项目3

Kubernetes集群配置与管理

学习目标

- 理解容器编排基本知识、Kubernetes 概述、Kubernetes 的设计理念、Kubernetes 体系结构、Kubernetes 核心概念、Kubernetes 集群部署方式、Kubernetes 集群管理策略以及 Kubectl 工具基本使用等相关理论知识。
- 掌握 Kubernetes 集群安装与部署、Kubectl 基本命令配置管理、Pod 的创建与管理、Deployment 控制器配置与管理、Server 的创建与管理以及 Kubernetes 容器管理等相关知识与技能。

3.1 项目概述

Docker 本身非常适合用于管理单个容器，但真正的生产环境还会涉及多个容器的封装和服务之间的协同处理。这些容器必须跨多个服务器主机进行部署与连接，单一的管理方式满足不了业务需求。Kubernetes 是一个可以实现跨主机管理容器化应用程序的系统，是容器化应用程序和服务生命周期管理平台，它的出现不仅解决了多容器之间数据传输与沟通的瓶颈，而且还促进了容器技术的发展。本章讲解容器编排基本知识、Kubernetes 概述、Kubernetes 的设计理念、Kubernetes 体系结构、Kubernetes 核心概念、Kubernetes 集群部署方式、Kubernetes 集群管理策略以及 Kubectl 工具基本使用等相关理论知识，项目实践部分讲解 Kubernetes 集群安装与部署、Kubectl 命令配置管理、Pod 的创建与管理、Deployment 控制器配置与管理、Server 的创建与管理以及 Kubernetes 容器管理等相关知识与技能。

3.2 必备知识

3.2.1 容器编排基础知识

企业中的系统架构是实现系统正常运行和服务高可用、高并发的基础。随着时代与科技的发展，系统架构经过了三个阶段的演变，实现了从早期单一服务器部署到现在的容器部署方式的改变。

1. 企业架构的演变

企业架构经历了传统时代、虚拟化时代与容器化时代的演变过程。

(1) 传统时代。

早期企业在物理服务器上运行应用程序，无法为服务器中的应用程序定义资源边界，导致系统资源分配不均匀。例如，一台物理服务器上运行着多个应用程序，可能存在一个应用程序占用大部分资源的情况，其他应用程序的可用资源因此减少，造成程序运行表现不佳。当然也可以在多台物理服务器上运行不同的应用程序，但这样资源并未得到充分利用，也增加了企业维护物理服务器的成本。

(2) 虚拟化时代。

虚拟化技术可以在物理服务器上虚拟出硬件资源，以便在服务器的 CPU 上运行多个虚拟机(VM)，每个 VM 不仅可以在虚拟化硬件上运行包括操作系统在内的所有组件，而且相互之间可以保持系统和资源的隔离，从而在一定程度上提高了系统的安全性。虚拟化有利于更好地利用物理服务器中的资源，实现更好的可扩展性，从而降低硬件成本。

(3) 容器化时代。

容器化技术类似于虚拟化技术，不同的是容器化技术是操作系统级的虚拟化，而不是硬件级的虚拟化。每个容器都具有自己的文件系统、CPU、内存、进程空间等，并且它们使用的计算资源是可以被限制的。应用服务运行在容器中，各容器可以共享操作系统。因此，容器化技术具有轻质、宽松隔离的特点。因为容器与底层基础架构和主机文件系统隔离，所以跨云和操作系统的快速分发得以实现。

2. 常见的容器编排工具

容器的出现和普及为开发者提供了良好的平台和媒介，使开发和运维工作变得更加简单与高效。随着企业业务和需求的增长，在大规模使用容器技术后，如何对这些运行的容器进行管理成为首要问题。在这种情况下，容器编排工具应运而生，最具代表性的有以下三种。

(1) Apache 公司的 Mesos。

Mesos 是 Apache 公司旗下的开源分布式资源管理框架，由美国加州大学伯克利分校的 AMPLab(Algorithms Machine and People Lab，算法、计算机和人实验室)开发。Mesos 早期通过了万台节点验证，2014 年之后又被广泛使用在 eBay、Twitter 等大型互联网公司的生产环境中。

