Kubernetes K8s 基础 理论

1. Kubernetes 简介

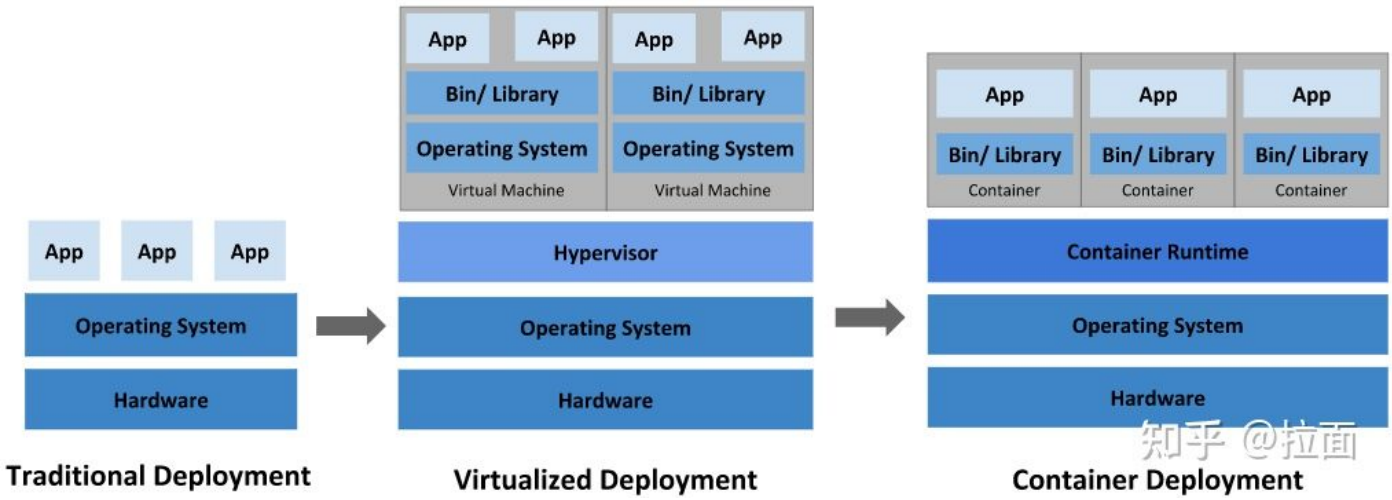
Kubernetes 又称 K8s（从开头字母 K 到末尾字母 s 中间一共有 8 个字符。。），是一款开源应用，用来对容器进行部署，扩展和管理（简称容器编排）。

K8s 中可以运行兼容 Kubernetes CRI (Container Runtime Interface)标准的容器。

K8s 需要与其它容器技术协同使用，比如 docker 做为底层容器基础。另外，K8s 本身不是用来构建镜像，而是管理由镜像生成的容器。

为了更好的理解 K8s，我们先简单回顾一下部署一般应用程序时，用到技术的进化。

图 1



说明：

- 传统部署时代：

物理服务器上直接安装操作系统（比如 Redhat Linux），然后在操作系统上直接安装应用软件（比如 Apache，tomcat）。

应用可以任意使用物理服务器的资源，没有边界保护且不易扩展；应用维护者一般还要同时维护操作系统及物理机。

- 虚拟化时代：

虚拟化技术(VM)诞生，我们在一台物理机上利用虚拟技术虚拟硬件，然后在虚拟硬件的基础上创建虚拟系统，每个虚拟系统具有完整的操作系统且彼此隔绝。

应用安装在虚拟系统之中，与物理机隔绝开。应用维护者只负责应用本身及虚拟系统。

虚拟系统的下面可能是一台物理服务器，也可能是多个物理服务器组成的集群。

虚拟系统的维护简便，可以方便的升级，扩展及回滚。

- 容器部署时代

容器与虚拟机（VM）类似，但不像 VM 之间那么隔绝。容器之间会共享安装容器的主机操作系统上的资源（CPU，内存，硬盘），但又与主机操作系统分隔开。

容器像是一个小的操作系统，可以在其中安装各种应用，这些应用不会对安装容器的主机操作系统产生影响（卷映射，端口映射等除外）。

所以容器可以方便地迁移到支持容器技术（比如 Docker）的各种操作系统上，不论是在云、VM 还是物理机上。

K8s 到底是做什么的?

当使用了某种容器技术之后（比如说安装了 Docker），我们就可以方便地利用镜像生成容器运行，就像我们在前几篇 Docker 相关的文章中做的那样。

这时如果访问量很大，我们想扩展容量多跑一个容器怎么办?

Docker 本身没法自动扩展，所以只能手工用同样的镜像再生成一个容器。

但这时新生成的容器端口如何映射（显然不能与之前用同一个端口），卷如何映射，前面有 Load Balancer 的话，该如何调整?

