Docker 与 Kube...

> 本篇是第八部分“生态篇”的第一篇。在这个部分，我会为你介绍 Docker 生态中的相关项目，以及如何参与到 Docker 项目中，最后会聊聊 Docker 未来的走向，本篇，我为你介绍下 Docker 与 Kubernetes 相关的内容。

整个专栏我们基本都在聊 Docker 的核心原理及其应用，本篇，我们进入“生态篇”的学习，一起来看看在 Docker 生态中，还有哪些值得关注的技术。

使用 Docker 的优势和痛点

在聊其他技术之前，我们先来聊聊使用 Docker 带来了哪些优势以及有哪些痛点，以便于对它以及其他技术有更好的认识。

优势

Docker 为我们提供了比较好的隔离性，应用程序可以通过将运行时的依赖一同打入镜像进行交付。

在构建镜像时，你可以将整个应用的编译，打包环境全部封装在 Dockerfile 中，再利用多阶段构建的特性缩小镜像体积。交付的时候甚至可以从交付镜像转变为交付 Dockerfile。

应用程序运行时，由于容器中已经包含了应用所需的基本依赖，也可以排除掉绝大多数系统环境差异造成的影响。这就为我们带来了很好的可 **移植性**。从根本上杜绝了类似于“在我电脑上运行是正常的”这种问题。

对于 Docker 镜像而言，可以为其指定对应的 repo:tag，可以将这些信息与代码的版本控制相结合。这就带来了良好的 **版本控制** 能力。

此外，正如我在前面内容中花了很大部分介绍的，通过使用 Docker 可以为我们提供非常多样性的安全策略，为我们的应用保驾护航。这便是良好的**安全性**。

另一方面，Docker 为我们提供了对容器进行资源管理相关的功能。可以较好的控制资源使用，以及进行资源隔离。

痛点

几十个容器，可能你还管的过来，但如果面临成百上千的容器，同时部署在多台机器上时，是否还那么好管理呢？

这就是我们在大规模使用 Docker 时，最大的痛点了。

大规模场景下，完全通过手工完成这些事情太繁杂了，所以我们需要一些工具或者平台来做这个事情。

你可能会想到比如 Ansible 之类的运维工具，你要是没有过于复杂的需求，或者靠 Docker 自身的功能都完全能满足业务的话，这也是一种可行的办法。

想象一下，在部署业务容器的时候，为了避免单点故障，通常我们会选择尽量将其分散部署。此时，就需要考虑服务器资源容量以及故障自愈相关的问题了。

通过人工和自动化脚本也能完成，但略麻烦了点。 这个时候出现了很多的解决方案。

Kubernetes 的诞生

2014 年 Google 基于其大规模生产实践经验开源出来的 Kubernetes（K8S），集容器编排、服务发现、故障自愈、资源管理调度等功能于一身，在开源后，迅速得到了众多公司的青睐。

同时，也存在着其他比较活跃的解决方案，如 Mesos 和 Docker Swarm 等，一度形成了三足鼎立的局面。

随着各公司的采纳度以及厂商的支持，Kubernetes 已经成为容器编排领域的事实标准。所以当前主流的推荐是 Kubernetes。

很多人觉得 Kubernetes 是一个 PaaS（Platform as a Service）系统，但 Kubernetes 并不是真正意义上的 PasS 系统。它实际工作在容器层，提供了一些与 PasS 类似或者共同的功能，比如部署、扩容、监控、负载均衡、日志记录等。

然而它并不是个完全一体化的平台，这些功能基本都是可选可配置的。

Docker 与 Kubernetes 之间的联系

Kubernetes 中最基本的部署单元叫做 Pod，每个 Pod 中可以包含一个或者多个容器。

而启动容器则必须有一个运行时来完成。Docker 最主要的作用便是这个了。

起初，Docker 作为容器生态的领头羊，是 Kubernetes 的默认运行时。后来，CoreOS 公司发布了一个名为 rkt 的项目，Kubernetes 为了能给用户提供更多的选择，也将 rkt 作为了一个可选的容器运行时。

再之后，由于出现了更多的容器运行时，比如从 Docker 中拆分出来的 containerd 等，Kubernetes 发现为每个容器运行时提供支持，会带来很多的麻烦。而且需要改动不少代码。所以 Kubernetes 提出了一个概念，称之为 CRI（Container Runtime Interface——容器运行时接口）。

凡是符合 CRI 规范的容器运行时，均可作为容器运行时使用。

到现在为止，Docker 仍然是 Kubernetes 中使用最多的容器运行时。 同时，Docker 也是使用最多的容器平台（或工具）。

Docker x Kubernetes

介绍了这么多，那如何能快速的使用 Kubernetes 呢?

Docker Desktop for Windows/Mac 均内置了 Kubernetes，如果你已经安装了最新版本的 Docker Desktop，可以通过设置中启动其内置的 Kubernetes。

除了这种方式外，我要特别推荐另一种方式：KIND（Kubernetes IN Docker），将 Kubernetes 运行在 Docker 中。

使用这种方式的好处在于，你可以轻松的在已经安装了 Docker 的机器上，很方便的启动一个单节点或者多节点的 Kubernetes 集群。（注：多节点需要更多的系统资源）

示例如下。

1. 先安装 kind 的二进制文件：

```
(MoeLove) ➜  ~ wget -q -O kind https://github.com/kubernetes-sigs/kind/releases/d
(MoeLove) ➜  ~ chmod +x kind
(MoeLove) ➜  ~ ./kind version
kind v0.7.0 go1.13.6 linux/amd64
```

2. 使用 kind 启动一个单节点的 Kubernetes 集群：

```
(MoeLove) ➜  ~ ./kind create cluster --name moelove
Creating cluster "moelove" ...
 ✓ Ensuring node image (kindest/node:v1.17.0) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-moelove"
You can now use your cluster with:

kubectl cluster-info --context kind-moelove

Not sure what to do next? 😅  Check out https://kind.sigs.k8s.io/docs/user/quic
```

注：这里需要下载一个 `kindest/node:v1.17.0` 的镜像，托管在 DockerHub 上，镜像体积是 1.25GB。下载速度取决于你的网速情况。

3. 安装 kubectl

安装完成后，输出内容中提示你可以通过 kubectl 命令操作集群。所以我们需要安装 kubectl。这是一个 Kubernetes 集群的 CLI 工具。它目前托管在 Kubernetes 的官方存储仓库中，你可以在 GitHub 的 Release 页面进行下载。当然，**我们可以有更简单的办法。**

复制

```
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE                  COMMAND                  CREATED
ee85ab2746a6        kindest/node:v1.17.0   "/usr/local/bin/entr…"   4 minutes ago
(MoeLove) ➜  ~ docker cp moelove-control-plane:/kind/bin/kubectl .
(MoeLove) ➜  ~ ./kubectl get nodes
NAME                     STATUS   ROLES    AGE    VERSION
moelove-control-plane    Ready    master   5m2s   v1.17.0
```

可以直接从容器中将 kubectl 二进制文件拷贝到本地。

总结

本篇，我为你大概介绍了 Kubernetes 诞生的背景，以及使用 Docker 的优势及痛点。

同时介绍了如何在本地使用 Docker 创建一个单节点的 Kubernetes 集群。此集群可用于本地快速的测试和体验。