Docker

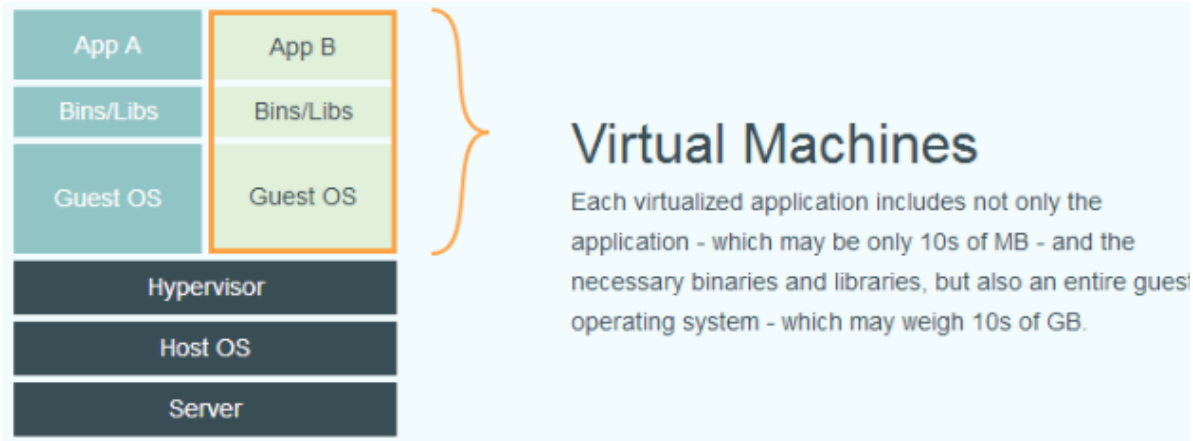
一、Docker简介

1.1 什么是Docker

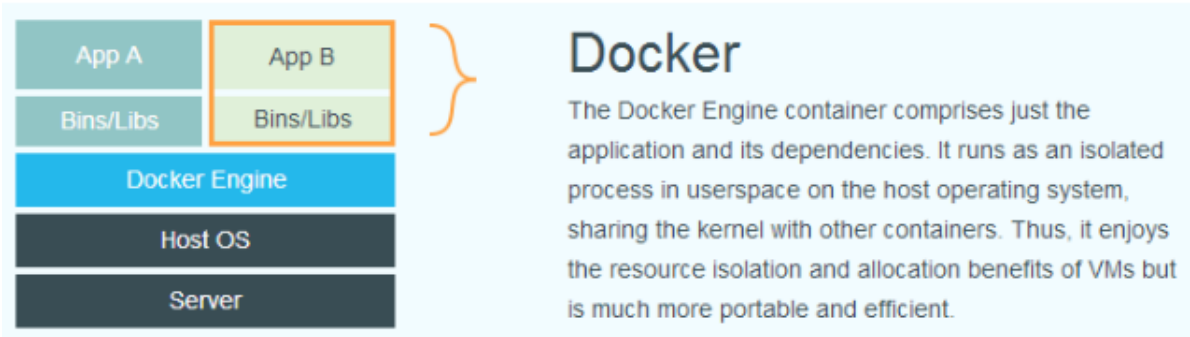
Dockcer是一个开源的应用容器引擎，Docker可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的Linux机器上。

Docker 利用 Linux 核心中的资源分脱机制，例如 cgroups，以及 Linux 核心名字空间（name space），来创建独立的软件容器（containers），属于操作系统层面的虚拟化技术。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。Docker 在容器的基础上进行了进一步的封装，从文件系统、网络互联到进程隔离等等，极大的简化了容器的创建和维护，使得其比虚拟机技术更为轻便、快捷。

