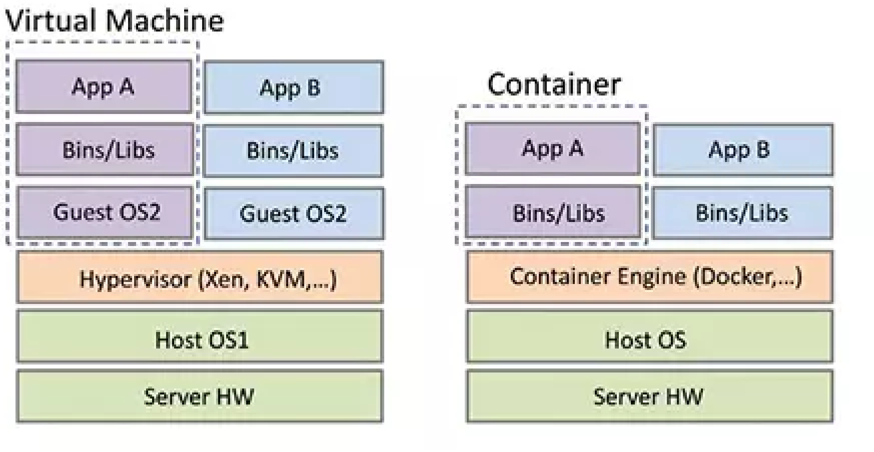
概念

A container is a standard unit of software that packages up **code and all its dependencies** so the application runs quickly and reliably from one computing environment to another.

A computer program running on an ordinary operating system can see all resources of that computer. *However*, programs running inside of a container can only see the container's contents and devices assigned to the container.

与虚拟化技术比较：



特点

1. **轻量化**：只打包了必要的Bin/Lib。[1]
2. **部署快**：毫秒级/秒级部署速度。
3. **易于移植**：“Build once, run anywhere”(Docker's slogan)
4. **弹性伸缩**：elastic，根据需求自动调整分配的计算资源量。

技术与相关概念

LXC

- **Namespaces**

The key feature of namespaces is that they isolate processes from each other.

