一、基础入门

1、Kubernetes 简介

Kubernetes 是谷歌开源的容器集群管理系统，是 Google 多年大规模容器管理技术 Borg 的开源版本，主要功能包括：

- 基于容器的应用部署、维护和滚动升级
- 负载均衡和服务发现
- 跨机器和跨地区的集群调度
- 自动伸缩
- 无状态服务和有状态服务
- 广泛的 Volume 支持
- 插件机制保证扩展性

Kubernetes 发展非常迅速，已经成为容器编排领域的领导者。

1.1 Kubernetes 是一个平台

Kubernetes 提供了很多的功能，它可以简化应用程序的工作流，加快开发速度。通常，一个成功的应用编排系统需要有较强的自动化能力，这也是为什么 Kubernetes 被设计作为构建组件和工具的生态系统平台，以便更轻松地部署、扩展和管理应用程序。

用户可以使用 Label 以自己的方式组织管理资源，还可以使用 Annotation 来自定义资源的描述信息，比如为管理工具提供状态检查等。

此外，Kubernetes 控制器也是构建在跟开发人员和用户使用的相同的 API 之上。用户还可以编写自己的控制器和调度器，也可以通过各种插件机制扩展系统的功能。

这种设计使得可以方便地在 Kubernetes 之上构建各种应用系统。

1.2 Kubernetes 不是什么

Kubernetes 不是一个传统意义上，包罗万象的 PaaS (平台即服务) 系统。它给用户预留了选择的自由。

- 不限制支持的应用程序类型，它不插手应用程序框架, 也不限制支持的语言 (如 Java, Python, Ruby 等)，只要应用符合 12 因素 即可。Kubernetes 旨在支持极其多样化的工作负载，包括无状态、有状态和数据处理工作负载。只要应用可以在容器中运行，那么它就可以很好的在 Kubernetes 上运行。
- 不提供内置的中间件 (如消息中间件)、数据处理框架 (如 Spark)、数据库 (如 mysql) 或集群存储系统 (如 Ceph) 等。这些应用直接运行在 Kubernetes 之上。
- 不提供点击即部署的服务市场。

- 不直接部署代码，也不会构建您的应用程序，但您可以在 Kubernetes 之上构建需要的持续集成 (CI) 工作流。
- 允许用户选择自己的日志、监控和告警系统。
- 不提供应用程序配置语言或系统 (如 jsonnet)。
- 不提供机器配置、维护、管理或自愈系统。

