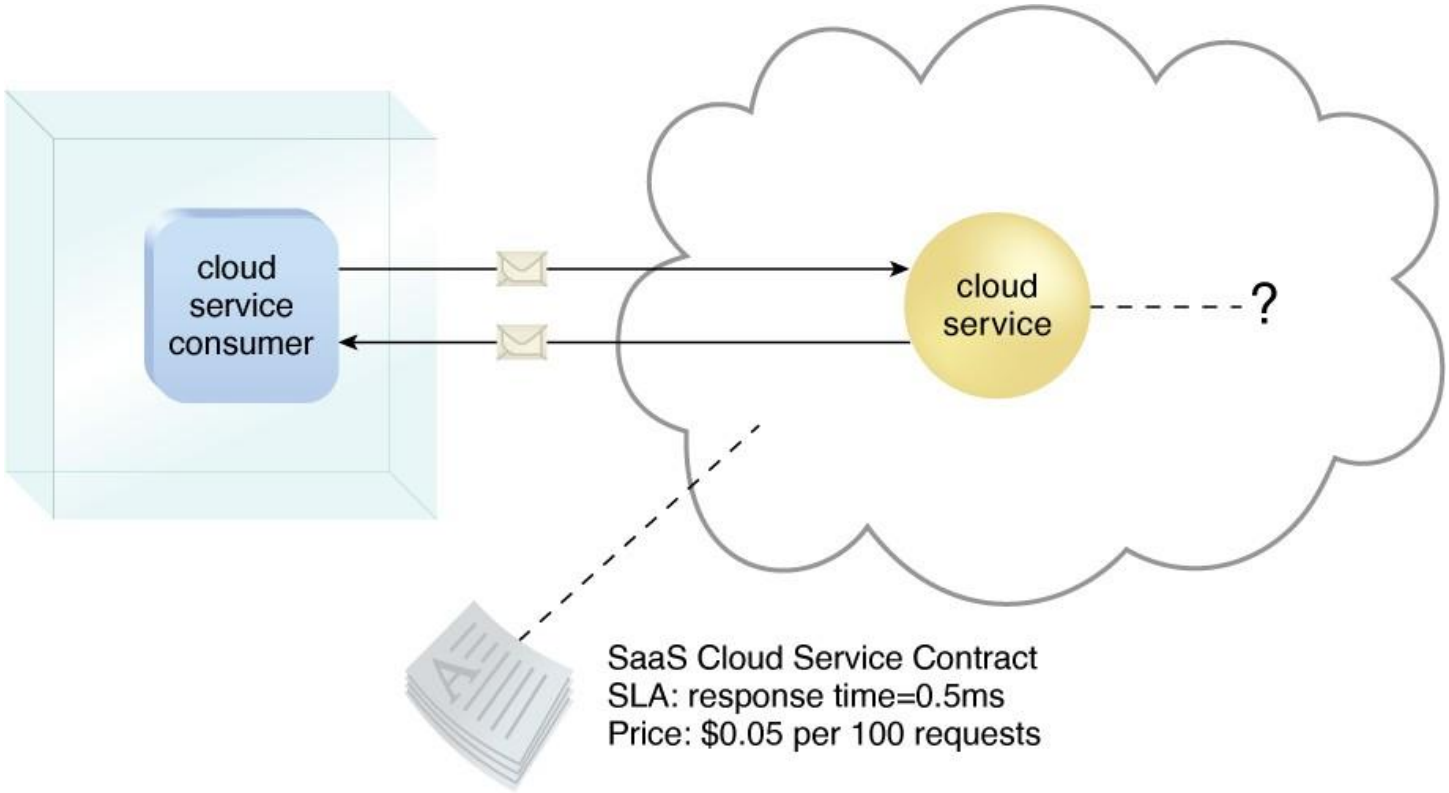
新浪云计算 e.sae.sina.com.cn

Windows Azure

云交付模型-SaaS

- 软件作为服务Software-as-a-Service (SaaS)
- SaaS通常是把软件程序定位成共享的云服务，作为“产品”或通用的工具进行提供。
- 通常，云用户对SaaS实现的管理权限非常有限。

云交付模型-SaaS

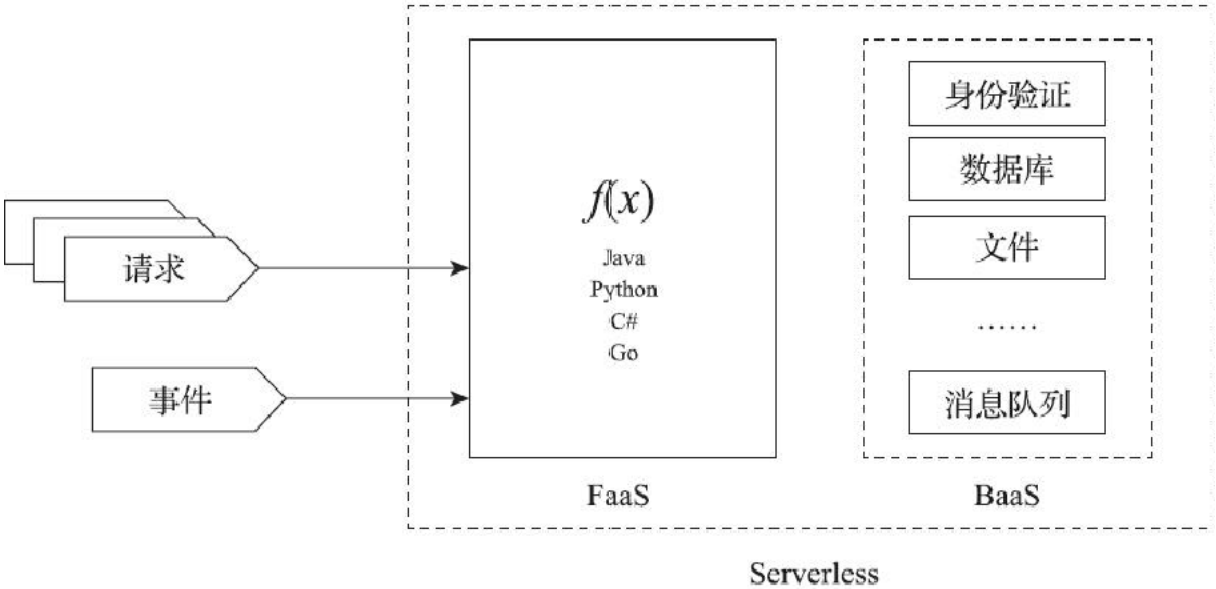


SaaS

- Software/Application as a Service
- 将运行于云中的应用软件的功能交付给用户。
- 关键技术
  - 呈现技术
  - 多租户技术（Multitenancy）
- 典型产品
  - Salesforce 的CRM服务

Severless与FaaS

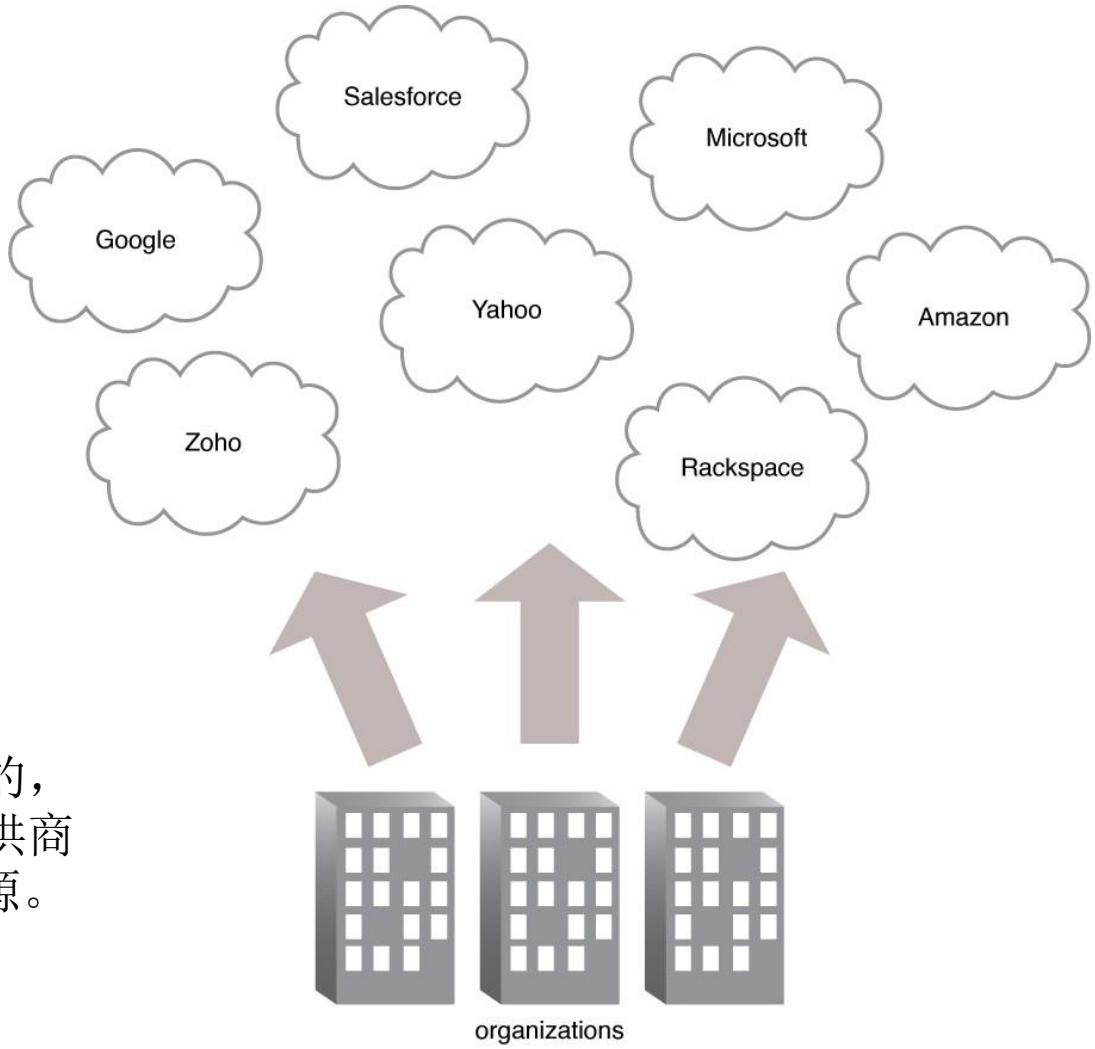
- Serverless架构即“无服务器”架构
  - 是一种软件系统架构思想和方法，它的核心思想是用户无须关注支撑应用服务运行的底层主机。
- Serverless需要两个方面的支持：
  - 函数即服务（Function as a Service, FaaS）
  - 后台即服务（Backend as a Service, BaaS）



交付模型比较

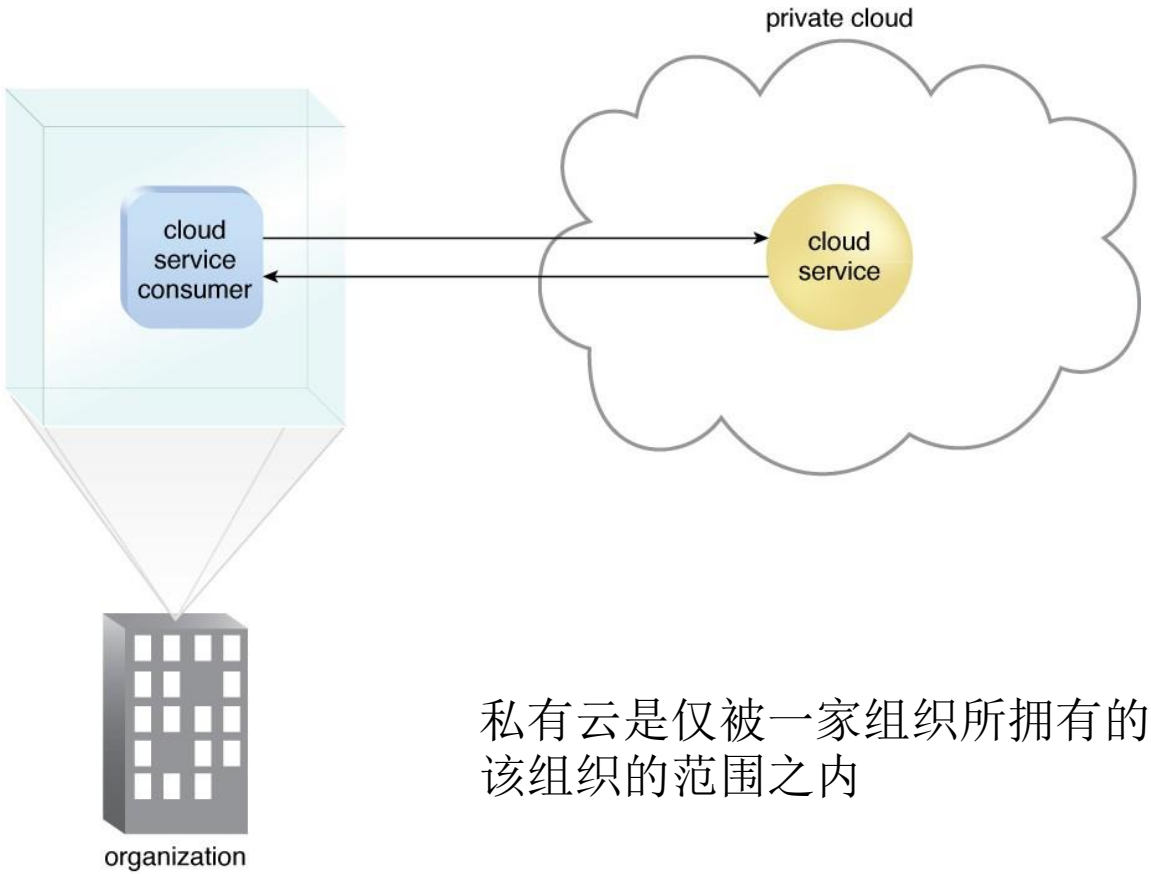
- IaaS：
  - 向用户提供对基于“原始”基础设施的IT资源的高等级管理控制。
- PaaS：
  - 使得云提供者可以提供预先配置好的环境。
  - 云用户可以使用这个环境来构建和部署云服务和解决方案，不过管理控制权有所下降。
- SaaS：
  - 是共享云服务的交付模型
  - 这些共享云服务可以是云承载的商业产品。
- 组合：
  - IaaS, PaaS and SaaS可以通过组合实现交付模型的合作。