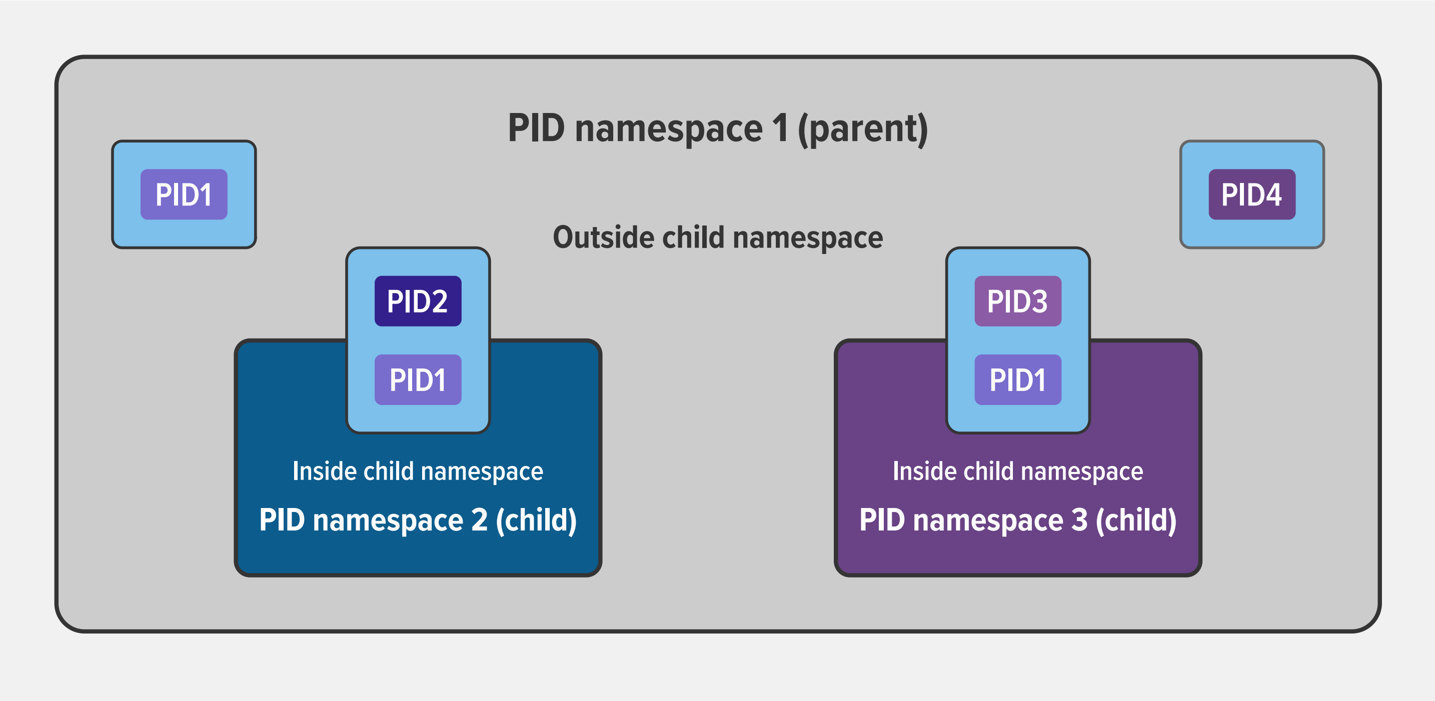
(1) **好处**：

1. A smaller blast radius for changes.[2]
2. A smaller footprint for security-related concerns.
3. Meets the architectural style of microservices as described by Martin Fowler.

(2) **Types of Namespaces**[3]

Within the Linux kernel, there are different types of namespaces. Each namespace has its own unique properties:

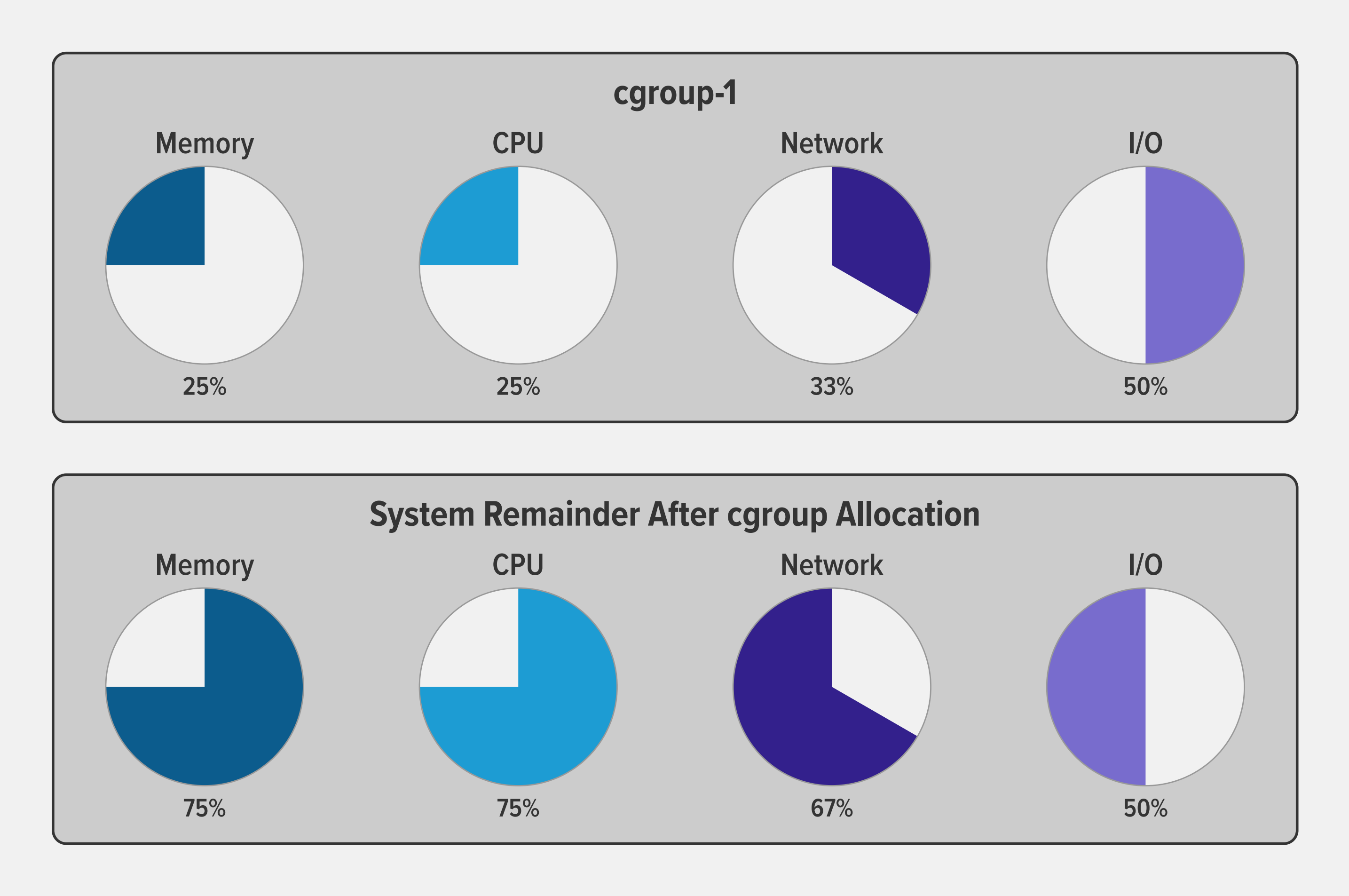
1. A **user namespace** has its own set of user IDs and group IDs for assignment to processes. In particular, this means that a process can have **root privilege within its user namespace** without having it in other user namespaces.
2. A **process ID (PID) namespace** assigns a set of PIDs to processes that are independent from the set of PIDs in other namespaces. **The first process created in a new namespace has PID 1 and child processes are assigned subsequent PIDs.** If a child process is created with its own PID namespace, it has PID 1 in that namespace as well as its PID in the parent process' namespace.
3. A **network namespace** has an independent network stack: its own private routing table, set of IP addresses, socket listing, connection tracking table, firewall, and other **network-related resources**.
4. A **mount namespace** has an independent list of mount points seen by the processes in the namespace. This means that you can mount and unmount filesystems in a mount namespace without affecting the host filesystem.(子文件系统的独立创建与挂载)
5. An **interprocess communication (IPC) namespace** has its own IPC resources, for example POSIX message queues.
6. A **UNIX Time-Sharing (UTS) namespace** allows a single system to appear to have different host and domain names to different processes.



