1. 设置Docker环境
2. 生成镜像并将其作为一个容器运行
3. 在开发机器上设置和使用Kubernetes环境
4. 在开发机器上设置和使用Swarm环境
5. 在Docker Hub上共享您的容器化应用程序

## 定向和设置

### 1、Docker概念

Docker是供开发人员和系统管理员使用容器构建、共享和运行应用程序的平台。使用容器来部署应用程序称为容器化。

容器化受欢迎的原因：

* 灵活：即使最复杂的应用程序也可以容器化。
* 轻量级：容器利用并共享主机内核，在系统资源方面比虚拟机更有效。
* 可移植：可以在本地构建，部署到云并在任何地方运行。
* 松散耦合：容器是高度自给自足并封装的容器，可以在不破坏其他容器的情况下更换或升级它们。
* 可扩展：可以在数据中心内增加并自动分发容器副本。
* 安全：容器将积极约束和隔离应用于流程，而无需用户方面的任何配置。

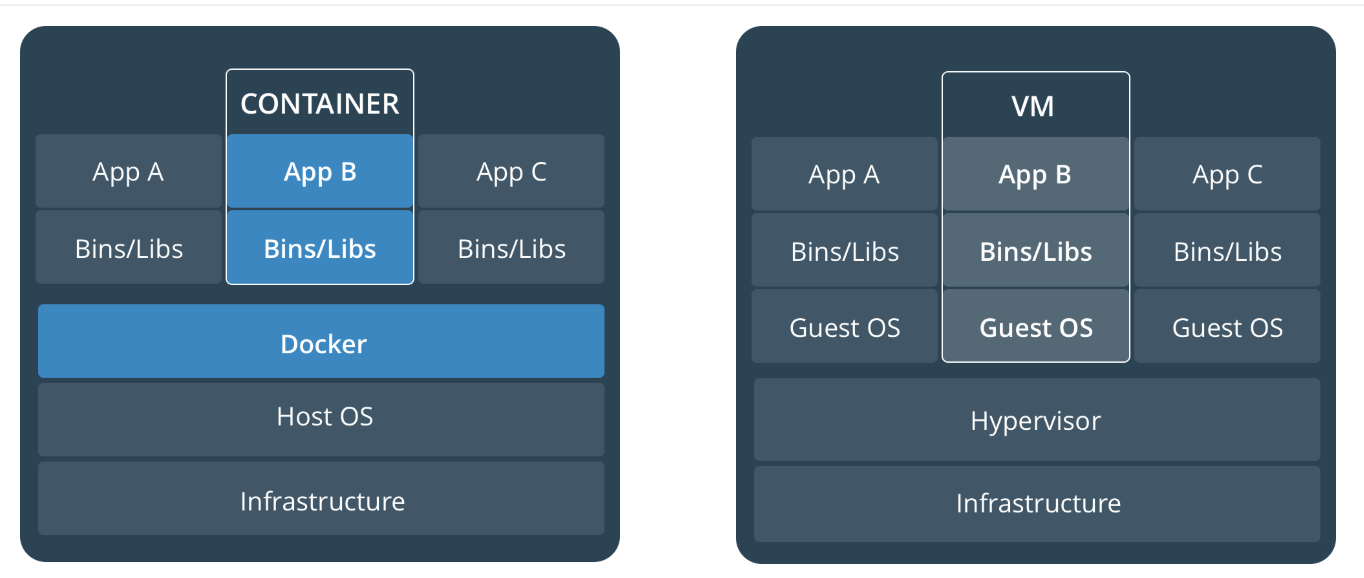
### 2、镜像和容器

本质上，一个容器不过是一个正在运行的进程，并对其应用了一些附加的封装功能，以使其与主机和其他容器隔离。容器隔离的最重要的方面之一是每个容器都与自己的私有文件系统进行交互。该文件系统由Docker镜像提供。镜像包括运行应用程序所需的所有内容-代码或二进制文件，运行时，依赖以及所需的任何其他文件系统对象。

