

这可能是最为详细的Docker入门吐血总结

转载 邓彪1022 最后发布于2019-01-15 15:21:55 阅读数 162132 ☆ 收藏

Docker是什么？

在计算机技术日新月异的今天， Docker 在国内发展的如火如荼，特别是在一线互联网公司， Docker 的使用是十分普遍的，甚至成为了企业面试的话看看下面这张图。

这是我在某招聘网站上看到的招聘 **Java开发工程师** 的招聘要求，其中有一条熟悉 docker 成为了你快速入职的加分项，由此可见熟悉 docker 在互联网行业的重要。

当然对于我们 **CTF选手** 而言，熟悉 docker 可以快速搭建 CTF环境，完美地还原比赛真实漏洞的场景，帮助我们快速提升自己。

市面上已经有很多优秀的教程，但是很多原理性的东西，笔者认为那些教程对初学者而言还是很难理解，感觉没有说清楚(笔者自己都觉得挺懵逼的)，少走弯路，我将以我的学习经历以及作为一个 **CTF选手** 的角度，编写此套教程，来带大家去了解并熟练运用 docker，祝愿各位读者朋友们学完此套教程来企业面试中能够多一项加分的筹码，能够帮助到大家，我觉得就很值了。

既然说了这么多， docker 到底是个什么东西呢？

我们在理解 docker 之前，首先我们得先区分清楚两个概念，**容器和虚拟机**。

可能很多读者朋友都用过虚拟机，而对容器这个概念比较的陌生。

我们用的传统虚拟机如 VMware，VisualBox 之类的需要模拟整台机器包括硬件，每台虚拟机都需要有自己的操作系统，虚拟机一旦被开启，预分配全部被占用。每一台虚拟机包括应用，必要的二进制和库，以及一个完整的用户操作系统。

而容器技术是和我们的宿主机共享硬件资源及操作系统，可以实现资源的动态分配。容器包含应用和其所有的依赖包，但是与其他容器共享内核。容器系统中，在用户空间以分离的进程运行。

容器技术是实现操作系统虚拟化的一种途径，可以让您在资源受到隔离的进程中运行应用程序及其依赖关系。通过使用容器，我们可以轻松打包应用配置和依赖关系，将其变成容易使用的构建块，从而实现环境一致性、运营效率、开发人员生产力和版本控制等诸多目标。容器可以帮助保证应用程序快速、一致地部署，其间不受部署环境的影响。容器还赋予我们对资源更多的精细化控制能力，让我们的基础设施效率更高。通过下面这幅图我们可以很直观的区别所在。

Docker 属于 Linux 容器的一种封装，提供简单易用的容器使用接口。它是目前最流行的 Linux 容器解决方案。

而 Linux 容器是 Linux 发展出了另一种虚拟化技术，简单来讲，Linux 容器不是模拟一个完整的操作系统，而是对进程进行隔离，相当于是在正常进程外增加一个保护层。对于容器里面的进程来说，它接触到的各种资源都是虚拟的，从而实现与底层系统的隔离。

Docker 将应用程序与该程序的依赖，打包在一个文件里面。运行这个文件，就会生成一个虚拟容器。程序在这个虚拟容器里运行，就好像在真实的物一样。有了 Docker，就不用担心环境问题。