- **cgroups**

在之前的历史回顾中，已经看到cgroups的产生和引入LXC使容器化技术的发展迈出了关键性的一步。cgroups是一种隔离策略，它提供了如下功能特性

1. **Resource limits** – You can configure a cgroup to limit how much of a particular resource (memory or CPU, for example) a process can use. 限制一个进程所能拥有的资源。
2. **Prioritization**(优先级) – You can control how much of a resource (CPU, disk, or network) a process can use compared to processes in another cgroup when there is resource contention.设定优先级，处理资源竞争的情形。
3. **Accounting** – Resource limits are monitored and reported at the cgroup level.监控和报告对资源的限制情况。
4. **Control** – You can change the status (frozen, stopped, or restarted) of all processes in a cgroup with a single command.改变进程的状态（冻结、停止、重启）



- **Conclusion**

Namespaces provide **isolation** of system resources, and **cgroups** allow for fine-grained **control** and enforcement of limits for those resources.

Docker

介绍容器离不开介绍Docker。其实Docker本身并不是容器，它是创建容器的工具，是应用容器引擎。Docker发展到现在，几乎已经成为了容器的代名词了。

Docker技术的三大核心概念，分别是：

- 镜像（Image）
- 容器（Container）
- 仓库（Repository）

两句Slogan

- **Build, Ship and Run**

“搭建、发送、运行”，三板斧。

- **Build once, run anywhere**

“搭建一次，到处运行”，可移植性。

知乎**一个比喻**： https://zhuanlan.zhihu.com/p/53260098

我来到一片空地，想建个房子，于是我搬石头、砍木头、画图纸，一顿操作，终于把这个房子盖好了。（写代码）

结果，我住了一段时间，想搬到另一片空地去。这时候，按以往的办法，我只能再次搬石头、砍木头、画图纸、盖房子。（在另一台机器上写代码）

但是，跑来一个老巫婆，教会我一种魔法。这种魔法，可以把我盖好的房子复制一份，做成“镜像”，放在我的背包里。（镜像和仓库的概念）

等我到了另一片空地，就用这个“镜像”，复制一套房子，摆在那边，拎包入住。（把镜像做成容器）

image 镜像

一个特殊的**文件系统**。它除了提供容器运行时所需的**程序、库、资源、配置**等文件外，还包含了一些为运行时准备的一些**配置参数**（例如环境变量）。

镜像不包含任何动态数据，其内容在构建之后也不会被改变。（房子都是一样的，生活用品由住户添置）

Container 容器

由镜像生成的容器，镜像的一个实例。

容器管理：早期 Docker 使用了 **LXC**（LinuX Containers）来管理容器的运行，后来 Docker 废弃了 LXC 并开发了一套自己的容器管理库，并命名为**libcontainer**。现在 libcontainer 更名为 **runc**，并由

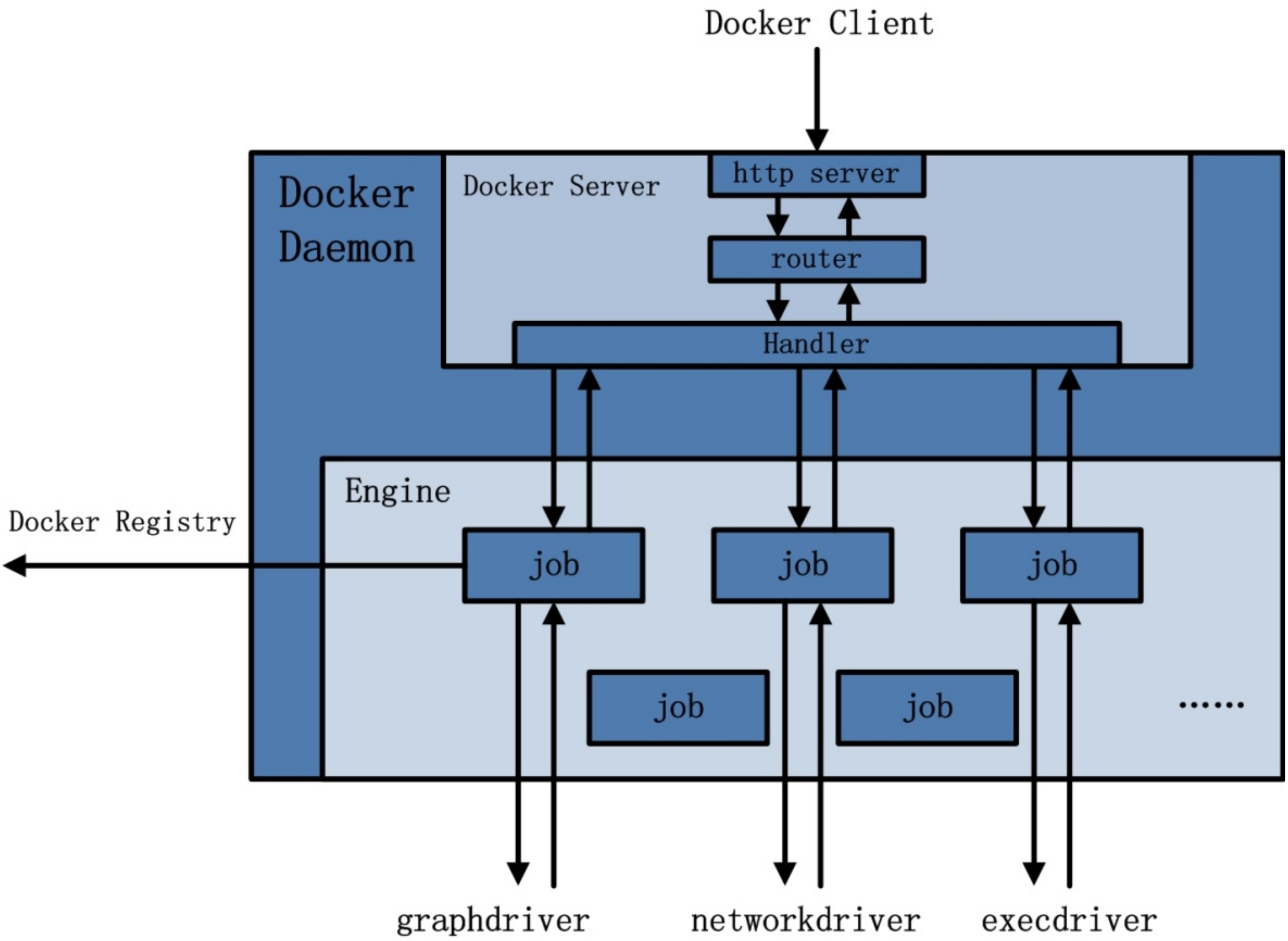
开放容器基金会（Open Container Foundation）负责运营和维护。

runc是一个容器的入口和守护进程，负责容器的创建（设置 cgroups 和 namespace 等资源隔离，准备文件系统）、启动（运行容器内的第一个进程）和资源回收。

