

02 | 如何将容器镜像部署到K8s?

2022-12-12 王伟 来自北京

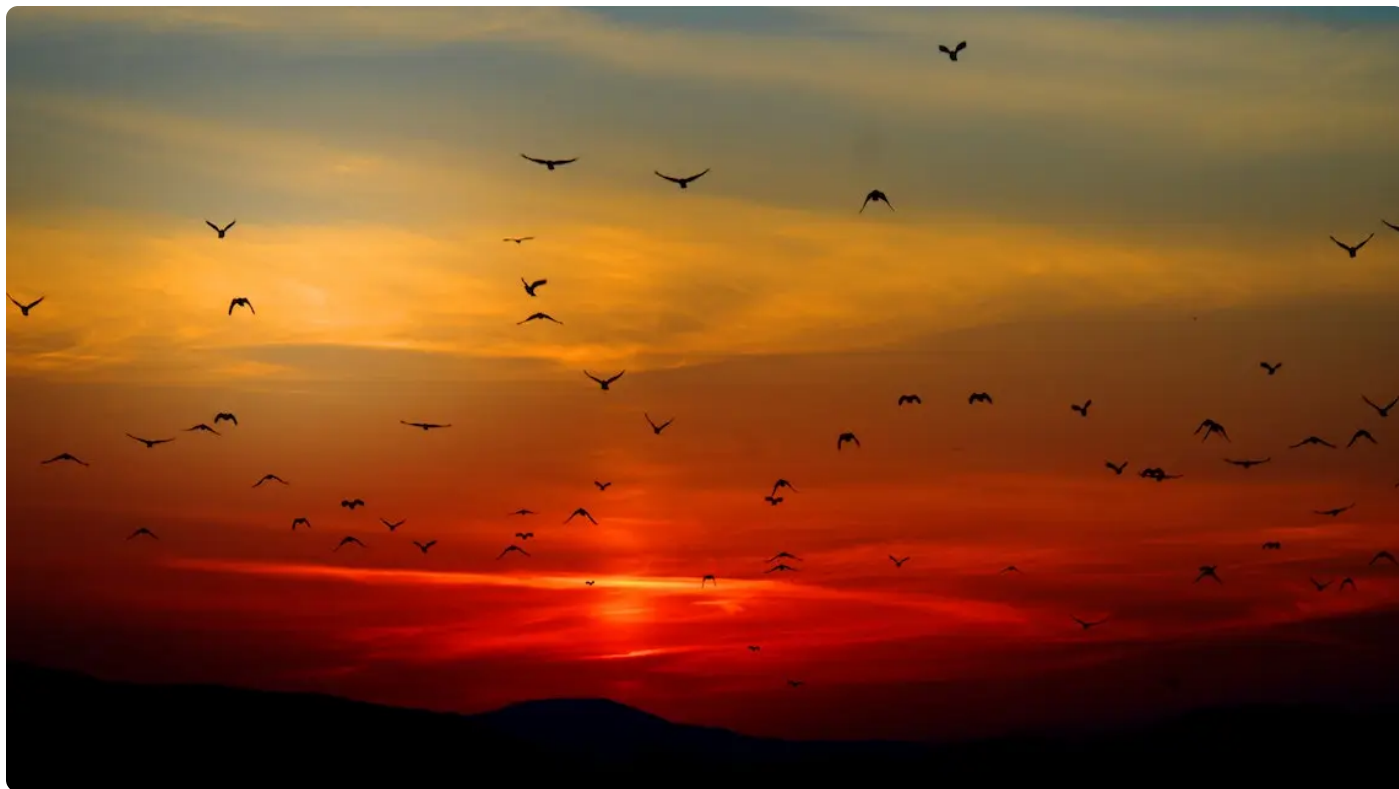


天下无鱼

<https://shikey.com/>

《云原生架构与GitOps实战》

[课程介绍 >](#)



讲述：王伟

时长 10:40 大小 14.62M



你好，我是王伟。

上节课，我们学习了容器镜像的概念以及如何将业务构建成容器镜像。容器镜像是我们学习云原生的起点，同时也会贯穿在云原生的各个环节之中，希望你能多花点时间巩固。这节课，我们一起来看看如何将容器镜像部署到 K8s。