云部署模型-Public Cloud



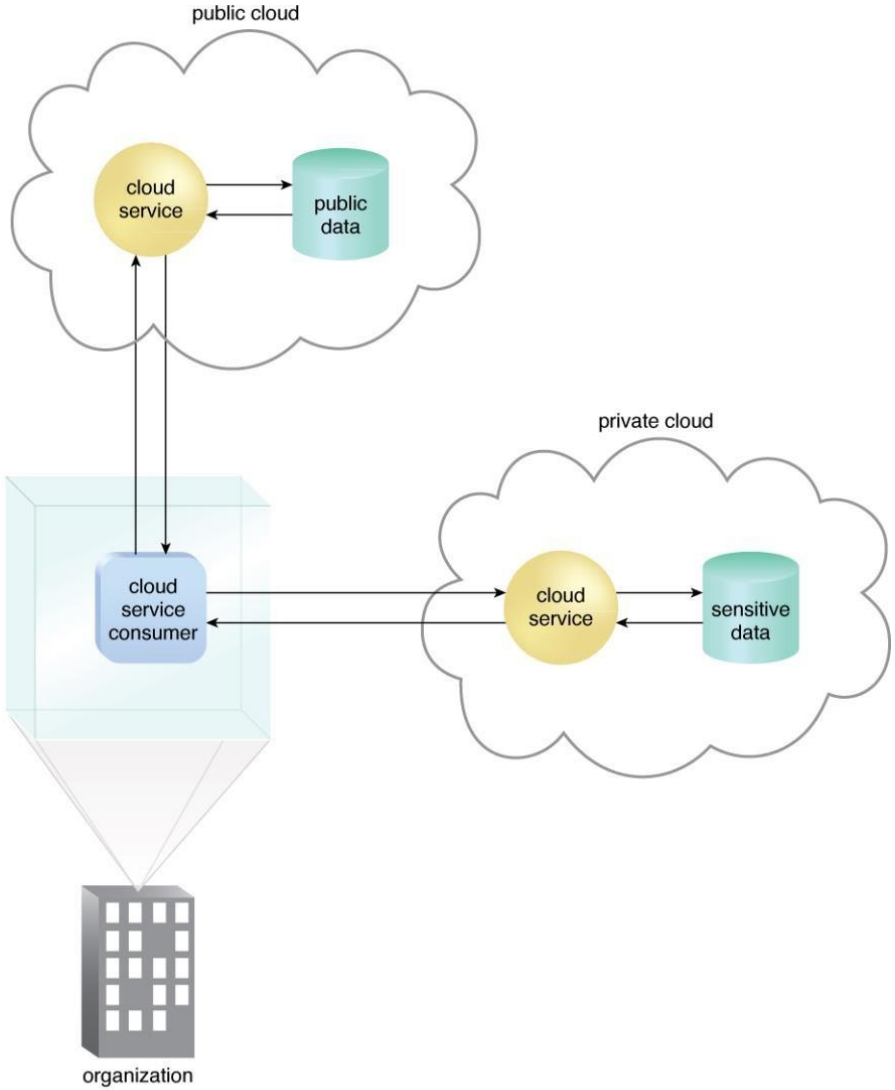
公有云是第三方所有的，通常向云用户组织提供商业化的云服务和IT资源。

云部署模型-Private Cloud



私有云是仅被一家组织所拥有的，并且位于该组织的范围之内

云部署模型-Hybrid Cloud



§10.2 虚拟机管理平台OpenStack

计算资源管理

OpenStack可以规划并管理大量虚拟机，从而允许企业或服务提供商按需提供计算资源

存储资源管理

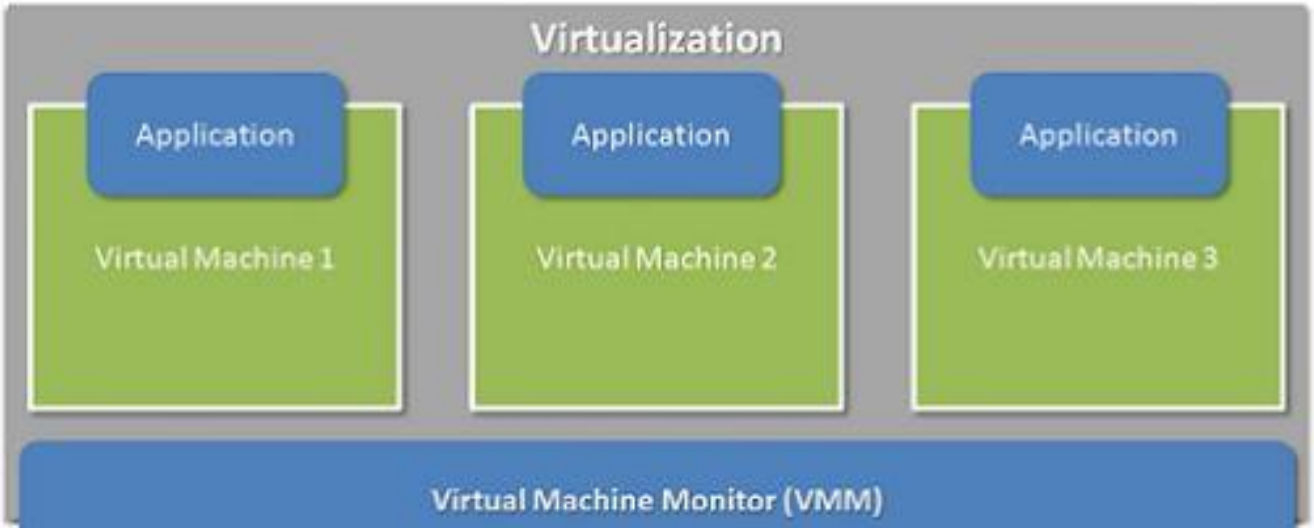
OpenStack可以为云服务或云应用提供所需的对象及块存储资源

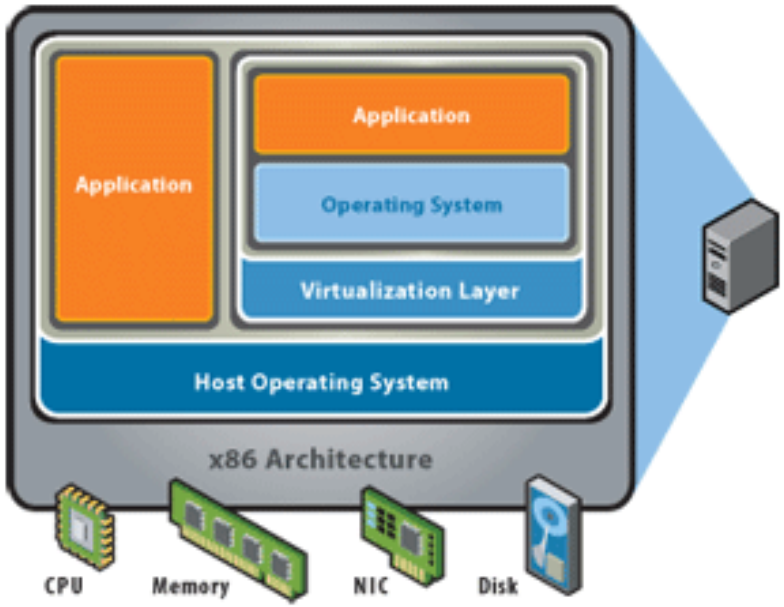
网络资源管理

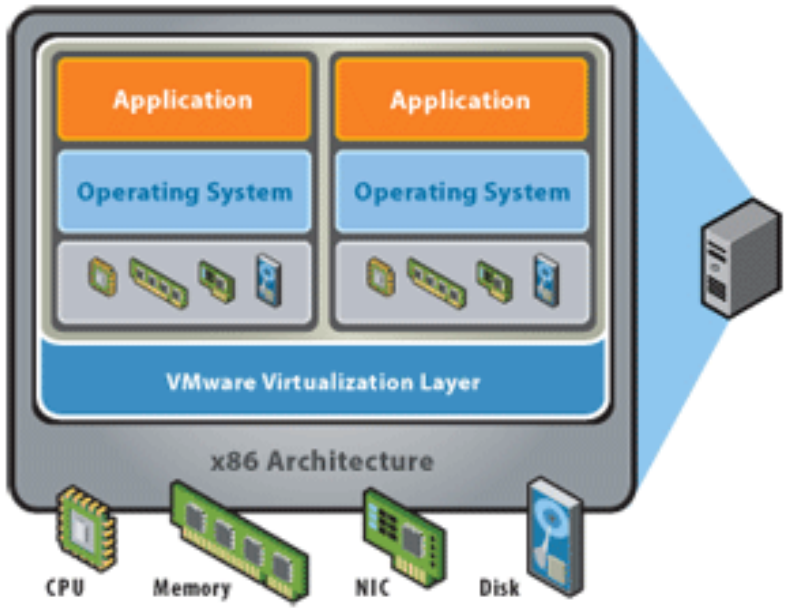
IP地址的数量、路由配置、安全规则将爆炸式增长；传统的网络管理技术无法真正高扩展、高自动化地管理下一代网络

openstack
CLOUD SOFTWARE

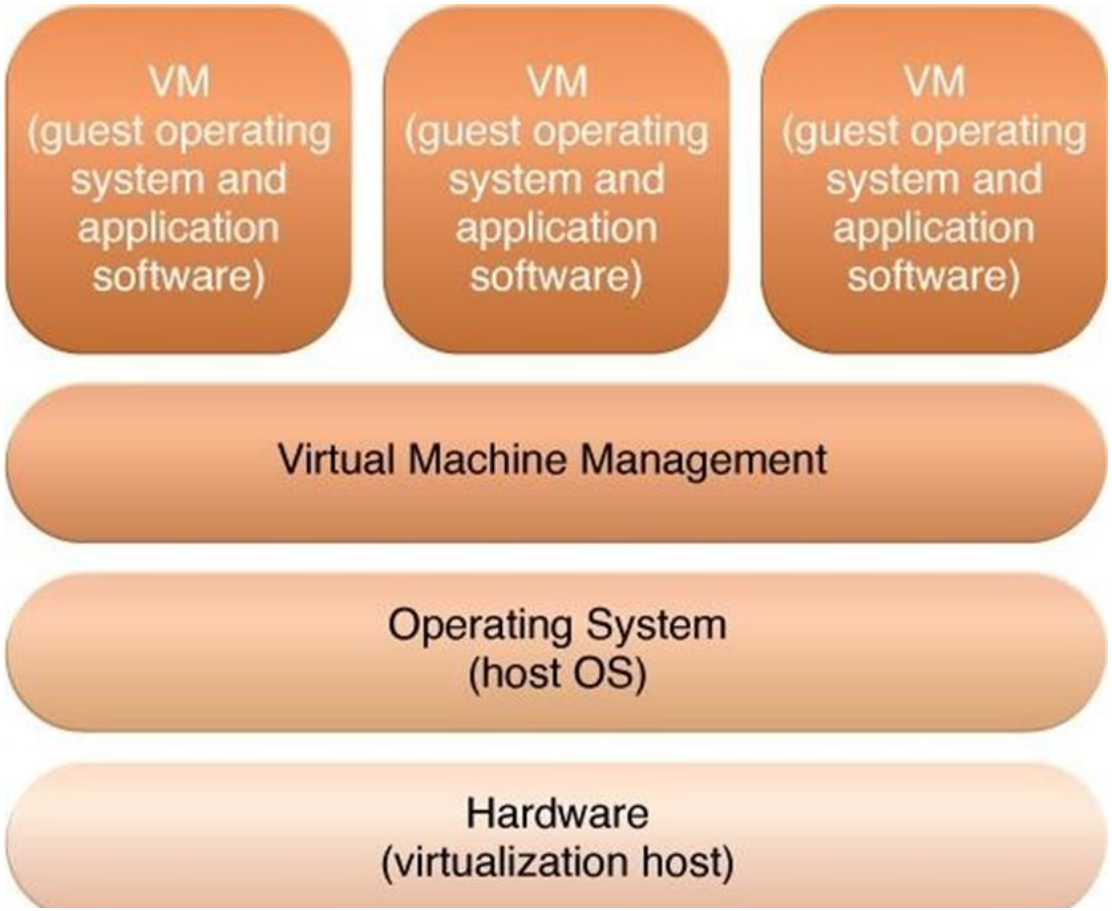
虚拟机架构


Virtualization
Application
Application
Application
Virtual Machine 1
Virtual Machine 2
Virtual Machine 3
Virtual Machine Monitor (VMM)