总体来说，Docker 的接口相当简单，用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像代码一样。

Docker的优势

Docker相比于传统虚拟化方式具有更多的优势：

- docker 启动快速属于秒级别。虚拟机通常需要几分钟去启动
- docker 需要的资源更少，docker 在操作系统级别进行虚拟化，docker 容器和内核交互，几乎没有性能损耗，性能优于通过 Hypervisor 层与硬件交互
- docker 更轻量，docker 的架构可以共用一个内核与共享应用程序库，所占内存极小。同样的硬件环境，Docker 运行的镜像数远多于虚拟机数量，资源利用率非常高
- 与虚拟机相比，docker 隔离性更弱，docker 属于进程之间的隔离，虚拟机可实现系统级别隔离
- 安全性：docker 的安全性也更弱。Docker 的租户 root 和宿主机 root 等同，一旦容器内的用户从普通用户权限提升为root权限，它就具备root权限，进而可进行无限制的操作。虚拟机租户 root 权限和宿主机的 root 虚拟机权限是分离的，并且虚拟机利用如 Intel 的 VT-d 和 VT-x 的隔离技术，这种隔离技术可以防止虚拟机突破和彼此交互，而容器至今还没有任何形式的硬件隔离，这使得容器容易受到攻击



180



28



举报

- 可管理性： docker 的集中化管理工具还不算成熟。各种虚拟化技术都有成熟的管理工具，例如 VMware vCenter 提供完备的虚拟机管理能力
- 高可用和可恢复性： docker 对业务的高可用支持是通过快速重新部署实现的。虚拟化具备负载均衡，高可用，容错，迁移和数据保护等经过生产熟保障机制， VMware 可承诺虚拟机 99.999% 高可用，保证业务连续性
- 快速创建、删除：虚拟化创建是分钟级别的， Docker 容器创建是秒级别的， Docker 的快速迭代性，决定了无论是开发、测试、部署、交付、部署：虚拟机可以通过镜像实现环境交付的一致性，但镜像分发无法体系化。 Docker 在 Dockerfile 中记录了容器构建过程，和快速部署

我们可以从下面这张表格很清楚地看到容器相比于传统虚拟机的特性的优势所在：

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为MB	一般为GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般是几十个

Docker的三个基本概念

从上图我们可以看到， Docker 中包括三个基本的概念：

- Image(镜像)
- Container(容器)
- Repository(仓库)

镜像 是 Docker 运行容器的前提，仓库是存放镜像的场所，可见镜像更是 Docker 的核心。

Image (镜像)

那么镜像到底是什么呢？

Docker 镜像可以看作是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

镜像（ Image ）就是一堆只读层（ read-only layer ）的统一视角，也许这个定义有些难以理解，下面的这张图能够帮助读者理解镜像的定义。

从左边我们看到了多个只读层，它们重叠在一起。除了最下面一层，其它层都会有一个指针指向下一层。这些层是Docker 内部的实现细节，并且能够系统上访问到。统一文件系统 (union file system) 技术能够将不同的层整合成一个文件系统，为这些层提供了一个统一的视角，这样就隐藏了多层的7的角度看来，只存在一个文件系统。我们可以在图片的右边看到这个视角的形式。

Container (容器)

容器 (container) 的定义和镜像 (image) 几乎一模一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。

由于容器的定义并没有提及是否要运行容器，所以实际上，容器 = 镜像 + 读写层。

Repository (仓库)

Docker 仓库是集中存放镜像文件的场所。镜像构建完成后，可以很容易的在当前宿主上运行，但是， 如果需要在其它服务器上使用这个镜像，我们就的存储、分发镜像的服务， Docker Registry (仓库注册服务器)就是这样的服务。有时候会把仓库 (Repository) 和仓库注册服务器 (Registry) 混为一谈区分。 Docker 仓库的概念跟 Git 类似，注册服务器可以理解为 GitHub 这样的托管服务。实际上，一个 Docker Registry 中可以包含多个仓库 (Repo个仓库可以包含多个标签 (Tag)，每个标签对应着一个镜像。所以说，镜像仓库是 Docker 用来集中存放镜像文件的地方类似于我们之前常用的代码仓

通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本。我们可以通过<仓库名>:<标签>的格式来指定本镜像。如果不给出标签，将以 latest 作为默认标签。

仓库又可以分为两种形式：

- public(公有仓库)

- private(私有仓库)

Docker Registry 公有仓库是开放给用户使用、允许用户管理镜像的 Registry 服务。一般这类公开服务允许用户免费上传、下载公开的镜像，并可能供用户管理私有镜像。