(2) Docker 公司的三剑客。

容器诞生后，Docker 公司就意识到单一容器体系的弊端，为了能够有效地解决用户的需求和

集群中的瓶颈,Docker 公司相继推出 Machine、Compose、Swarm 项目。

Machine 项目用 Go 语言编写,可以实现 Docker 运行环境的安装与管理,还可以实现批量在指定节点或平台上安装并启动 Docker 服务。

Compose 项目用 Python 语言编写,可以实现基于 Docker 容器多应用服务的快速编排,其前身是开源项目 Fig。Compose 项目使用户可以通过单独 YAML 文件批量创建自定义的容器,并通过应用程序接口(Application Programming Interface,API)对集群中的 Docker 服务进行管理。

Swarm 项目基于 Go 语言编写,支持原生态的 Docker API 和 Docker 网络插件,很容易实现跨主机集群部署。

(3) Google 公司的 Kubernetes。

Kubernetes(来自希腊语,意为"舵手",因为 K 与 s 之间有 8 个字母,所以业内人士喜欢称其为 K8s)基于 Go 语言开发,是 Google 公司发起并维护的开源容器集群管理系统,底层基于 Docker、rkt 等容器技术,其前身是 Google 公司开发的 Borg 系统。Borg 系统在 Google 内部已经应用了多年,曾管理超过 20 亿个容器。经过多年的经验积累,Google 公司将 Borg 系统完善后贡献给了开源社区,并将其重新命名为 Kubernetes。

3.2.2 Kubernetes 概述

Kubernetes 系统支持用户通过模板定义服务配置,用户提交配置信息后,系统会自动完成对应用容器的创建、部署、发布、伸缩、更新等操作。系统发布以来吸引了 Red Hat、CentOS 等知名互联网公司与容器爱好者的关注,是目前容器集群管理系统中优秀的开源项目之一。

1. Kubernetes 简介

Kubernetes 是开源的容器集群管理系统,可以实现容器集群的自动化部署、自动扩缩容、维护等功能。它既是一款容器编排工具,也是全新的基于容器技术的分布式架构领先方案。在 Docker 技术的基础上,Kubernetes 为容器化的应用提供部署运行、资源调度、服务发现和动态伸缩等功能,提高了大规模容器集群管理的便捷性。

Kubernetes 一个核心的特点就是能够自主地管理容器,来保证云平台中的容器按照用户的期望状态运行(如用户想让 Apache 一直运行,用户不需要关心怎么去做,Kubernetes 会自动去监控,然后重启、新建。总之,让 Apache 一直提供服务),管理员可以加载一个微型服务,让规划器来找到合适的位置。同时,Kubernetes 也提供系统提升工具以及人性化服务,让用户能够方便地部署自己的应用。

在 Kubernetes 中,基本调度单元称为 Pod,通过该种抽象类别可以把更高级别的抽象内容增加到容器化组件,所有的容器均在 Pod 中运行,一个 Pod 可以承载一个或者多个相关的容器,同一个 Pod 中的容器会部署在同一个物理机器上并且能够共享资源。容器集为分组容器增加了一个抽象层,可帮助调用工作负载,并为这些容器提供所需的联网和存储等服务。

一个 Pod 也可以包含 0 个或者多个磁盘卷组(Volumes),这些卷组将会以目录的形式提供给一个容器,或者被所有 Pod 中的容器共享。对于用户创建的每个 Pod,系统会自动选择那个健康并且有足够容量的机器,然后创建类似容器的容器。当容器创建失败时,容器会被 Node Agent 自动重启,这个 Node Agent 叫作 Kubelet。但是,如果是 Pod 失败或者机器故障,它不会自动转移并且启动,除非用户定义了 Replication Controller。

