Chapter3 虚拟化技术

§3.1 虚拟机技术
§3.2 容器技术

§3.1 虚拟机技术

虚拟化是指将物理IT资源转换为虚拟IT资源的过程。

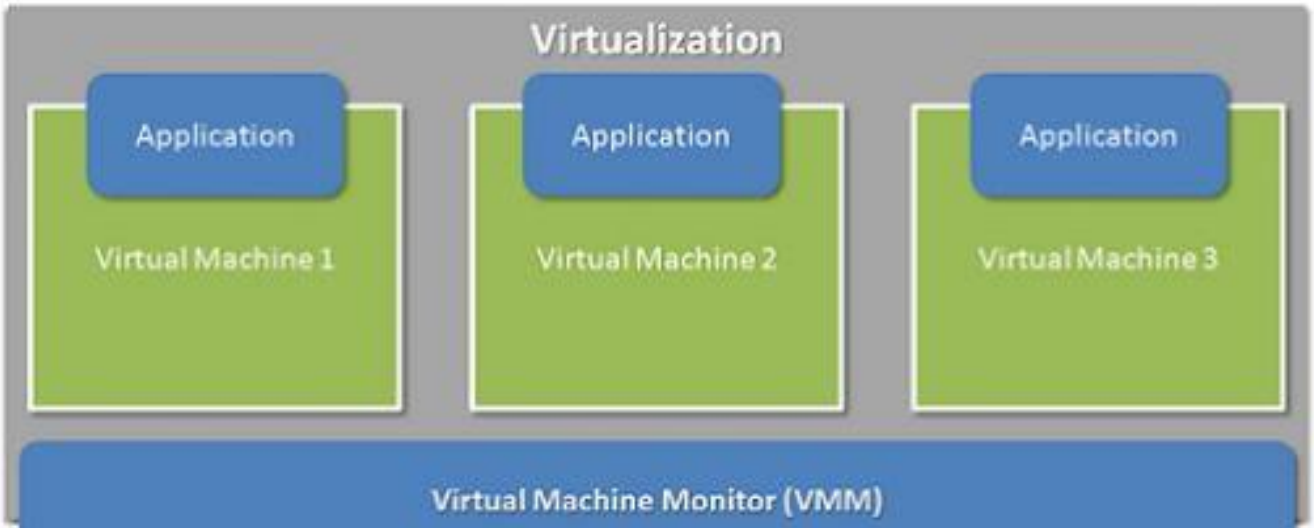
大多数IT资源都能被虚拟化，包括：

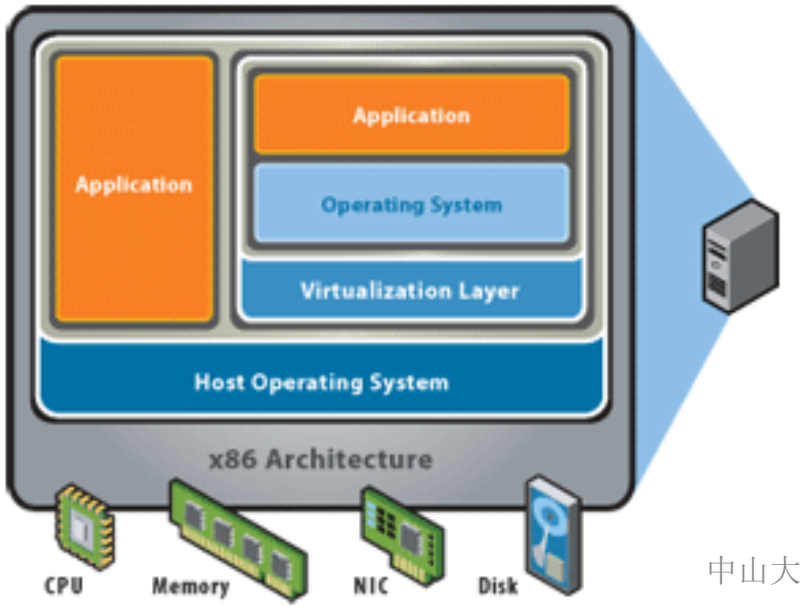
- 服务器（server）——一个物理服务器可以抽象为一个虚拟服务器。
- 存储设备（storage）——一个物理存储设备可以抽象为一个虚拟存储设备或一个虚拟磁盘。
- 网络（network）——物理路由器和交换机可以抽象为逻辑网络，如VLAN。
- 电源（power）——一个物理UPS和电源分配单元可以抽象为通常意义上的虚拟UPS。

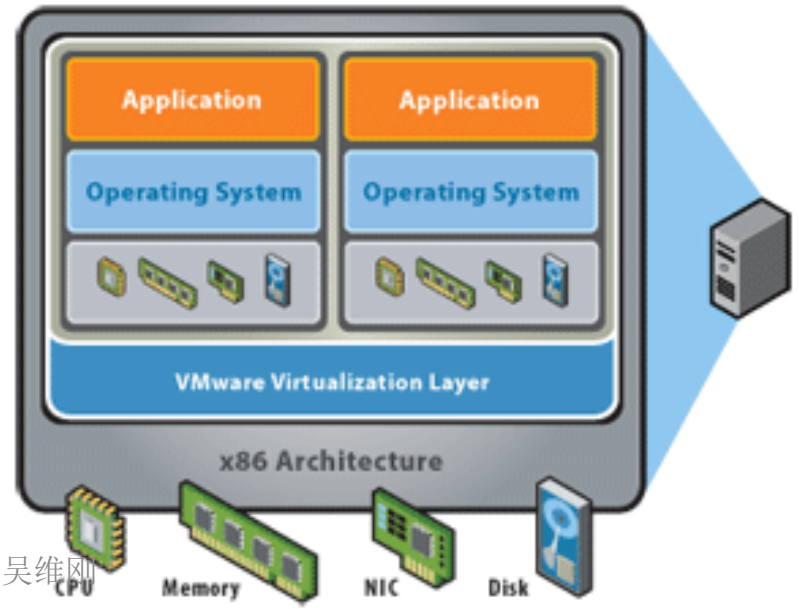
虚拟机

- 一台物理机→多台虚拟机（Virtual Machine, VM）
  - 虚拟CPU
  - 虚拟内存
  - 虚拟I/O
  - 独立软件
  - 数据隔离