除了使用公开服务外，用户还可以在本地搭建私有 Docker Registry 。Docker 官方提供了 Docker Registry 镜像，可以直接使用做为私有 Registry 服务。创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库 pull 下来。

我们主要把 Docker 的一些常见概念如 Image ， Container ， Repository 做了详细的阐述，也从传统虚拟化方式的角度阐述了 docker 的优势，我们直观地看到 Docker 的架构：



Docker 使用 C/S 结构，即**客户端/服务器**体系结构。Docker 客户端与 Docker 服务器进行交互，Docker 服务端负责构建、运行和分发容器。客户端和服务端可以运行在一台机器上，也可以通过 RESTful 、 stock 或网络接口与远程 Docker 服务端进行通信。



这张图展示了 Docker 客户端、服务端和 Docker 仓库（即 Docker Hub 和 Docker Cloud ），默认情况下 Docker 会在 Docker 中央仓库管理镜像的设计理念类似于 Git ，当然这个仓库是可以通过修改配置来指定的，甚至我们可以创建我们自己的私有仓库。

Docker的安装和使用

Docker 的安装和使用有一些前提条件，主要体现在体系架构和内核的支持上。对于体系架构，除了 Docker 一开始就支持的 X86-64 ，其他体系架构在不断地完善和推进中。

Docker 分为 CE 和 EE 两大版本。CE 即社区版（免费，支持周期 7 个月），EE 即企业版，强调安全，付费使用，支持周期 24 个月。

我们在安装前可以参看官方文档获取最新的 Docker 支持情况，官方文档在这里：

<https://docs.docker.com/install/>

Docker 对于内核支持的功能，即内核的配置选项也有一定的要求(比如必须开启 Cgroup 和 Namespace 相关选项，以及其他网络和存储驱动等)，中提供了一个检测脚本来检测和指导内核的配置，脚本链接在这里：

<https://raw.githubusercontent.com/docker/docker/master/contrib/check-config.sh>

在满足前提条件后，安装就变得非常的简单了。

Docker CE 的安装请参考官方文档：

- MacOS：<https://docs.docker.com/docker-for-mac/install/>
- Windows：<https://docs.docker.com/docker-for-windows/install/>
- Ubuntu：<https://docs.docker.com/install/linux/docker-ce/ubuntu/>
- Debian：<https://docs.docker.com/install/linux/docker-ce/debian/>
- CentOS：<https://docs.docker.com/install/linux/docker-ce/centos/>
- Fedora：<https://docs.docker.com/install/linux/docker-ce/fedora/>
- 其他 Linux 发行版：<https://docs.docker.com/install/linux/docker-ce/binaries/>

这里我们以 CentOS7 作为本文的演示。


环境准备


- 阿里云服务器(1核2G，1M带宽)
- CentOS 7.4 64位


由于 Docker-CE 支持 64 位版本的 CentOS7 ，并且要求内核版本不低于 3.10


首先我们需要卸载掉旧版本的 Docker


```
1 $ sudo yum remove docker \
2   docker-client \
3   docker-client-latest \
```



180



Docker Registry 服务



28 条回复














举报

```
4 | docker-common \ 5 | docker-latest \
6 | docker-latest-logrotate \
7 | docker-logrotate \
8 | docker-selinux \
9 | docker-engine-selinux \
10 | docker-engine
```



180



28



我们执行以下安装命令去安装依赖包：

```
1 | $ sudo yum install -y yum-utils \
2 |     device-mapper-persistent-data \
3 |     lvm2
```

这里我事先已经安装过了，所以提示我已经安装了最新版本

安装Docker

Docker 软件包已经包括在默认的 CentOS-Extras 软件源里。因此想要安装 docker，只需要运行下面的 yum 命令

```
$ sudo yum install docker
```

当然在测试或开发环境中 Docker 官方为了简化安装流程，提供了一套便捷的安装脚本，CentOS 系统上可以使用这套脚本安装：

```
1 | curl -fsSL get.docker.com -o get-docker.sh
2 | sh get-docker.sh
```