Kubernetes 的目标是让部署容器化的应用简单并且高效，它提供了应用部署、规划、更新、维护的一种机制。Kubernetes 是一种可自动实施 Linux 容器操作的开源平台。它可以帮助用户省去应用容器化过程的许多手动部署和扩展操作。也就是说，用户可以将运行 Linux 容器的多组主机聚集在一起，借助 Kubernetes 编排功能，用户可以构建跨多个容器的应用服务，跨集群调度、扩展这些容器，并长期持续管理这些容器的健康状况。

有了 Kubernetes 便可切实采取一些措施来提高 IT 安全性。而且，这些集群可跨公共云、私有云或混合云部署主机。因此，对于要求快速扩展的云原生应用而言，Kubernetes 是理想的托管平台。Kubernetes 于 2015 年发布，并迅速成为事实上的容器编排标准。Kubernetes 还需要与联网、存储、安全性、遥测和其他服务整合，以提供全面的容器基础架构。

2. Kubernetes 的优势

Kubernetes 系统不仅可以实现跨集群调度、水平扩展、监控、备份、灾难恢复，还可以解决大型互联网集群中多任务处理的瓶颈。Kubernetes 遵循微服务架构理论，将整个系统划分为多个功能各异的组件。各组件结构清晰、部署简单，可以非常方便地运行于系统环境之中。利用容器的扩容机制，系统将容器归类，形成“容器集”(Pod)，用于帮助用户调度工作负载(Work Load)，并为这些容器提供联网和存储服务。

2017 年 Google 公司的搜索热度报告中显示，Kubernetes 搜索热度已经超过了 Mesos 和 Docker Swarm，这也标志着 Kubernetes 在容器编排市场逐渐占有主导地位。

近几年容器技术得到广泛应用，使用 Kubernetes 系统管理容器的企业也在不断增加，Kubernetes 系统的主要功能如表 3.1 所示。

表 3.1　Kubernetes 系统的主要功能

主要功能	详　解
自我修复	在节点产生故障时，会保证预期的副本数量不会减少，会在产生故障的同时，停止健康检查失败的容器并部署新的容器，保证上线服务不会中断
存储部署	Kubernetes 挂载外部存储系统，将这些存储作为集群资源的一部分来使用，增加存储使用的灵活性
自动部署和回滚更新	Kubernetes 采用滚动更新策略更新应用，一次更新一个 Pod，当更新过程中出现问题时，Kubernetes 会进行回滚更新，保证升级业务不受影响
弹性伸缩	Kubernetes 可以使用命令或基于 CPU 使用情况，自动快速扩容和缩容应用程序，保证在高峰期的高可用性和业务低档期回收资源，减少运行成本
提供认证和授权	可以控制用户是否有权限使用 API 进行操作，精细化权限分配
资源监控	工作节点中集成 Advisor 资源收集工具，可以快速实现对集群资源监控
密匙和配置管理	Kubernetes 允许存储和管理敏感信息，如密码、OAuth 令牌和 SSH 密钥。用户可以部署和更新机密和应用程序配置，而无须重建容器映像，也不会在堆栈配置中暴露机密
服务发现和负载均衡	为多个容器提供统一的访问入口(内部 IP 和一个 DNS 名称)，并且将所有的容器进行负载均衡，集群内应用可以通过 DNS 名称完成相互之间的访问

Kubernetes 提供的这些功能去除了不必要的限制和规范，使应用程序开发者能够从繁杂的运维中解放出来，获得了更大的发挥空间。

3. 深入理解 Kubernetes

Kubernetes 在容器层面而非硬件层面运行，因此它不仅提供了 PaaS 产品的部署、扩展、负载

平衡、日志记录和监控功能,还提供了构建开发人员平台的构建块,在重要的地方保留了用户选择灵活性。Kubernetes 的特征如下。

(1) Kubernetes 支持各种各样的工作负载,包括无状态、有状态和数据处理的工作负载。如果应用程序可以在容器中运行,那么它也可以在 Kubernetes 上运行。