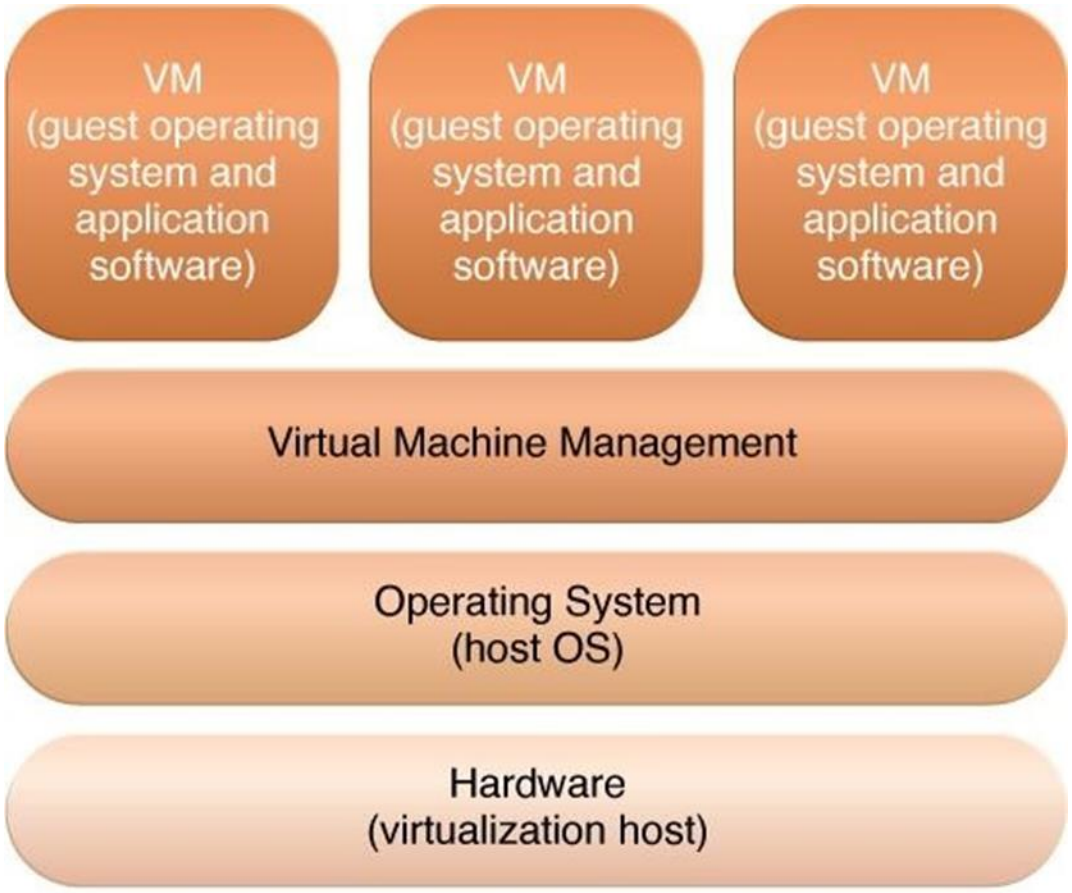
虚拟机架构


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

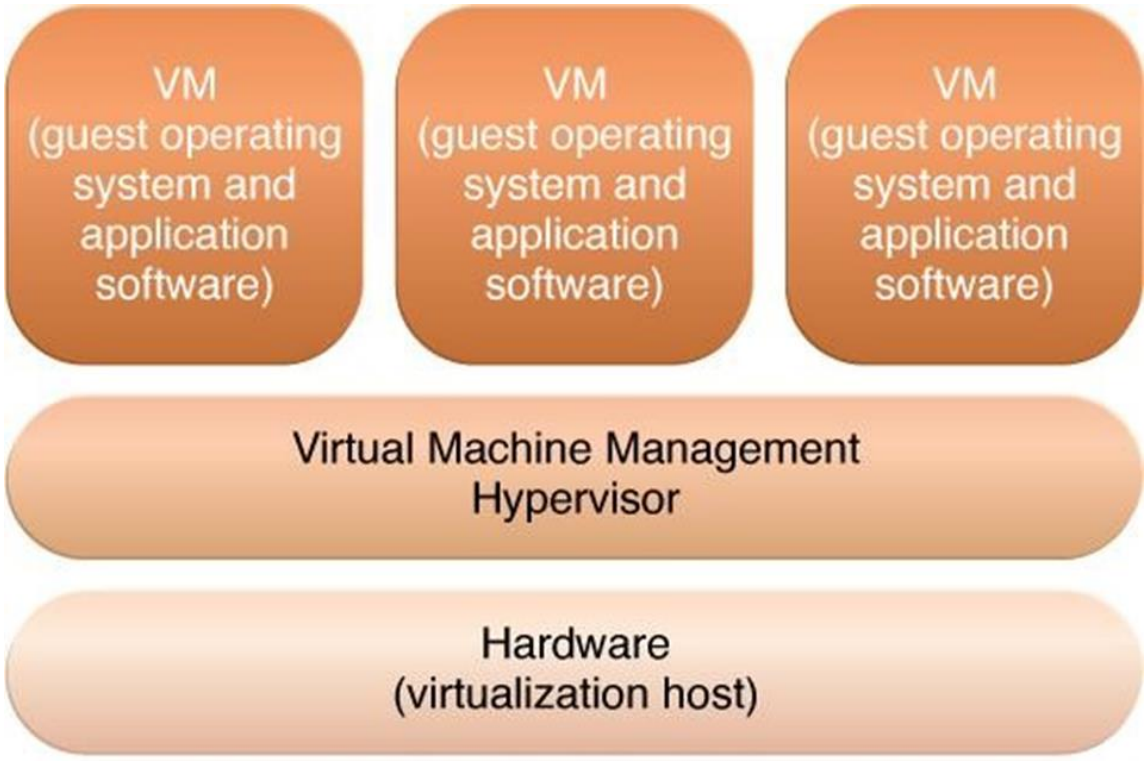

Application
Application
Operating System
Operating System
VMware Virtualization Layer
x86 Architecture
CPU
Memory
NIC
Disk

基于操作系统的虚拟化--寄生架构（Hosted）



基于操作系统虚拟化的逻辑分层。
其中，VM首先被安装在完整的宿主操作系统上，然后被用于产生虚拟机

基于硬件的虚拟化--裸金属架构（Bare-metal）



基于硬件虚拟化的逻辑分层，不再需要另一个宿主操作系统

Hosted vs. Bare-metal

Bare-metal (Type I)	Hosted (Type II)
效率高 （不受Host OS影响、I/O优化）	
安全性高	
	使用方便
	功能丰富（e.g. 3D加速）
适于服务器系统	适于桌面系统

OpenStack

- 既是一个**社区**，也是一个**项目**和一个**开源软件**
- 提供了一个部署云的操作平台或工具集
- 易于构建虚拟计算或存储服务的云
- 提供可扩展、灵活的云计算：公有云、私有云；大云、小云