具体可以参看 docker-install 的脚本：

```
https://github.com/docker/docker-install
```

执行这个命令后，脚本就会自动的将一切准备工作做好，并且把 Docker CE 的 Edge 版本安装在系统中。

安装完成后，运行下面的命令，验证是否安装成功：

```
1 | docker version
2 | or
3 | docker info
```

返回docker的版本相关信息，证明 docker 安装成功

启动Docker-CE

```
1 | $ sudo systemctl enable docker
2 | $ sudo systemctl start docker
```

Docker的简单运用---Hello World

由于服务器日常崩溃了，docker 出了点问题，所以下案例的演示是基于 Kali Linux 环境下进行的。

我们通过最简单的 image 文件 hello world，感受一下 Docker 的魅力吧！

我们直接运行下面的命令，将名为 hello-world 的 image 文件从仓库抓取到本地。

```
docker pull library/hello-world
```



举报

docker pull images 是抓取 image 文件， library/hello-world 是 image 文件在仓库里面的位置，其中 library 是 image 文件所在的组， hello-world 是 image 文件的名字。

抓取成功以后，就可以在本机看到这个 image 文件了。

```
docker images
```

我们可以看到如下结果：

现在，我们可以运行 hello-world 这个 image 文件

```
docker run hello-world
```

我们可以看到如下结果：

输出这段提示以后，hello world 就会停止运行，容器自动终止。有些容器不会自动终止，因为提供的是服务，比如Mysql镜像等。

是不是很容易呢？我们从上面可以看出， docker 的功能是十分强大的，除此之外，我们还可以拉去一些 Ubuntu ， Apache 等镜像，在未来的教程一提到。

Docker 提供了一套简单实用的命令来创建和更新镜像，我们可以通过网络直接下载一个已经创建好了的应用镜像，并通过 Docker RUN 命令就可以直接通过 RUN 命令运行成功后，这个运行的镜像就是一个 Docker 容器啦，容器可以理解为一个轻量级的沙箱， Docker 利用容器来运行和隔离应用被启动、停止、删除的，这并不会影响 Docker 镜像。

我们可以看看下面这幅图：

Docker 客户端是 Docker 用户与 Docker 交互的主要方式。当您使用 docker 命令行运行命令时， Docker 客户端将这些命令发送给服务器端，服务器端再执行命令。 docker 命令使用 docker API 。 Docker 客户端可以与多个服务端进行通信。

我们将剖析一下 Docker 容器是如何工作的，学习好Docker容器工作的原理，我们就可以自己去管理我们的容器了。

Docker架构

在上面的学习中，我们简单地讲解了Docker的基本架构。了解到了Docker 使用的是 C/S 结构，即**客户端/服务器**体系结构。明白了 Docker 客户端与服务器进行交互时， Docker 服务端负责构建、运行和分发 Docker 镜像。 也知道了Docker 客户端和服务端可以运行在一台机器上，可以通过 RESTful 、 络接口与远程 Docker 服务端进行通信。

我们从下图可以很直观的了解Docker的架构：

Docker 的核心组件包括：

- 1. Docker Client
- 2. Docker daemon
- 3. Docker Image
- 4. Docker Registry
- 5. Docker Container

Docker 采用的是 Client/Server 架构。客户端向服务器发送请求，服务器负责构建、运行和分发容器。客户端和服务端可以运行在同一个 Host 上，也可以通过 socket 或 REST API 与远程的服务器通信。可能很多朋友暂时不太理解一些东西，比如 REST API 是什么东西等，不过没关系，在后面的文章中会讲解清楚。

Docker Client

Docker Client ，也称 Docker 客户端。它其实就是 Docker 提供命令行界面 (CLI) 工具，是许多 Docker 用户与 Docker 进行交互的主要方式。客户端可以运行和停止应用程序，还可以远程与Docker_Host进行交互。最常用的 Docker 客户端就是 docker 命令，我们可以通过 docker 命令很方便地在 hos