### 3、容器和虚拟机

容器在Linux上本地运行，并与其他容器共享主机的内核。它运行一个分离的进程，不占用任何其他可执行文件更多的内存，从而使其轻巧。

相比之下，VM运行成熟的“guest”操作系统，并通过管理程序对主机资源进行虚拟访问。通常，VM会产生大量开销，超出了应用程序逻辑所消耗的开销。



### 4、安装Docker桌面

开始开发容器化应用程序的最佳方法是使用OSX或Windows的Docker桌面。Docker Desktop将允许您轻松地在本地开发计算机上设置Kubernetes或Swarm，因此您可以立即使用正在开发应用程序的Orchestrator的所有功能，而无需集群。请遵循适合您的操作系统的安装说明：

* [OSX](https://docs.docker.com/docker-for-mac/install/)
* [Windows](https://docs.docker.com/docker-for-windows/install/)

1）安装完成后，检查版本docker –version。

2）运行简单的Docker镜像来检测安装是否正常：docker run hello-world

如果本地找不到该镜像，则Docker将其从Docker Hub中拉出。

3）启动Dockerized Web服务器。

docker run --detach --publish=80:80 --name=webserver nginx

本地找不到该镜像，则docker将其从Docker Hub中拉出。

在web浏览器中访问<http://localhost/>已查看nginx主页。指定了默认的HTTP端口号，所以无需在URL末尾附加:80。



4）在Web服务器运行时吗，使用命令查看容器上的详细信息。

docker ps

或docker container ls

5）使用以下命令停止并删除容器和镜像。使用“all”标志（--all 或 -a）查看停止的容器。

docker container stop webserver

docker container ls -a

docker container rm webserver

[Docker Desktop相关配置](https://docs.docker.com/docker-for-mac/)

#### 4.1 Kubernetes

Docker Desktop包含一个可在Mac上运行的独立Kubernetes服务器，因此可以测试在Kubernetes上部署Docker工作负载。

Kubernetes客户端命令kubectl已包括在内，并配置为连接到本地Kubernetes服务器。如果已经安装kubectl并指向其他环境，例如minikube GKE集群，请确保更改上下文，以使kubectl指向docker-for-desktop。

$ kubectl config get-contexts

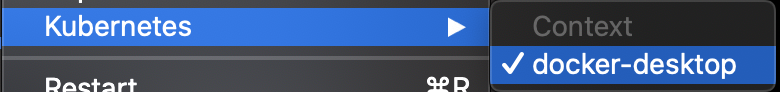
$ kubectl config use-context docker-desktop

要启用Kubernetes支持并安装作为Docker容器运行的Kubernetes独立实例，请选择Enable Kubernetes。要将Kubernetes设置为默认协调器，默认选择将Docker堆栈部署到Kubernetes。



应用并重启，这将实例化将Kubernetes服务器作为容器运行所需的镜像，并将/usr/local/bin/kubectl命令安装在Mac上。

Kubernetes的状态显示在Docker菜单中，上下文指向docker-desktop。



* 默认情况下，Kubernetes容器对诸如docker service ls之类的命令是隐藏的，因为不支持手动管理它们。要使其可见，请选择显示系统容器（高级），然后应用并重启。大多数用户不需要此选项。
* 要随时禁用Kubernetes支持，请清除“启动Kubernetes”复选框。Kubernetes容器将停止并删除，并且/usr/local/bin/kubectl命令也将删除。

更多关于Kubernetes信息参考[Deploy on Kubernetes](https://docs.docker.com/docker-for-mac/kubernetes/)。

#### 4.2 添加TLS证书

可以将受信任的证书颁发机构（CAs）（用于验证注册表服务器证书）和客户端证书（用于对注册表进行身份验证）添加到Docker守护进程。

#### 4.3 添加自定义CA证书（服务器端）

支持所有受信任的CA（root or intermediate）。 Docker Desktop根据Mac密钥链创建所有用户信任的CA的证书捆绑包，并将其附加到Moby信任的证书。 因此，如果主机上的用户信任企业SSL证书，则Docker Desktop会信任它。