Application
Application
Operating System
Virtualization Layer
Host Operating System
x86 Architecture
CPU
Memory
NIC
Disk


Application
Application
Operating System
Operating System
VMware Virtualization Layer
x86 Architecture
CPU
Memory
NIC
Disk

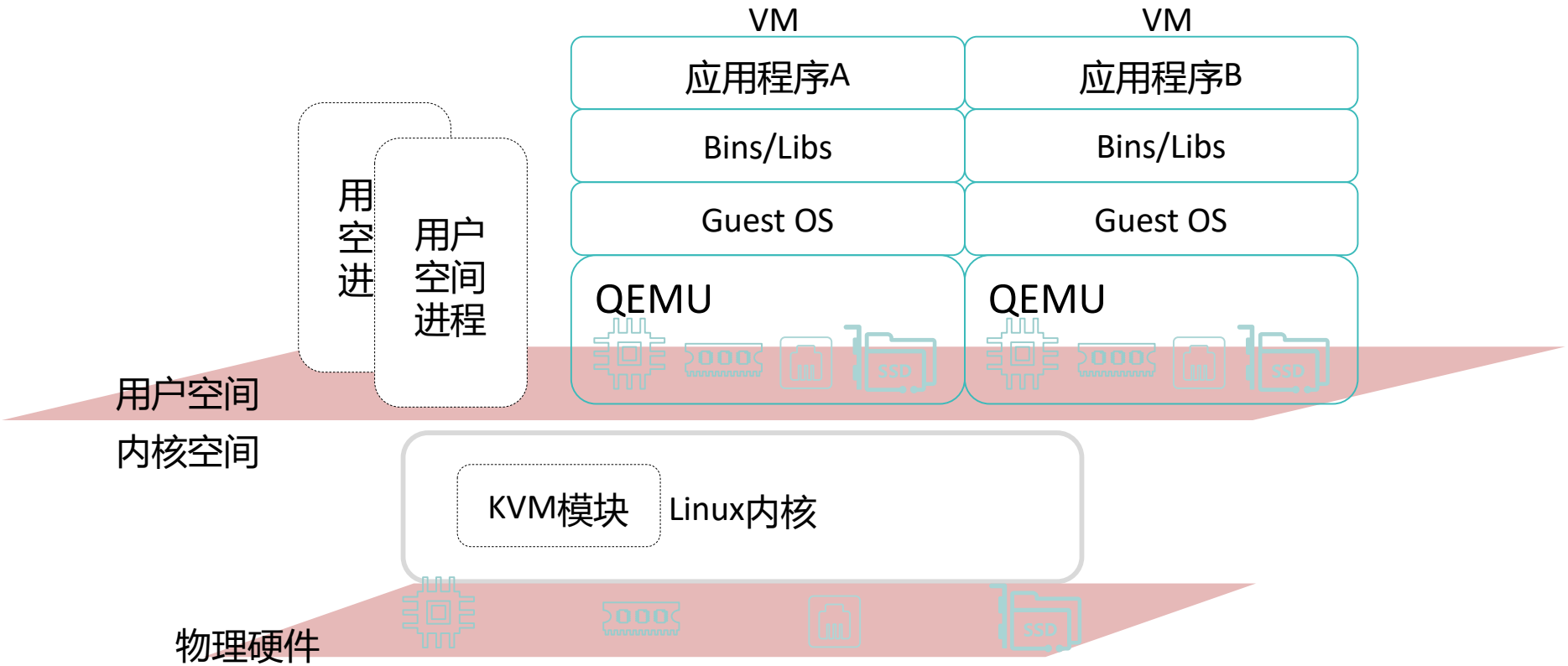
基于操作系统的虚拟化--寄生架构（Hosted）



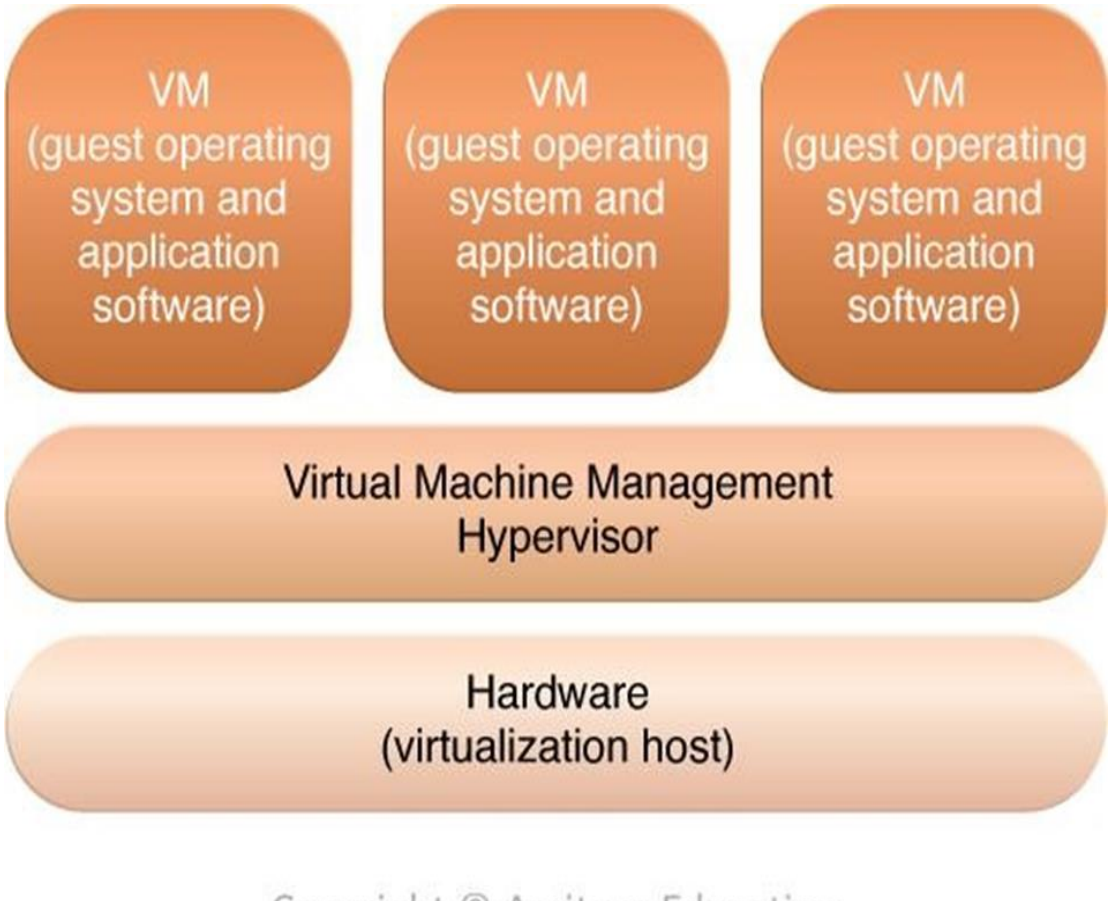
- ✓ 宿主操作系统：管理所有硬件资源，VMM为该操作系统上的一个应用程序
- ✓ 虚拟机监控器（VMM）：模拟硬件的一些行为
- ✓ 客户操作系统：运行用户的应用

寄生架构--KVM

- Linux安装KVM模块：
  - Guest Mode：主要是指虚拟机，包括CPU、内存、磁盘等虚拟设备；
  - User Mode：运行QEMU，为VM模拟执行I/O类的操作请求；
  - Kernel Mode：可以真正操作硬件，当Guest OS执行I/O操作或特权指令操作时，需要向用户模式提交请求，再由用户模式向内核模式请求。