👍
180

🔗

💬
28

📖

☆

📱

<

>

🔔

举报

行 docker 容器。

Docker daemon

Docker daemon 是服务器组件，以 Linux 后台服务的方式运行，是 Docker 最核心的后台进程，我们也把它称为守护进程。它负责响应请求，然后将这些请求翻译成系统调用完成容器管理操作。该进程会在后台启动一个 API Server，负责接收由 Docker Client 发送的请求，Docker daemon 内部的一个路由分发调度，由具体的函数来执行请求。

我们大致可以将其分为以下三部分：

- Docker Server
- Engine
- Job

Docker Daemon的架构如下所示：

Docker Daemon 可以认为是通过 Docker Server 模块接受 Docker Client 的请求，并在 Engine 中处理请求，然后根据请求类型，创建出指定的 Job 行。Docker Daemon 运行在 Docker host 上，负责创建、运行、监控容器，构建、存储镜像。

运行过程的作用有以下几种可能：

- 向 Docker Registry 获取镜像
- 通过 graphdriver 执行容器镜像的本地化操作
- 通过 networkdriver 执行容器网络环境的配置
- 通过 execdriver 执行容器内部运行的执行工作

由于 Docker Daemon 和 Docker Client 的启动都是通过可执行文件 docker 来完成的，因此两者的启动流程非常相似。Docker 可执行文件运行时，通过不同的命令行 flag 参数，区分两者，并最终运行两者各自相应的部分。

启动 Docker Daemon 时，一般可以使用以下命令来完成

```
1 | docker --daemon = true
2 | docker -d
3 | docker -d = true
```

再由 docker 的 main() 函数来解析以上命令的相应 flag 参数，并最终完成 Docker Daemon 的启动。

下图可以很直观地看到 Docker Daemon 的启动流程：

默认配置下，Docker daemon 只能响应来自本地 Host 的客户端请求。如果要允许远程客户端请求，需要在配置文件中打开 TCP 监听。我们可以照着配置：

1、编辑配置文件 /etc/systemd/system/multi-user.target.wants/docker.service，在环境变量 ExecStart后面添加 -H tcp://0.0.0.0，允许来自任意连接。

2、重启 Docker daemon

```
1 | systemctl daemon-reload
2 | systemctl restart docker.service
```

3、我们通过以下命令即可实现与远程服务器通信

```
docker -H 服务器IP地址 info
```

👍
180

🔖 Docker 到的请

💬
28

📖

☆

📱

<

>

🔊

举报

-H 是用来指定服务器主机， info 子命令用于查看 Docker 服务器的信息

Docker Image

Docker 镜像可以看作是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的环境变量、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。我们可将 Docker 镜像看成只读模板，通过它可

镜像有多种生成方法：

- 1. 从无到有开始创建镜像
- 2. 下载并使用别人创建好的现成的镜像
- 3. 在现有镜像上创建新的镜像

我们可以将镜像的内容和创建步骤描述在一个文本文件中，这个文件被称作 Dockerfile ，通过执行 docker build <docker-file> 命令可后续的教程中，我们会用一篇专门讨论这个问题。

Docker Registry

Docker registry 是存储 docker image 的仓库，它在 docker 生态环境中的位置如下图所示：

运行docker push、docker pull、docker search时，实际上是通过 docker daemon 与 docker registry 通信。

Docker Container

Docker 容器就是 Docker 镜像的运行实例，是真正运行项目程序、消耗系统资源、提供服务的地方。 Docker Container 提供了系统硬件环境，我们用 Docker Images 这些制作好的系统盘，再加上我们所编写好的项目代码， run 一下就可以提供服务啦。

Docker组件是如何协作运行容器

看到这里，我相信各位读者朋友们应该已经对Docker基础架构已经熟悉的差不多了，我们还记得运行的第一个容器吗？现在我们再通过hello-world这一下 Docker 各个组件是如何协作的。