主要功能

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

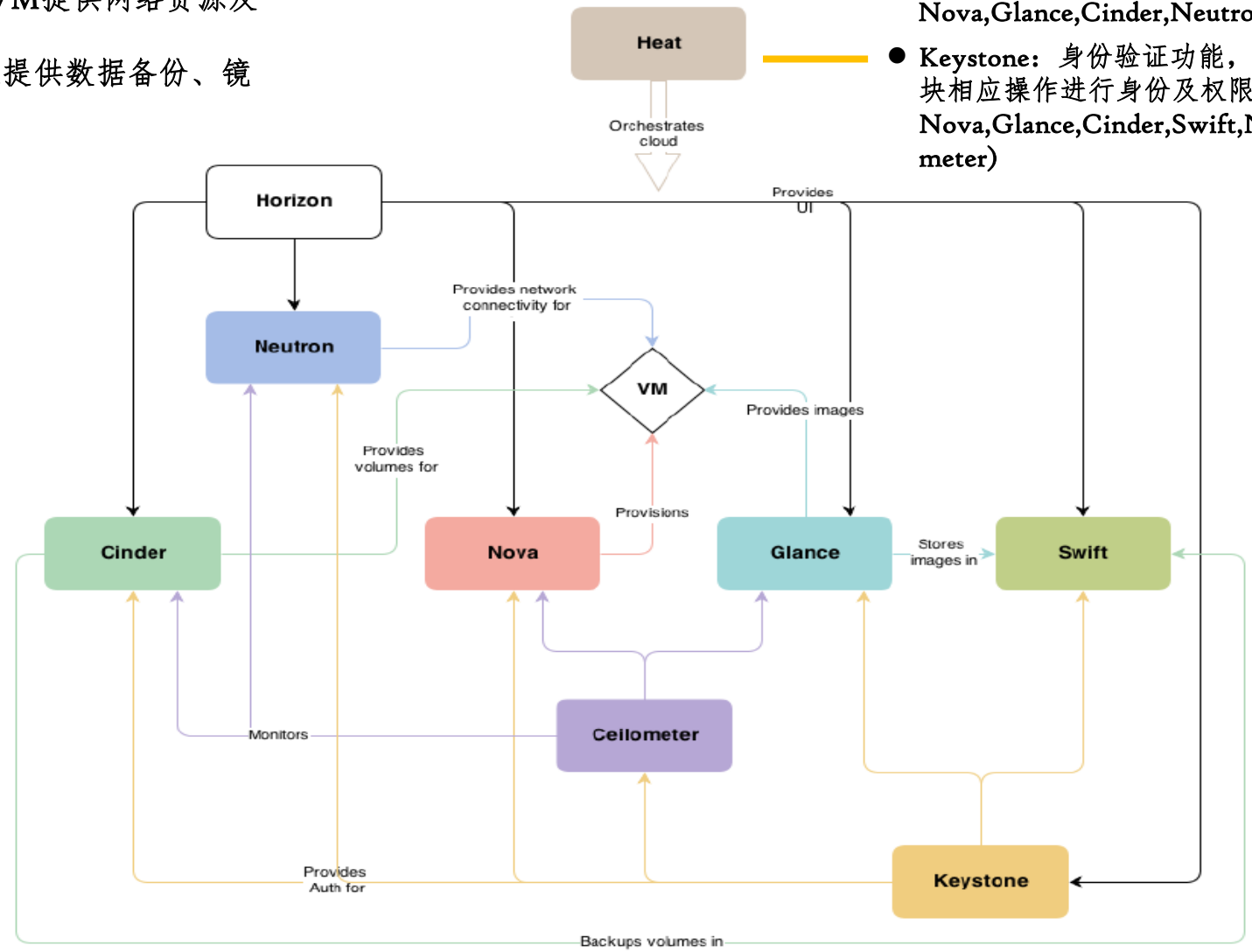
网络资源管理

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

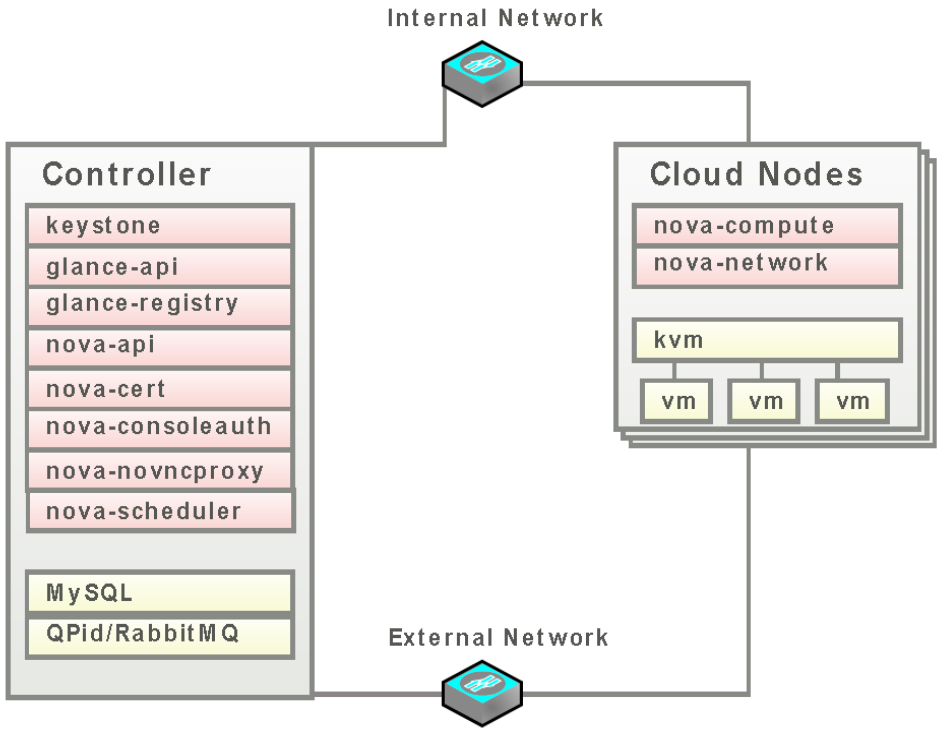
OpenStack主要服务/组件

- 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 最简物理架构



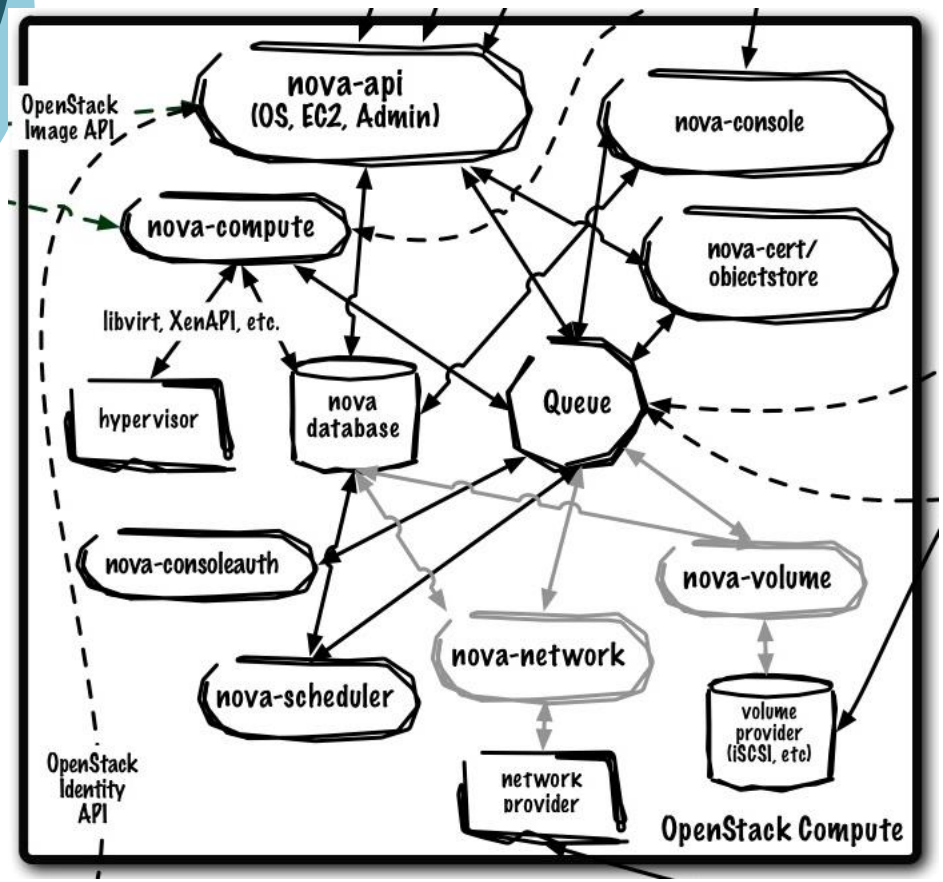
2个节点：

- 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：

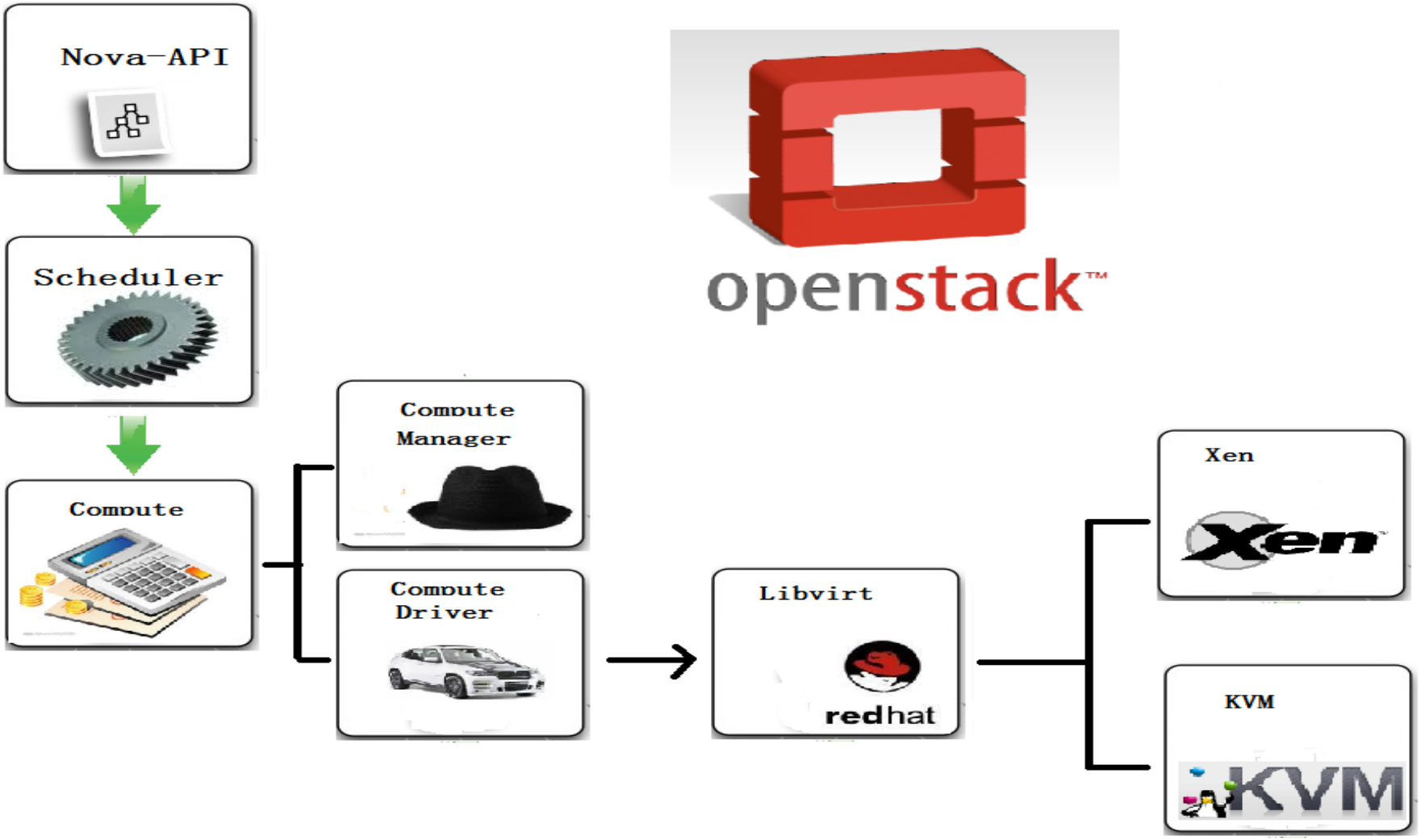
- Internal Network(内部网络)
  - 用于提供Provider网络(VM to Provider)
  - 用于tenant网络(VM to VM)
- External Network(外部网络）
  - 用于外部用户与VM通信及控制(VM to Internet)

Nova的架构

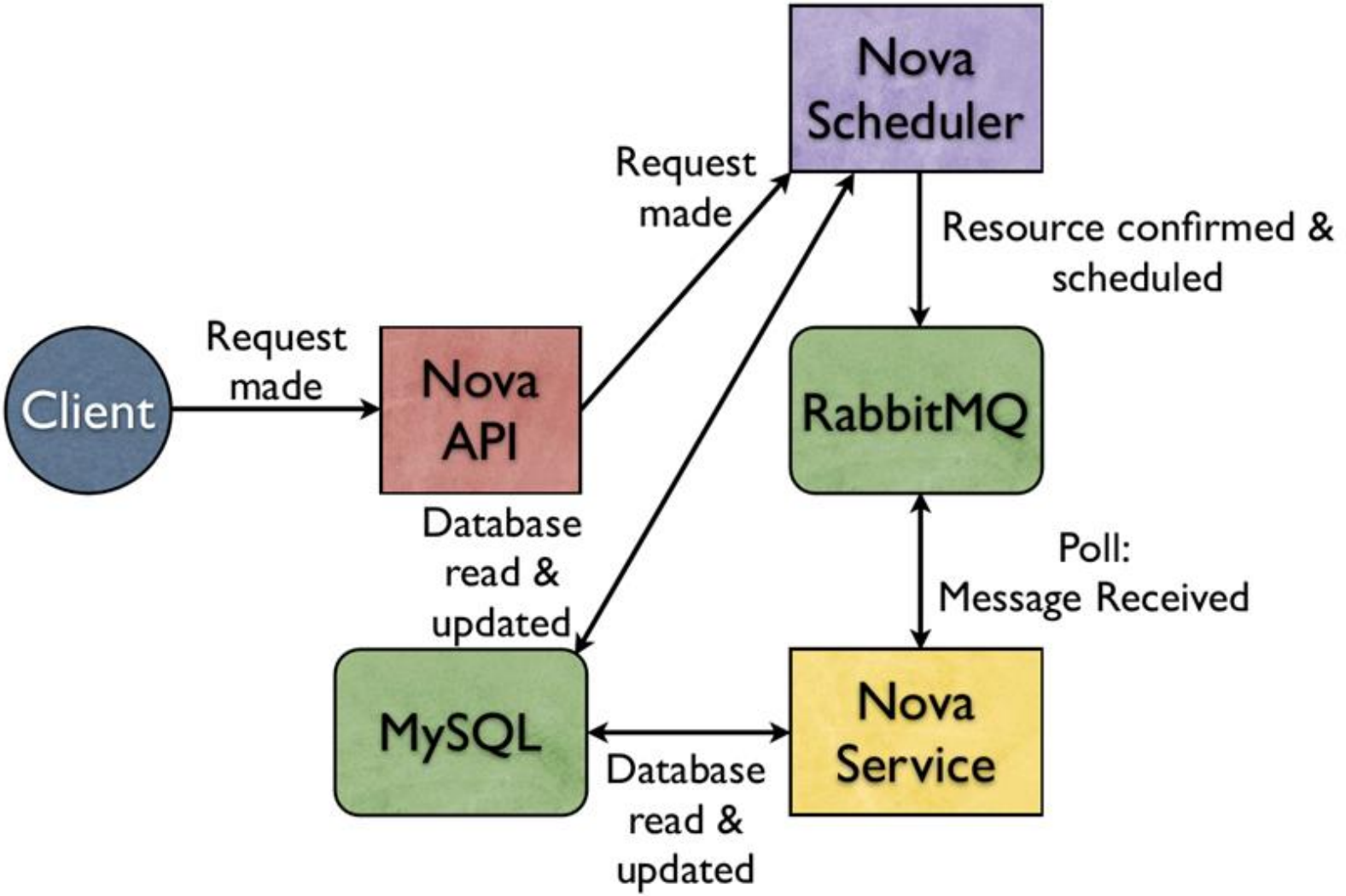


- Nova-API:对外统一提供标准化接口。接受和响应最终用户的请求,实现与其他各逻辑模块的通讯与服务提供。
- Nova-Scheduler:从队列上得到一个虚拟机实例请求并且决定它应该在哪里运行。
- Queue：提供了守护进程之间传递消息的中央枢纽，还是与其他各逻辑模块间通信的连接枢纽
- Nova-Database:存储云基础设施的编译时和运行时的状态，主要是sqlite3（只适用于测试和开发工作），MySQL和PostgreSQL。
- Nova-Compute:一个人工守护进程，对虚拟机生命周期的管理,支持多种虚拟机。
- Nova还提供控制台的服务，让最终用户通过代理服务器访问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