基于硬件的虚拟化--裸金属架构（Bare-metal）

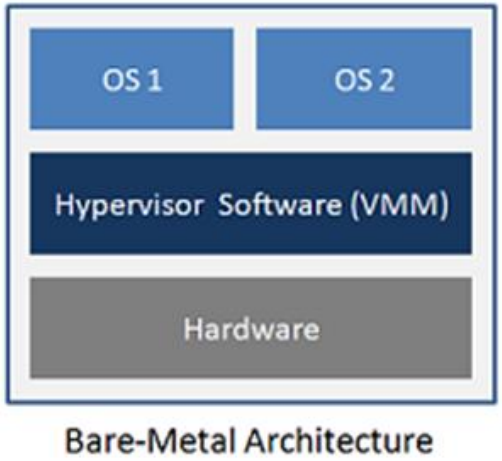


虚拟机监控器（VMM）：实现核心功能的轻量级操作系统，也称Hypervisor

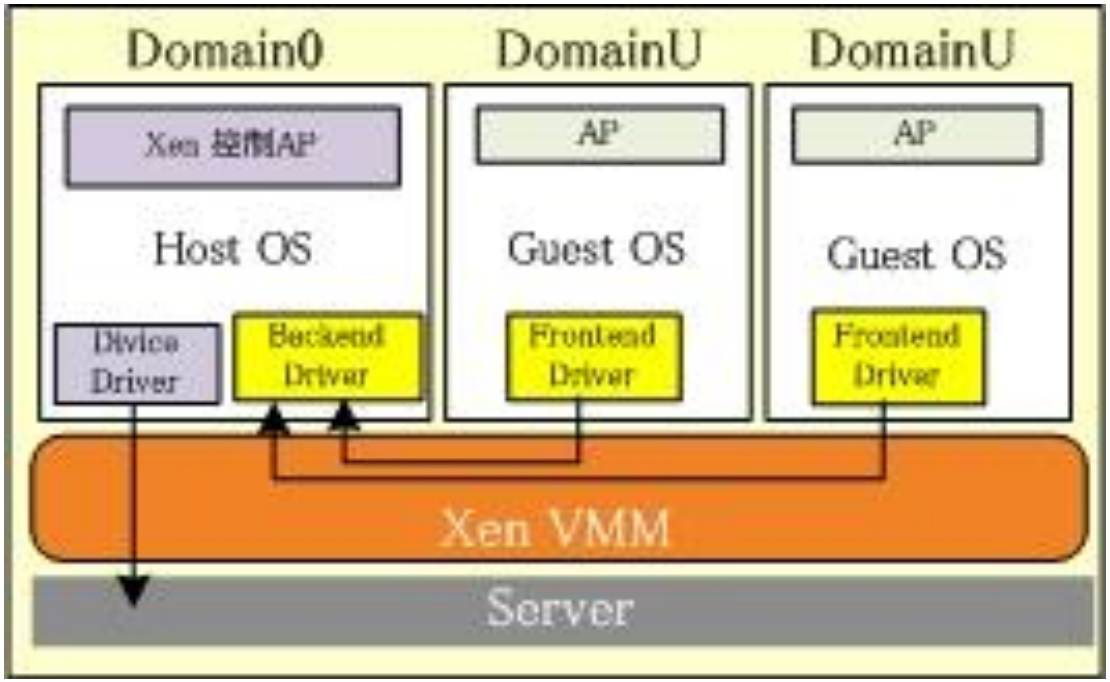
客户操作系统：运行用户的应用

裸金属架构（Bare-metal）

- VMM + (Host OS +) OS



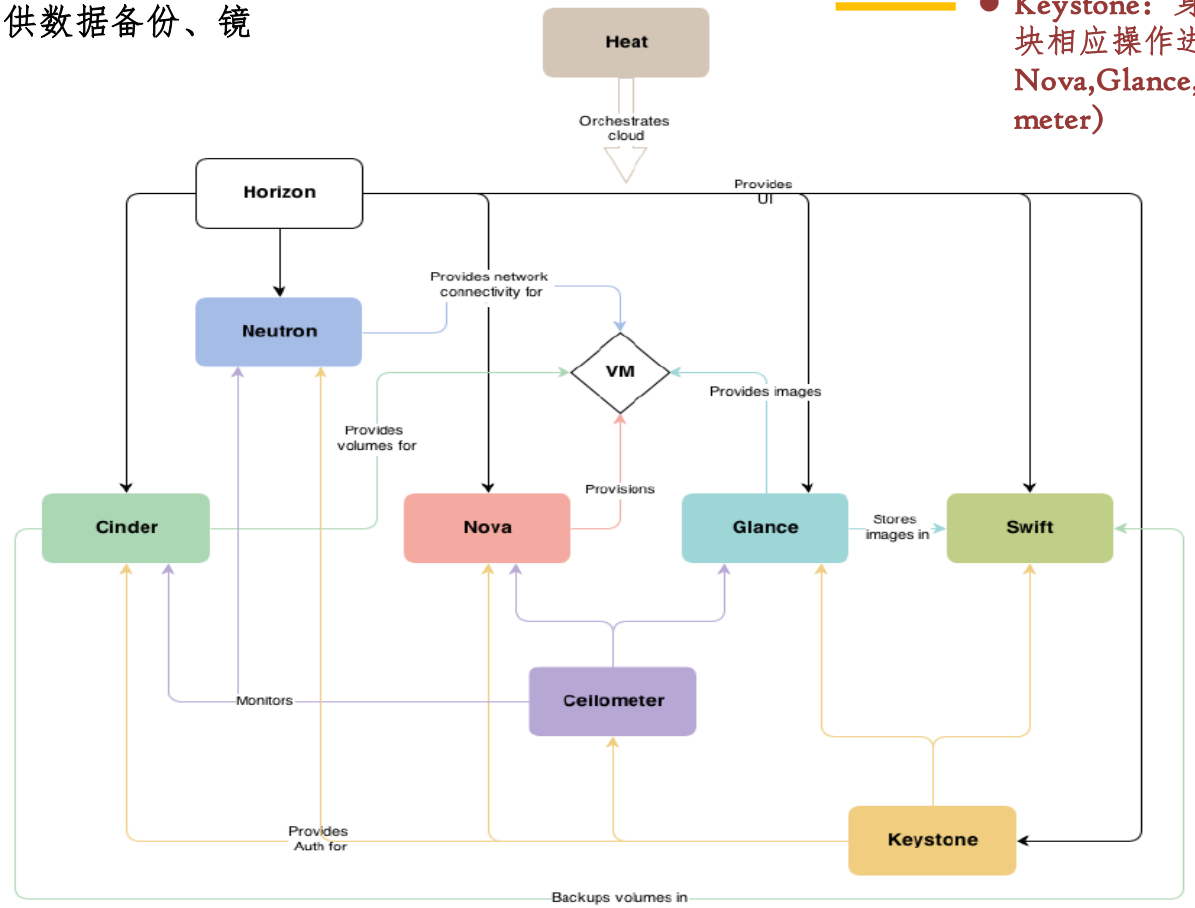
Bare-Metal Architecture



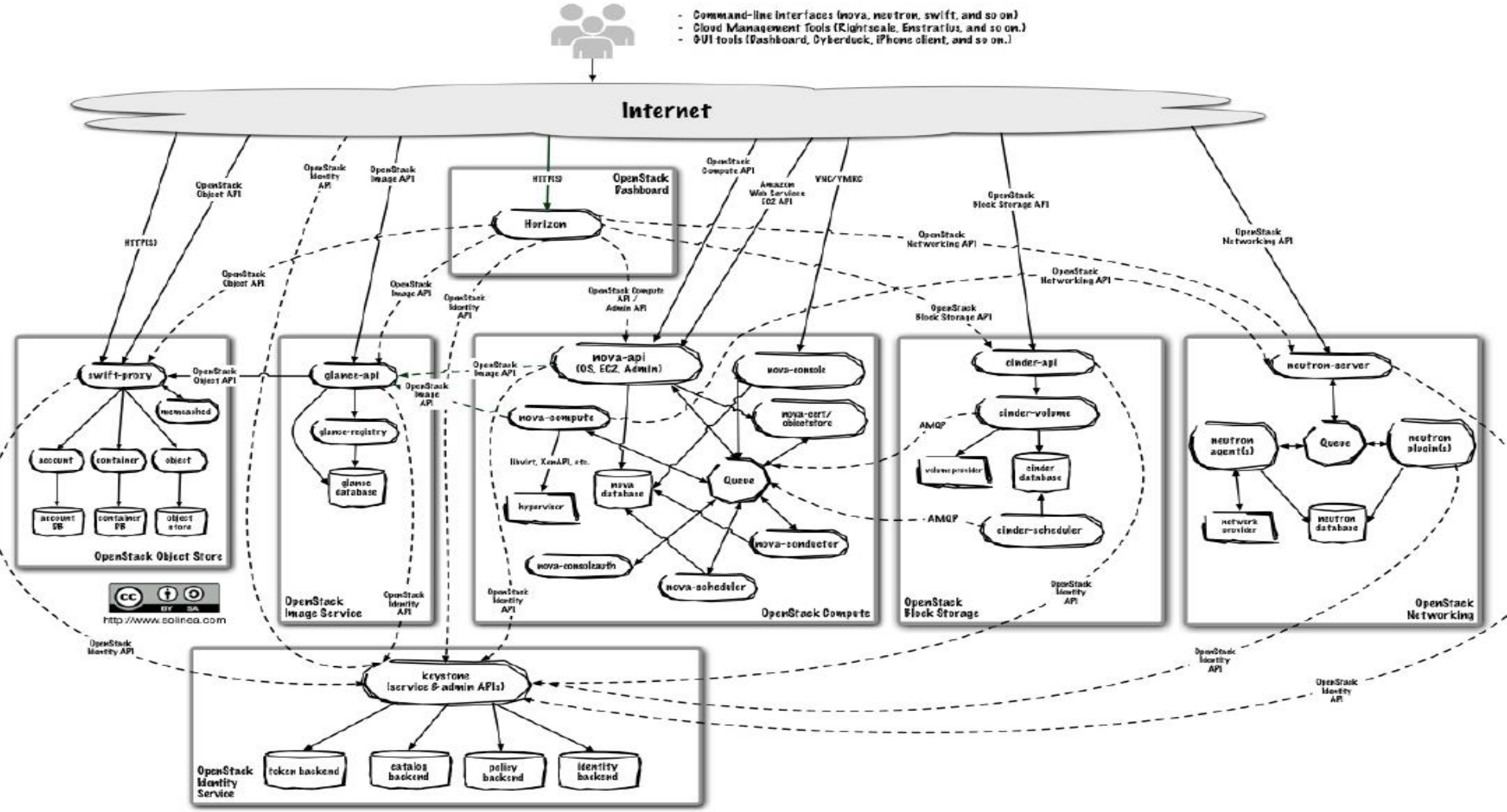
OpenStack主要服务/组件

- Nova为VM提供计算资源
- Glance为VM提供镜像
- Cinder为VM提供块存储资源
- Neutron为VM提供网络资源及网络连接
- Swift为VM提供数据备份、镜像备份

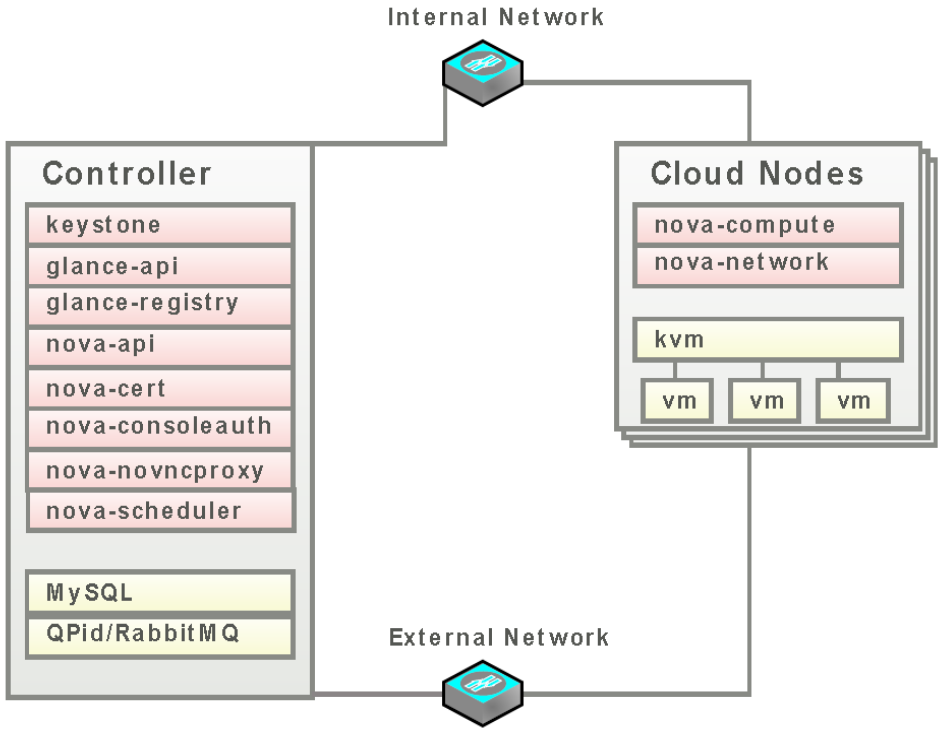
- Horizon(Dashboard)：控制台
- Cellometer：监控功能，Nova,Glance,Cinder,Neutron
- Keystone：身份验证功能，可以对其他模块相应操作进行身份及权限验证(包括Nova,Glance,Cinder,Swift,Neutron,Ceilometer)



OpenStack整体架构



OpenStack 最简物理架构



2个节点：

- Cloud Controller Node:
  - Keystone(身份验证服务)
  - Glance(镜像管理服务)
  - Nova (计算资源管理服务)
  - 数据库服务(MySQL)
  - 消息服务(RabbitMQ或QPid)
- Compute Node:
  - Nova-Compute
  - Nova-Network
  - KVM虚拟化系统

2种网络：

- Internal Network(内部网络)
  - 用于提供Provider网络(VM to Provider)
  - 用于tenant网络(VM to VM)
- External Network(外部网络）
  - 用于外部用户与VM通信及控制(VM to Internet)

Nova是OpenStack云中的计算组织控制器

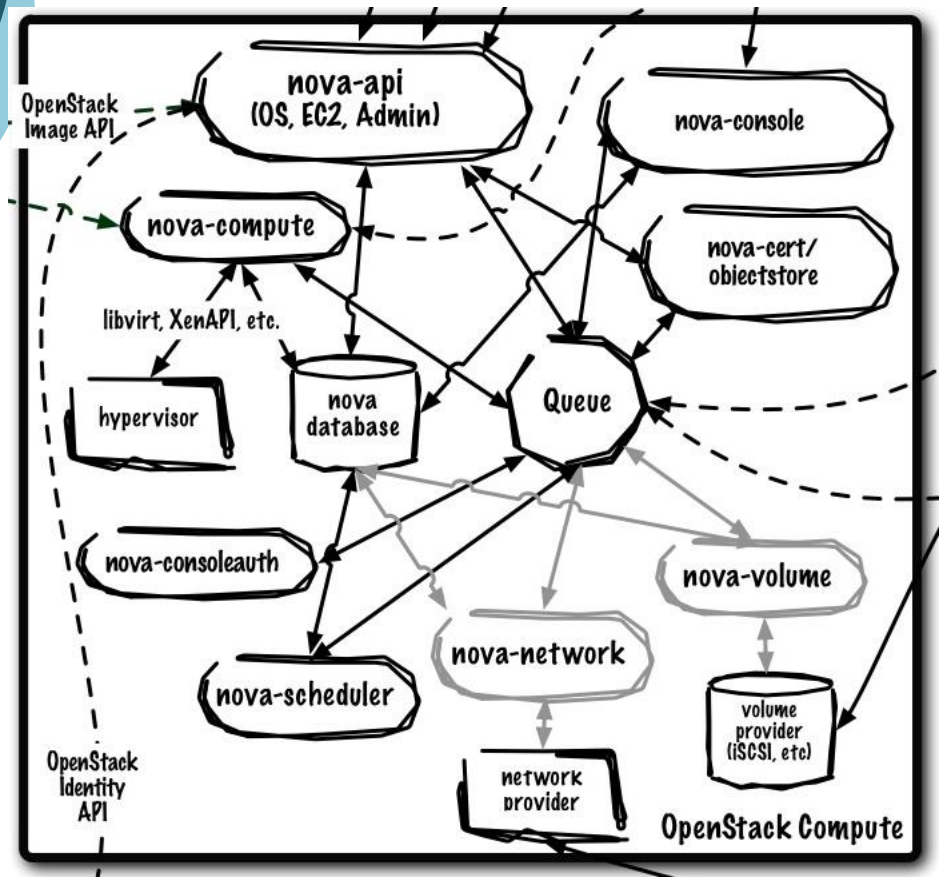
Nova处理OpenStack云中实例（instances）生命周期的所有活动。