容器启动过程如下：

- Docker 客户端执行 docker run 命令
- Docker daemon 发现本地没有 hello-world 镜像
- daemon 从 Docker Hub 下载镜像
- 下载完成，镜像 hello-world 被保存到本地
- Docker daemon 启动容器

具体过程可以看如下这幅演示图：

我们可以通过docker images 可以查看到 hello-world 已经下载到本地


我们可以通过docker ps 或者 docker container ls 显示正在运行的容器，我们可以看到， hello-world 在输出提示信息以后就会停止运行，容器自动们在查看的时候没有发现有容器在运行。


我们把 Docker 容器的工作流程剖析的十分清楚了，我们大体可以知道 Docker 组件协作运行容器可以分为以下几个过程：


- 1. Docker 客户端执行 docker run 命令
- 2. Docker daemon 发现本地没有我们需要的镜像
- 3. daemon 从 Docker Hub 下载镜像
- 4. 下载完成后，镜像被保存到本地
- 5. Docker daemon 启动容器


了解了这些过程以后，我们再来理解这些命令就不会觉得很突兀了，下面我来给大家讲讲 Docker 常用的一些命令操作吧。


Docker常用命令


180





28













举报

我们可以通过 `docker -h` 去查看命令的详细的帮助文档。在这里我只会讲一些平常日常比赛或者生活中我们可能会用的比较多的一些命令。

例如，我们需要拉取一个 docker 镜像，我们可以用如下命令：

```
docker pull image_name
```

image_name 为镜像的名称，而如果我们想从 Docker Hub 上去下载某个镜像，我们可以使用以下命令：

```
docker pull centos:latest
```

centos:latest 是镜像的名称，Docker daemon 发现本地没有我们需要的镜像，会自动去 Docker Hub 上去下载镜像，下载完成后，该镜像默认存到 `/var/lib/docker` 目录下。

接着我们如果想查看下主机下存在多少镜像，我们可以用如下命令：

```
docker images
```

我们要想知道当前有哪些容器在运行，我们可以用如下命令：

```
docker ps -a
```

-a 是查看当前所有的容器，包括未运行的

我们该如何去对一个容器进行启动，重启和停止呢？我们可以用如下命令：

```
1 | docker start container_name/container_id
2 | docker restart container_name/container_id
3 | docker stop container_name/container_id
```

这个时候我们如果想进入到这个容器中，我们可以使用 attach 命令：

```
docker attach container_name/container_id
```

那如果我们想运行这个容器中的镜像的话，并且调用镜像里面的 bash，我们可以使用如下命令：

```
docker run -t -i container_name/container_id /bin/bash
```

那如果这个时候，我们想删除指定镜像的话，由于 image 被某个 container 引用（拿来运行），如果不将这个引用的 container 销毁（删除），那 image 不能被删除。我们首先得先去停止这个容器：

```
1 | docker ps
2 | docker stop container_name/container_id
```

然后我们用如下命令去删除这个容器：

```
docker rm container_name/container_id
```

然后这个时候我们再去删除这个镜像：

```
docker rmi image_name
```

此时，常用的 Docker 相关的命令就讲到这里为止了，我们在后续的文章中还会反复地提到这些命令。

Dockerfile是什么

180



28













举报

前面我们已经提到了 Docker 的一些基本概念。以 CTF 选手的角度来看，我们可以去使用 Dockerfile 定义镜像，依赖镜像来运行容器，可以去模拟出洞场景。因此毫无疑问的说，Dockerfile 是镜像和容器的关键，并且 Dockerfile 还可以很轻易的去定义镜像内容，说了这么多，那么 Dockerfile 到那西呢？

Dockerfile 是自动构建 docker 镜像的配置文件，用户可以使用 Dockerfile 快速创建自定义的镜像。Dockerfile 中的命令非常类似于 linux 的 shell 脚本。我们可以通过下面这幅图来直观地感受下 Docker 镜像、容器和 Dockerfile 三者之间的关系。