Repository 仓库

好多镜像可以放在一起，用仓库进行管理。负责管理Docker镜像的是Docker Registry服务。

Docker Daemon 守护进程



Docker Daemon是Docker架构中运行在后台的守护进程，大致可以分为Docker Server、Engine和Job三部分。

Docker Daemon可以认为是通过Docker Server模块接受Docker Client的请求，并在Engine中处理请求，然后根据请求类型，创建出指定的Job并运行。

运行过程的作用有以下几种可能：

- 向Docker Registry获取镜像，
- 通过graphdriver执行容器镜像的本地化操作，

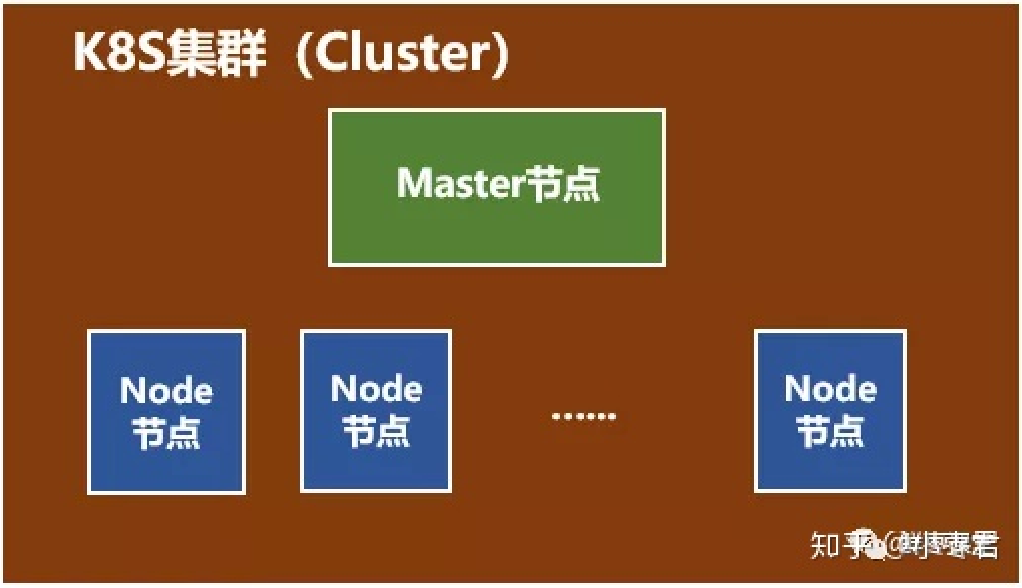
- 通过networkdriver执行容器网络环境的配置，
- 通过execdriver执行容器内部运行的执行工作等。

Kubernetes(K8S)