Nova负责**管理计算资源、网络、认证**、所需可扩展性的平台。

Nova并不具有虚拟化能力，它使用Libvirt API来与被支持的Hypervisors交互。

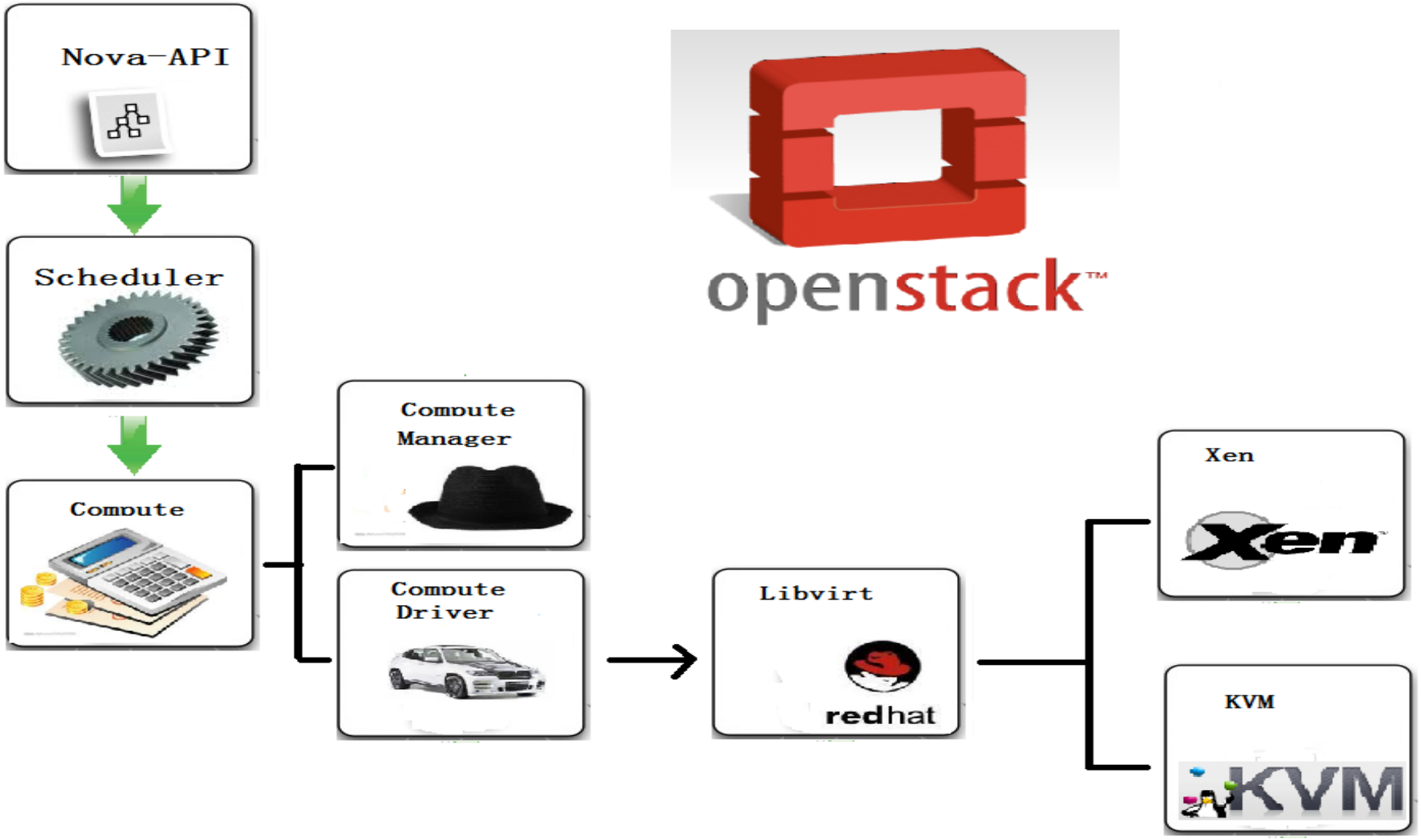
Nova通过一个与Amazon Web Services（AWS）EC2 API兼容的Web Services API来对外提供服务。

Nova的架构



- Nova-API:对外统一提供标准化接口。接受和响应最终用户的请求,实现与其他各逻辑模块的通讯与服务提供。
- Nova-Scheduler:从队列上得到一个虚拟机实例请求并且决定它应该在哪里运行。
- Queue: 提供了守护进程之间传递消息的中央枢纽，还是与其他各逻辑模块间通信的连接枢纽
- Nova-Database:存储云基础设施的编译时和运行时的状态，主要是sqlite3（只适用于测试和开发工作），MySQL和PostgreSQL。
- Nova-Compute:一个人工守护进程，它可以通过虚拟机管理程序的API来创建和终止虚拟机实例。支持多种虚拟化平台。
- Nova还提供控制台的服务，让最终用户通过代理服务器访问他们的虚拟实例的控制台。

Nova工作流程


Nova-API
Scheduler
Compute
Compute Manager
Compute Driver
Libvirt
redhat
Xen
KVM
openstack™

Nova组件

1. Nova-API

API Server，

对外提供一个与云基础设施交互的接口，

也是外部可用于管理基础设施的唯一组件。

2. Rabbit MQ Server

Message Queue，

OpenStack节点之间通过消息队列使用AMQP（Advanced Message Queue Protocol）完成通信。

Nova组件

3. Nova-Compute

Compute Worker，管理实例生命周期，

通过Message Queue接收实例生命周期管理的请求，并承担操作工作。

4. Nova-Network

Network Controller，

处理主机的网络配置，包括IP地址分配、为项目配置VLAN、实现安全组、配置计算节点网络。

Nova组件

5. Nova-Volume

Volume Workers，

用来管理基于LVM的实例卷。

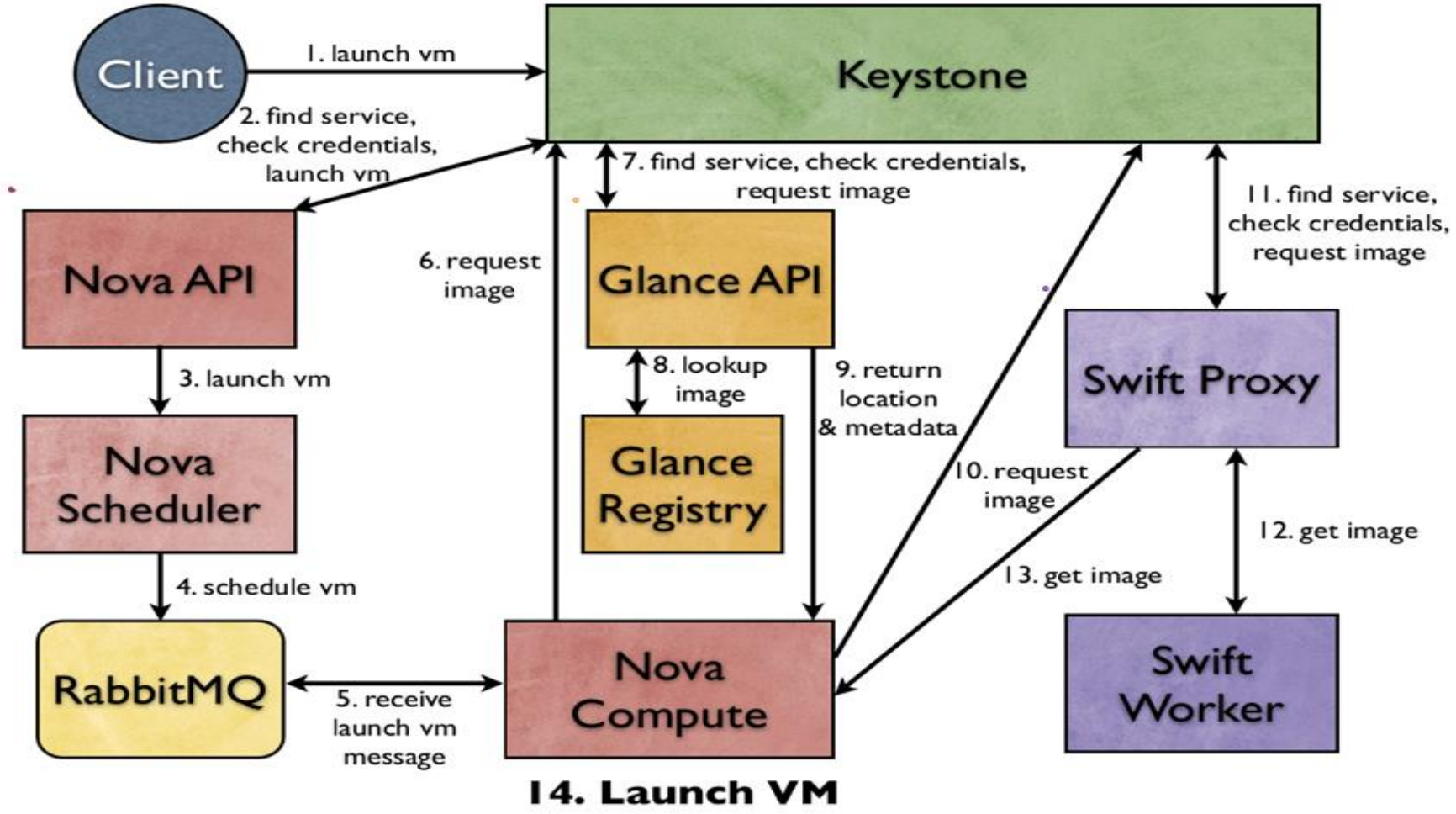
有卷的相关功能，例如新建卷、删除卷、为实例附加卷、为实例分离卷。

6. Nova-Scheduler

调度器Scheduler，

通过恰当的调度算法从可用资源池获得一个计算服务。

Nova启动虚拟机流程



Nova Scheduler

Scheduler(调度器）服务nova-schecduler来裁决虚拟机的空间与资源分配；它会通过各种规则（内存使用率、CPU负载等因素）选择主机。

不同的调度器并不能共存，需要在/etc/nova/nova.conf中通过scheduler_driver选项指定，默认使用的是FilterScheduler：scheduler_driver = nova. scheduler. filter_scheduler.FilterScheduler

实现自己的调度器：继承类SchedulerDriver，实现接口nova.scheduler.driver.Scheduler

从Juno开始，社区也在致力于剥离nova-scheduler为Gantt，从而提供一个通用的调度服务为多个项目使用

Nova支持的调度器和过滤器

Nova支持的调度器
•ChanceScheduler随机调度器
•SimpleScheduler简单调度器
•FilterScheduler过滤调度器
•MultiScheduler多重调度器

Nova支持的过滤器
•All HostFilter
•ImagePropertiesFilter
•AvailabilityZoneFilter
•ComputeFilter
•CoreFilter
•IsolateHostFilter
•RamFilter
•SimpleCIDRAffinityFilter
•DifferentHostFilter
•SameHostFilter

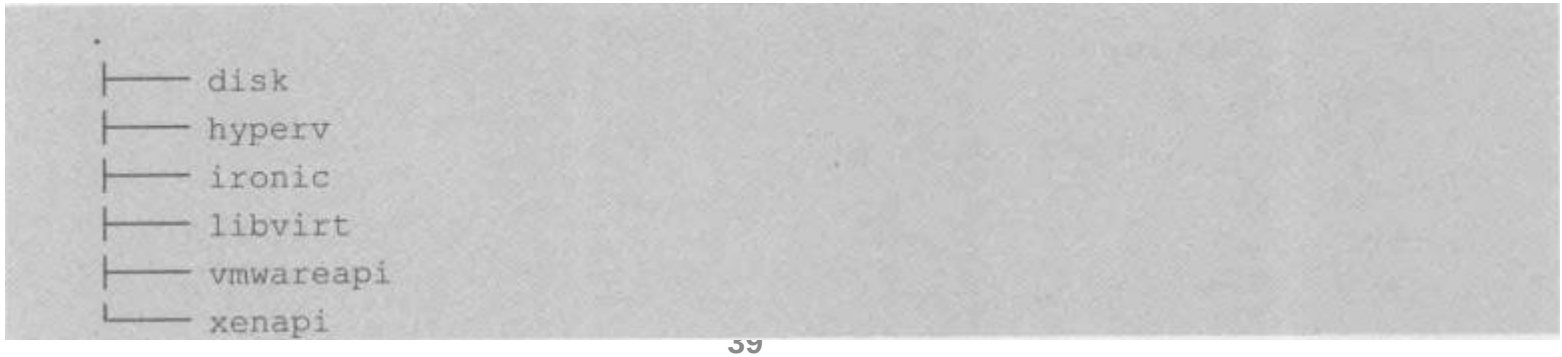
Nova Compute

Nova控制虚拟机的状态变迁和生老病死

对虚拟机生命周期的管理具体由Compute服务nova-compute来完成。

Nova本身并不提供虚拟化技术，只是借助于各种主流的虚拟化，比如Xen、KVM等，实现虚拟机的创建与管理，因此作为Nova的核心，nova-compute需要和不同的Hypervisor进行交互。

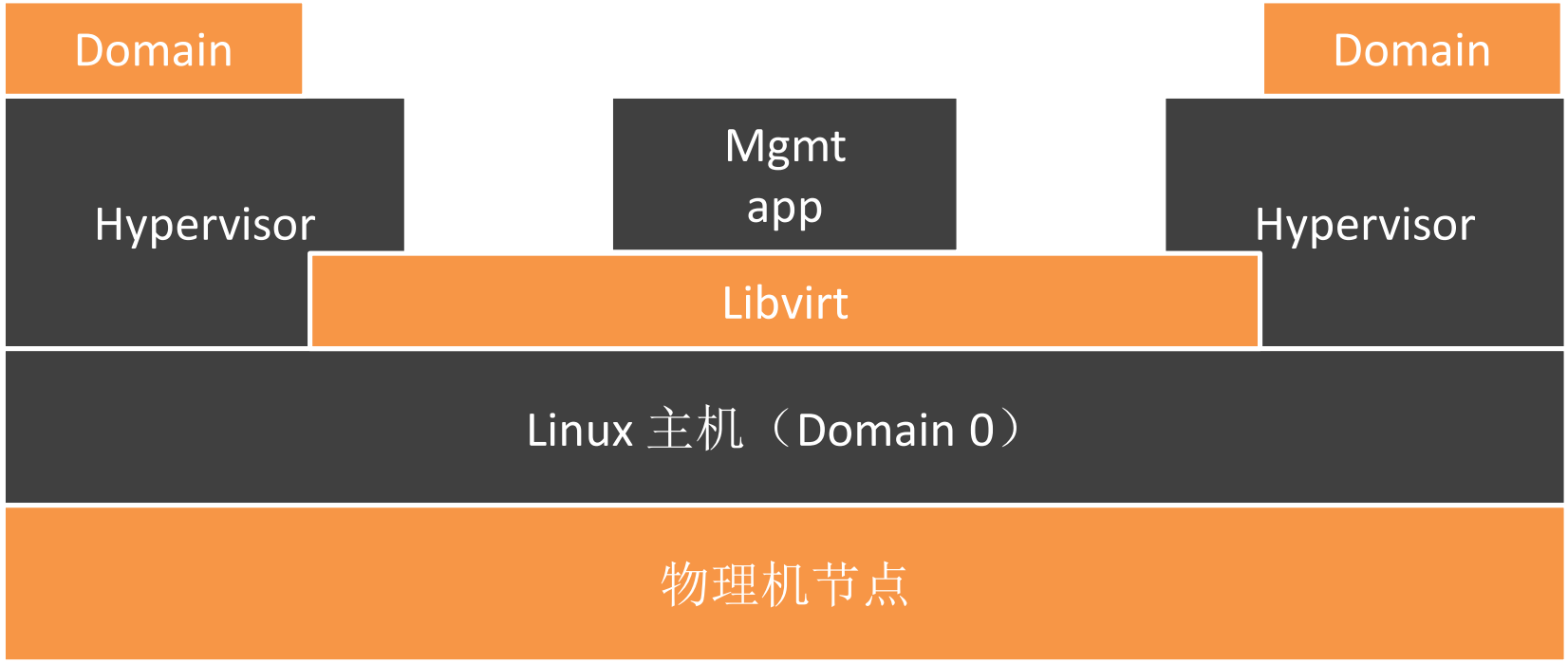
各种Hypervisor的支持通过Virt Driver的方式实现，比如Libvirt Driver，各种Virt Driver的实现位于nova/virt目录。