我们从上图中可以看到，Dockerfile 可以自定义镜像，通过 Docker 命令去运行镜像，从而达到启动容器的目的。

Dockerfile 是由一行行命令语句组成，并且支持以 # 开头的注释行。

一般来说，我们可以将 Dockerfile 分为四个部分：

- 基础镜像(父镜像)信息指令 FROM
- 维护者信息指令 MAINTAINER
- 镜像操作指令 RUN、ENV、ADD 和 WORKDIR 等
- 容器启动指令 CMD、ENTRYPOINT 和 USER 等

下面是一段简单的Dockerfile的例子：

```
1 FROM python:2.7
2 MAINTAINER Angel_Kitty <angelkitty6698@gmail.com>
3 COPY . /app
4 WORKDIR /app
5 RUN pip install -r requirements.txt
6 EXPOSE 5000
7 ENTRYPOINT ["python"]
8 CMD ["app.py"]
```

我们可以分析一下上面这个过程：

- 1、从 Docker Hub 上 pull 下 python 2.7 的基础镜像
- 2、显示维护者的信息
- 3、copy 当前目录到容器中的 /app 目录下 复制本地主机的 <src> (Dockerfile 所在目录的相对路径)到容器里 <dest>
- 4、指定工作路径为 /app
- 5、安装依赖包
- 6、暴露 5000 端口
- 7、启动 app

这个例子是启动一个 python flask app 的 Dockerfile (flask 是 python 的一个轻量的 web 框架)，相信大家从这个例子中能够稍微理解了Dockerfile的编写过程。

Dockerfile常用的指令

根据上面的例子，我们已经差不多知道了Dockerfile的组成以及指令的编写过程，我们再来理解一下这些常用命令就会得心应手了。

由于 Dockerfile 中所有的命令都是以下格式：INSTRUCTION argument，指令 (INSTRUCTION) 不分大小写，但是推荐大写，和sql语句是不是很像，我们正式来讲解一下这些指令集吧。

FROM

FROM 是用于指定基础的 images，一般格式为 FROM <image> or FROM <image>:<tag>，所有的 Dockerfile 都用该以 FROM 开头，FROM 指明 Dockerfile 所创建的镜像文件以什么镜像为基础，FROM 以后的所有指令都会在 FROM 的基础上进行创建镜像。可以在同一个 Dockerfile 中多次用 FROM 命令用于创建多个镜像。比如我们要指定 python 2.7 的基础镜像，我们可以像如下写法一样：

```
FROM python:2.7
```

MAINTAINER

MAINTAINER 是用于指定镜像创建者和联系方式，一般格式为 MAINTAINER <name>。这里我设置成我的 ID 和邮箱：

👍
180

🔗

💬
28

📖

☆

📱

<

>

🔁

举报

COPY

COPY 是用于复制本地主机的 <src> (为 Dockerfile 所在目录的相对路径)到容器中的 <dest>。

当使用本地目录为源目录时，推荐使用 COPY 。一般格式为 COPY <src><dest> 。例如我们要拷贝当前目录到容器中的 /app 目录下，

```
COPY . /app
```

WORKDIR

WORKDIR 用于配合 RUN ， CMD ， ENTRYPOINT 命令设置当前工作路径。可以设置多次，如果是相对路径，则相对前一个 WORKDIR 格式为 WORKDIR /path/to/work/dir 。例如我们设置/app 路径，我们可以进行如下操作：

```
WORKDIR /app
```

RUN

RUN 用于容器内部执行命令。每个 RUN 命令相当于在原有的镜像基础上添加了一个改动层，原有的镜像不会有变化。一般格式为 RUN <command> 要安装 python 依赖包，我们做法如下：

```
RUN pip install -r requirements.txt
```

EXPOSE

EXPOSE 命令用来指定对外开放的端口。一般格式为 EXPOSE <port> [<port>...]

