docker学习

2023-11-15 14:11

目录

学习书籍链接：https://yeasy.gitbook.io/docker_practice/

学习书籍链接：https://yeasy.gitbook.io/docker_practice/

一、概念

1.1 镜像：

Docker镜像是一个特殊的文件系统，除了提供容器运行时所需要的程序、库、资源和配置等文件外，

还包含了一些运行时准备的参数（如匿名卷、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。

1.2 容器：

镜像和容器的关系，就像面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。

1.3 仓库：

镜像构建完成后，可以方便的在宿主主机上运行，但是需要在其他服务器上使用这个镜像，就需要一个集中的存储、分发镜像的服务，

Docker Register就是这样。一个Docker Register可以包含多个仓库，每个仓库可以包含多个标签，每个标签对应一个镜像。

通过**<仓库名>:<标签>**的格式来指定具体是哪个版本的镜像，如果不给**<标签>**，将以 **latest** 作为默认标签。

二、使用镜像

2.1 获取镜像

获取镜像命令格式：

```
$ docker pull [选项] [Docker Registry 地址[:端口号]/]仓库名[:标签]
```

运行镜像：

```
$ docker run -it --rm ubuntu:18.04 bash
```

- **-it**：这是两个参数，-i 交互式，-t 终端
- **--rm**：容器退出后，随之将其删除，默认情况下容器退出后不会立即删除，除非手动rm，
- **ubuntu:18.04**：指定镜像为基础启动容器，ubuntu为镜像名，18.04为tag标记
- **bash**：放在镜像后的是命令

2.2 查看镜像

```
$ docker image ls
```

2.3 删除镜像

```
$ docker image rm [选项] <镜像1> [<镜像2> ...]
```

2.4 利用commit理解镜像构建

https://yeasy.gitbook.io/docker_practice/image/commit

2.5 使用Dockerfile构建镜像

$ mkdir mynginx
$ cd mynginx
$ touch Dockerfile

然后输入下面内容到Dockerfile：

```
FROM nginx
RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
```

2.5.1 FROM：指定基础镜像

- 指定基础镜像，如果以 **scratch** 为基础镜像的话，意味着你不以任何镜像为基础，
- 不以任何系统为基础，直接将可执行文件复制进镜像做法并不罕见，对于linux下静态编译的程序来说，并不需要操作系统提供运行时支持，所需要的一切库都在可执行文件里面了，直接以 FROM scratch 会让镜像更小，常用于go语言开发的应用。

2.5.2 RUN：执行命令

- 用于执行命令行命令，
- shell格式：RUN <命令>
- exec格式：RUN ["可执行文件", "参数1", "参数2"]

Dockerfile中每一个指令都会建立一层，RUN也不例外，每一个RUN的行为就和手工建立镜像一样，执行结束后，commit这一层构成新镜像。

```
root@VM_79_85_centos:/data/mm64/sshuangzhu/docker>cat Dockerfile.one
FROM scratch

WORKDIR /go/src/github.com/go/helloworld/

COPY app .

WORKDIR /go/src/github.com/go/helloworld/

CMD ["./app"]
root@VM_79_85_centos:/data/mm64/sshuangzhu/docker>docker build -t go/helloworld:2 -f Dockerfile.one .
Sending build context to Docker daemon  2.032MB
Step 1/5 : FROM scratch
 --->
Step 2/5 : WORKDIR /go/src/github.com/go/helloworld/
 ---> Using cache
 ---> 1fd4b35078df   1
Step 3/5 : COPY app .
 ---> Using cache
 ---> 4cdfa9c96ce1   2
Step 4/5 : WORKDIR /go/src/github.com/go/helloworld/
 ---> Using cache
 ---> 4a72b99be600   3
Step 5/5 : CMD ["./app"]
 ---> Using cache
 ---> 07d45c92159c   4
Successfully built 07d45c92159c
Successfully tagged go/helloworld:2
root@VM_79_85_centos:/data/mm64/sshuangzhu/docker>
```