K8s 的系统设计非常复杂，这导致它的学习曲线非常陡峭。尤其是对于刚入门的同学而言，面对繁杂的概念时往往会不知所措，也无从下手。再加上学习过程缺乏实战性的反馈，很容易导致“从入门到放弃”的情况出现。

所以这节课，我打算摒弃概念先行的教学方法，从实战的角度出发，带你从零入门 K8s。

我会继续延伸上一节课程的内容，带你把之前构建的容器镜像部署到 K8s 集群中。在实战的过程中，我也会讲解 K8s 的一些基本概念。

在开始实践之前，你需要做好以下准备：

- 准备一台电脑（首选 Linux 或 macOS，Windows 也适用，需要稍微注意一下操作上的差异）；
- [安装 Docker](#)；
- [安装 Kubectl](#)。

安装 Kubernetes

如何获得一个 K8s 集群是每一个开发者需要熟练掌握的内容。

安装 K8s 的方法非常多，有生产级的安装方法，也有以测试为目标的安装方法。为了方便测试，这里我推荐一种在本地安装 K8s 集群的办法：Kind。

首先，你需要根据[官方步骤](#)安装 Kind，它是一个命令行工具，使用非常简单。

接下来，我们开始创建 K8s。

将以下内容保存为 config.yaml：

 复制代码

```
1 kind: Cluster
2 apiVersion: kind.x-k8s.io/v1alpha4
3 nodes:
4 - role: control-plane
5   kubeadmConfigPatches:
6   - |
7     kind: InitConfiguration
8     nodeRegistration:
9       kubeletExtraArgs:
10         node-labels: "ingress-ready=true"
11   extraPortMappings:
12   - containerPort: 80
13     hostPort: 80
14     protocol: TCP
15   - containerPort: 443
16     hostPort: 443
17     protocol: TCP
```

执行 `kind create` 命令，创建 K8s 集群：



```
1 > kind create cluster --config config.yaml
2 Creating cluster "kind" ...
3   ✓ Ensuring node image (kindest/node:v1.23.4)
4   ✓ Preparing nodes
5   ✓ Writing configuration
6   ✓ Starting control-plane
7   ✓ Installing CNI
8   ✓ Installing StorageClass
9 Set kubectl context to "kind-kind"
10 You can now use your cluster with:
11
12 kubectl cluster-info --context kind-kind
```

通过这两个简单的步骤，我们的本地 K8s 集群就创建好了。

初识 Kubernetes

K8s 的概念既多又复杂，如果像教科书一样把这些概念一条一条给你讲出来，最后你可能也记不住。所以，我也不打算这么做。

我想先问你一个问题：Docker 已经挺好用了，我们为什么还需要 K8s 呢？你可以试着用 30 秒来思考一下。

我用一个场景来给你解释一下。还记得我们在上节课提到的启动一个容器的方法吗？这条命令非常简单，只需要运行 `docker run` 就可以了。

但是试想一下，如果你需要同时启动 10 个不同的容器镜像，是不是需要运行 10 次 `docker run` 命令呢？此外，如果容器之间有依赖顺序，你是不是还需要额外记住这 10 条命令特定的启动顺序？

K8s 的独特之处在于，它为我们抽象了诸如“启动 10 个容器镜像”这样的过程式的命令，你只需要向 K8s 描述“我需要 10 个容器”。10 个容器是我期望的最终状态，我不管怎么执行命令，执行了多少次命令等过程，我想要的就是这个结果。

用来向 K8s 描述“期望最终状态”的文件，就叫做 K8s Manifest，也可以称之为清单文件。

Manifest 就好比餐厅的菜单，你只管点菜，做菜的过程我不管。



到这里，我们引出了第一个概念：**Manifest**。简单地说，它是用来描述如何将容器镜像部署到集群中的。Manifest 的概念非常重要，它会贯穿整个 K8s 的生态系统，所以请你务必要理解。