再有，如果运行容器的服务器宕机了，运行在这台服务器上的容器可是不会自动迁到其它可用服务器上去的。

上面这些问题，都可以通过 K8s 来解决。K8s 提供了一个框架，用来在集群（单机上也可以用）上管理容器的启停、扩展、部署方式等。

说 K8s 是一个框架，是因为 K8s 本身并不是一个完整的、拿来即用的产品。它提供了一个框架，但还需要额外的插件和资源来实现具体应用。

这也是各个云对 K8s 提供的服务的不同之处，内核都是 K8s，但是插件、可配合使用的资源各不相同。

K8s 提供的功能

1. 服务发现和负载均衡： K8s 可以通过 DNS 或者 IP 来暴露一个容器。如果对一个容器的请求过高，K8s 可以通过负载均衡来分配流量
2. 存储管理： K8s 允许使用本地或者第三方存储来保存持久化文件
3. 自动部署回滚： K8s 允许指定部署容器的状态，控制状态切换的时间
4. 自动装箱： 可以理解为通过指定需要的 CPU，内存，K8s 会为容器自动分配所在的 node
5. 自动愈合： K8s 可以自动重启失败的容器，并删除健康检查失败的容器
6. 密钥和配置管理： K8s 可以单独存储密码，SSH Key 等关键信息，这样更新这些密码信息时并不需要重新生成镜像

上面说的这些功能不理解的话，没关系。知道有这些概念即可，在实践中碰到了自然会知道。

2. K8s 的组件

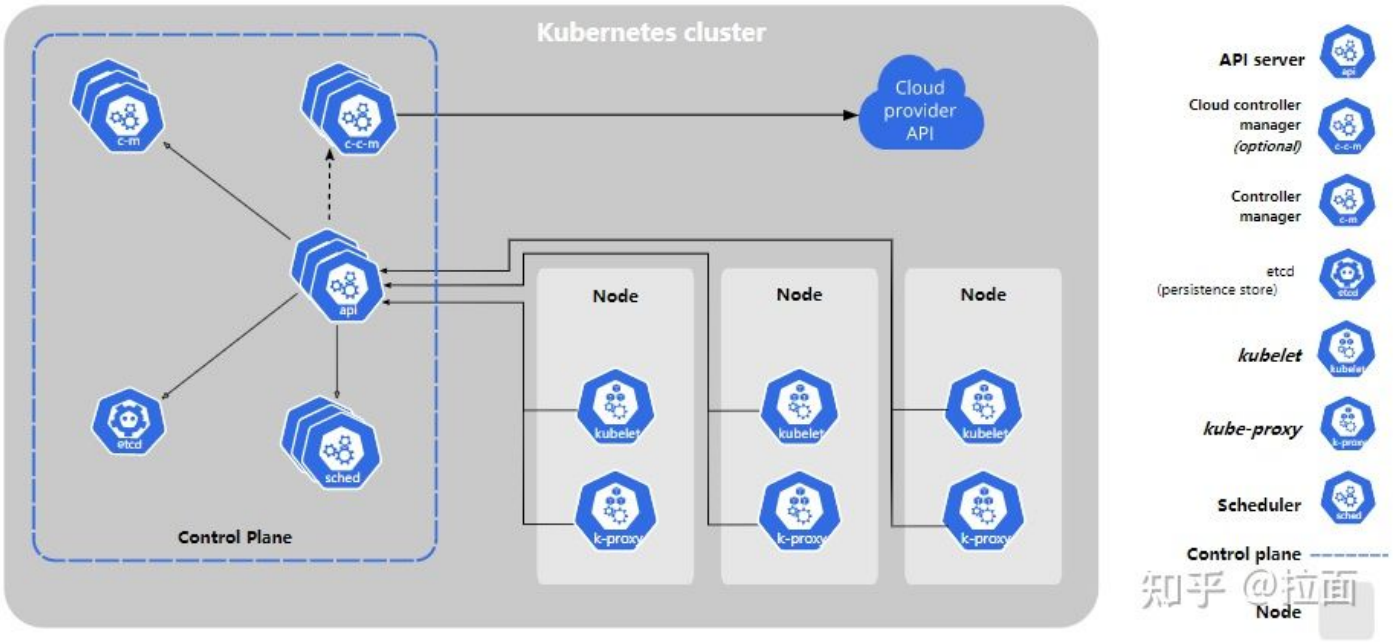
一个 K8s 集群由 Node 和 Control plane 两大组件构成

- Node：一个 node 对应一个服务器，可以是物理机，可以是 VM，也可以是云上的服务器（比如 EC2）。node 上会运行 pod（可以简单理解为容器），pod 是应用的具体载体，所以 node 是为应用 pod 提供计算服务的设备。
- Control plane：是用来管理 Node 的组件，Control Plane 可以由一个或多个服务器组成。