Nova Compute

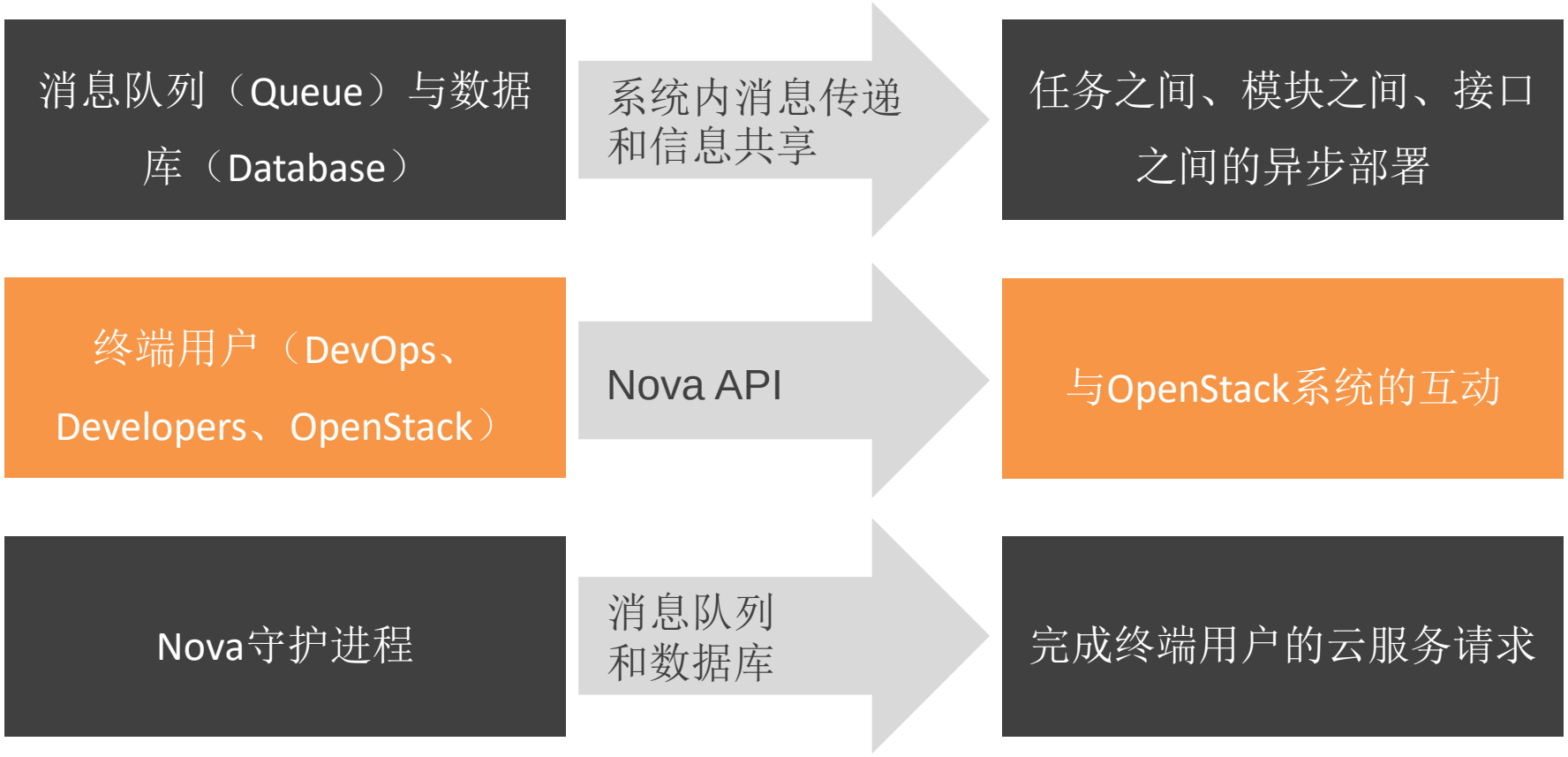
Libvirt体系结构

管理应用程序通过Libvirt工作，以控制本地域



Nova Compute

Nova中的RabbitMQ解析

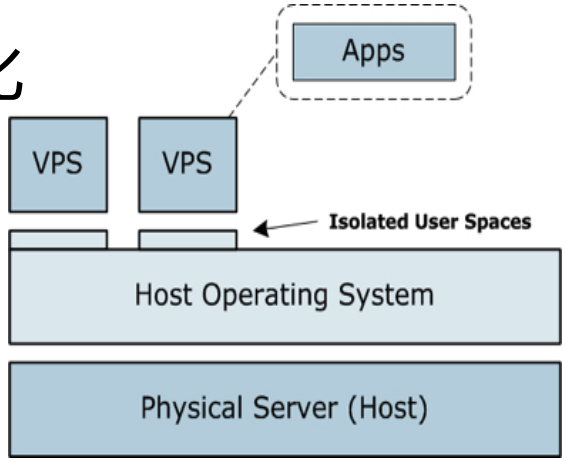


§10.3 容器管理平台K8S

- 起源于谷歌内部的Borg 系统
- 一个集群和容器管理工具（事实标准）
  - 可以让开发者把容器部署到集群/用网络连接起来的服务器上。
  - 可以应用于不同的容器引擎上（不仅仅是Docker）。
- 基本理念：进一步把机器、存储、网络从它们的物理实现层面抽象出来。

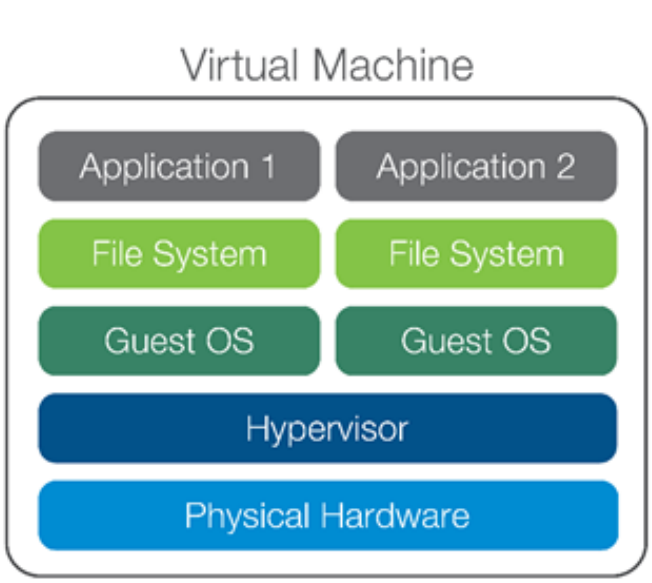
容器技术（Container）

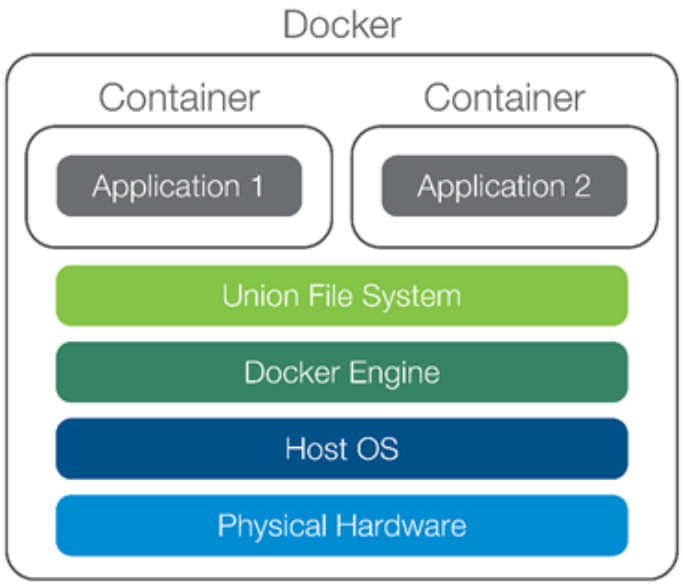
- 容器在操作系统中的运行
  - ✓ 容器自带环境在操作系统中启动进程
  - ✓ 所有容器共用一个操作系统
  - ✓ 通过命名空间和进程组来提供隔离性
- Docker容器
  - ✓ 开源的应用容器引擎
  - ✓ 开发者可以打包他们的应用以及应用的依赖包，放到一个可移植的容器中
  - ✓ 然后发布到任意的机器上以实现虚拟化



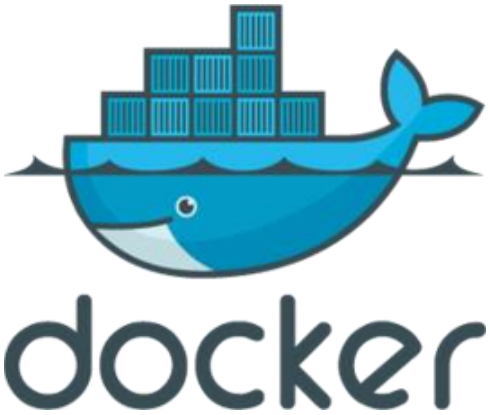
容器虚拟化

- 优势
  - ✓ 轻量级虚拟化，没有独立的操作系统，额外开销很小
  - ✓ 每个主机可以支撑上千容器，且容器启动时间在毫秒级
  - ✓ 自带运行环境，开发过程中打包好的应用，在生产环境中能迅速部署
- 缺点
  - ✓ 隔离性、安全性较差
  - ✓ 多容器共享操作系统，不能随意修改操作系统相关的配置





容器技术



- 三个基本技术
- Namespace：
  - 视图隔离
  - 让进程只能看到Namespace中的世界；
- Cgroups
  - 资源隔离
  - 让这个“世界”围着一个看不见的墙；
- Rootfs
  - 文件系统隔离
  - rootfs 只是一个操作系统所包含的文件、配置和目录，但不包括内核。

容器技术-- Namespace

- 容器的本质是—进程
  - **Namespace 技术**则是用来修改进程视图的主要方法

在容器中执行ps命令，结果如下：

```
/ # ps
PID   USER     TIME COMMAND
1 root       0:00 /bin/sh
10 root      0:00 ps
```

这个容器（进程）将会“看到”一个全新的进程空间，在这个进程空间里，它的 PID 是 1。

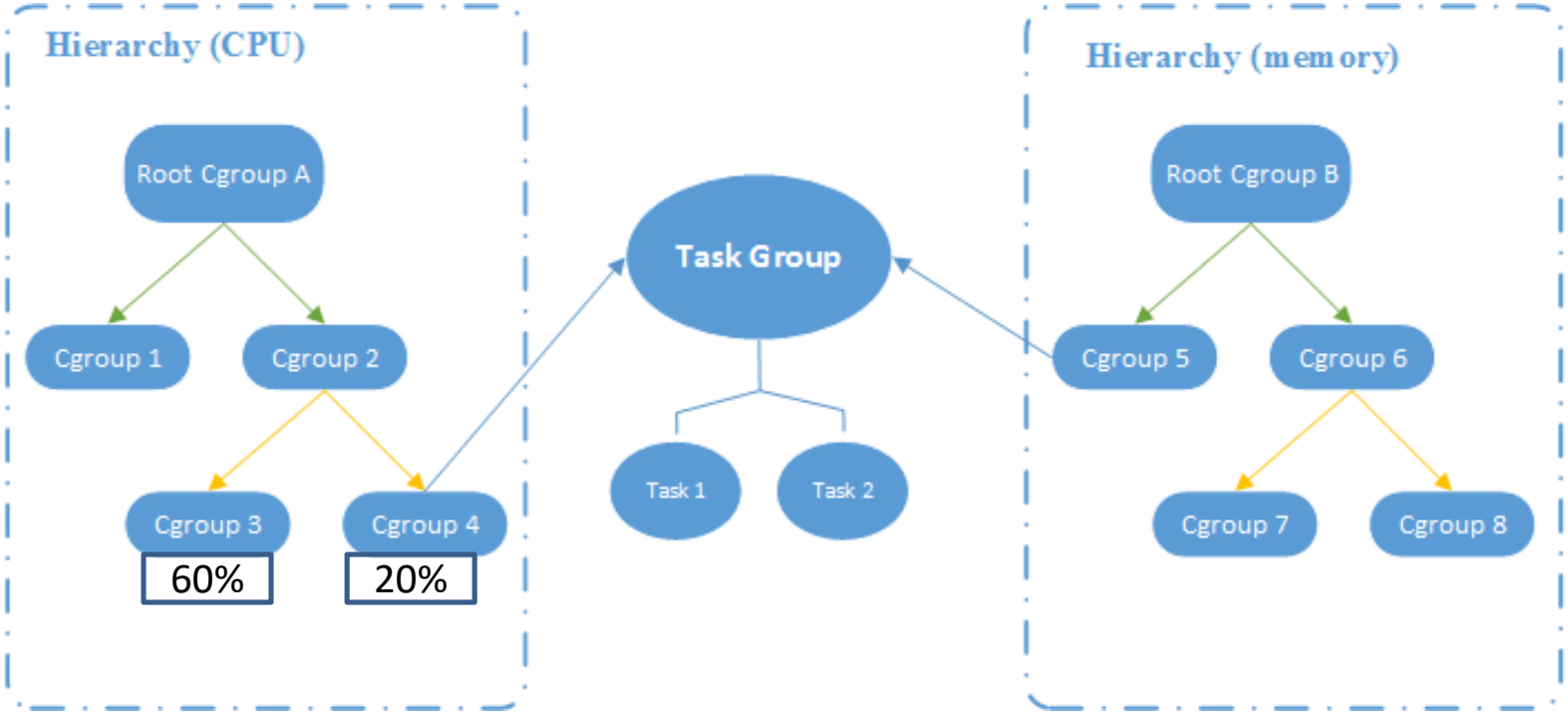
除了PID Namespace，Linux 操作系统还提供了 Mount、UTS、IPC、Network 和 User 这些 Namespace，用来对各种不同的进程上下文进行“障眼法”操作。