要手动添加自定义的自签名证书，请先将证书添加到macOS钥匙串中，然后由Docker Desktop获取。 下面是一个例子：

$ sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain ca.crt

如果只想将证书添加到自己的本地钥匙串中（而不是对所有用户），请运行以下命令。

$ security add-trusted-cert -d -r trustRoot -k ~/Library/Keychains/login.keychain ca.crt

参阅[证书的目录结构](https://docs.docker.com/docker-for-mac/#directory-structures-for-certificates)。

#### 4.4 添加客户端证书

可以将客户端证书放入~/.docker/certs.d/<MyRegistry>:<Port>/client.cert and ~/.docker/certs.d/<MyRegistry>:<Port>/client.key

Docker Desktop应用程序启动时，会将Mac上的~/.docker/certs.d文件夹复制到Moby（Docker Desktop xhyve虚拟机）上的/etc/docker/certs.d目录。

#### 4.5 完成安装shell

Docker Desktop附带了用于启动docker，docker-machine和docker-compose命令的脚本。可以在Docker.app中的Contents/Resource/etc目录中找到completion脚本，并且可以将其安装在Bash和Zsh中。

Bash

Bash内置了对要激活Docker命令的完成的支持，需要将这些文件复制或符号链接到bash\_completion.d/目录。例如通过homebrew安装的bash：

etc=/Applications/Docker.app/Contents/Resources/etc

ln -s $etc/docker.bash-completion $(brew --prefix)/etc/bash\_completion.d/docker

ln -s $etc/docker-machine.bash-completion $(brew --prefix)/etc/bash\_completion.d/docker-machine

ln -s $etc/docker-compose.bash-completion $(brew --prefix)/etc/bash\_completion.d/docker-compose

将下列配置加入~/.bash\_profile文件中：

[ -f /usr/local/etc/bash\_completion ] && . /usr/local/etc/bash\_completion

或者

if [ -f $(brew --prefix)/etc/bash\_completion ]; then

. $(brew --prefix)/etc/bash\_completion

fi

Zsh

在Zsh中，completion system负责处理处理。要激活Docker命令的completion功能，需要将这些文件复制或符号链接到Zsh site-functions/目录。例如，如果是通过Homebrew安装了Zsh，则：

etc=/Applications/Docker.app/Contents/Resources/etc

ln -s $etc/docker.zsh-completion /usr/local/share/zsh/site-functions/\_docker

ln -s $etc/docker-machine.zsh-completion /usr/local/share/zsh/site-functions/\_docker-machine

ln -s $etc/docker-compose.zsh-completion /usr/local/share/zsh/site-functions/\_docker-compos

#### 4.6 帮助论坛[Docker Desktop for Mac forum](https://forums.docker.com/c/docker-for-mac)

#### 4.7 Docker Hub

从Docker Desktop菜单中选择登陆/创建Docker ID以访问Docker Hub账户。登陆后，可以直接从Docker Desktop菜单访问Docker Hub存储库和组织。

双因素验证：

Docker Desktop使您可以使用双因素身份验证登录Docker Hub。当您访问Docker Hub帐户时，双要素身份验证提供了额外的安全性。

您必须先在Docker Hub中启用双重身份验证，然后才能通过Docker Desktop登录到您的Docker Hub帐户。

启用两因素身份验证后：

1）转到Docker Desktop菜单，然后选择登录/创建Docker ID。

2）输入您的Docker ID和密码，然后单击登录。

3）成功登录后，Docker Desktop会提示您输入身份验证代码。输入手机的六位数代码，然后点击验证。

成功通过身份验证后，您可以直接从Docker Desktop菜单访问组织和存储库。

### 5、启用Kubernetes

Docker Desktop上启动Kubernetes后，为确保Kubernetes已启动并运行，创建一个pod.yaml：

apiVersion: v1

kind: Pod

metadata:

name: demo

spec:

containers:

- name: testpod

image: alpine:3.5

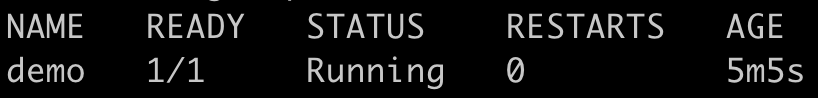
command: ["ping","8.8.8.8"]

在pod.yaml位置执行命令创建你的pod：

kubectl apply -f pod.yaml

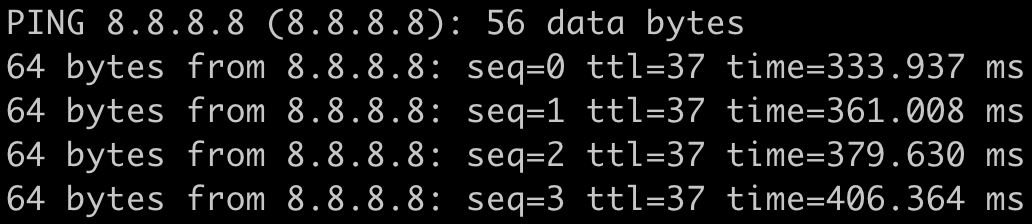
检查pod是否启动并运行：

kubectl get pods



检查是否获得了ping操作所需的日志：

kubectl logs demo



拆除测试pod：

kubectl delete -f pod.yaml

### 6、启用Docker Swarm

Docker Desktop主要在Docker Engine上运行，它具有运行Swarm所需的一切内置功能。按照适用系统的设置和验证说明进行操作：

1）打开终端，初始化

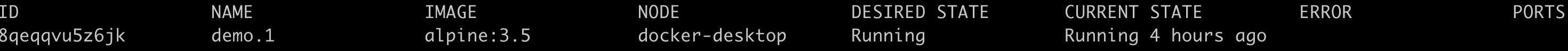
docker swarm init

2）运行一个简单的Docker服务，该服务使用alpine-bases的文件系统，并将ping隔离到8.8.8.8。

docker service create --name demo alpine:3.5 ping 8.8.8.8

3）检查服务是否创建了一个正在运行的容器。

docker service ps demo



4）检查是否获得了ping操作所需的日志。

docker service logs demo

5）最后关闭测试服务。

docker service rm demo

### 7、结论

至此，已经在开发机器上安装了Docker Desktop，并确认可以在Kubernetes和Swarm中运行简单的容器化工作负载。

### 8、CLI（Command Line Interface） References——命令行界面参考

以下提供了文本中使用的所有CLI命令的文档：

[kubectl apply](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#apply)

[kubectl get](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#get)

[kubectl logs](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#logs)

[kubectl delete](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#delete)

[docker swarm init](https://docs.docker.com/engine/reference/commandline/swarm_init/)

[docker service \*](https://docs.docker.com/engine/reference/commandline/service/)

## 应用容器化

### 1、介绍

借助Docker Desktop，已经在开发环境中设置了协调器，可以开始开发容器化的应用程序。通常，开发工作流程如下所示：

1. 首先创建Docker镜像，为应用程序的每个组件创建和测试单独的容器。
2. 将容器和支持基础架构组装成一个完整的应用程序，以Docker堆栈文件或Kubernetes YAML表示。
3. 测试，共享和部署完整的容器化应用程序。

在本教程的此阶段，重点放在第1步：创建容器将基于的镜像。Docker镜像捕获了我们的容器化进程将在其中运行私有文件系统；我们需要创建一个镜像，其中包含我们的应用程序需要运行的内容。

一旦学习了如何构建镜像，容器化的开发环境就比传统的开发环境更容易建

立，这是因为一个容器化的开发环境会将应用程序所需的所有依赖项隔离在Docker映像中；不需在开发机器上安装Docker以外的任何东西。这样你可以轻松地为不同的堆栈开发应用程序，而无需在开发计算机上进行任何更改。

### 2、配置

1）从GitHub克隆一个示例项目

git clone -b v1 https://github.com/docker-training/node-bulletin-board.git

这是一个简单的公告板应用程序，node.js编写。现在正在尝试对其进行容器化。

2）查看Dockerfile文件。描述了如何为容器组装私有文件系统，并且还可以包含一些元数据，这些元数据描述了如何基于该镜像运行容器。

FROM node:6.11.5

WORKDIR /usr/src/app

COPY package.json .