(2) 不支持部署源代码和构建的应用程序,其持续集成、交付和部署工作流程由企业自行部署。

(3) Kubernetes 只是一个平台,它不提供应用程序级服务,包括中间件(如消息总线)、数据处理框架(如 Spark)、数据库(如 MySQL)、高速缓存、集群存储系统(如 Ceph)等。

(4) Kubernetes 不提供或授权配置语言(如 Jsonnet),只提供了一个声明性的 API,用户可以通过任意形式的声明性规范来实现所需要的功能。

3.2.3 Kubernetes 的设计理念

大多数用户希望 Kubernetes 项目带来的体验是确定的:有应用的容器镜像,可在一个给定的集群上把这个应用运行起来。此外,用户还希望 Kubernetes 具有提供路由网关、水平扩展、监控、备份、灾难恢复等一系列运维的能力。这些其实就是经典 PaaS 项目的能力,用户使用 Docker 公司的 Compose+Swarm 项目,完全可以很方便地自己开发出这些功能。而如果 Kubernetes 项目只停留在拉取用户镜像、运行容器和提供常见的运维功能,就很难和"原生态"的 Docker Swarm 项目竞争,与经典的 PaaS 项目相比也难有优势可言。

1. Kubernetes 项目着重解决的问题

运行在大规模集群中的各种任务之间存在着千丝万缕的关系。如何处理这些关系,是作业编排和管理系统的难点。这种关系在各种技术场景中随处可见,比如,Web 应用与数据库之间的访问关系、负载均衡器和后端服务之间的代理关系、门户应用与授权组件之间的调用关系等。同属于一个服务单位的不同功能之间,也存在这样的关系,比如,Web 应用与日志搜集组件之间的文件交换关系。

在容器普及前,传统虚拟化环境对这种关系的处理方法都是"粗粒度"的。很多并不相关的应用被部署在同一台虚拟机中,也许是因为这些应用之间偶尔会互相发起几个 HTTP 请求。更常见的是,把应用部署在虚拟机里之后,还需要手动维护协作处理日志搜集、灾难恢复、数据备份等辅助工作的守护进程。

容器技术在功能单位的划分上有着独一无二的"细粒度"优势。使用容器技术可以将那些原先挤在同一个虚拟机里的应用、组件、守护进程分别做成镜像,然后运行在专属的容器中。进程互不干涉,各自拥有资源配额,可以被调度到整个集群里的任何一台机器上。这正是 PaaS 系统最理想的工作状态,也是所谓"微服务"思想得以落地的先决条件。为了解决容器间需要"紧密协作"的难题,Kubernetes 系统中使用了 Pod 这种抽象的概念来管理各种资源;当需要一次性启动多个应用实例时,可以通过系统中的多实例管理器 Deployment 实现;当需要通过一个固定的 IP 地址和端口以负载均衡的方式访问 Pod 时,可以通过 Service 实现。

2. Kubernetes 项目对容器间的访问进行了分类

在服务器上运行的应用服务频繁进行交互访问和信息交换。在常规环境下,这些应用往往会被直接部署在同一台机器上,通过本地主机(Local Host)通信并在本地磁盘目录中交换文件。在

Kubernetes 项目中，这些运行的容器被划分到同一个 Pod 内，并共享 Namespace（命名空间）和同一组数据卷，从而达到高效率交换信息的目的。

还有另外一些常见的需求，比如 Web 应用对数据库的访问。在生产环境中它们不会被部署在同一台机器上，这样即使 Web 应用所在的服务器宕机，数据库也不会受影响。容器的 IP 地址等信息不是固定的，为了使 Web 应用可以快速找到数据库容器的 Pod，Kubernetes 项目提供了一种名为 Service 的服务。Service 服务的主要作用是作为 Pod 的代理入口（Portal），代替 Pod 对外暴露一个固定的网络地址。这样，运行 Web 应用的 Pod，就只需要关心数据库 Pod 提供的 Service 信息。