容器技术--CGroups

- **Linux Control Groups，一种linux内核功能**
- 用于限制、记录、隔离进程组所使用的物理资源
  - 如 cpu memory i/o 等等
- 实现将任意进程进行分组化管理
- Cgroups 给用户暴露出来的操作接口是文件系统
  - 即它以文件和目录的方式组织在操作系统的 /sys/fs/cgroup 路径下

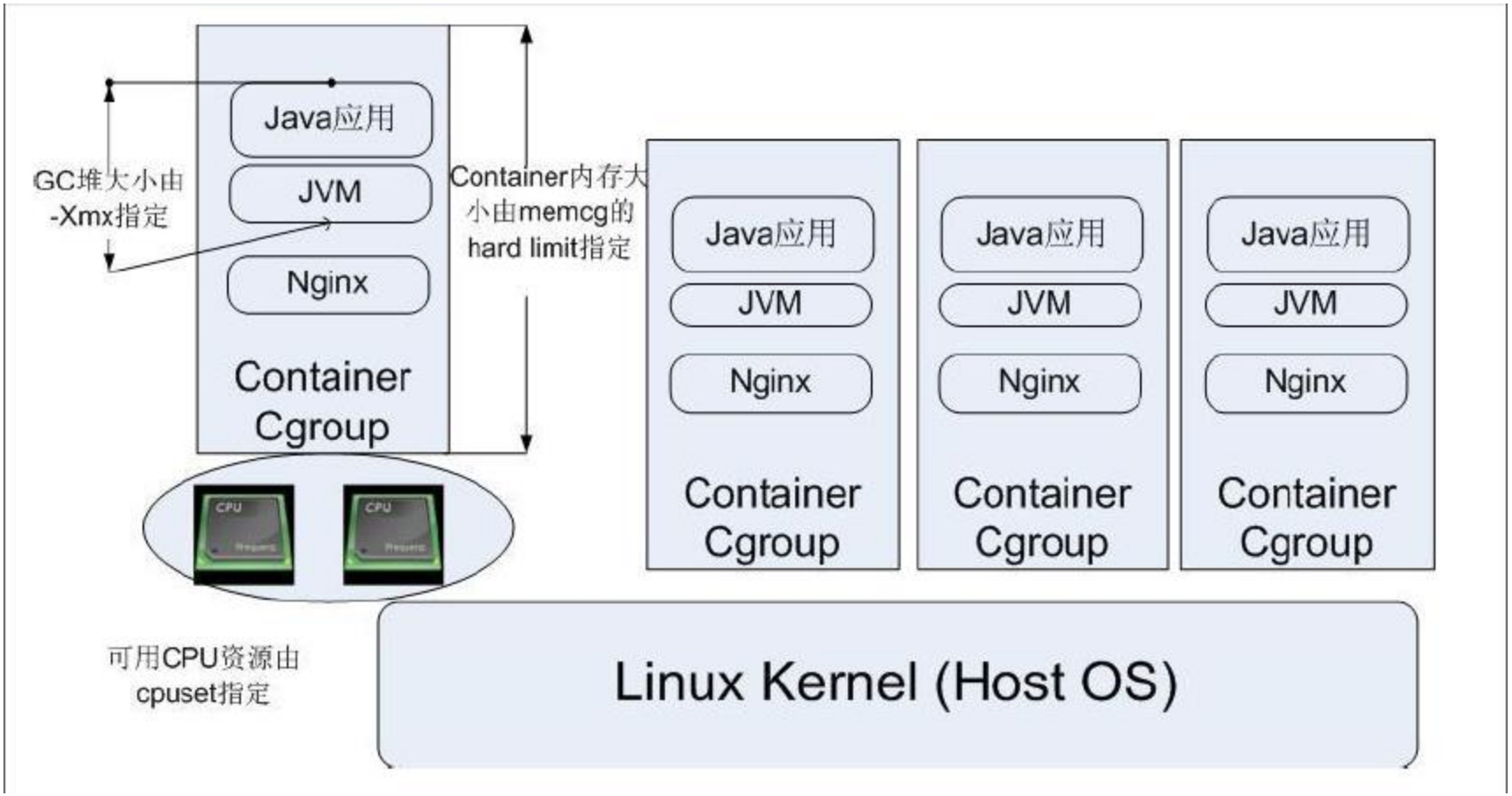
```
$ mount -t cgroup
cpuset on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cpu on /sys/fs/cgroup/cpu type cgroup (rw,nosuid,nodev,noexec,relatime,cpu)
cpuacct on /sys/fs/cgroup/cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpuacct)
blkio on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
memory on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
...
```

容器技术--CGroups



- 任务（task）：就是系统的一个进程；
- 控制族群（control group）：按照某种标准划分的一组进程，资源控制的基本单位；
- 层级（hierarchy）：控制族群树，子节点族群是父节点族群的孩子，继承父控制族群的特定的属性；
- 子系统（subsystem）：资源控制器，对应一种资源，比如 cpu 子系统就是控制 cpu 时间分配的一个控制器。

容器技术--CGroups

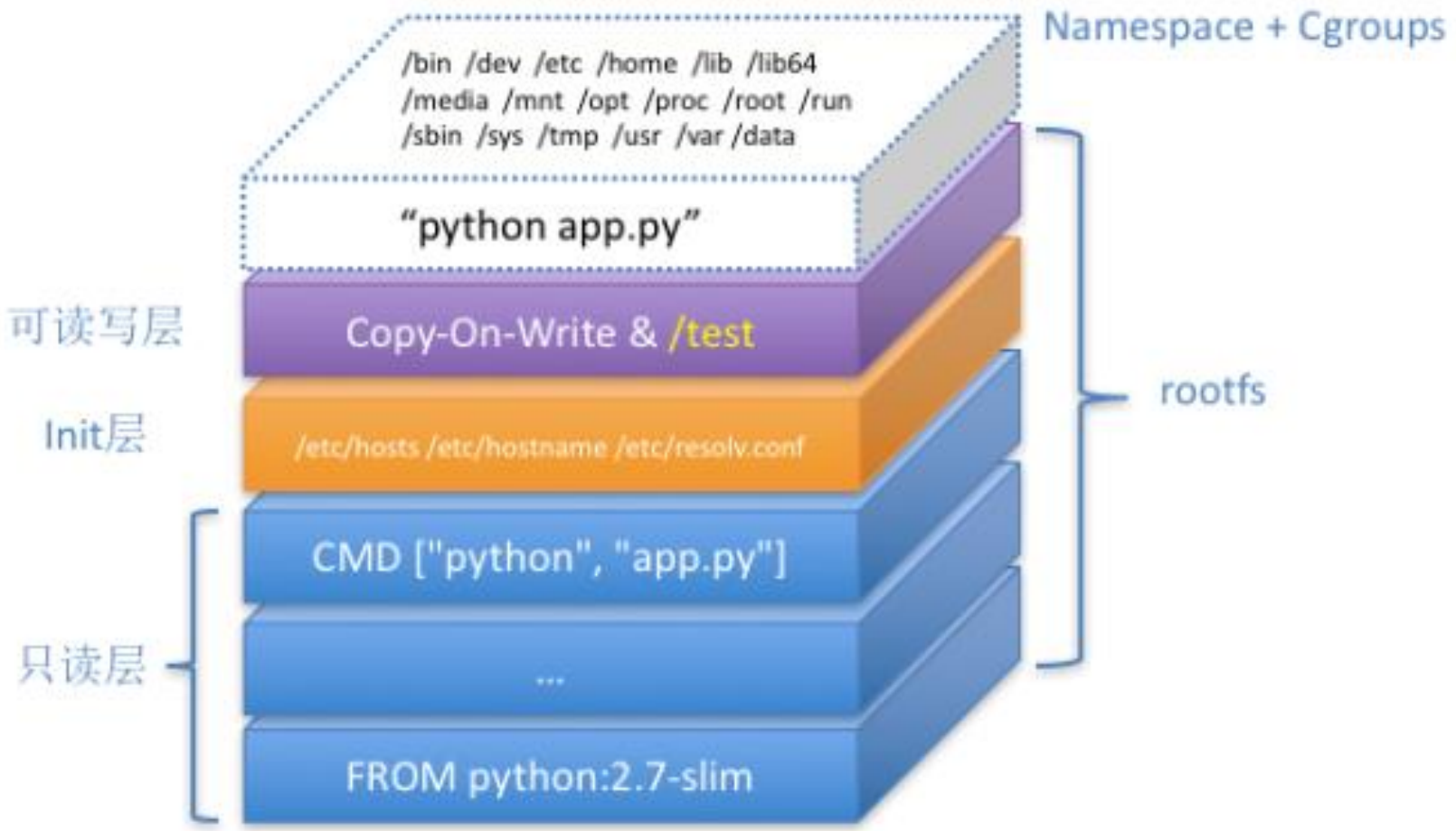


CGroup 典型应用架构图

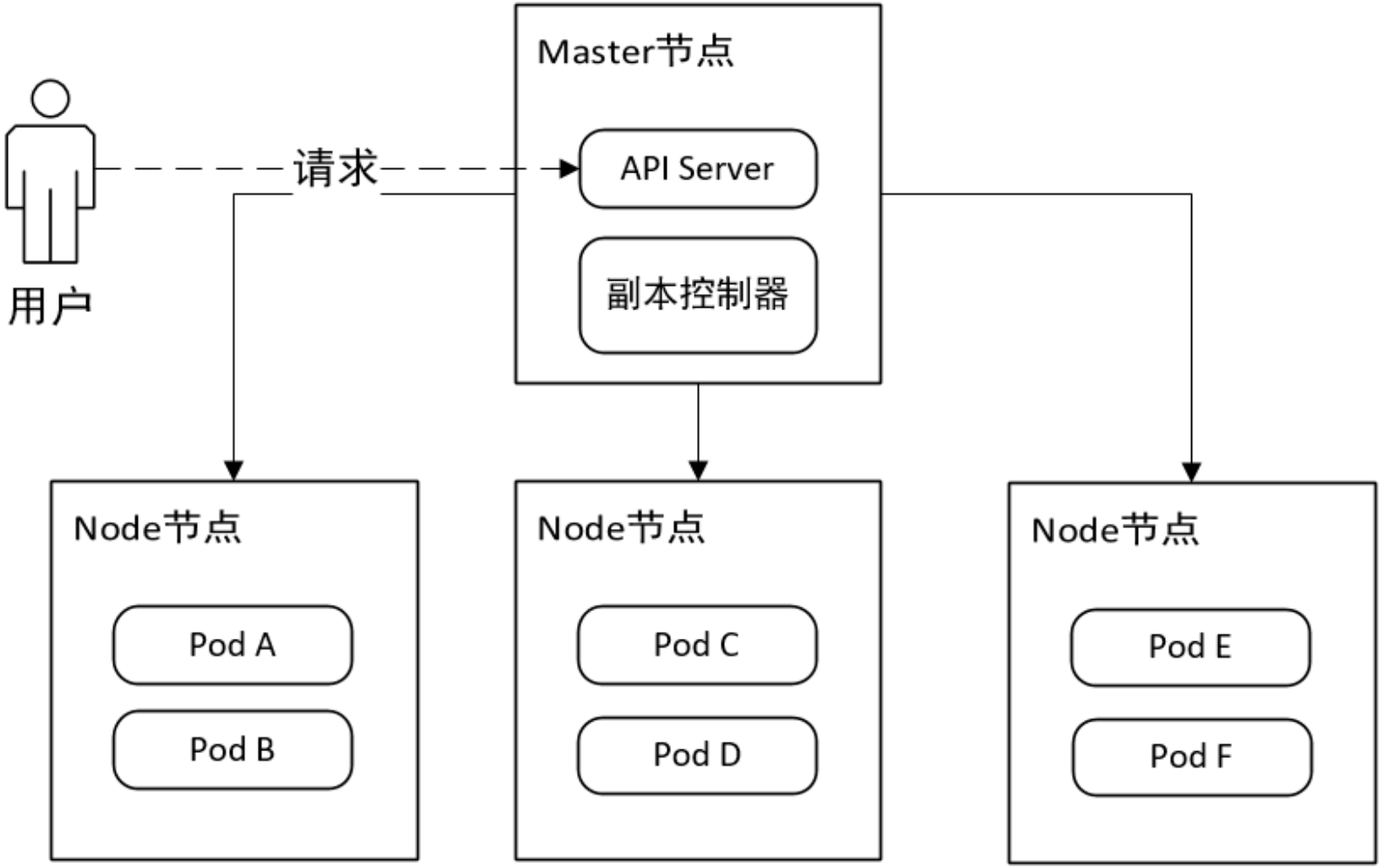
容器技术-- rootfs

- 就是所谓的“容器镜像”
- 挂载在容器根目录上、用来为容器进程提供隔离后执行环境的文件系统。
- 由三部分组成
  - 只读层：各以增量的方式分别包含了操作系统的一部分；
  - 可读写层：存放用户修改后的增量：增、删、改；
  - init层： Docker 项目单独生成的一个内部层，专门用来存放 /etc/hosts、/etc/resolv.conf 等信息。