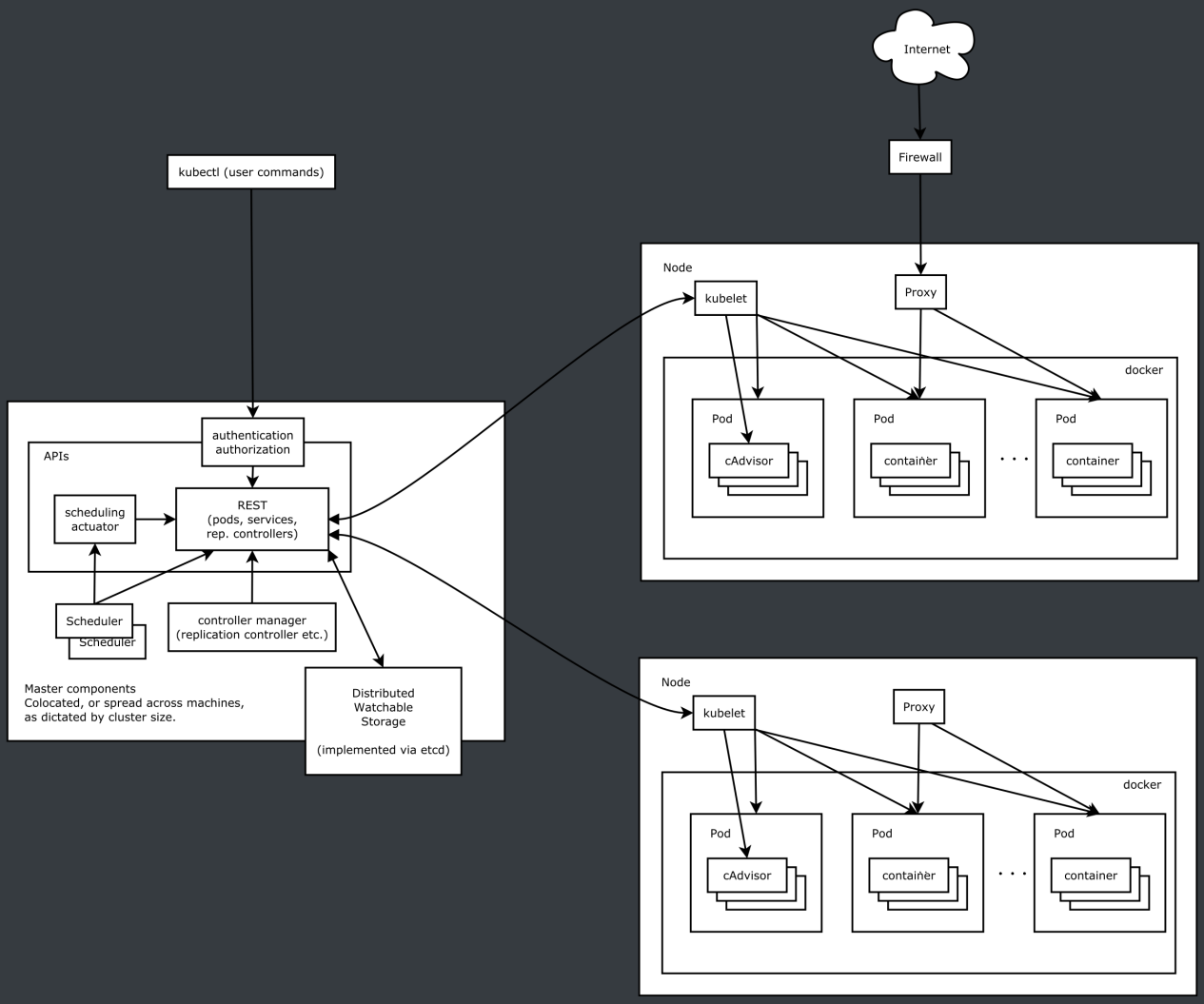
另外，已经有很多 PaaS 系统运行在 Kubernetes 之上，如 Openshift, Deis 和 Eldarion 等。 您也可以构建自己的 PaaS 系统，或者只使用 Kubernetes 管理您的容器应用。

当然了，Kubernetes 不仅仅是一个 “编排系统”，它消除了编排的需要。Kubernetes 通过声明式的 API 和一系列独立、可组合的控制器保证了应用总是在期望的状态，而用户并不需要关心中间状态是如何转换的。这使得整个系统更容易使用，而且更强大、更可靠、更具弹性和可扩展性。

1.3 核心组件

Kubernetes 主要由以下几个核心组件组成：

- etcd 保存了整个集群的状态；
- apiserver 提供了资源操作的唯一入口，并提供认证、授权、访问控制、API 注册和发现等机制；
- controller manager 负责维护集群的状态，比如故障检测、自动扩展、滚动更新等；
- scheduler 负责资源的调度，按照预定的调度策略将 Pod 调度到相应的机器上；
- kubelet 负责维护容器的生命周期，同时也负责 Volume（CVI）和网络（CNI）的管理；
- Container runtime 负责镜像管理以及 Pod 和容器的真正运行（CRI）；
- kube-proxy 负责为 Service 提供 cluster 内部的服务发现和负载均衡



2、Kubernetes 基本概念

2.1 Container

Container（容器）是一种便携式、轻量级的操作系统级虚拟化技术。它使用 namespace 隔离不同的软件运行环境，并通过镜像自包含软件的运行环境，从而使得容器可以很方便的在任何地方运行。

由于容器体积小且启动快，因此可以在每个容器镜像中打包一个应用程序。这种一对一的应用镜像关系拥有很多好处。使用容器，不需要与外部的基础架构环境绑定，因为每一个应用程序都不需要外部依赖，更不需要与外部的基础架构环境依赖。完美解决了从开发到生产环境的一致性问题。

容器同样比虚拟机更加透明，这有助于监测和管理。尤其是容器进程的生命周期由基础设施管理，而不是被进程管理器隐藏在容器内部。最后，每个应用程序用容器封装，管理容器部署就等同于管理应用程序部署。

其他容器的优点还包括

- 敏捷的应用程序创建和部署: 与虚拟机镜像相比，容器镜像更易用、更高效。
- 持续开发、集成和部署: 提供可靠与频繁的容器镜像构建、部署和快速简便的回滚（镜像是不可变