3.2.4　Kubernetes 体系结构

Kubernetes 对计算资源进行了更高层次的抽象，通过将容器进行细致的组合，将最终的应用服务交给用户。Kubernetes 在模型建立之初就考虑了容器跨机连接的要求，支持多种网络解决方案。同时在 Service 层构建集群范围的软件定义网络（Software Defined Network，SDN），其目的是将服务发现和负载均衡放置到容器可达的范围。这种透明的方式便利了各个服务间的通信，并为微服务架构的实践提供了平台基础。而在 Pod 层次上，作为 Kubernetes 可操作的最小对象，其特征更是对微服务架构的原生支持。

1. 集群体系结构

Kubernetes 集群主要由控制节点 Master（部署高可用需要两个以上）和多个工作节点 Node 组成，两种节点上分别运行着不同的组件来维持集群高效稳定的运转，另外还需要集群状态存储系统（etcd）来提供数据存储服务，一切都基于分布式的存储系统。Kubernetes 集群中各节点和 Pod 的对应关系，如图 3.1 所示。

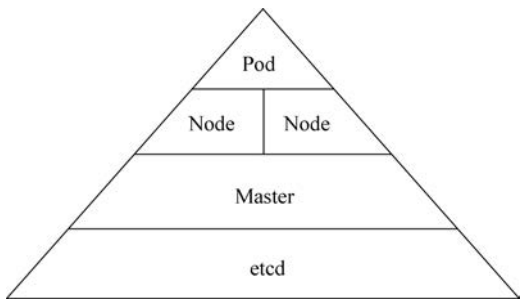

图 3.1　Kubernetes 集群中各节点和 Pod 的对应关系

在 Kubernetes 的系统架构中，Kubernetes 节点有运行应用容器必备的服务，而这些都是受 Master 控制的，Master 节点上主要运行着 API Server、Controller Manager 和 Scheduler 组件，而每个 Node 节点上主要运行着 Kubelet、Kubernetes Proxy 和容器引擎。除此之外，完整的集群服务还依赖一些附加的组件，如 kubeDNS、Heapster、Ingress Controller 等。

2. Master 节点与相关组件

控制节点 Master 是整个集群的网络中枢，主要负责组件或者服务进程的管理和控制，例如，追踪其他服务器健康状态、保持各组件之间的通信、为用户或者服务提供 API。

Master 中的组件可以在集群中的任何计算机上运行。但是，为简单起见，设置时通常会在一

台计算机上部署和启动所有主组件,并且不在此计算机上运行用户容器。在控制节点 Master 中所部署的组件包括以下三种。

(1) API Server。

API Server 是整个集群的网关,作为 Kubernetes 系统的入口,其内部封装了核心对象的"增""删""改""查"操作,以 REST API 方式供外部客户和内部组件调用,就像是机场的"联络室"。

(2) Scheduler(调度器)。

该组件监视新创建且未分配工作节点的 Pod,并根据不同的需求将其分配到工作节点中,同时负责集群的资源调度、组件抽离。

(3) Controller Manager(控制器管理器)。

Controller Manager 是所有资源对象的自动化控制中心,大多数对集群的操作都是由几个被称为控制器的进程执行的,这些进程被集成于 kube-controller-manager 守护进程中。实现的主要功能如下。