现在，假设我们有了一个 Manifest，我要如何告诉 K8s 呢？换句话说，我们怎么和 K8s 交流？这时候我们需要引入一个工具：**Kubectl**。

Kubectl 是一个与 K8s 集群交互的工具，通过 Kubectl，我们可以非常方便地以 Manifest 为媒介操作 K8s 集群的对象。就像操作数据库一样，我们可以对 Manifest 所描述的对象进行创建、删除、修改、查找等操作。

接下来，我们从编写 Manifest 开始，看看怎么将上节课制作的容器镜像部署到 K8s 集群中。

部署容器镜像到 K8s

要将容器镜像部署到 K8s 集群中，我们首先需要书写 Manifest。我们需要将下面的内容保存为 flask-pod.yaml：


复制代码

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: hello-world-flask
5 spec:
6   containers:
7     - name: flask
8       image: lyzhang1999/hello-world-flask:latest
9       ports:
10        - containerPort: 5000
```

接下来，我们通过 kubectl apply 应用这段 Manifest：

复制代码

```
1 $ kubectl apply -f flask-pod.yaml
2 pod/hello-world-flask created
```

需要注意的是，当使用 Kind 创建集群后，它会自动配置 Kubectl 和 Kind 集群的连接。如果连接配置不正确，你可以执行 `kind export kubeconfig` 来切换集群连接的上下文。 <https://shikey.com/>

如果成功看到输出内容 `pod/hello-world-flask created`，说明我们已经把这段 Manifest 提交到集群里了。参数 `-f` 表示“指定一个 Manifest 文件”。

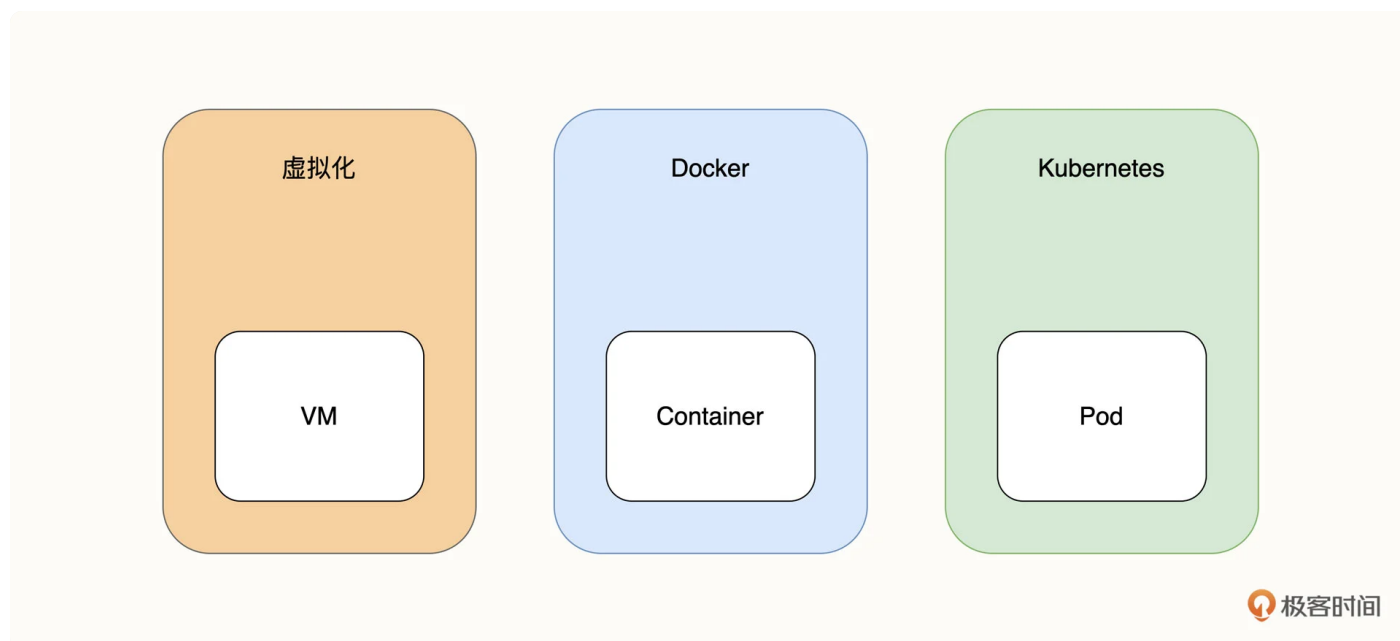
也就是说，当我们想要向 K8s 提交 Manifest 的时候，只需要记住一条命令，那就是 **kubectl apply**。

我们对上面的 Manifest 做一些必要的解释。其实这里只需要聚焦四个字段，分别是 **Kind**、**Containers**、**Image**、**Ports**。

Kind 字段表示 K8s 的工作负载类型。在 K8s 中，我们不能像 Docker 一样直接运行一个容器镜像，镜像需要依赖于 K8s 更上层的封装方式运行，这种封装方式也就是工作负载，Pod 是工作负载的一种类型。

每个工作负载有自己的名字，也就是 Metadata 节点下的 Name 字段。

为了更好地帮助你理解，我们可以把 Pod 类比成虚拟化技术中的 VM，或者是 Docker 技术的容器，它们都是一种调度对象，如下图所示：



在实际的项目中，我们一般不会直接创建 Pod 类型的工作负载，而是会通过其它的工作负载间接创建它们。如果你现在还不能理解 Pod 的概念也没关系，只需要记住它是 K8s 调度的最小单位就可以了。



Containers 字段代表 Pod 要运行的容器配置，例如名称、镜像和端口等。细心的你会注意到，它是一个数组类型，这意味着我们可以在一个 Pod 里面配置多个容器。

Image 字段表示容器镜像。显然，这里我们希望启动的是上节课制作的镜像，也就是 lyzhang1999/hello-world-flask。需要注意将这里的 lyzhang1999 替换为你自己的 docker hub 账户 ID。

Ports 字段表示容器要暴露的端口。它有点像我们在上节课提到的启动镜像时暴露的容器端口，也就是 docker run 的 p 参数。很显然我们希望暴露 5000 端口，也就是业务进程的监听端口。

接下来，我们继续回到实战环节。

当我们把 Manifest 部署到 K8s 集群之后，会面临两个问题：

1. 如何查看刚才提交的 Pod？
2. 如何访问 Pod？

查看和访问 Pod

先看第一个问题。要查看 K8s 集群正在运行中的 Pod，你可以使用 `kubectl get pods`：

复制代码