Union FS 是有最大层数限制的，比如 AUFS，曾经是最大不得超过 42 层，现在是不得超过 127 层。

可以使用 **'&&'和'\'** 把多个命令放在一个RUN指令中：

```
FROM debian:stretch

RUN set -x; buildDeps='gcc libc6-dev make wget' \
    && apt-get update \
    && apt-get install -y $buildDeps \
    && wget -O redis.tar.gz "http://download.redis.io/releases/redis-5.0.3.tar.gz" \
    && mkdir -p /usr/src/redis \
    && tar -xzf redis.tar.gz -C /usr/src/redis --strip-components=1 \
    && make -C /usr/src/redis \
    && make -C /usr/src/redis install \
    && rm -rf /var/lib/apt/lists/* \
    && rm redis.tar.gz \
    && rm -r /usr/src/redis \
    && apt-get purge -y --auto-remove $buildDeps
```

2.5.3 构建镜像

构建镜像命令：

```
$ docker build [选项] <上下文路径/URL/->
例如：
$ docker build -t nginx:v3 .
$ docker build -t nginx:test -f Dockerfile .
```

2.5.4 构建镜像上下文（Context）

如果注意，会看到 docker build 命令最后有一个 .。. 表示当前目录，而 Dockerfile 就在当前目录，因此不少初学者以为这个路径是在指定 Dockerfile 所在路径，这么理解其实是不准确的。如果对应上面的命令格式，你可能会发现，这是在指定 ***上下文路径****。那么什么是上下文呢?*

首先我们要理解 docker build 的工作原理。Docker 在运行时分为 Docker 引擎（也就是服务端守护进程）和客户端工具。Docker 的引擎提供了一组 REST API，被称为 Docker Remote API，而如 docker 命令这样的客户端工具，则是通过这组 API 与 Docker 引擎交互，从而完成各种功能。因此，虽然表面上我们好像是在本机执行各种 docker 功能，但实际上，一切都是使用的远程调用形式在服务端（Docker 引擎）完成。也因为这种 C/S 设计，让我们操作远程服务器的 Docker 引擎变得轻而易举。

当我们进行镜像构建的时候，并非所有定制都会通过 RUN 指令完成，经常会需要将一些本地文件复制进镜像，比如通过 COPY 指令、ADD 指令等。而 docker build 命令构建镜像，其实并非在本地构建，而是在服务端，也就是 Docker 引擎中构建的。那么在这种客户端/服务端的架构中，如何才能让服务端获得本地文件呢?

这就引入了上下文的概念。当构建的时候，用户会指定构建镜像上下文的路径，docker build 命令得知这个路径后，会将路径下的所有内容打包，然后上传给 Docker 引擎。这样 Docker 引擎收到这个上下文包后，展开就会获得构建镜像所需的一切文件。

如果在Dockerfile中这么写：COPY ./package.json /app/

这并不是要复制执行 docker build 命令所在的目录下的 package.json，也不是复制 Dockerfile 所在目录下的 package.json，而是复制 ***上下文（context）*** *目录下的 package.json。*

因此，COPY 这类指令中的源文件的路径都是相对路径。这也是初学者经常会问的为什么 COPY ../package.json /app 或者 COPY /opt/xxxx /app 无法工作的原因，因为这些路径已经超出了上下文的范围，Docker 引擎无法获得

这些位置的文件。如果真的需要那些文件，应该将它们复制到上下文目录中去。

现在就可以理解刚才的命令 docker build -t nginx:v3 . 中的这个 .，实际上是在指定上下文的目录，docker build 命令会将该目录下的内容打包交给 Docker 引擎以帮助构建镜像。

那么为什么会有人误以为 . 是指定 Dockerfile 所在目录呢？这是因为在默认情况下，如果不额外指定 Dockerfile 的话，会将上下文目录下的名为 Dockerfile 的文件作为 Dockerfile。