① 生命周期功能: Namespace 的创建,Event、Pod、Node 和级联垃圾的回收。

② API 业务逻辑功能: ReplicaSet 执行的 Pod 扩展等。

Kubernetes 主要控制器功能如表 3.2 所示。

表 3.2 Kubernetes 主要控制器功能

控制器名称	功 能
Deployment Controller	管理维护 Deployment,关联 Deployment 和 Replication Controller,保证运行指定数量的 Pod。当 Deployment 更新时,控制实现 Replication Controller 和 Pod 的更新
Node Controller	管理维护 Node,定期检查 Node 的健康状态,标识出(失效|未失效)的 Node
Namespace Controller	管理维护 Namespace,定期清理无效的 Namespace,包括 Namespace 下的 API 对象,比如 Pod、Service 等
Service Controller	管理维护 Service,提供负载以及服务代理
Endpoints Controller	管理维护 Endpoints,关联 Service 和 Pod,创建 Endpoints 为 Service 的后端,当 Pod 发生变化时,实时更新 Endpoints
Service Account Controller	管理维护 Service Account,为每个 Namespace 创建默认的 Service Account,同时为 Service Account 创建 Service Account Secret
Persistent Volume Controller	管理维护 Persistent Volume 和 Persistent Volume Claim,为新的 Persistent Volume Claim 分配 Persistent Volume 进行绑定,为释放的 Persistent Volume 执行清理回收
Daemon Set Controller	管理维护 Daemon Set,负责创建 Daemon Pod,保证指定的 Node 上正常地运行 Daemon Pod
Job Controller	管理维护 Job,为 Jod 创建一次性任务 Pod,保证完成 Job 指定完成的任务数目
Pod Autoscaler Controller	实现 Pod 的自动伸缩,定时获取监控数据,进行策略匹配,当满足条件时执行 Pod 的伸缩动作

另外 Kubernetes v1.16 以后的版本还加入了云控制器管理组件,用来与云提供商交互。

3. Node 节点与相关组件

Node 节点是集群中的工作节点(在早期的版本中也被称为 Minion),主要负责接收 Master 的工作指令并执行相应的任务。当某个 Node 节点宕机时,Master 节点会将负载切换到其他工作节点上,Node 节点与 Master 节点的关系如图 3.2 所示。

Node 节点上所部署的组件包括以下三种。

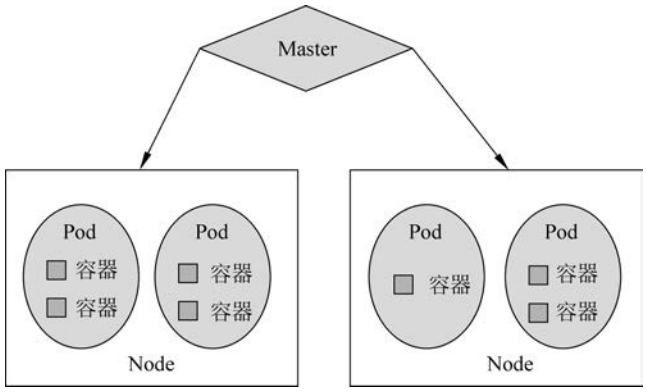

图 3.2　Node 节点与 Master 节点的关系

(1) Kubelet。

Kubelet 组件主要负责管控容器，它会从 API Server 接收 Pod 的创建请求，然后进行相关的启动和停止容器操作。同时，Kubelet 监控容器的运行状态并“汇报”给 API Server。

(2) Kubernetes Proxy。

Kubernetes Proxy 组件负责为 Pod 创建代理服务，从 API Server 获取所有的 Service 信息，并创建相关的代理服务，实现 Service 到 Pod 的请求路由和转发。Kubernetes Proxy 在 Kubernetes 层级的虚拟转发网络中扮演着重要的角色。

(3) Docker Engine。

Docker Engine 主要负责本机的容器创建和管理工作。

4. 集群状态存储组件

Kubernetes 集群中所有的状态信息都存储于 etcd 数据库中。etcd 以高度一致的分布式键值存储，在集群中是独立的服务组件，可以实现集群发现、共享配置以及一致性保障(如数据库主节点选择、分布式锁)等功能。在生产环境中，建议以集群的方式运行 etcd 并保证其可用性。

etcd 不仅可提供键值存储，还可以提供监听(Watch)机制。键值发生改变时 etcd 会通知 AIP Server，并由其通过 Watch API 向客户端输出。读者可以访问 Kubernetes 官方网站查看更多的 etcd 说明。

5. 其他组件

Kubernetes 集群还支持 DNS、Web UI 等插件，用于提供更完善的集群功能，这些插件的命名空间资源属于命名空间 kube-system，下面列出常用的插件及其主要功能。