```
1 $ kubectl get pods
2 NAME                READY   STATUS    RESTARTS   AGE
3 hello-world-flask    1/1     Running   0           1m
```

可以看到，结果中返回了刚才部署名为 hello-world-flask 的 Pod，状态为“运行中”。还记得上节课我们提到怎么查看运行中容器的命令吗？是不是非常类似？

接下来我们尝试访问 Pod。要在本地访问集群内的 Pod，我们可以使用 `kubectl port-forward` 命令进行端口转发操作，打通容器和本地网络：



复制代码

```
1 $ kubectl port-forward pod/hello-world-flask 8000:5000
2 Forwarding from 127.0.0.1:8000 -> 5000
3 Forwarding from [::1]:8000 -> 5000
```

你可能会问，既然已经在 Manifest 里定义了 Ports 参数，为什么还需要进行端口转发操作呢？那是因为 Ports 参数只定义了 Pod 在集群内部暴露的端口，这个端口可以在集群内部进行访问。但我们本地和集群的网络是隔离的，自然不能在集群外部，也就是本地电脑访问 Pod。

执行 `kubectl port-forward` 之后，我们打开浏览器访问 `127.0.0.1:8000` 就可以看到输出了：

复制代码

```
1 Hello, my first docker images! hello-world-flask
```

需要注意的是，端口转发的进程是在前台运行的，你可以使用 `ctrl+c` 终止转发进程。

除了访问 Pod 以外，上节课我们还提到可以用 `docker exec` 进入运行中的容器。同样，我们也可以进入到 Pod 运行中的容器内。

你可以使用 `kubectl exec` 进入到 Pod 容器内部：

复制代码