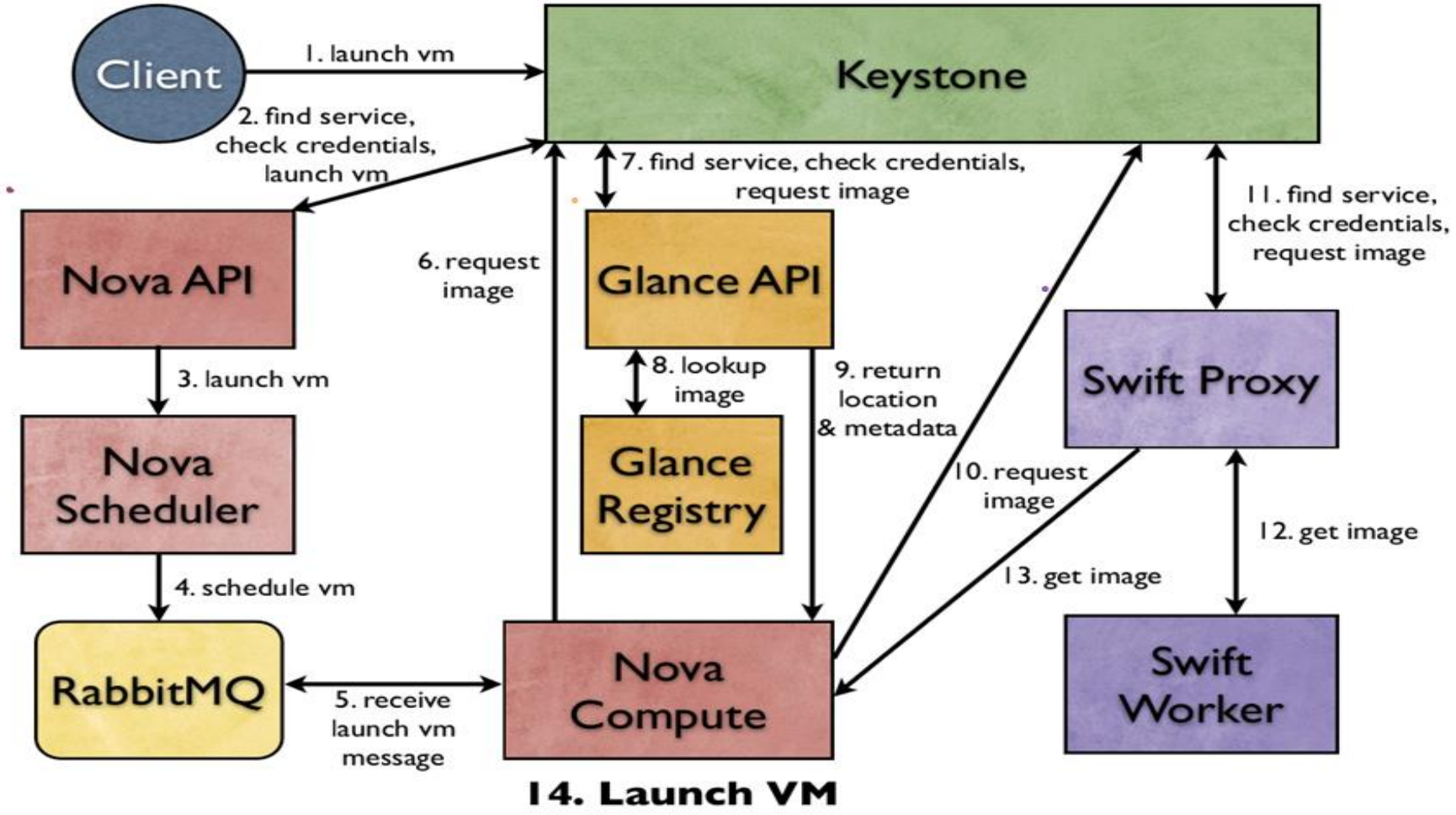
Nova工作流程


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

Nova处理过程


Nova Scheduler
Request made
Resource confirmed & scheduled
Client
Request made
Nova API
RabbitMQ
Database read & updated
Poll: Message Received
MySQL
Nova Service
Database read & updated

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 调度器

Nova支持的调度器和过滤器

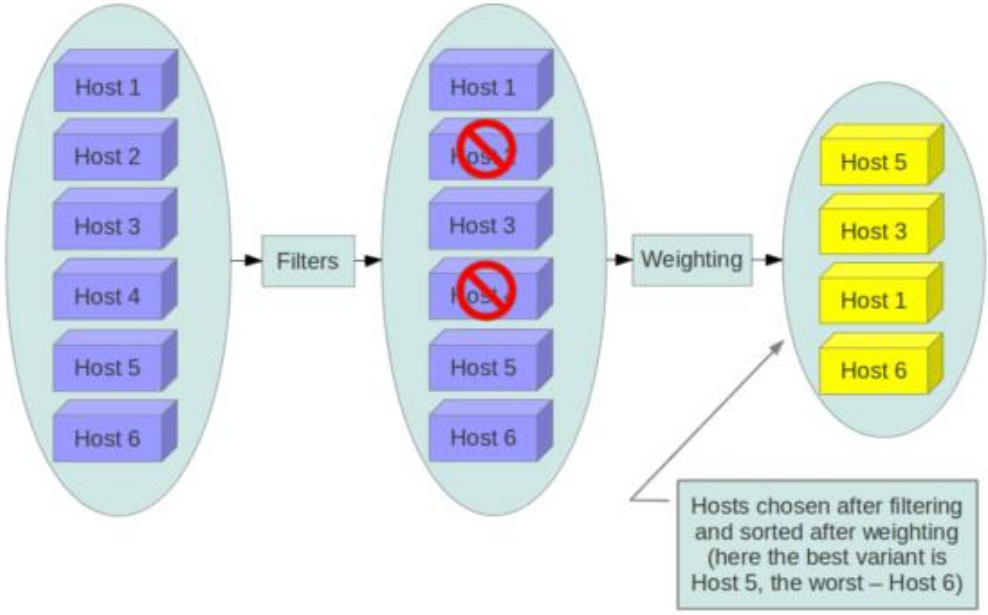
Nova支持的调度器
•ChanceScheduler随机调度器
•SimpleScheduler简单调度器
•FilterScheduler过滤调度器
•MultiScheduler多重调度器

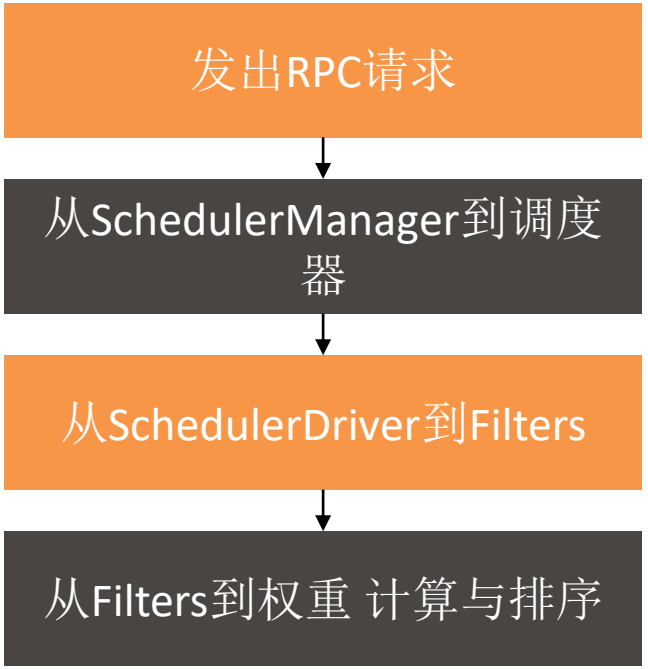
Nova支持的过滤器
•All HostFilter
•ImagePropertiesFilter
•AvailabilityZoneFilter
•ComputeFilter
•CoreFilter
•IsolateHostFilter
•RamFilter
•SimpleCIDRAffinityFilter
•DifferentHostFilter
•SameHostFilter

Nova Scheduler工作流程

FilterScheduler的工作流程如图所示：

FilterScbeduler首先使用指定的Filters (过滤器）得到符合条件的主机， 比如内存使用率小于50%,然后对得到的主机列表计算权重并排序， 获得最佳的一个。





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。

1. Instance

对象Instance：描述一个虚拟机特征与状态的一些信息或结构。

```
# nova/objects/instance.py

class Instance(base.NovaPersistentObject, base.NovaObject):
    fields = {
        'id': fields.IntegerField(),

        'user_id': fields.StringField(nullable=True),
        'project_id': fields.StringField(nullable=True),
            ...
        'power_state': fields.IntegerField(nullable=True),
        'vm_state': fields.StringField(nullable=True),
        'task_state': fields.StringField(nullable=True),

        'memory_mb': fields.IntegerField(nullable=True),
        'vcpus': fields.IntegerField(nullable=True),
        'root_gb': fields.IntegerField(nullable=True),
        'ephemeral_gb': fields.IntegerField(nullable=True),
        'ephemeral_key_uuid': fields.UUIDField(nullable=True),

        'host': fields.StringField(nullable=True),
        'node': fields.StringField(nullable=True),

        'instance_type_id': fields.IntegerField(nullable=True),

        'user_data': fields.StringField(nullable=True),
        'reservation_id': fields.StringField(nullable=True),
        'scheduled_at': fields.DateTimeField(nullable=True),
        'launched_at': fields.DateTimeField(nullable=True),
        'terminated_at': fields.DateTimeField(nullable=True),

        'availability_zone': fields.StringField(nullable=True),

        'display_name': fields.StringField(nullable=True),
        'display_description': fields.StringField(nullable=True),
        ...
        }
```