Docker容器技术-- rootfs



Kubernetes集群架构



Kubernetes集群架构

- Master节点
  - 集群的核心节点
  - 负责管理与调度整个集群的 Pod
  - 设置了一个 API Server 用于接收集群外部的命令，以便于管理者对集群进行配置与管理。

- Node节点（也称作slave）
  - 集群中执行具体业务的节点
  - 负责接收来自 Master 的调度命令，并启动与部署相应的 Pod

Kubernetes核心概念

- **Pod**：
  - 一个 Pod 通常由多个互相协作的 Docker 容器组成
  - 是 Kubernetes 的基本管理单元，也是容器应用部署与运行的基本单元
- **Service**：
  - 一组提供相同服务的Pod的路由代理抽象
  - 用于解决 Pod 之间服务发现的问题
- **Replication Controller**：
  - Pod 的副本控制器，用于管理 Pod 的当前运行副本数量，并解决自动扩容缩容的问题。

Kubernetes核心概念

- **Label**:
  - 用于区分集群中 Pod、Service、Replication Controller 这些资源的键值对
  - 为集群提供管理信息。

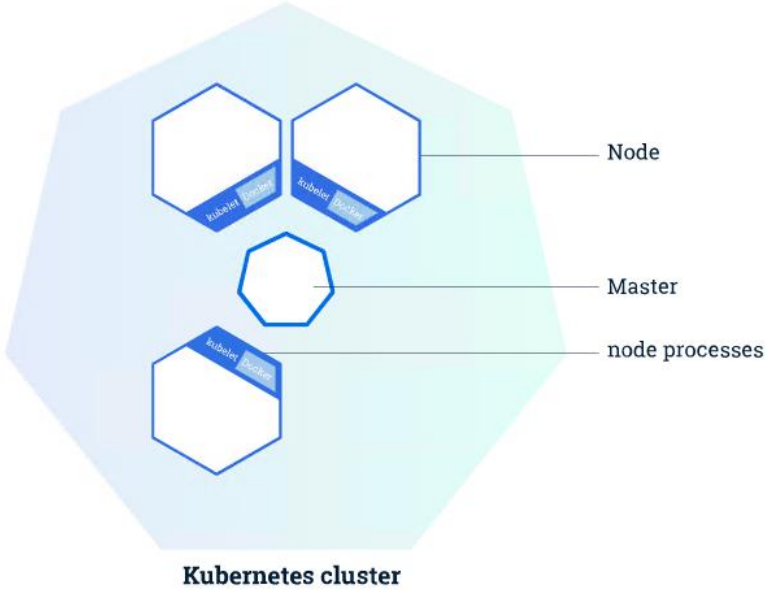
- **Namespace**:
  - 集群中的命名空间，
  - 不同命名空间中可以存在相同命名的资源，从而有利于设计逻辑上独立的资源组或者用户组

Kubernetes组件——Master的组件

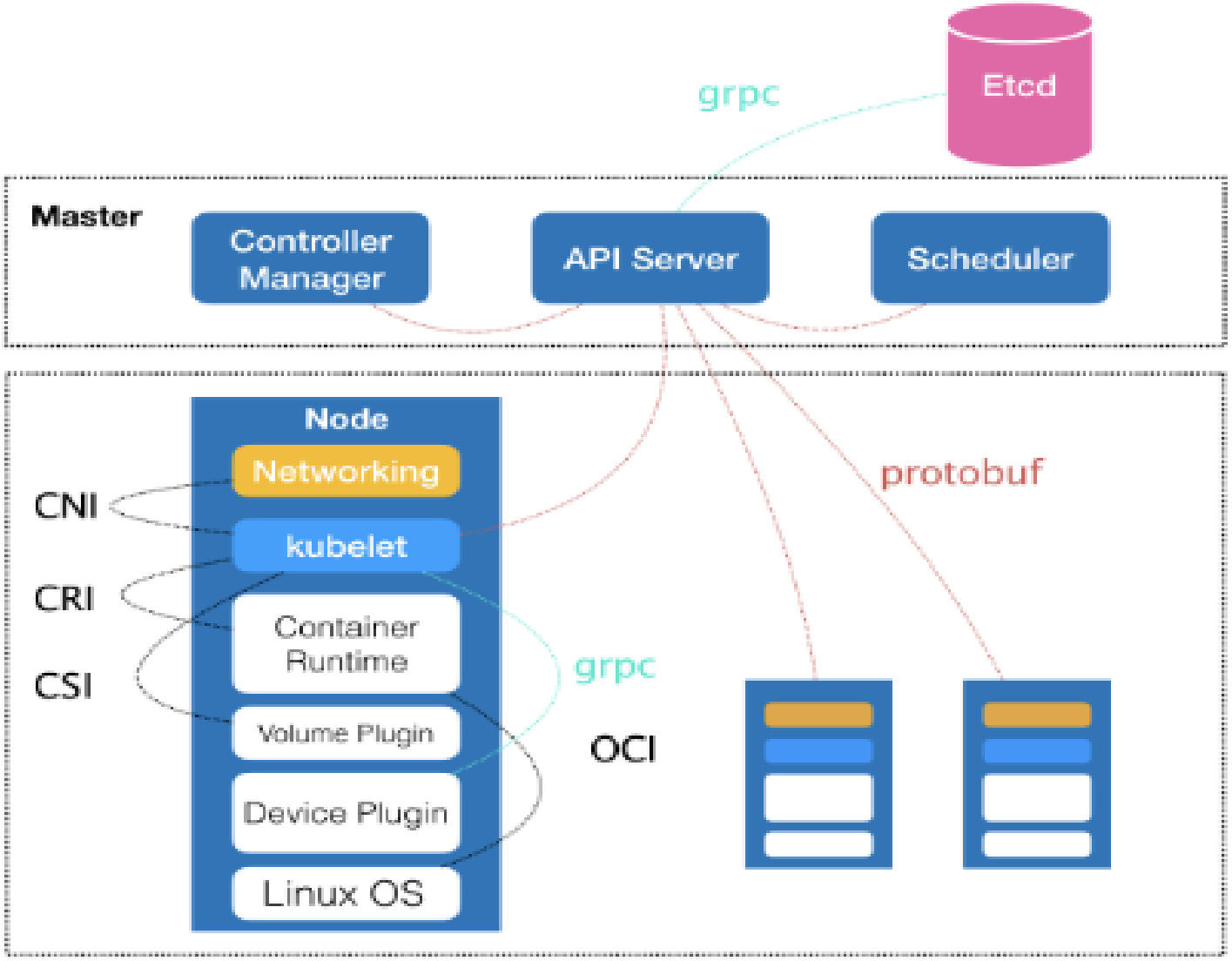
- **apiserver**：
  - 作为 kubernetes 集群的入口，封装了核心对象的增删改查操作，以 RESTFul 接口方式提供给外部客户和内部组件调用。
  - 它维护的 REST 对象将持久化到 etcd（一个分布式强一致性的 key/value 存储）。
- **scheduler**：
  - 负责集群的资源调度，为新建的 pod 分配宿主机。
- **controller-manager**：
  - 负责执行各种控制器，目前有两类：
  - **endpoint-controller**：维护 service 到 pod 的映射。
  - **replication-controller**：保证 replicationController 定义的复制数量与实际运行 pod 的数量总是一致的。

Kubernetes组件——Slave的组件

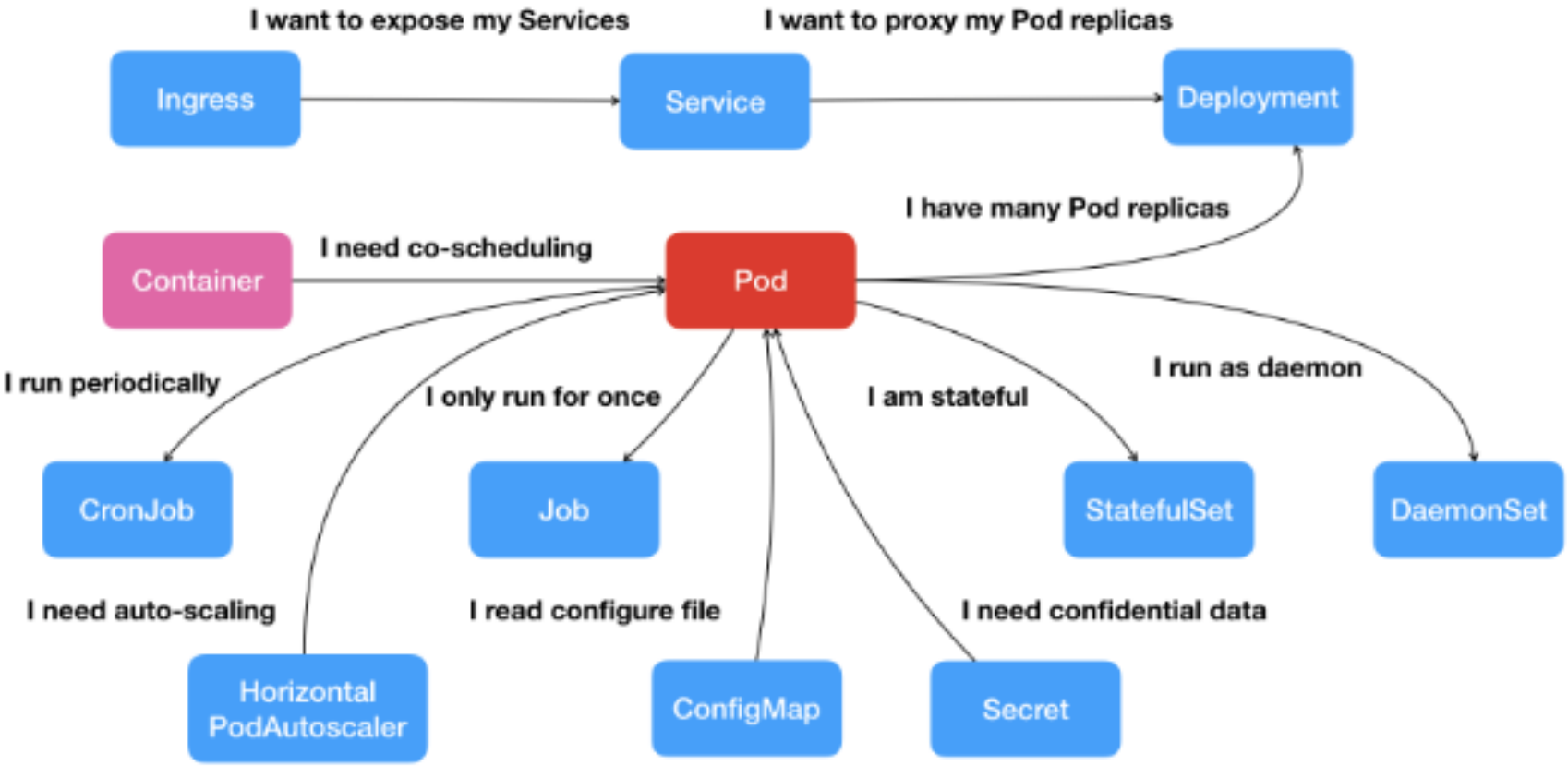
- **kubelet**：
  - 负责管控 Docker 容器，如启动/停止、监控运行状态等。
- **proxy**：
  - 负责为 pod 提供代理。



Kubernetes


Etcd
grpc
Master
Controller Manager
API Server
Scheduler
Node
Networking
kubelet
Container Runtime
Volume Plugin
Device Plugin
Linux OS
CNI
CRI
CSI
grpc
OCI
protobuf

Kubernetes—容器编排



小结

- 云计算是分布式计算的新技术阶段
- 云计算实现弹性、透明、高扩展、服务化的资源供给
- 虚拟化资源管理云计算的关键技术
- 虚拟机集群管理平台-OpenStack
- 容器集群管理平台-K8S

谢谢！

wuweig@mail.sysu.edu.cn