(1) DNS。

DNS(Domain Name System，域名系统)插件用于集群中的主机名、IP 地址的解析。

(2) Web UI。

Web UI(用户界面)是提供可视界面的插件，允许用户通过界面来管理集群中运行的应用程序。

(3) Container Resource Monitoring。

Container Resource Monitoring(容器资源监视器)用于容器中的资源监视，并在数据库中记录这些资源分配。

(4) Cluster-level Logging。

Cluster-level Logging(集群级日志)是用于集群中日志记录的插件,负责保存容器日志与搜索存储的中央日志信息。

(5) Ingress Controller。

Ingress Controller 可以定义路由规则并在应用层实现 HTTP(S)负载均衡机制。

3.2.5 Kubernetes 核心概念

要想深入理解 Kubernetes 系统的特性与工作机制,不仅需要理解系统关键资源对象的概念,还要明确这些资源对象在系统中所扮演的角色。下面将介绍与 Kubernetes 集群相关的概念和术语,Kubernetes 集群架构如图 3.3 所示。

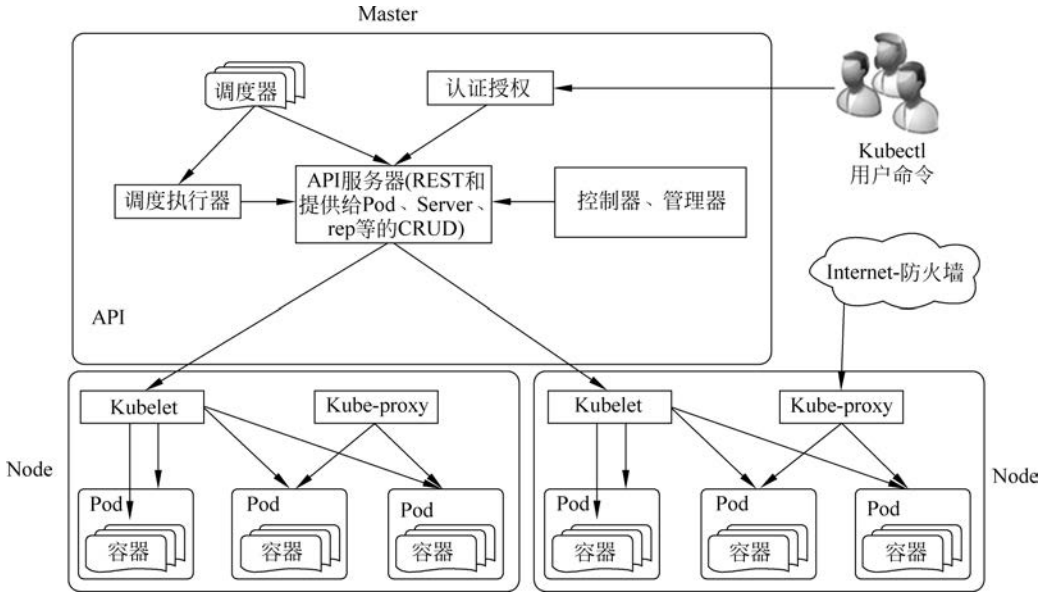

图 3.3 Kubernetes 集群架构

1. Pod

Pod(直译为豆荚)是 Kubernetes 中最小管理单位(容器运行在 Pod 中),一个 Pod 可以包含一个或多个相关容器。在同一个 Pod 内的容器可以共享网络名称空间和存储资源,也可以由本地的回环接口(lo)直接通信,但彼此又在 Mount、User 和 PID 等命名空间上保持隔离,Pod 抽象图如图 3.4 所示。

2. Label 和 Selector

Label(标签)是资源标识符,用来区分不同对象的属性。Label 本质上是一个键值对(key:value),可以在对象创建时或者创建后进行添加和修改。Label 可以附加到各种资源对象上,一个资源对象可以定义任意数量的 Label。用户可以通过给指定的资源对象捆绑一个或多个 Label 来实现多维度的资源分组管理功能,以便于灵活地进行资源分配、调度、配置、部署等管理工作。