Nova Compute

2. Flavor

Flavor：虚拟机在创建时为其预先指定一组资源的设置，包括了计算、存储、内存等能力的大小。

每个虚拟机Instance对象的 instance_type _id字段就代表了它的Flavor。

Flavor默认包含Disk、Memory、 VCPU 、RootDisk, Ephemeral Disk、Swap等信息。

执行命令 "nova flavor-list"可以查看Nova中默认的 一些Flavor内容。

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	1			1	1.0	N/A
2	m1.small	2048	20			1	1.0	N/A
3	m1.medium	4096	40			2	1.0	N/A
4	m1.large	8192	80			4	1.0	N/A
42	m1.nano	64	0			1	1.0	N/A
451	m1.heat	512	0			1	1.0	N/A
5	m1.xlarge	16384	160			8	1.0	N/A
84	m1.micro	128	0			1	1.0	N/A

Nova Compute

3. 虚拟机状态

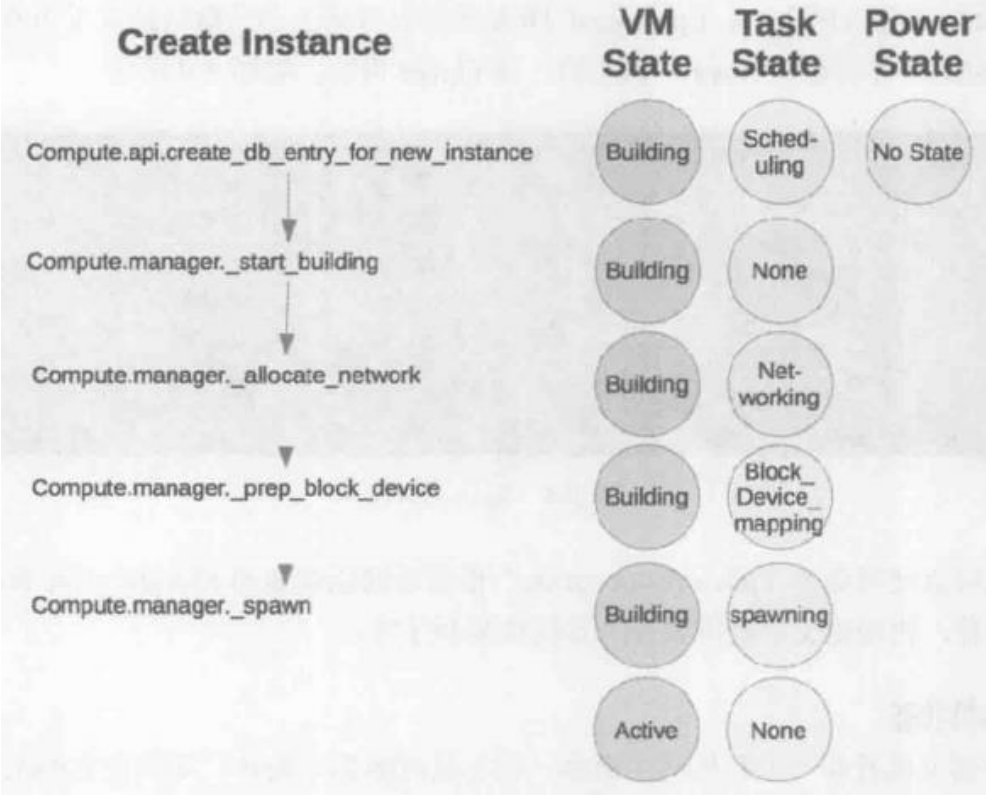
Instance对象中有三个字段与描述状态相关：

1	power_state: 使用Libvirt等Virt Driver提供的接口从Hypervisor中获取的虚拟机的状态，比如Running、Shutdown、NoState等 从所有nova-compute服务正常运行的节点中随机选择。
2	vm_state: 虚拟机的稳定状态 如果正在执行某个ComputeAPI，或者说执行某个任务，则vm_state就是任务结束时用户所预期的那个状态 比如Active表示运行良好。
3	task_state: 一种过渡状态，与ComputeAPI的执行紧密相关 表示虚拟机正在运行什么任务，只有正在执行的任务才有task_state，处于vm_state的虚拟机task_state为None。

Nova Compute

Compute 服务

创建虚拟机的过程中，需要申请网络、准备块设备、调用VirtDriver创建虚拟机，那么上述三种状态的变化就如图所示。



Nova Compute

Compute 服务

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。

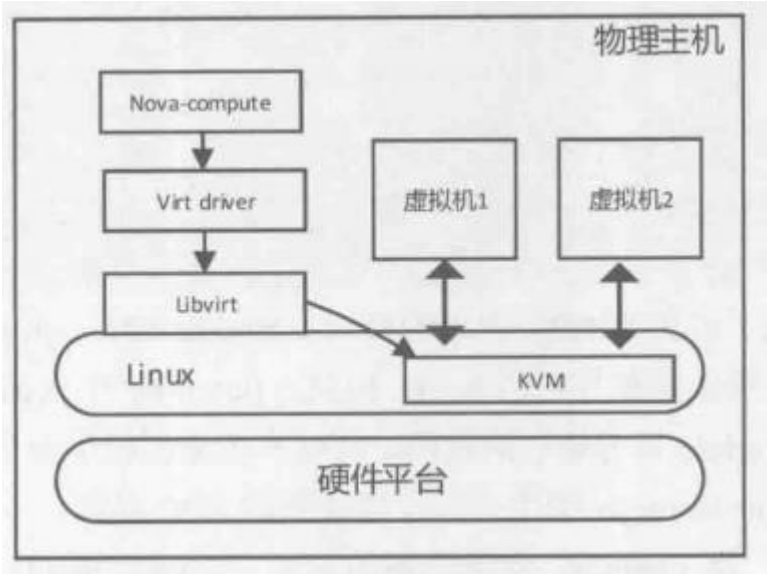
各种Hypervisor的支持通过Virt Driver的方式实现，比如Libvirt Driver，各种Virt Driver的实现位于nova/virt目录。

```
.
├── disk
├── hyperv
├── ironic
├── libvirt
├── vmwareapi
└── xenapi
```

Nova Compute

- **Compute 服务**

除了提供一些工具函数的disk目录，以及提供Baremetal支持的ironic目录，目前Nova中共实现了Hyperv、Libvvirt、Vmware以及XenAPI四种Virt Driver。



nova-compute通过Libvirt 与 KVM交互

Nova Compute

Resource Tracker

Nova使用ComputeNode对象保存计算节点的配置信息以及资源使用状况。 在数据库中存储主机的资源使用情况，包括内存、CPU、磁盘等。

每创建、迁移、删除一个虚拟机，都要更新数据库中的相关的内容。

为nova-scheduler提供依据，及系统监控等。。

Nova-compute为每一个主机创建一个ResourceTracker对象

任务就是更新ComputeNode对象在数据库中对应的表compute_nodes。

有两种更新数据库中资源数据的方式：一是使用ResourceTracker的Claim机制，二是使 用周期性任务(PeriodicTask)。

Nova Compute

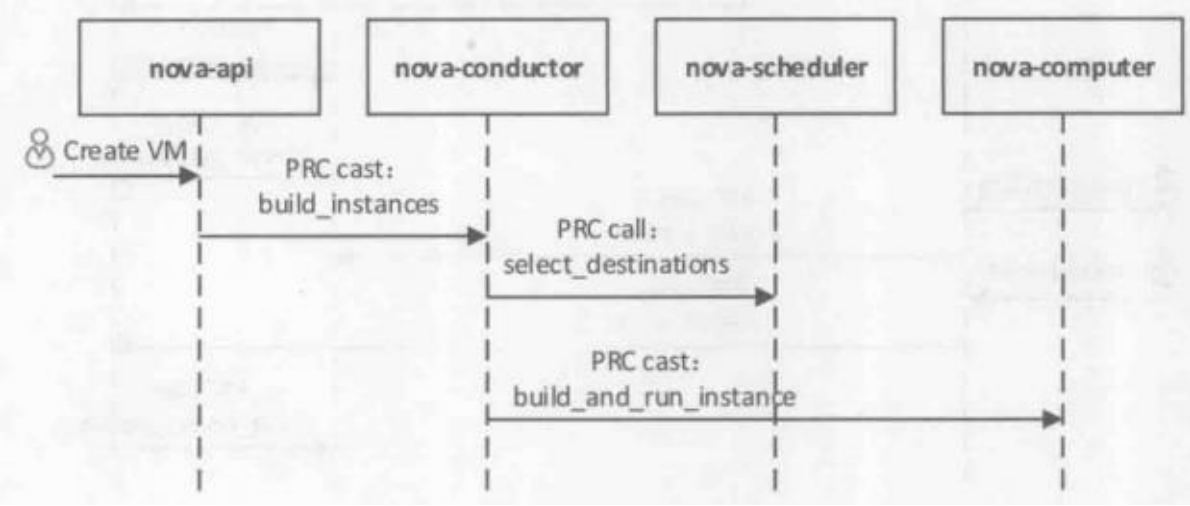
典型工作流程：创建虚拟机

创建一个虚拟机至少需要指定的参数有三个：虚拟机名字、镜像、Flavor。

执行“nova image-list”命令可以看到目前可用的虚拟机镜像。

比如创建一个名为test的虚拟机，使用flavor类型m1.tiny

```
+--------------------------------------+------+--------+------------+-------------+-----------------+
| ID                                   | Name | Status | Task State | Power State | Networks        |
+--------------------------------------+------+--------+------------+-------------+-----------------+
| 2dd52014-78f5-4d9b-a65e-8190421d89d6 | test | ACTIVE | -          | Running     | private=10.0.0.2 |
+--------------------------------------+------+--------+------------+-------------+-----------------+
```