这只是默认行为，实际上 Dockerfile 的文件名并不要求必须为 Dockerfile，而且并不要求必须位于上下文目录中，比如可以用 -f ../Dockerfile.php 参数指定某个文件作为 Dockerfile。

2.6 Dockerfile命令详解

- COPY 复制文件
- ADD 更高级的复制文件
- CMD 容器启动命令
- ENTRYPOINT 入口点
- ENV 设置环境变量
- ARG 构建参数
- VOLUME 定义匿名卷
- EXPOSE 暴露端口
- WORKDIR 指定工作目录
- USER 指定当前用户
- HEALTHCHECK 健康检查
- ONBUILD 为他人作嫁衣

三、操作容器

3.1 启动

3.1.1 新建并启动

所需要的命令为：docker run，例如：

```
$ docker run ubuntu:18.04 /bin/echo 'Hello world'
Hello world
$
$ docker run -t -i ubuntu:18.04 /bin/bash
root@af8bae53bdd3:/#
```

-t 指docker分配一个伪终端并绑定到标准输入上，**-i** 则让容器的标准输入保持打开（也就是交互式）

容器启动失败查看：**docker logs -f -t --tail 20 【容器名】**

3.1.2 启动已终止容器

启动已终止（exited）的容器，可以使用：docker container start命令。

```
$ docker container stop 4b87f3074a7b
4b87f3074a7b
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
4b87f3074a7b        ubuntu              "/bin/bash"         2 minutes ago       Exited (0) 3
seconds ago                         focused_germain
$ docker container restart 4b87f3074a7b
```

3.2 守护态运行

可以通过添加 –d 参数，实现运行守护态。例如：

```
$ docker run -d ubuntu:18.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
77b2dc01fe0f3f1265df143181e7b9af5e05279a884f4776ee75350ea9d8017a
```

此时容器会在后台运行，并不会把输出到stdout，可以使用docker container logs查看输出。

3.3 终止

可以使用：docker container stop来终止运行中的容器。

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
4b87f3074a7b        ubuntu              "/bin/bash"         2 minutes ago       Up 2 seconds
focused_germain
$ docker container stop 4b87f3074a7b
4b87f3074a7b
$
```

3.4 进入容器

在使用–d启动的容器，会进入后台运行，可以使用docker attach命令或者docker exec命令进入容器。

3.4.1 attach命令

```
$ docker run -dit ubuntu
243c32535da7d142fb0e6df616a3c3ada0b8ab417937c853a9e1c251f499f550

$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
243c32535da7        ubuntu:latest       "/bin/bash"         18 seconds ago      Up 17
seconds                                 nostalgic_hypatia

$ docker attach 243c
root@243c32535da7:/#
```

注意：如果从这个stdin中exit，会导致容器的停止。

3.4.2 exec命令

```
$ docker run -dit ubuntu
69d137adef7a8a689cbcb059e94da5489d3cddd240ff675c640c8d96e84fe1f6

$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
69d137adef7a        ubuntu:latest       "/bin/bash"         18 seconds ago      Up 17
seconds                                 zealous_swirles

$ docker exec -i 69d1 bash
ls
bin
boot
dev
...

$ docker exec -it 69d1 bash
root@69d137adef7a:/#
```

如果从这个 stdin 中 exit，不会导致容器的停止。这就是为什么推荐大家使用 docker exec 的原因。

3.5 导出和导入

使用docker export命令导出容器：

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
7691a814370e        ubuntu:18.04        "/bin/bash"         36 hours ago        Exited (0)
21 hours ago                        test
$ docker export 7691a814370e > ubuntu.tar
```

可以使用 docker import 从容器快照文件中再导入为镜像，例如：

```
$ cat ubuntu.tar | docker import - test/ubuntu:v1.0
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL
SIZE
test/ubuntu         v1.0                9d37a6082e97        About a minute ago  171.3 MB
```

3.6 删除

可以使用 docker container rm 来删除一个处于终止状态的容器。例如：

```
$ docker container rm trusting_newton
trusting_newton
```