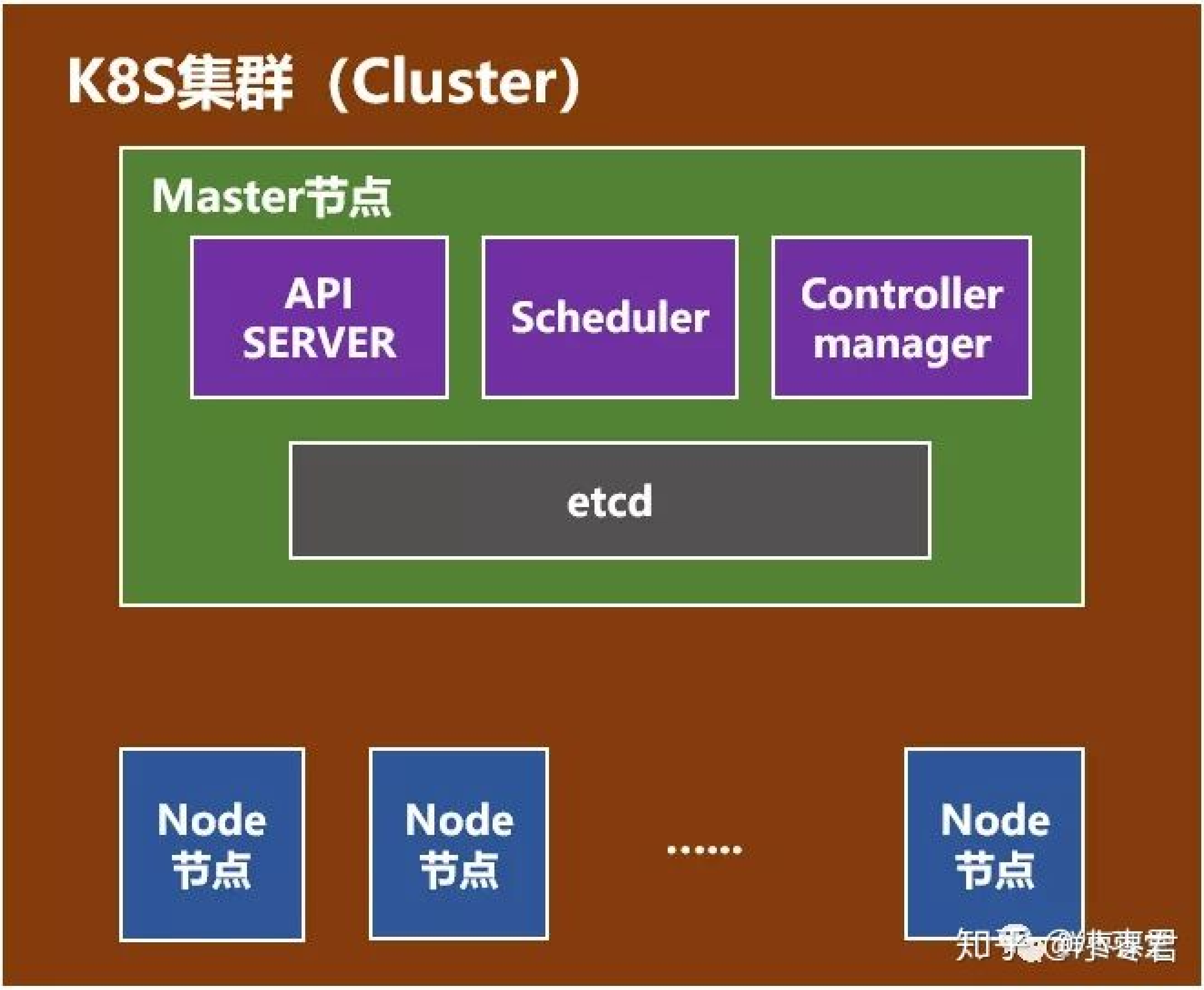
Kubernetes是基于容器的集群管理平台，简称K8S（8表示字母KS之间的“ubernete”是8个字符），由Google创建。