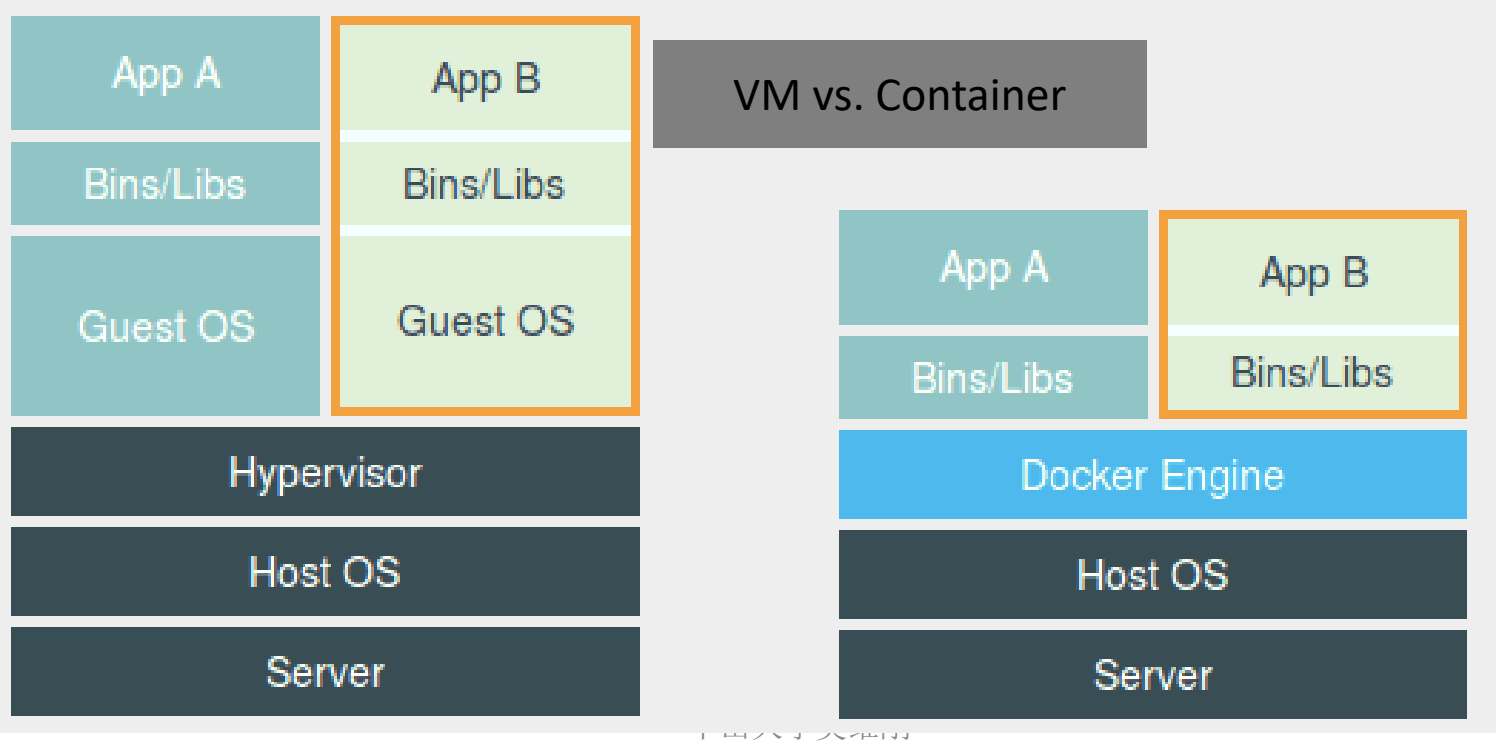


小结

- 虚拟机是计算资源虚拟化的基本技术
- 虚拟机技术本身有很多不同类型
  - 不同架构
  - 不同OS：Windows，Linux
  - 不同具体产品实现
- 虚拟机管理技术
  - 管理物理集群及虚拟机
  - 如：OpenStack、Windows Azure等

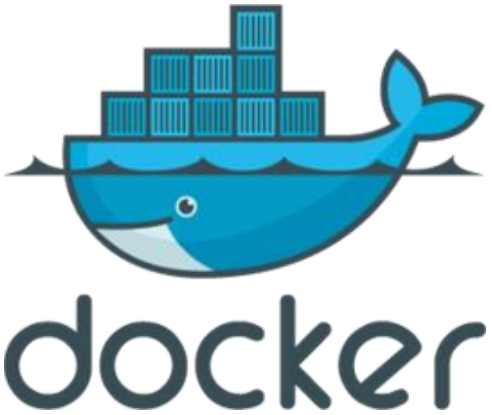
§3.2 容器技术

- 容器，Container：轻量级虚拟化
- 在用户空间层次进行隔离、虚拟化



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。

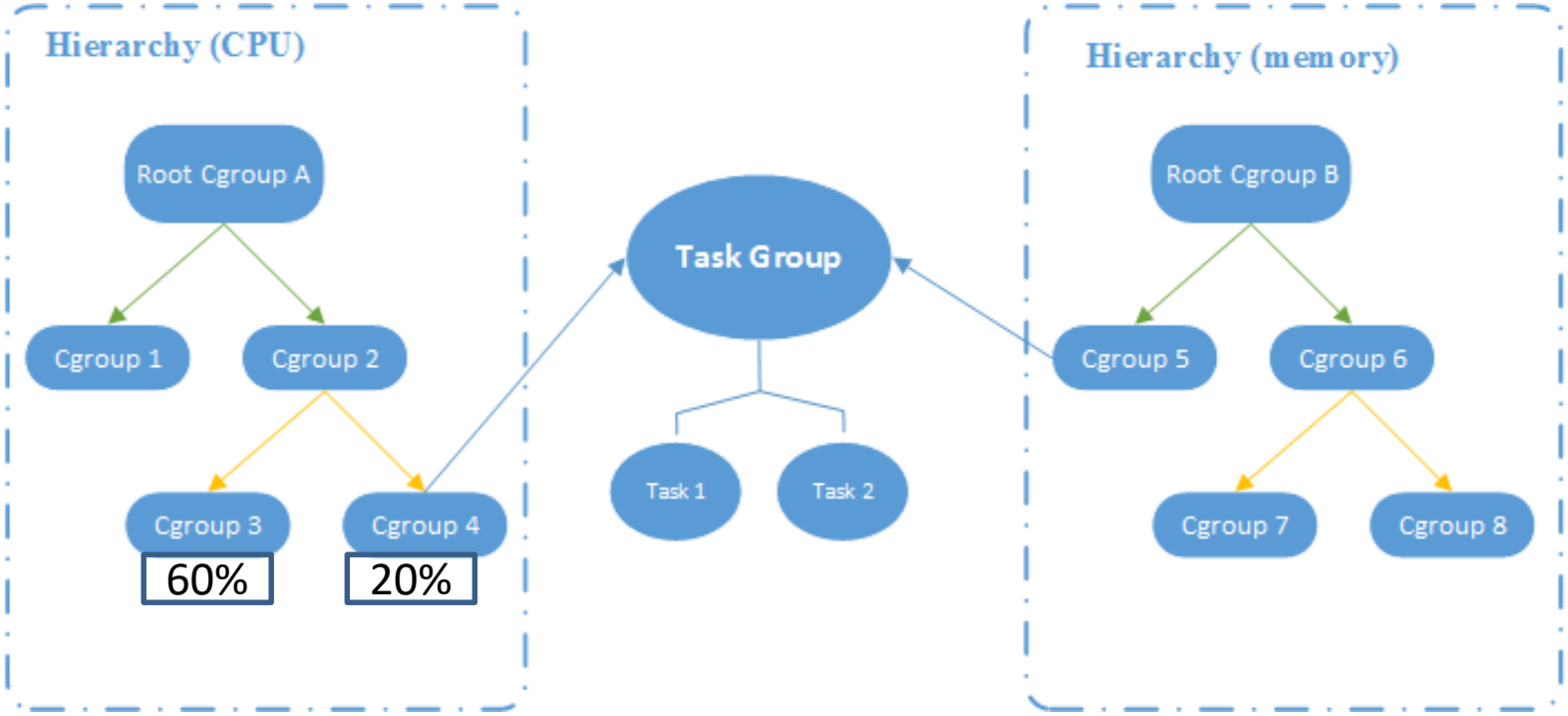
除了PID Namespace，Linux 操作系统还提供了 Mount、UTS、IPC、Network 和 User 这些 Namespace，用来对各种不同的进程上下文进行“障眼法”操作。