如果有多个终止容器想要删除，可以使用 docker container prune 清理所有已终止的容器。

四、访问仓库

4.1 Docker Hub

1. 注册：你可以在 https://hub.docker.com 免费注册一个 Docker 账号。
2. 登录：可以通过执行 docker login 命令交互式的输入用户名及密码来完成在命令行界面登录 Docker Hub。你可以通过 docker logout 退出登录。
3. 拉取镜像：你可以通过 docker search 命令来查找官方仓库中的镜像，并利用 docker pull 命令来将它下载到本地。
4. 推送镜像：用户也可以在登录后通过 docker push 命令来将自己的镜像推送到 Docker Hub。

4.2 私有仓库

4.3 私有仓库高级配置

五、数据管理

5.1 数据卷

5.1.1 创建数据卷

$ docker volume create test-volume

5.1.2 查看所有数据卷

$ docker volume ls

5.1.3 启动一个挂载数据卷的容器

$ docker run -dit --name ubuntu --mount source=test-volume,target=/usr/share/data ubuntu:20.04

-d：是守护态运行，这样容器就不会立马退出

-it：指定使用交互式运行

--mount：用于挂载数据卷，source=test-volume,target=/usr/share/data表示挂载数据卷test-volume到容器的/usr/share/data目录

5.1.4 查看数据卷的具体信息

$ docker inspect ubuntu

数据卷信息在“Mounts”Key下面：

```
{
//...
        "Mounts": [
            {
                "Type": "volume",
                "Name": "test-volume",
                "Source": "/var/lib/docker/volumes/test-volume/_data",
                "Destination": "/usr/share/data",
                "Driver": "local",
                "Mode": "z",
                "RW": true,
                "Propagation": ""
            }
        ],
// ...
}
```

这里可以看到，数据卷这里的含义：

- Name=test-volume，表示数据卷名字为test-volume
- Source=/var/lib/docker/volumes/test-volume/_data，这个是数据卷在主机的本地文件路径
- Destination=/usr/share/data，这个是容器内部的路径

5.1.5 删除数据卷

$ docker volume rm test-volume

数据卷是用来持久化数据的，它的声明周期独立于容器，docker不会在删除容器后自动删除数据卷，如果需要删除容器同时移除数据卷，可以在删除容器的时候使用：docker rm –v命令。

清理数据卷：*$ docker volume prune*

5.2 挂载主机目录

5.2.1 挂载一个主机目录作为数据卷

```
$ docker run -dit --name ubuntu --mount type=bind,source=/data2,target=/usr/share/data ubuntu:20.04
ad6a52cc58e3b270f1eecab61c9db4d102799b1d09ce01e0f1b5d95779bd4846
$ docker exec -it ad6a52cc58e3 bash
# cd /usr/share/data/
# echo -e "Hello" > hello.txt
# exit
$ cat /data2/hello.txt
Hello
```

5.2.2 查看数据卷具体信息

$ docker inspect ubuntu

```
        "Mounts": [
            {
                "Type": "bind",
                "Source": "/data2",
                "Destination": "/usr/share/data",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
```

5.2.3 挂载一个本地主机文件作为数据卷

$ docker run --rm -it --mount type=bind,source=$HOME/.bash_history,target=/root/.bash_history ubuntu:20.04 bash

这样就可以把$HOME/.bash_history文件挂载到容器的/root/.bash_history文件了。

六、使用网络

6.1 外部访问容器

$ docker run -d -p 80:80 nginx

docker run的时候可以通过-p或-P参数指定端口映射。使用-p时，docker会随机映射一个端口到内部开放网络端口。

同样，可以使用docker logs 【containerID】来查看访问记录输出日志。

-p支持的格式有：ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort。

docker port 【containerID】 【port】可以查看绑定的地址。

6.2 容器互联

6.2.1 新建网络

$ docker network create -d bridge test-net

-d参数指定Docker网络类型，有bridge，overlay。其中overlay网络类型属于Swarm mode，暂时忽略。

云开发机执行可能会遇到错误：“Error response from daemon: could not find an available, non-overlapping IPv4 address pool among the defaults to assign to the network”