一个K8S系统，通常称为一个**K8S集群（Cluster）**。

集群构成为：



其中，Master节点和Node节点构成如下：



K8S集群（Cluster）

Master节点

API SERVER

Scheduler

Controller manager

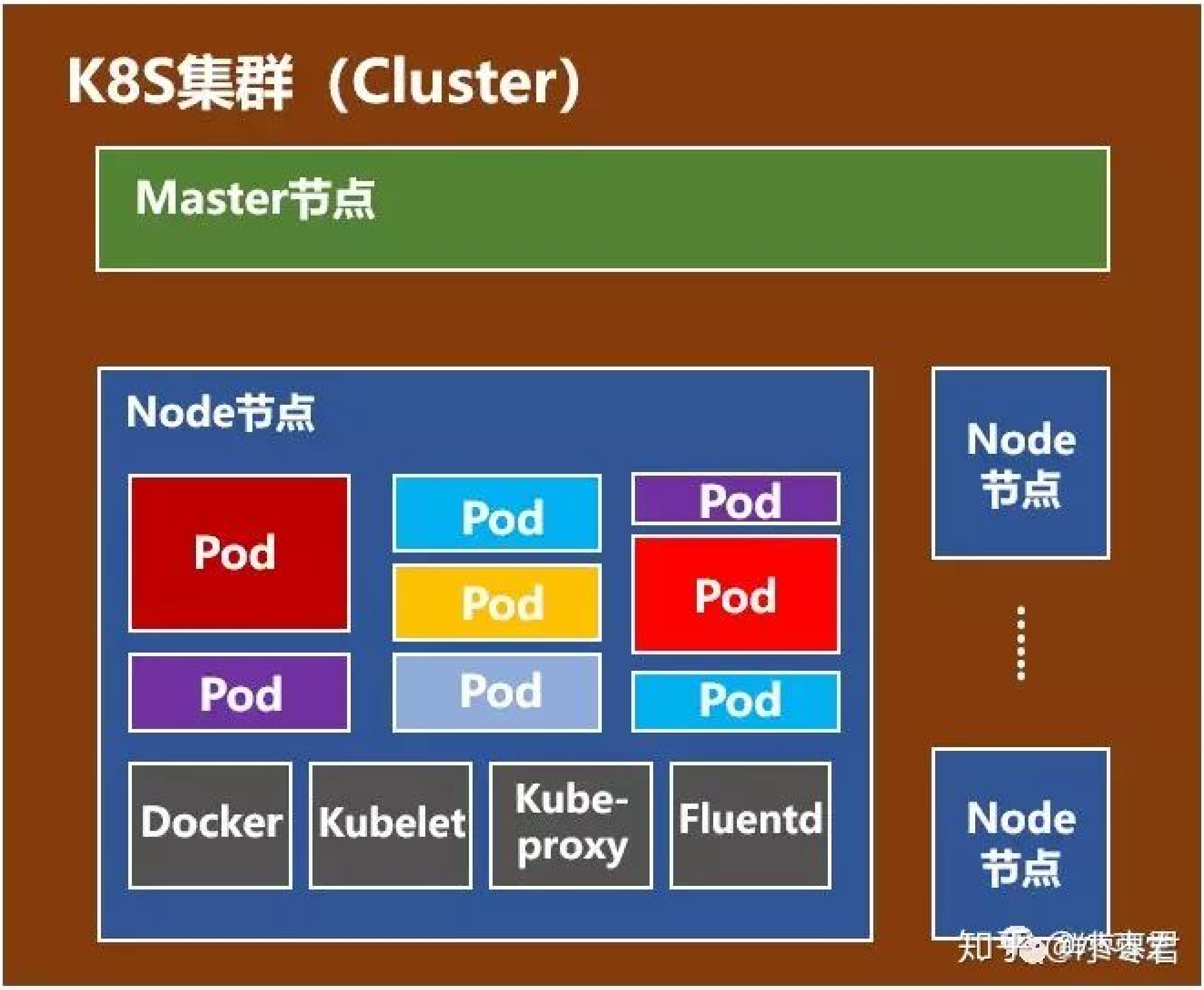
etcd

Node 节点

Node 节点

……

Node 节点



Master节点主要还是负责**管理和控制**。**Node**节点是工作负载节点，里面是**具体的容器**。具体层次如下

- **K8S Cluster**
  - **Master**
    - **API Server**
      API Server是整个系统的**对外接口**，供客户端和其它组件调用，相当于“营业厅”。
    - **Scheduler**
      Scheduler负责对集群内部的资源进行调度(**资源调度**)，相当于“调度室”。
    - **Controller manager**
      Controller manager负责**管理控制器**，相当于“大总管”。
    - **etcd**
      used as Kubernetes' **backing store** for all cluster data，相当于“资料室”。
  - **Node**
    - **Pod**
      一个Pod代表着集群中运行的一个进程，它内部封装了一个或多个紧密相关的容器。
    - **Docker**
      上面已经介绍了。用于**创建容器**。
    - **kubelet**
      监视指派到它所在Node上的Pod，包括创建、修改、监控、删除等。

- **kube-proxy**
  为Pod对象提供代理。
- **Fluentd**
  日志收集、存储与查询。
- **kube-dns**（可选）

---

[1] *Bins/Libs*: The Bins/Libs layer supplies software libraries and services needed by the top layer apps. ↩

[2] *Blast Radius*: The reach that a faulty configuration change or problem might cause. ↩

[3] Source: https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/ ↩