```
1 $ kubectl exec -it hello-world-flask -- bash
2 root@hello-world-flask:/app#
```

可以看到，这里的 `-it` 参数和 `docker exec` 的参数是完全一致的。在进入容器后，你可以试着像第一节那样，修改 `app.py` 的输出，保存后刷新浏览器，观察一下是不是也能立即看到修改结果呢？

最后，我们可以通过 `kubectl delete` 命令来删除 Pod：

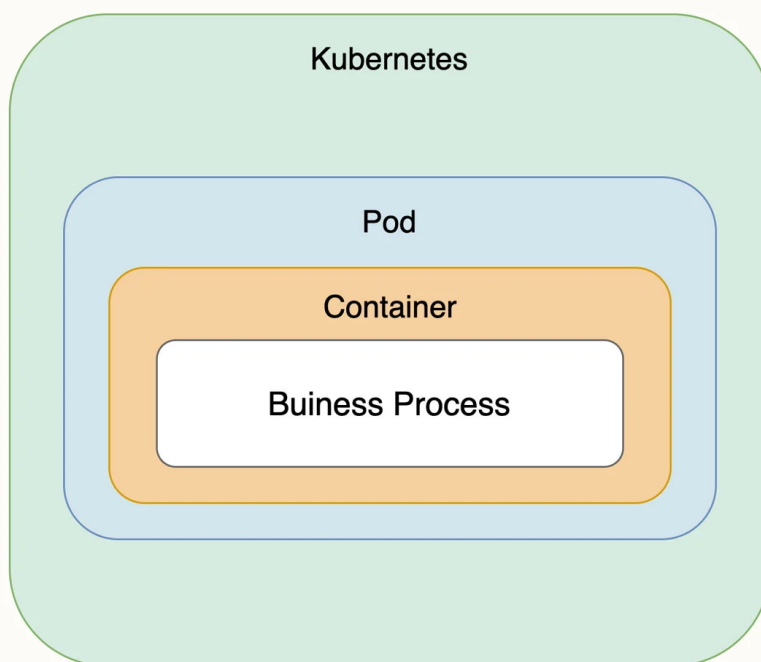
```
1 $ kubectl delete pod hello-world-flask
2 pod "hello-world-flask" deleted
```

到这里，我们就完成了将容器镜像部署到 K8s，还有查看、访问和删除 Pod 这一系列操作。

业务进程、容器镜像和工作负载之间的关系

通过上面的实战，我们将一个容器镜像以工作负载的方式部署到了 K8s 集群中。那么如果再往前追溯到业务层面，业务进程、容器镜像和工作负载之间的关系又是怎样的呢？

当我初次接触 K8s 的时候，我对业务进程、容器镜像和工作负载层层封装，以及他们相互之间的关系感到非常疑惑。或许你也有同样的疑问，我为你总结了一张图，可以帮助你梳理一下它们之间的关系。



通过这张图可以看出，最内层是我们的业务应用进程，外层通过 Docker 镜像以容器化的形式运行，再往外是 K8s 的最小调度单位 Pod。

总结

这节课，我们通过 Kind 在本地创建了一个 K8s 集群。同时，在将容器镜像部署到 K8s 集群的过程中，我还给你介绍了 Manifest 以及 K8s 的最小调度单位 Pod 的概念。



此外，我们还介绍了与 K8s 集群交互的工具 Kubectl 及其简单操作，例如向集群提交 Manifest、端口转发、查看、进入以及删除 Pod 等。

最后，我们还梳理了业务进程、镜像、容器和 Pod 之间的关系，理解它们将有助于你未来学习 K8s 的其他复杂概念。

到这里，你可能会问：学习了这么多概念，也花了这么多努力把容器镜像部署到了 K8s 集群，可是为什么还没看到 K8s 究竟能给我们带来什么实际的好处呢？

别着急，下节课，我会带你直观地感受 K8s 的魅力和强大之处：**自动扩容和自愈**。

思考题

最后，给你留两道思考题吧。

1. Kind 的全称是 Kubernetes in Docker，它的设计理念非常有意思。请你结合相关资料，说一说你对它的理解。
2. 请你尝试为 hello-world-flask Pod 增加第二个容器，镜像为 nginx，端口为 80，并通过端口转发访问 Nginx Pod。

欢迎你给我留言交流讨论，你也可以把这节课分享给更多的朋友。我们下节课见。

分享给需要的人，Ta 购买本课程，你将得 18 元

 生成海报并分享

 赞 4  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

更多课程推荐

云原生架构与 GitOps 实战

即学即用，攻破云原生核心技术

王伟

前腾讯云 CODING 架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言 (4)

 写留言



陈志成

2022-12-14 来自广东

kind 有很多设计理念，比如

1. 优雅降级

因为 kind 是专为测试 k8s 设计的，所以自身很顽强。支持所有的 k8s 官方版本，并且如其名字所示，docker 里搞出来的 k8s，集群依赖的所有的镜像都打包在了它的节点镜像里，厉害的是，如果这些镜像损坏了或者丢失了，节点镜像还能帮你重新拉下来，真是打不死的小强。

2. 不重复发明轮子

虽然现在已经有了很多功能类似的软件，比如大名鼎鼎的 kubeadm，但是 kind 并不是来搅局的，他很谦虚的抱着协作的态度，能复用就复用，站稳巨人的肩膀，专注解决痛点，合作共赢。

3. 最小化假设