RUN npm install

COPY . .

CMD [ "npm", "start" ]

编写Dockerfile是应用程序容器化的第一步。可以将这些Dockerfile命令视为有关如何构建镜像的逐步指南。采取以下步骤：

* 从先前存在的node:6.11.5镜像开始。这是由node.js供应商构建的官方镜像，并已由Docker验证为包含node 6.11.5解释器和基本依赖项的高质量镜像。
* 使用WORKDIR可以指定所有后续操作均应从镜像文件系统中的目录/usr/src/app中执行（而不是主机的文件系统中）。
* 从你的主机当前COPY package.json文件到你的镜像的当前位置（.）。在这种情况下，复制到 /usr/src/app/package.json。
* 运行你的镜像文件系统内部的命令npm install（该命令将读取package.json以确定应用程序的节点依赖性，并安装它们）。
* 从主机将应用程序其余部分的源代码复制到镜像文件系统。

上面的步骤构建了镜像的文件系统，但是Dockerfile中还有一行。CMD指令是我们在镜像中指定一些元数据的第一个示例，该元素描述了如何基于该镜像运行容器。这就是说该镜像应支持的容器化过程是npm start。

[Dockerfile参考](https://docs.docker.com/engine/reference/builder/)

### 3、建立并测试你的镜像

现在我们有一些源代码和一个Dockerfile，现在该构建第一个镜像，并确保从其启动的容器可以按预期工作。

1）确保终端位于node-bulletin-board/bulletin-board-app目录中，并构建bulletin board镜像：

docker image build -t bulletinboard:1.0 .

将看到Docker逐步完成Dockerfile中的每条指令，逐步构建镜像。如果成功，则构建过程应以Successfully tagged bulletinboard:1.0结束。

2）根据你的新镜像启动一个容器：

docker container run --publish 8000:8080 --detach --name bb bulletinboard:1.0

--publish 要求Docker将主机端口8000上传入的流量转发到容器的端口8080

（容器具有自己的专用端口集,因此如果要从网络访问一个端口，则必须以这种方式转发流量；否则，防火墙规则将阻止所有网络流量到达您的容器，这是默认的安全状态）。

--detach 要求Docker在后台运行此容器。

--name 指定一个名称，在以后的命令中，可以使用该名称来引用我们的容器bb。

注意：我们没有指定我们要运行容器的进程。我们没有必要，因为使用CMD在构建Dockerfile时使用了指令。因此，Docker知道npm start在容器启动时会自动运行该过程。

3）在浏览器中访问应用程序localhost:8000，可以看到公告板应用程序已启动并正在运行。

4）对公告板容器正常工作感到满意后，将其删除。

docker container rm --force bb

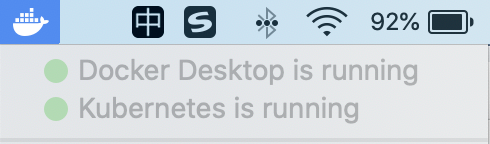
### 4、结论

至此，我们已经对应用程序进行了简单的容器化，并确认我们的应用程序已在其容器中成功运行。下一步将是编写Kubernetes yaml，以描述如何在Kubernetes上运行和管理这些容器，或者编写将使我们能够在Docker Swarm上进行相同操作的堆stack文件。

## 部署到Kubernetes

### 1、先决条件

* 完成对应用程序的容器化。
* 确保在Docker桌面上启用了Kubernetes。



### 2、介绍

应用程序的各个组件都是作为独立容器运行的，现在该安排它们由Kubernetes这样的协调器来管理。Kubernetes提供了许多工具来扩展，联网，保护和维护你的容器化应用程序。