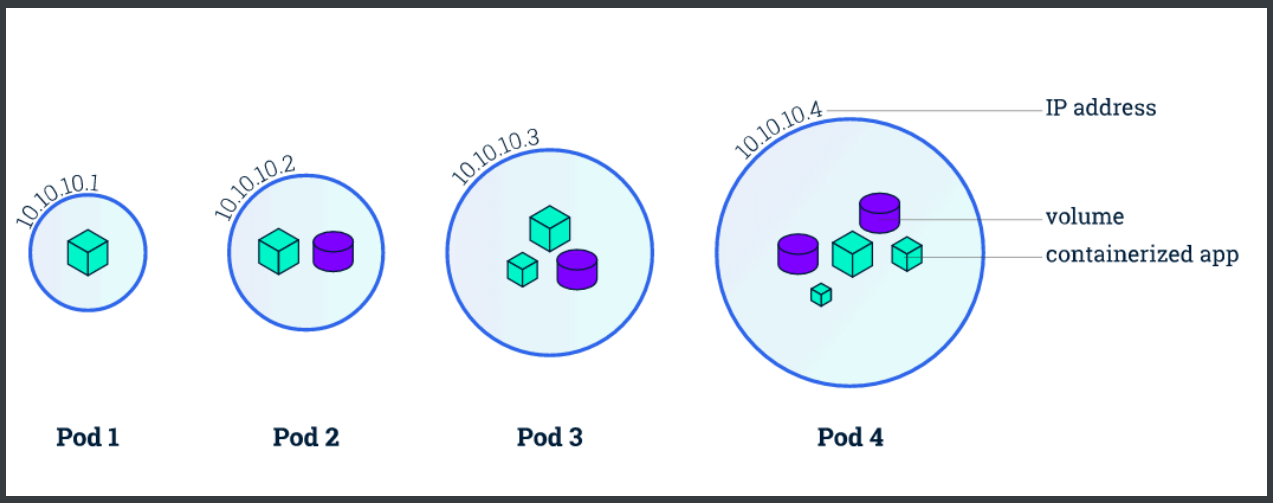
的）。

- 开发与运维的关注分离: 在构建/发布时即创建容器镜像，从而将应用与基础架构分离。
- 开发、测试与生产环境的一致性: 在笔记本电脑上运行和云中一样。
- 可观测：不仅显示操作系统的信息和度量，还显示应用自身的信息和度量。
- 云和操作系统的分发可移植性: 可运行在 Ubuntu, RHEL, CoreOS, 物理机, GKE 以及其他任何地方。
- 以应用为中心的管理: 从传统的硬件上部署操作系统提升到操作系统中部署应用程序。
- 松耦合、分布式、弹性伸缩、微服务: 应用程序被分成更小，更独立的模块，并可以动态管理和部署 - 而不是运行在专用设备上的大型单体程序。
- 资源隔离：可预测的应用程序性能。
- 资源利用：高效率和高密度。

2.2 Pod

Kubernetes 使用 Pod 来管理容器，每个 Pod 可以包含一个或多个紧密关联的容器。

Pod 是一组紧密关联的容器集合，它们共享 PID、IPC、Network 和 UTS namespace，是 Kubernetes 调度的基本单位。Pod 内的多个容器共享网络和文件系统，可以通过进程间通信和文件共享这种简单高效的方式组合完成服务。