1.2 Docker和虚拟机的区别与特点



对于虚拟机来说，需要模拟整台机器包括硬件，每台虚拟机都要有自己的操作系统。



容器技术和我们的宿主机共享硬件资源和操作系统，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。

Docker具有以下几个特点：

> 1、更快的启动速度

> 2、更高效的资源利用率
>
> 3、更高的系统支持量
>
> 4、持续交付与部署
>
> 5、更轻松的迁移

二、Docker的基本概念

2.1 核心概念：镜像、容器与仓库

Docker主要包含三个基本概念，分别是镜像、容器和仓库。

- **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的程序、库、资源、配置外，还包含了一些为运行时准备的一些配置参数。
- **容器**：容器的实质是进程，容器进行运行于属于自己的独立的命名空间。容易可以被创建、启动、停止、删除和暂停。镜像于容器之间的关系，可以类比面向对象中的类和实例
- **仓库**：镜像构建完成后，我们需要一个集中的存储、发布镜像的服务，Docker Registry就是这样的服务。一个 Docker Registry 中可以包含多个仓库；每个仓库可以包含多个标签；每个标签对应一个镜像，其中标签可以理解为镜像的版本号。

2.2 Docker 三剑客

docker-compose：Docker镜像在创建之后，往往需要自己手动pull来获取镜像，然后执行run命令来运行。当服务需要用到多种容器，容器之间又产生各种依赖和连接，可以使用docker-compose技术将所有的容器的部署方法、文件映射、容器连接等等一系列配置写在一个配置文件里，最后只需要执行docker-compose up脚本，就可以一个个自动部署他们，

docker-machine：Docker 技术是基于 Linux 内核的 cgroup 技术实现的，那么问题来了，在非 Linux 平台上是否就不能使用 docker 技术了呢？答案是可以的，不过显然需要借助虚拟机去模拟出 Linux 环境来。

docker-swarm：swarm 是基于 docker 平台实现的集群技术，他可以通过几条简单的指令快速的创建一个 docker 集群，

三、Docker的安装与使用

3.1 Docker安装、运行与加速

在centos7下的安装

- 1 安装

```
yum install docker
```

- 2 启动docker并设置成开机启动

```
systemctl start docker

systemctl enable docker
```

- 3 查看docker是否安装成功

```
docker info
```

```
[root@localhost ~]# docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.13.1
Storage Driver: overlay2
 Backing Filesystem: xfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: journald
Cgroup Driver: systemd
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: inactive
Runtimes: docker-runc runc
Default Runtime: docker-runc
Init Binary: /usr/libexec/docker/docker-init-current
containerd version:  (expected: aa8187dbd3b7ad67d8e5e3a15115d3eef43a7ed1)
runc version: 8891bca22c049cd2dcf13ba2438c0bac8d7f3343 (expected: 9df8b306d
01f59d3a8029be411de015b7304dd8f)
init version: fec3683b971d9c3ef73f284f176672c44b448662 (expected: 949e6facb
77383876aeff8a6944dde66b3089574)
Security Options:
 seccomp
  WARNING: You're not using the default seccomp profile
  Profile: /etc/docker/seccomp.json
 selinux
Kernel Version: 3.10.0-1160.59.1.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 3
CPUs: 2
Total Memory: 3.84 GiB
Name: localhost.localdomain
ID: 2CBB:35GT:YWNS:SS7U:Q62E:27M5:5JXT:AT3G:6O45:RK3G:VAA5:3O7B
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
 127.0.0.0/8
Registry Mirrors:
 https://13chw39k.mirror.aliyuncs.com
Live Restore Enabled: false
Registries: docker.io (secure)
[root@localhost ~]#
```

由于某些原因，国内从 Docker Hub 上拉取内容会非常缓慢，这个时候就可以配置一个镜像加速器环境。详情说明可以移步Docker 中国官方镜像加速，我们也可以配置一个阿里云镜像Docker配置阿里云镜像

3.2 Hello World

大多数编程语言以及一些软件的第一个示例都是Hello World，Docker也不例外。接下来我们运行一个 docker run hello-world 验证一下吧。

```
docker pull hello-world
```

```
[root@localhost ~]# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

3.3 运行一个MySQL

```
docker pull mysql:5.7

docker images

docker run -p 3306:3306 --name zdwMySQL -e MYSQL_ROOT_PASSWORD=123456 -d
docker.io/mysql:5.7
```

参数说明

docker run就是运行容器的命令，简要说明常见的参数

> -i 交互式操作
>
> -t 终端
>
> -it 当执行一些命令并查看返回结果，我们需要交互式终端
>
> -p指定端口映射 格式为：主机port：容器port

> -e 设置环境变量
>
> -d 后台运行
>
> /bin/bash 交互式

查看所有已经运行的容器