原因1：创建的网络数大于docker最大允许的数量（网上了解的是31个限制），这种需要删除掉不用的即可。

原因2：没有划分网络网段使用，可以通过下面方式解决：

解决方式：

```
// 删除一个网段，可选172和198
$ ip route del 172.16.0.0/12
// tlinux需要改路由，下面这个文件不删除重启又会占用网段
$ vim /etc/sysconfig/network-scripts/setdefaultgw-tlinux
// 删除下面一行
172.16.0.0/12
```

查看网络：***$ docker network ls***

6.2.2 连接容器

创建容器1：

$ docker run -dit --rm --name busybox1 --network my-net busybox sh

创建容器2：

$ docker run -dit --rm --name busybox2 --network my-net busybox sh

连接进入容器1，然后ping容器2查看网络是否联通：

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
efb276b11935        busybox             "sh"                3 minutes ago       Up 3 minutes
busybox2
71b059578c94        busybox             "sh"                3 minutes ago       Up 3 minutes
busybox1
$ docker exec -it 71b059578c94 sh
/ # ping busybox2
PING busybox2 (172.18.0.5): 56 data bytes
64 bytes from 172.18.0.5: seq=0 ttl=64 time=0.074 ms
64 bytes from 172.18.0.5: seq=1 ttl=64 time=0.061 ms
64 bytes from 172.18.0.5: seq=2 ttl=64 time=0.064 ms
^C
--- busybox2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.061/0.066/0.074 ms
/ #
```

6.3 配置DNS

文件挂载方式：

如何自定义配置容器的主机名和 DNS 呢？秘诀就是 Docker 利用虚拟文件来挂载容器的 3 个相关配置文件。在容器中使用 mount 命令可以看到挂载信息。

这种机制可以让宿主主机 DNS 信息发生更新后，所有 Docker 容器的 DNS 配置通过 /etc/resolv.conf 文件立刻得到更新。

配置全部容器的 DNS ， 也可以在 /etc/docker/daemon.json 文件中增加以下内容来设置。

```
{
   "dns":[
      "114.114.114.114",
      "8.8.8.8"
   ]
}
```

这样每次启动的容器 DNS 自动配置为 114.114.114.114 和 8.8.8.8。使用以下命令来证明其已经生效。

$ docker run -it --rm ubuntu:18.04 cat etc/resolv.conf

手动指定方式：

在docker run的时候，添加下面的参数：

-h HOSTNAME 或者 --hostname=HOSTNAME 设定容器的主机名，它会被写到容器内的 /etc/hostname 和 /etc/hosts。但它在容器外部看不到，既不会在 docker container ls 中显示，也不会在其他的容器的 /etc/hosts 看到。

--dns=IP_ADDRESS 添加 DNS 服务器到容器的 /etc/resolv.conf 中，让容器用这个服务器来解析所有不在 /etc/hosts 中的主机名。

--dns-search=DOMAIN 设定容器的搜索域，当设定搜索域为 .example.com 时，在搜索一个名为 host 的主机时，DNS 不仅搜索 host，还会搜索 host.example.com。

注意：如果在容器启动时没有指定最后两个参数，Docker 会默认用主机上的 /etc/resolv.conf 来配置容器。

七、底层实现

7.1 基本架构

Docker 采用了 C/S 架构，包括客户端和服务端。Docker 守护进程 （Daemon）作为服务端接受来自客户端的请求，并处理这些请求（创建、运行、分发容器）。

7.2 命名空间

Linux Namespace是Linux提供的一种内核级别环境隔离的方法。不知道你是否还记得很早以前的Unix有一个叫chroot的系统调用（通过修改根目录把用户jail到一个特定目录下），

chroot提供了一种简单的隔离模式：chroot内部的文件系统无法访问外部的内容。Linux Namespace在此基础上，提供了对UTS、IPC、mount、PID、network、User等的隔离机制。

查看进程命名空间：ls -l /proc/[pid]/ns，简单操作：ls -l /proc/$$/ns

扩展阅读：

DOCKER基础技术：LINUX NAMESPACE（上）： https://coolshell.cn/articles/17010.html

DOCKER基础技术：LINUX NAMESPACE（下）：https://coolshell.cn/articles/17029.html

关于centos7的命名空间启用：http://bingerambo.com/posts/2020/12/centos-7-%E5%90%AF%E7%94%A8-user-namespaces%E7%94%A8%E6%88%B7%E5%91%BD%E5%90%8D%E7%A9%BA%E9%97%B4/

7.3 控制组

控制组（cgroups）是 Linux 内核的一个特性，主要用来对共享资源进行隔离、限制、审计等。只有能控制分配到容器的资源，才能避免当多个容器同时运行时的对系统资源的竞争。

控制组技术最早是由 Google 的程序员在 2006 年提出，Linux 内核自 2.6.24 开始支持。

控制组可以提供对容器的内存、CPU、磁盘 IO 等资源的限制和审计管理。

Linux的CGROUP可以直接通过mount挂载，可以通过 mount -t cgroup 或者 lssubsys -m 查看：