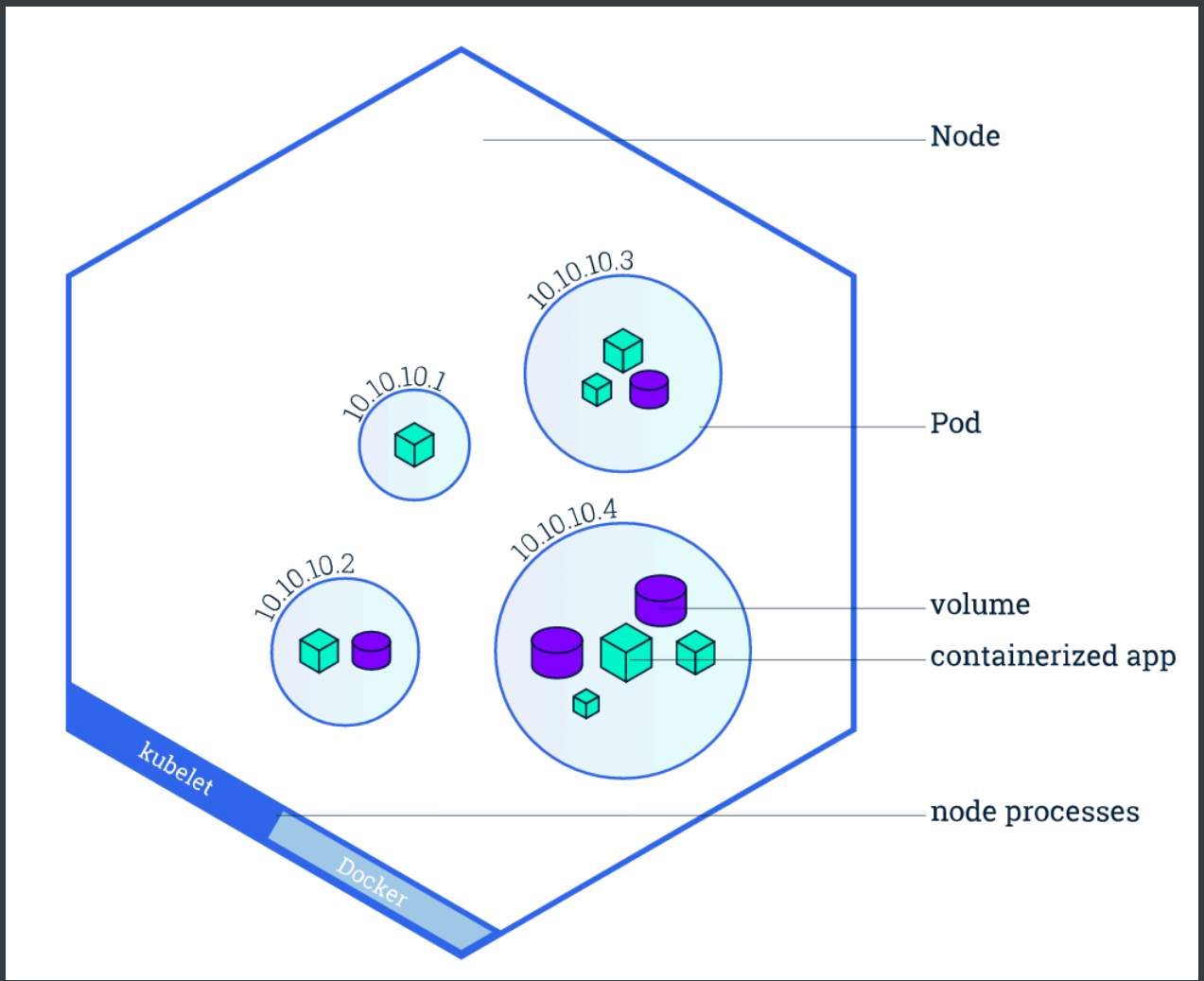
pod

在 Kubernetes 中，所有对象都使用 manifest（yaml 或 json）来定义，比如一个简单的 nginx 服务可以定义为 nginx.yaml，它包含一个镜像为 nginx 的容器：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

2.3 Node

Node 是 Pod 真正运行的主机，可以是物理机，也可以是虚拟机。为了管理 Pod，每个 Node 节点上至少要运行 container runtime（比如 docker 或者 rkt）、 `kubelet` 和 `kube-proxy` 服务。



node

2.4 Namespace

Namespace 是对一组资源和对象的抽象集合，比如可以用来将系统内部的对象划分为不同的项目组或用户组。常见的 pods, services, replication controllers 和 deployments 等都是属于某一个 namespace 的（默认是 default），而 node, persistentVolumes 等则不属于任何 namespace。

2.5 Service

Service 是应用服务的抽象，通过 labels 为应用提供负载均衡和服务发现。匹配 labels 的 Pod IP 和端口列表组成 endpoints，由 kube-proxy 负责将服务 IP 负载均衡到这些 endpoints 上。

每个 Service 都会自动分配一个 cluster IP（仅在集群内部可访问的虚拟地址）和 DNS 名，其他容器可以通过该地址或 DNS 来访问服务，而不需要了解后端容器的运行。