Selector(选择器)是一个通过匹配 Label 来定义资源之间关系的表达式。给某个资源对象定

义一个 Label，相当于给它打一个标签，随后可以通过 Label Selector（标签选择器）查询和筛选拥有某些 Label 的资源对象，Label 与 Pod 的关系如图 3.5 所示。

图 3.4　Pod 抽象图

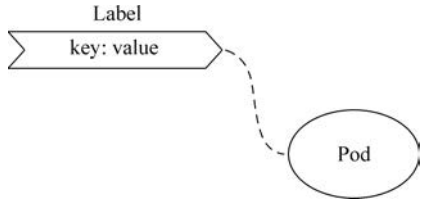

图 3.5　Label 与 Pod 的关系

3. Pause 容器

Pause 容器用于 Pod 内部容器之间的通信，是 Pod 中比较特殊的“根容器”。它打破了 Pod 中命名空间的限制，不仅是 Pod 的网络接入点，而且还在网络中扮演着“中间人”的角色。每个 Pod 中都存在一个 Pause 容器，其中运行着进程用来通信。Pause 容器与其他进程的关系如图 3.6 所示。

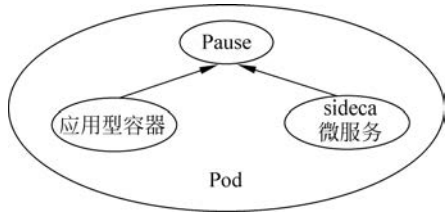

图 3.6　Pause 容器与其他进程的关系

4. Replication Controller

Replication Controller（RC，Pod 的副本控制器）在现在的版本中是一个总称。旧版本中使用 Replication Controller 来管理 Pod 副本（副本指一个 Pod 的多个实例），新版本增加了 ReplicaSet、Deployment 来管理 Pod 的副本，并将三者统称为 Replication Controller。

Replication Controller 保证了集群中存在指定数量的 Pod 副本。当集群中副本的数量大于指定数量，多余的 Pod 副本会停止；反之，欠缺的 Pod 副本则会启动，保证 Pod 副本数量不变。Replication Controller 是实现弹性伸缩、动态扩容和滚动升级的核心。

ReplicaSet（RS）是创建 Pod 副本的资源对象，并提供声明式更新等功能。

Deployment 是一个更高层次的 API 对象，用于管理 ReplicaSet 和 Pod，并提供声明式更新等功能，比旧版本的 Replication Controller 稳定性高。

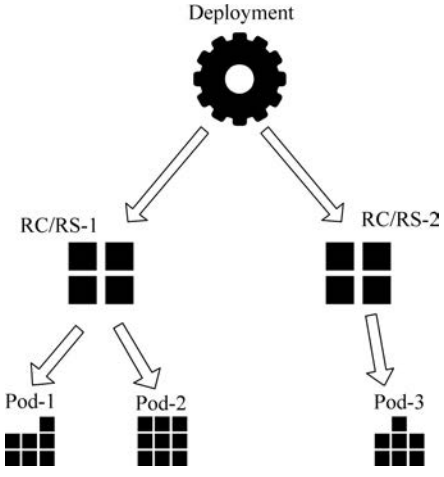

图 3.7　Deployment 与 ReplicaSet（RS）的关系

官方建议使用 Deployment 管理 ReplicaSet，而不是直接使用 ReplicaSet，这就意味着可能永远不需要直接操作 ReplicaSet 对象，而 Deployment 将会是使用最频繁的资源对象。Deployment 与 ReplicaSet（RS）的关系如图 3.7 所示。

5. StatefulSet

在 Kubernetes 系统集群中，Pod 的管理对象 StatefulSet 用于管理系统中有状态的集群，如 MySQL、MongoDB、ZooKeeper 集群等。这些集群中每个节点都有固定的 ID，集群中的成员通过 ID 相互通信，且集群规模是比较固定的。另外，为了能够在其他节