```
root@VM_79_85_centos:/data/mm64/sshuangzhu/docker>lssubsys -m
cpuset /sys/fs/cgroup/cpuset
cpu,cpuacct /sys/fs/cgroup/cpu,cpuacct
memory /sys/fs/cgroup/memory
devices /sys/fs/cgroup/devices
freezer /sys/fs/cgroup/freezer
net_cls /sys/fs/cgroup/net_cls
blkio /sys/fs/cgroup/blkio
perf_event /sys/fs/cgroup/perf_event
hugetlb /sys/fs/cgroup/hugetlb
```

pids /sys/fs/cgroup/pids
oom /sys/fs/cgroup/oom

扩展阅读：

DOCKER基础技术：LINUX CGROUP： https://coolshell.cn/articles/17049.html

Linux资源管理之cgroups简介： https://tech.meituan.com/2015/03/31/cgroups.html

REDHAT资源管理：https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/6/html-single/resource_management_guide/index#idm140538582929664

7.4 联合文件系统

联合文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。

联合文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

Docker 中使用的 AUFS（Advanced Multi-Layered Unification Filesystem）就是一种联合文件系统。

扩展阅读：

DOCKER基础技术：AUFS：https://coolshell.cn/articles/17061.html

7.5 容器格式

最初，Docker 采用了 LXC 中的容器格式。从 0.7 版本以后开始去除 LXC，转而使用自行开发的 libcontainer，从 1.11 开始，则进一步演进为使用 runC 和 containerd。

7.6 网络

Docker 的网络实现其实就是利用了 Linux 上的 **网络命名空间**和 **虚拟网络设备**（特别是 veth pair）。

7.6.1 基本原理

首先，要实现网络通信，机器需要至少一个网络接口（物理接口或虚拟接口）来收发数据包；此外，如果不同子网之间要进行通信，需要路由机制。

Docker 中的网络接口默认都是虚拟的接口。

Docker 容器网络就是在本地主机和容器内分别创建一个虚拟接口，让它们彼此连通。

7.6.2 创建网络参数

Docker 创建一个容器的时候，会执行如下操作：

- *创建一对虚拟接口，分别放到本地主机和新容器中；*
- *本地主机一端桥接到默认的 docker0 或指定网桥上，并具有一个唯一的名字，如 veth65f9；*
- *容器一端放到新容器中，并修改名字作为 eth0，这个接口只在容器的命名空间可见；*
- *从网桥可用地址段中获取一个空闲地址分配给容器的 eth0，并配置默认路由到桥接网卡 veth65f9。*