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

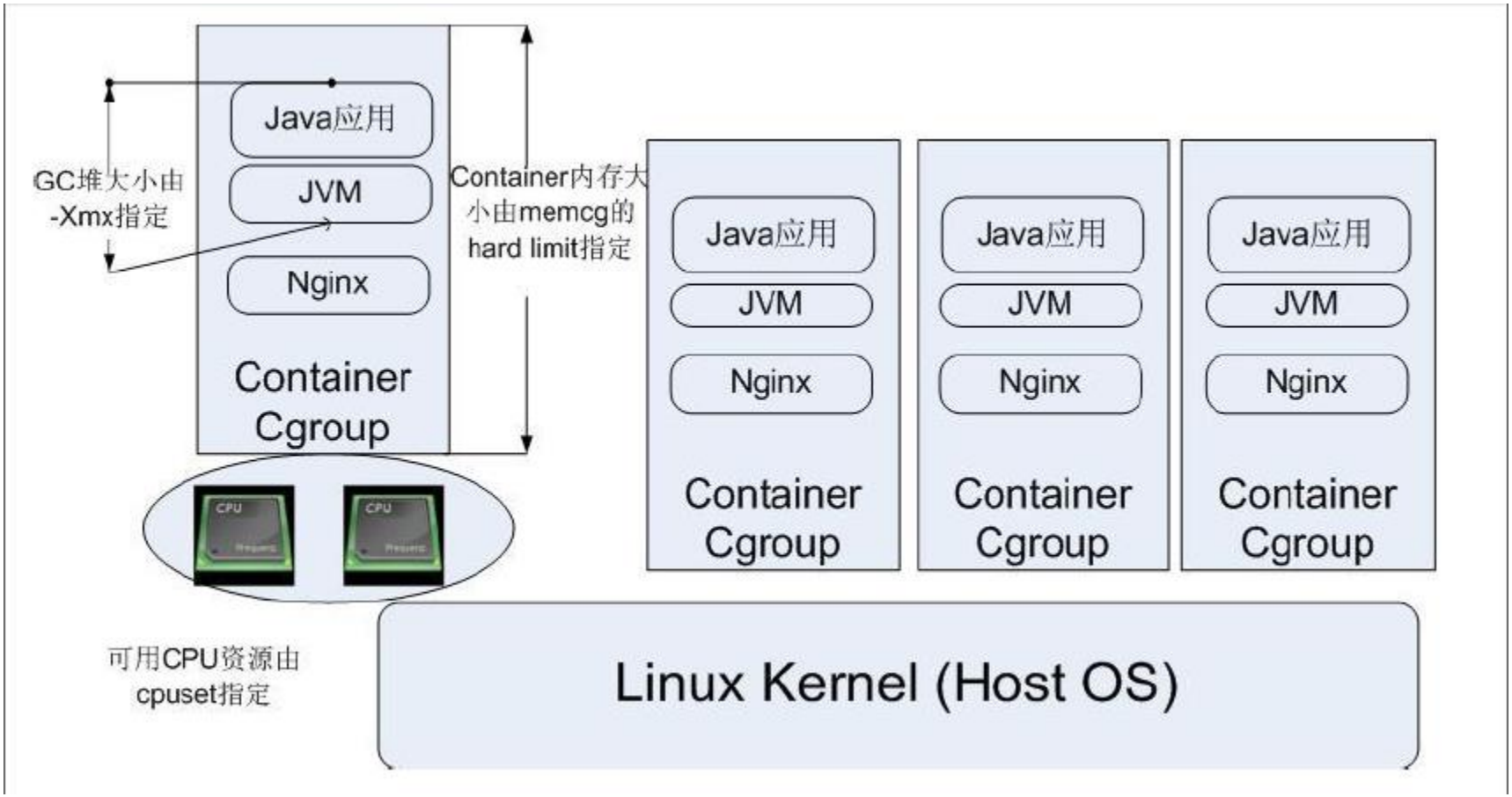
```
$ mount -t cgroup
cpuset on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cpu on /sys/fs/cgroup/cpu type cgroup (rw,nosuid,nodev,noexec,relatime,cpu)
cpuacct on /sys/fs/cgroup/cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpuacct)
blkio on /sys/fs/cgroup/blkio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

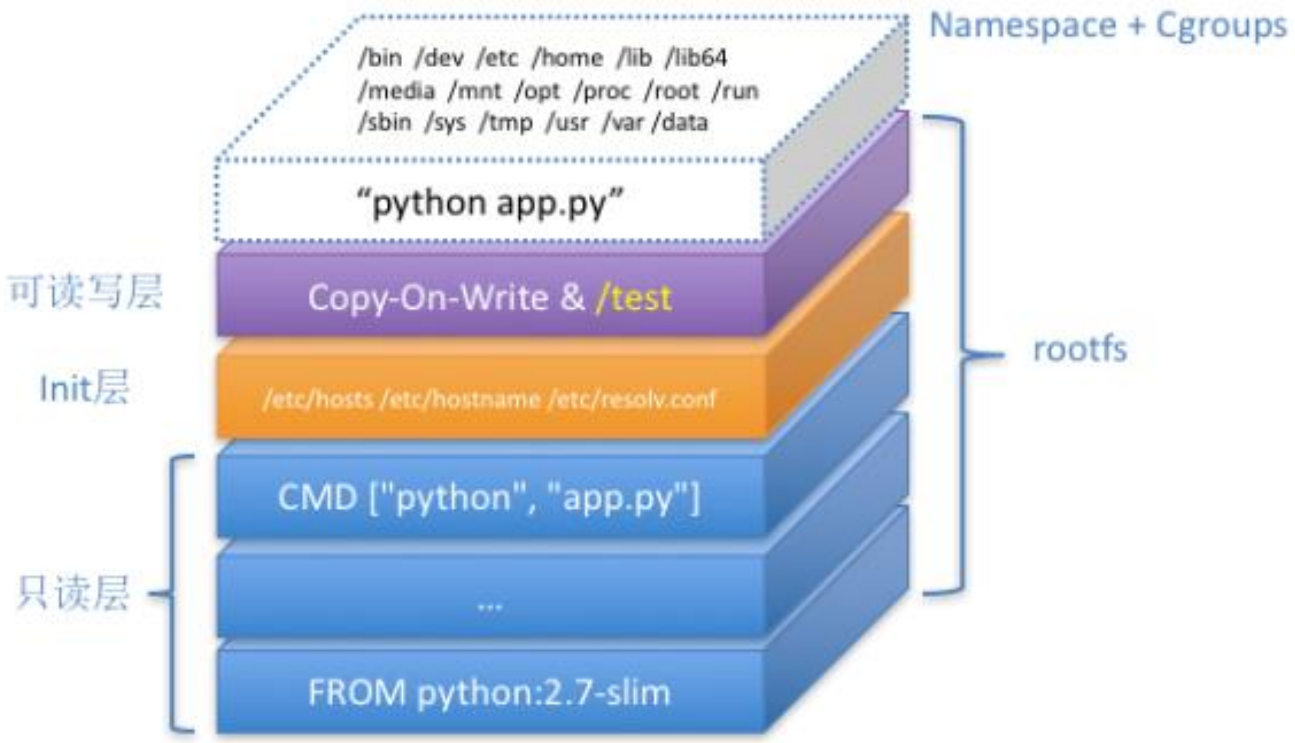


CGroup 典型应用架构图

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。

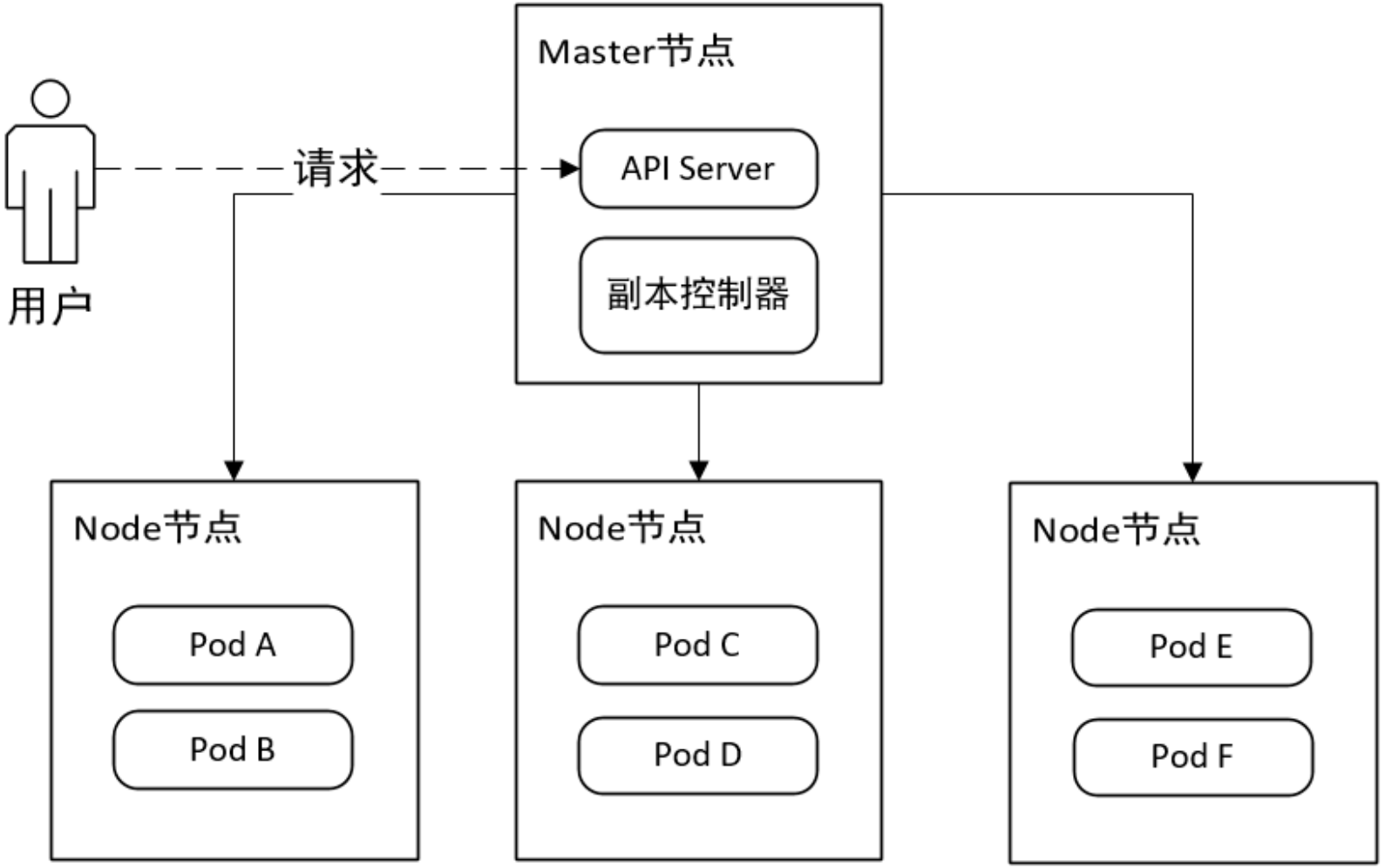
Docker容器技术


Namespace + Cgroups
/bin /dev /etc /home /lib /lib64
/media /mnt /opt /proc /root /run
/sbin /sys /tmp /usr /var /data
"python app.py"
可读写层
Copy-On-Write & /test
Init层
/etc/hosts /etc/hostname /etc/resolv.conf
CMD ["python", "app.py"]
只读层
...
FROM python:2.7-slim
rootfs

Kubernetes