为了验证我们的容器化应用程序是否可以在Kubernetes上正常工作，我们将在开发机器上直接使用Docker Desktop内置的Kubernetes环境部署我们的应用程序，然后再将其移交给生产环境中的完整Kubernetes集群运行。Docker Desktop创建的Kubernetes环境具有全部功能，这意味着它具有你的应用程序在真正集群上享受的所有Kubernetes功能，可通过开发机器便利地进行访问。

### 3、使用Kubernetes YAML描述应用

Kubernetes中的所有容器都计划为pods，这是一组共享资源的共享容器。此外，在实际应用中，我们几乎不创建单个pods。取而代之的是大多数工作负载都按期部署，它们是Kubernetes自动维护的可扩展pods组。最后，所有Kubernetes对象可以在Kubernetes YAML文件的清单中进行描述。这些YAML文件描述了Kubernetes应用程序的所有组件和配置，可用于在任何Kubernetes环境中轻松创建和销毁你的应用程序。

1）编写一个Kubernetes YAML文件，以允许和管理bulletin board。将以下内容放在名为bb.yaml的文件中。

apiVersion: apps/v1

kind: Deployment

metadata:

name: bb-demo

namespace: default

spec:

replicas: 1

selector:

matchLabels:

bb: web

template:

metadata:

labels:

bb: web

spec:

containers:

- name: bb-site

image: bulletinboard:1.0

---

apiVersion: v1

kind: Service

metadata:

name: bb-entrypoint

namespace: default

spec:

type: NodePort

selector:

bb: web

ports:

- port: 8080

targetPort: 8080

nodePort: 30001

在YAML文件中，有两个对象，以“--”分隔：

* 一个Deployment，描述一组可扩展的相同pods。在这种情况下，你将获得你的pod的一个副本，并且该pod（在template:键下进行描述）中仅包含一个容器。该容器基于bulletinboard：1.0镜像。
* 一个NodePort服务，它将流量从主机上的30001端口路由到内部的8080端口，从而可以从网络访问公告栏。

虽然Kubernetes YAML看起来很长很复杂，但几乎总是遵循相同的模式：

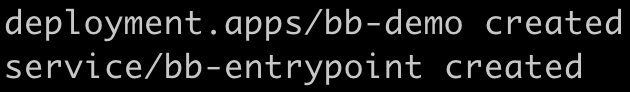
* apiVersion：指示解析此对象的Kubernetes API。
* kind：指定创建资源的角色/类型。
* metadata：资源的元数据/属性。
* spec：指定该资源的内容。

### 4、部署并检查应用程序

1）在终端中，导航到创建位置，bb.yaml然后将应用程序部署到Kubernetes:

kubectl apply -f bb.yaml

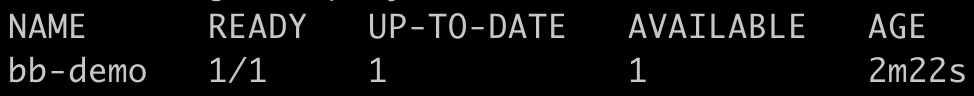
可以看到如下输出，表示你的Kubernetes对象已经成功创建：



2）查看部署确保一切正常：

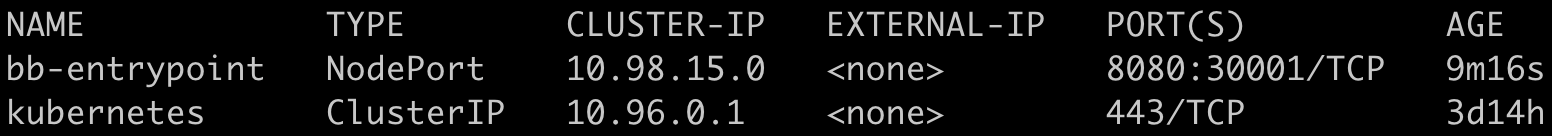
kubectl get deployment

如果一切顺利，则你的部署会被列出来：



这表明在YAML中要求的所有pods都已启动并正在运行。对你的服务进行检查：

kubectl get services



除了默认Kubernetes服务之外，还看到我们的bb-entrypoint服务，该服务接收端口30001/TCP上的流量。

3）打开浏览器，访问localhost:30001，可以看到你的bulletin board，就像前面将其作为独立容器运行时一样。

4）拆除你的应用程序：

kubectl delete -f bb.yaml

### 5、结论

至此，我们已经成功地使用Docker Desktop将我们的应用程序部署到开发机器上功能齐全的Kubernetes环境中。你可以在自己的机器上开始向应用程序添加其他组件，并利用Kubernetes的所有强大的功能。

除了部署到Kubernetes，还可以将应用程序描述为Kubernetes YAML文件。这个简单的文本文件包含在运行状态下创建应用程序所需的一切；我们可以将其检入版本控制并与同事共享，从而使我们可以轻松地将应用程序分发到其他集群（例如可能在我们开发环境之后的测试和生产集群）。

### 6、Kubernetes参考

有关本文中使用的所有新Kubernetes对象的更多文档，请参见：

* [Kubernetes Pods](https://kubernetes.io/docs/concepts/workloads/pods/pod/)
* [Kubernetes Deployments](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/)
* [Kubernetes Services](https://kubernetes.io/docs/concepts/services-networking/service/)

## 部署到Swarm

### 1、先决条件

* 完成对应用程序的容器化。
* 输入docker system info查找消息Swarm:active，确保在Docker Desktop上启用了Swarm。（如果Swarm没有运行，只需在shell提示符下键入docker swarm init即可进行设置）。

### 2、介绍

现在，我们已经演示了应用程序的各个组件可以作为独立容器运行，并展示了如何使用Kubernetes进行部署，让我们看看如何安排它们由Docker Swarm管理。Swarm提供了许多工具，可以扩展，联网，保护和维护您的容器化应用程序，而容器本身无法提供这些工具。

为了验证我们的容器化应用程序是否可以在Swarm上正常工作，我们将在开发机器上使用Docker Desktop内置的Swarm环境来部署我们的应用程序，然后再将其交付以在生产环境中的完整Swarm集群上运行。Docker Desktop创建的Swarm环境具有完整的功能，这意味着它具有你的应用程序将在真正的集群上享受的所有Swarm功能，可通过开发机器的便利地进行访问。

### 3、使用Stack文件描述应用

Swarm从来不会像教程那样创建单个容器。相反，所有Swarm工作负载都计划为services，它们是可伸缩的容器组，具有由Swarm自动维护的附加网络功能。此外，所有Swarm对象都在stack文件的清单中进行描述。这些YAML文件描述了Swarm应用程序的所有组件和配置，可用于在任何Swarm环境中轻松创建和销毁你的应用程序。

1）编写一个简单的stack文件来运行和管理bulletin board。将以下内容放入bb-stack.yaml：

version: '3.7'

services:

bb-app:

image: bulletinboard:1.0

ports:

- "8000:8080"

在这个Swarm YAML文件中，我们只有一个对象：一个service，描述一组可伸缩的相同容器。在这种情况下，你只会得到一个容器（默认），该容器将基于bulletinboard:1.0镜像。我们还要求Swarm将到达开发机器8000端口的所有流量转发到bulletinboard容器内的8080端口。

Kubernetes服务和Swarm服务有很大的不同！尽管名称相似，但两个协调器在

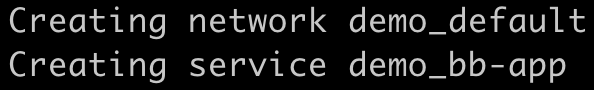
“service”一词中的含义却截然不同。在Swarm中，服务同时提供调度和联网功能，创建容器并提供用于将流量路由到它们的工具。在Kubernetes中，调度和联网分别处理：部署（或其他控制器）将容器的调度作为pods处理，而服务仅负责将网络功能添加到这些pods。