kind 表示只要求安装 docker 即可，剩下的就交给我了。避免做出任何不必要的假设，minikube 表示，你礼貌吗？

4. 封闭原则，无状态原则

力求操作幂等、避免依赖外部服务，本身不存储或管理状态。好家伙，低内聚，高耦合，本身还没有副作用，不愧是 kind，确实很友好。

5. 避免伤害用户，遵循 k8s API 约定

啥？还有这原则。来看看都啥内容，避免配置和命令行接口的不兼容性修改，严格遵守 k8s 的弃用政策，面向外部的特性考虑扩展性以及长期支持性，使用 k8s 的配置风格，减少参数个数...一上来就海誓山盟还真是不习惯。

另外还有自动化和支持 CRI，感兴趣的可以了解了解 <https://kind.sigs.k8s.io/docs/design/principles/>

作者回复: 幽默又不失知识点，优秀！



Promise

2022-12-14 来自浙江

1. Kind是一个使用Docker容器 "节点 "运行本地Kubernetes集群的工具。 Kind主要是为测试Kubernetes本身而设计，但也可用于本地开发或CI。

2.

apiVersion: v1

kind: Pod

metadata:

name: hello-world-flask

spec:

containers:

- name: flask

image: lyzhang1999/hello-world-flask:latest

ports:

- containerPort: 5000

- name: flask

image: nginx:latest

ports:

- containerPort: 80

kubectl apply -f flask-pod.yaml

kubectl port-forward pod/hello-world-flask 80:80

其实可以使用Service和Ingress来访问Nginx Pod



fireheart

freshon

2022-12-13 来自广东

Kind还没有看。



天下无鱼

<https://shikey.com/>

说一下思考题2:

将 flask-pod.yaml 更新为:

apiVersion: v1

kind: Pod

metadata:

name: hello-world-flask

spec:

containers:

- name: flask

image: lyzhang1999/hello-world-flask:latest

ports:

- containerPort: 5000

- name: nginx

image: nginx:1.23.2

ports:

- containerPort: 80

执行 `kubectl apply -f .\flask-pod.yaml` 就可以。

端口转发: `kubectl port-forward pod/hello-world-flask 80:80`

作者回复:



Jack

2022-12-12 来自广东

MacBook-Pro-2 gitOps % `kubectl get pods`

NAME	READY	STATUS	RESTARTS	AGE
hello-world-flask	0/1	ErrImagePull	0	4s

老师, `kubectl apply -f flask-pod.yaml`之后运行的结果是失败的, 如上图。这个如何解决呢?

flask-pod.yaml的配置文件如下:

apiVersion: v1

kind: Pod

metadata:

name: hello-world-flask

spec:

containers:

- name: flask

image: lyzhang1999/hello-world-flask:latest

ports:

- containerPort: 5000



作者回复: 看起来是镜像拉取失败了, 建议开一台香港 Linux 主机实验。

共 8 条评论 >