- 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。

Kubernetes集群结构



Kubernetes集群结构

- 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。

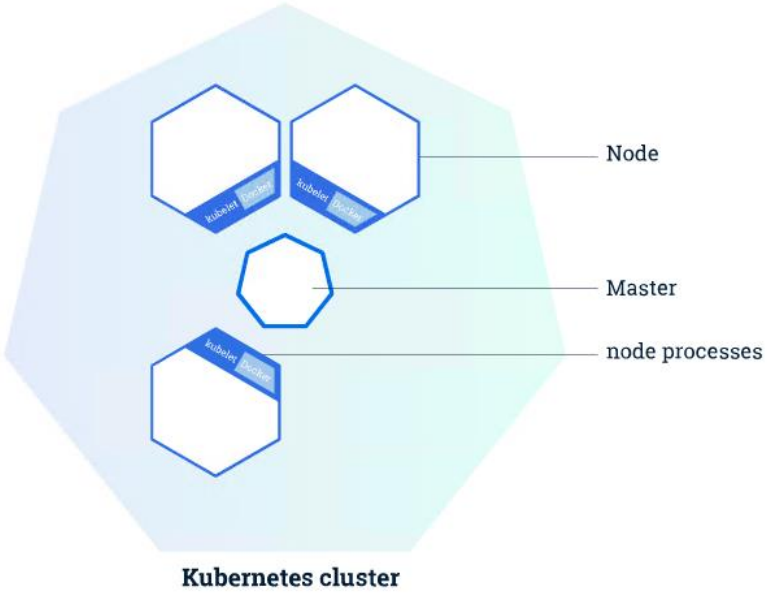
- **Namespace：**
  - 集群中的命名空间，
  - 不同命名空间中可以存在相同命名的资源，从而有利于设计逻辑上独立的资源组或者用户组

Kubernetes组件——Master的组件

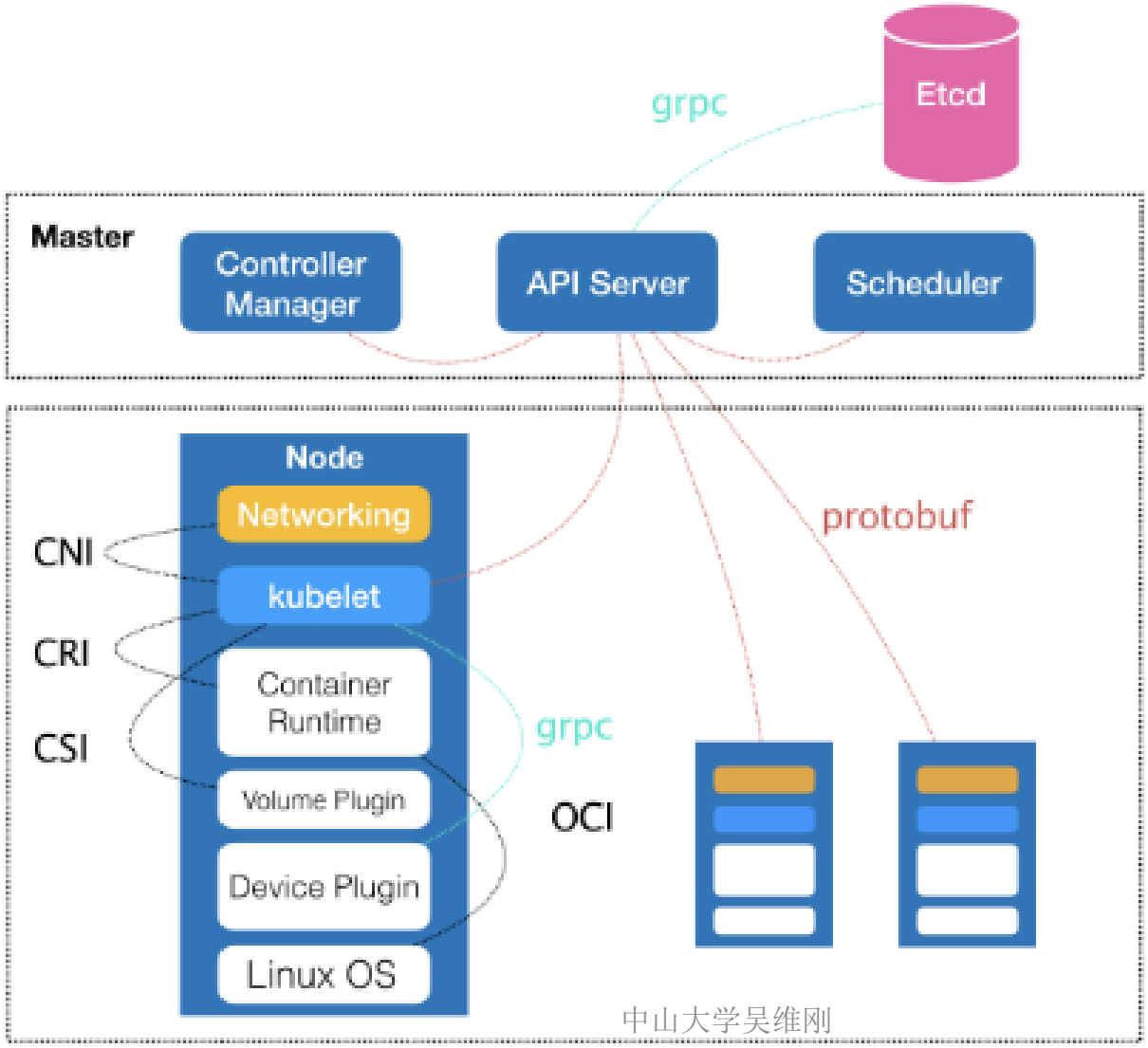
- **apiserver：**
  - 作为 kubernetes 集群的入口，封装了核心对象的增删改查操作，以 RESTFul 接口方式提供给外部客户和内部组件调用。
  - 它维护的 REST 对象将持久化到 etcd（一个分布式强一致性的 key/value 存储）。
- **scheduler：**
  - 负责集群的资源调度，为新建的 pod 分配宿主机。
- **controller-manager：**
  - 负责执行各种控制器，目前有两类：
  - **endpoint-controller：** 维护 service 到 pod 的映射。
  - **replication-controller：**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

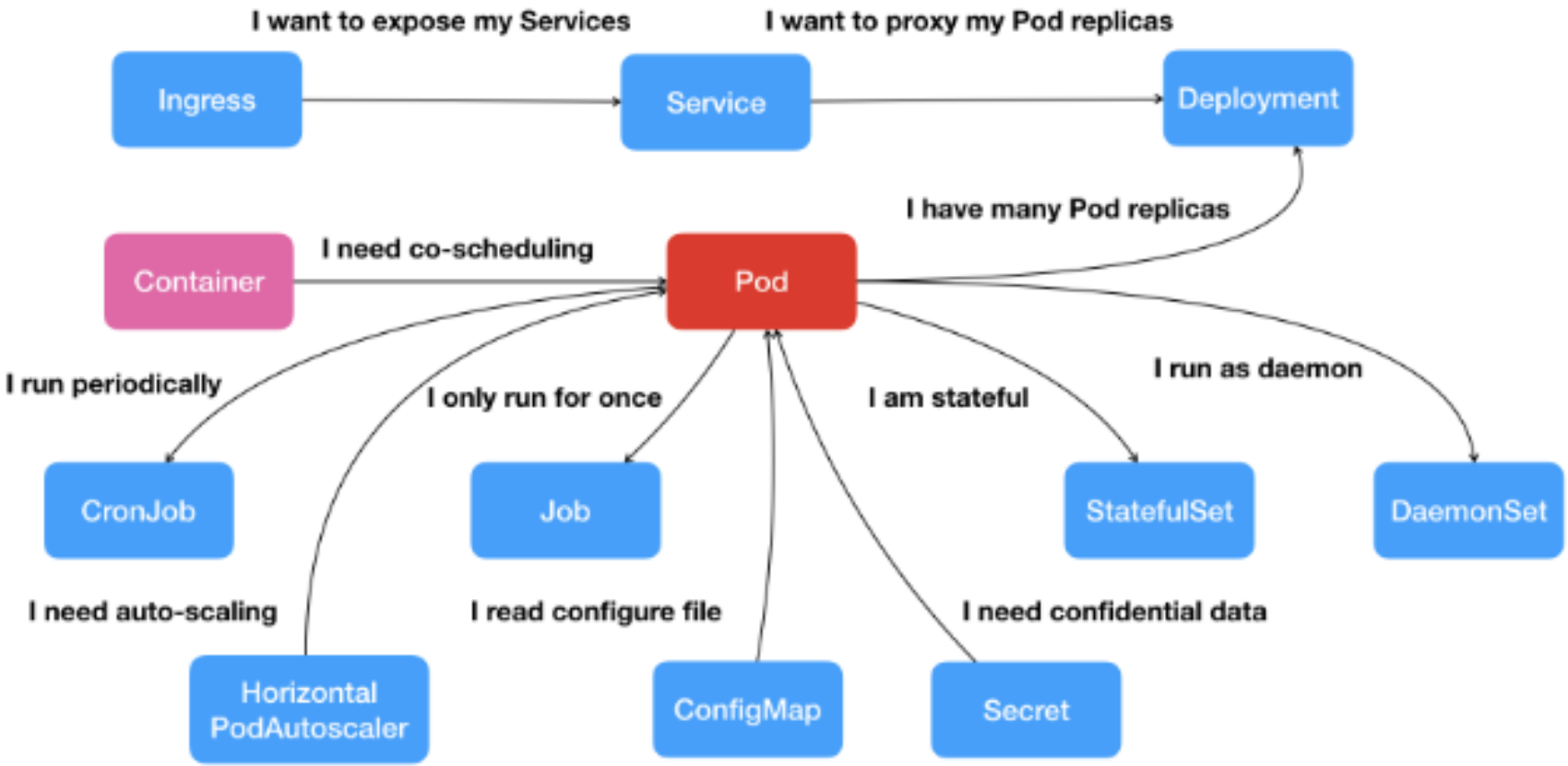


Kubernetes


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

- 容器技术是轻量化虚拟化
  - 容器共享OS
  - 比VM成本低效率高
- 逐渐成为主流的虚拟化技术
- 可以于VM结合一起使用