### 4、部署并检查你的应用程序

1）将应用程序部署到Swarm：

docker stack deploy -c bb-stack.yaml demo

如果一切顺利，Swarm将报告创建的所有stack对象没有任何问题：

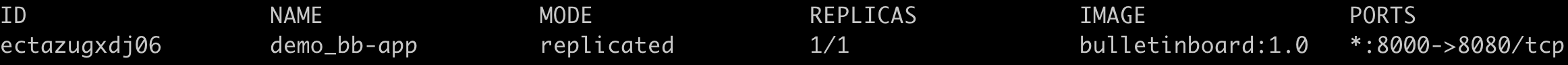


注意：除服务外，Swarm默认还会创建一个Docker网络，以隔离作为stack一部分的容器。

2）列出服务来确保一切正常：

docker service ls

你的服务将报告其创建的副本1/1：



这表示你在服务器中要求的1/1个容器已启动并正在运行。此外，我们看到开发计算机上的端口号8000已转发到bulletinboard容器中的端口8080。

3）访问localhost:8000

4）删除你的应用程序：

docker stack rm demo

### 5、结论

至此，已经成功地使用Docker Desktop将应用程序部署到开发机器上的功能齐全的Swarm环境中。现在可以在自己的计算机上向应用程序添加其他组件，并利用Swarm的强大功能。

除了部署到Swarm之外，我们还将应用程序描述为stack文件。这个简单的文本文件包含在运行状态下创建应用程序所需的一切；我们可以将其检入版本控制并与同事共享，从而使我们可以轻松地将应用程序分发到其他集群（例如可能在我们开发环境之后的测试和生产集群）。

Swarm和CLI参考

此处提供了本文中使用的所有新Swarm对象和CLI命令的更多文档：

* [Swarm Services](https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/)
* [Swarm Stacks](https://docs.docker.com/engine/swarm/stack-deploy/)
* [docker stack \*](https://docs.docker.com/engine/reference/commandline/stack/)
* [docker service \*](https://docs.docker.com/engine/reference/commandline/service/)

## 在Docker Hub上共享镜像

### 1、先决条件

* 完成对应用程序的容器化。

### 2、介绍

开发容器化应用程序的最后一步是在Docker Hub之类的注册中心上共享镜像，以便可以轻松下载它们并在任何目标集群上运行它们。

### 3、设置你的Docker Hub账户

## 4、创建并推送到Docker Hub存储库

至此，已经设置了Docker Hub账户并将其连接到Docker Desktop。现在进行第一个存储库并在那里共享公告栏应用程序。

1）在Docker Hub上新建一个存储库。

2）创建完成后准备在Docker Hub上共享镜像，但是我们首先要做的是：必须正确命名镜像的名称才能在Docker Hub上共享。具体来说，镜像必须命名为<Docker Hub ID>/<Repository Name>:<tag>。我们可以这样重新标记bulletinboard:1.0镜像：

docker image tag bulletinboard:1.0 chancewu/bulletinboard:1.0

3）将镜像推送到Docker Hub

在Docker Hub中访问存储库，将在此处看到新镜像。默认情况下，Docker Hub存储库是公共的。

### 5、结论

现在，镜像已在Docker Hub上可用，可以在任何地方运行它。如果尝试在尚未安装的新群集上使用它，则Docker将自动尝试从Docker Hub下载它。通过以这种方式移动镜像，我们不再需要在要运行软件的机器上安装Docker和Orchestrator以外的任何依赖项；容器化应用程序的依赖关系已完全封装并隔离在我们的镜像中，我们可以按上述方式通过Docker Hub进行共享。

需要记住的另一件事：目前，我们仅将您的映像推送到Docker Hub；你的Dockerfile，Kube YAML和Stack File呢？关键的最佳实践是将它们保留在版本控制中，或者与应用程序的源代码一起保留，并在Docker Hub存储库描述中添加链接或注释，以指示可以在何处找到这些文件，不仅保存图像的记录，是构建的，但它是作为完整的应用程序运行的。