例如上面那个例子，开放5000端口：

```
EXPOSE 5000
```

ENTRYPOINT

ENTRYPOINT 可以让你的容器表现得像一个可执行程序一样。一个 Dockerfile 中只能有一个 ENTRYPOINT，如果有多个，则最后一个生效。

ENTRYPOINT 命令也有两种格式：

- ENTRYPOINT ["executable", "param1", "param2"] ：推荐使用的 exec形式
- ENTRYPOINT command param1 param2 ：shell 形式

例如下面这个，我们要将 python 镜像变成可执行的程序，我们可以这样去做：

```
ENTRYPOINT ["python"]
```

CMD

CMD 命令用于启动容器时默认执行的命令，CMD 命令可以包含可执行文件，也可以不包含可执行文件。不包含可执行文件的情况下就要用 ENTRYPOINT 命令，然后 CMD 命令的参数就会作为ENTRYPOINT的参数。

CMD 命令有三种格式：

- CMD ["executable","param1","param2"]：推荐使用的 exec 形式。
- CMD ["param1","param2"]：无可执行程序形式
- CMD command param1 param2：shell 形式。

一个 Dockerfile 中只能有一个CMD，如果有多个，则最后一个生效。而 CMD 的 shell 形式默认调用 /bin/sh -c 执行命令。

CMD 命令会被 Docker 命令行传入的参数覆盖：docker run busybox /bin/echo Hello Docker 会把 CMD 里的命令覆盖。

例如我们要启动 /app ，我们可以用如下命令实现：

🔖
180

🔗

💬
28

📖

☆

📱

<

>

可以这样
默认路

🔊

举报

CMD ["app.py"]

当然还有一些其他的命令，我们在用到的时候再去一一讲解一下。

构建Dockerfile

我们大体已经把Dockerfile的写法讲述完毕，我们可以自己动手写一个例子：

```
1 mkdir static_web
2 cd static_web
3 touch Dockerfile
4 然后 vi Dockerfile 开始编辑该文件
5 输入 i 开始编辑
6
7 以下是我们构建的Dockerfile内容
8 .....
9 FROM nginx
10 MAINTAINER Angel_Kitty <angelkitty6698@gmail.com>
11 RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
12 .....
13
14 编辑完后 按 esc 退出编辑
15 然后 :wq 写入 退出
```

我们在 Dockerfile 文件所在目录执行：

```
docker build -t angelkitty/nginx_web:v1 .
```

我们解释一下，-t 是为新镜像设置仓库和名称，其中 angelkitty 为仓库名，nginx_web 为镜像名，:v1为标签（不添加为默认 latest）

我们构建完成之后，使用 docker images 命令查看所有镜像，如果存在 REPOSITORY 为 nginx 和 TAG 是 v1 的信息，就表示构建成功。

接下来使用 docker run 命令来启动容器

```
docker run --name nginx_web -d -p 8080:80 angelkitty/nginx_web:v1
```

这条命令会用 nginx 镜像启动一个容器，命名为 nginx_web，并且映射了 8080 端口，这样我们可以用浏览器去访问这个 nginx 服务器：
http://localhost:8080/ 或者 http://本机的IP地址:8080/，页面返回信息：



这样一个简单使用 Dockerfile 构建镜像，运行容器的示例就完成了！

参考文献

- [Docker — 从入门到实践](#)
- [Docker 入门教程](#)

作者：[Angel_Kitty](#)

出处：<https://www.cnblogs.com/ECJTUACM-873284962/>

关于作者：阿里云ACE，目前主要研究方向是Web安全漏洞以及反序列化。如有问题或建议，请多多赐教！

👍 点赞 180 ☆ 收藏 ➦ 分享 ...



邓彪1022

发布了45 篇原创文章 · 获赞 199 · 访问量 22万+

👍
180

➦

💬
28

📖

☆

📱

<

>

🔊

举报

私信