提示：Node 和 Control plane 可以理解为一个软件的两个组成部分，它们可以安装在一台机器上也可以分开安装在不同的机器上

K8s 集群结构如下图

图 2



说明：

在 AWS 中，当我们创建 EKS 集群的时候，创建的就是图中 Control Plane 这部分。AWS 会帮我们维护 Control Plane，并且不收费。

node 部分对应的就是 AWS 的 EC2，我们选定 EC2 的型号，数量之后，AWS 会启动 EC2 并在其上安装 K8s 相应的软件。

当我们自己做实验时，可以在本机安装一套 K8s，完整包括了 Node 和 Control Plane 两个组件。

下面我们简单说明一下 Control Plane 和 Nodes 各自拥有的主要组件。

这一部分内容主要涉及到 K8s 的管理，对于单纯使用 K8s 而言，这些内容了解即可。

Control Plane

Control Plane 的组件理论上可以运行在集群中的任意服务器上，但一般会默认装在同一台机器上。

对于 AWS 这样的云环境，Control Plane 完全由 AWS 托管，不需要我们进行维护。

Control Plane 主要由以下部分构成

- kube-apiserver： 用来暴露 K8s API。比如当我们用 kubectl 命令行部署 pod 的时候，Kubectl 命令行就是与 kube-apiserver 进行通信
- etcd： 键值数据库，用来存放 K8s 自身运行时需要的信息
- kube-scheduler： 为新建的，没有指定 node 的 pod，根据资源要求、policy 设定等安排工作 node
- kube-controller-manager：有各种 controller 控制不同的功能
  - Node controller： 负责通知及响应 node 宕机
  - Job controller： 负责运行一次性任务
  - Endpoints controller： 产生 Endpoints 对象（Joins service 和 pods）
  - Service Account & Token controllers： 创建默认帐号及 API access tokens

- cloud-controller-manager： 为各个云平台提供支持的服务，与云平台相关，在本地服务器运行 K8s 时没有此组件

Nodes

- kubelet：在每个 node 运行的代理程序，管理 Pod 中运行的容器，不负责非 Pod 中运行的容器（比如，我 node 上用 docker 命令单独启动的容器不在 kubelet 管辖范围内）
- kube-proxy： 在 node 上运行的网络代理，维护 K8s 的网络规则，这些规则控制集群内外访问 Pods 的通信
- Container runtime：容器的运行环境，比如说 Docker，这是需要单独安装的
- Addons： 插件利用 K8s 资源去实现集群的一些特性
- DNS： K8s DNS 是一个 DNS 服务器，它与环境中的其它 DNS 服务器一起负责 K8s 服务的 DNS 解析工作。K8s 启动容器时，DNS 会自动加入容器的 DNS 搜索列表中
- Web UI（Dashboard）： 提供图像界面管理 K8s
- Container Resource Monitoring： 记录各种 metric，但 K8s 只提供了 API 接口，无法直接使用
- Cluster-level Logging： 容器中默认日志输出到标准输出，会随着容器的消灭而灭亡，所以日志需要输出到外挂持久存储或者输出到第三方日志存储应用

3. K8s 对象

“在 Kubernetes 系统中，Kubernetes 对象是持久化的实体。Kubernetes 使用这些实体去表示整个集群的状态”

这是官网上 K8s 对象的定义，非常抽象。简单理解就是，在 K8s 中跑的一个 Pod 就是一个对象，在 K8s 中建的 Namespace，Service 等也是对象。

我们用 K8s 时，创建的所有内容都是 K8s 对象。以后在实际使用中，我们也是主要和 K8s 对象打交道。

对象的 Spec 和 Status

对象用 Spec 来描述其定义，用 Status 指定其运行时所在的状态。

K8s 会根据对象的 Spec 创建对象，根据 Status 来维护对象的运行情况（比如，pod failed 的时候 K8s 会自动帮你重启对象）。

描述对象

K8s 利用 yaml 文件来描述对象的 Spec 和 Status。

我们在 Yaml 文件中定义好需要的对象，然后利用 K8s API（kubectl 命令）把 Yaml 文件中的对象创建到 K8s 中。

下面是官网的一个 Yaml 文件“deployment.yaml”的例子

YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

说明：

我们在这个 Yaml 文件中指定对象的类型为“Deployment”，运行容器的镜像为“nginx:1.14.2”并打开 80 端口

这个文件写好之后，我们可以用下列命令创建 Deployment

Plain Text

```
kubectl apply -f deployment.yaml --record
```

可见，创建一个 Deployment 很简单。对于其它 K8s 对象也是这样，使用 K8s 的难点主要在于编写 K8s 的 yaml 文件。