```
docker ps
```

进入我们的MySQL容器内部

```
docker attach

docker exec：推荐大家使用 docker exec 命令，因为此命令会退出容器终端，但不会导致容器的停止。
```

```
Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                               NAMES
2b64722069ed        docker.io/mysql:5.7   "docker-entrypoint..."   3 minutes ago
    Up 3 minutes        0.0.0.0:3306->3306/tcp, 33060/tcp   zdwMySQL
[root@localhost ~]# docker exec -it 2b64722069ed
"docker exec" requires at least 2 argument(s).
See 'docker exec --help'.

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container
[root@localhost ~]# docker exec -it 2b64722069ed  /bin/bash
root@2b64722069ed:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.36 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
```

四、镜像使用与发布

4.1 镜像拉取

Docker Hub上有大量优秀的镜像，如何获取一个镜像呢，和git类似，docker也使用pull命令

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

我们拉取一个protainer试试 *(Portainer是Docker的图形化管理工具,提供状态显示面板、应用模板快速部署、容器镜像网络数据卷的基本操作)*

```
docker pull portainer
```

4.2 Docker镜像一些常用操作

- docker images -列出本地已下载的镜像
- docker rmi [选项]<镜像1> [<镜像2>] -删除镜像
- docker logs <id /container_name> -查看容器日志
- docker search images_name - 从Docker Hub检索镜像
- docker history -显示镜像历史
- docker push new_image_name -发布镜像
- docker ps -查看当前所有运行容器 docker ps -a 查看正在运行的容器

4.3 docker commit命令

该命令的主要功能是把当前容器提交打包为镜像

当我们运行一个容器的时候，我们做的任何文件修改都会被记录于容器存储层。而Docker提供了

一个docker commit命令，可以将容器的存储层保存下来成为镜像。换句话说，就是在原有镜像的基础上，再叠加上容器的存储层，并构成新的镜像。

docker commit语法为

```
docker commit [选项] <容器ID或容器名>  [<仓库名> [:<标签>]]
```

我们可以用以下命令将容器保存为镜像

```
docker commit  \
--author 'dawei zhang' \
--message  "Hello Docker! 我修改了nginx默认欢迎页" \
edf25f309293   \
nginx:v2
```

```
[root@localhost share]# docker images
REPOSITORY            TAG       IMAGE ID       CREATED         SIZE
nginx                 v2        9c0185ce74fd   2 hours ago     141MB
nginx                 latest    605c77e624dd   3 months ago    141MB
mysql                 latest    3218b38490ce   3 months ago    516MB
portainer/portainer   latest    580c0e4e98b0   13 months ago   79.1MB
[root@localhost share]#
```

新的镜像制作好后，我们可以来运行这个镜像

```
docker run --name  myNginx -d -p 81:80 nginx:v2
```

查看我们修改后的nginx首页，发现修改成功

```
#进入我们的nginx容器内部
docker exec -it a8fdb09ea54e /bin/bash

###执行修改命令
echo '<h1>Hello Docker! 我修改了nginx默认欢迎页！</h1>' >
/usr/share/nginx/html/index.html
```

```
1 centos7  ×  +
[root@localhost docker_compose_sprinbootservice]# curl 127.0.0.1:81
<h1>Hello Docker! 我修改了nginx默认欢迎页！</h1>
[root@localhost docker_compose_sprinbootservice]#
```

慎用docker commit

使用docker commit命令虽然可以比较直观的理解镜像分成存储的概念，但是实际环境中并不会这样使用

4.4容器卷和主机互通互联

卷的目的就是**数据的持久化**,完全独立于容器的生成周期，因此Docker不会在容器删除时删除其挂载的数据卷。

- 有点类似Redis里面的rdb和aof文件
- 将docker容器内的数据保存进宿主机的磁盘中
- 运行一个带有容器卷存储功能的容器实例
    - docker run -it --privileged=true **-v** /宿主机绝对路径目录:/容器内目录　　镜像名

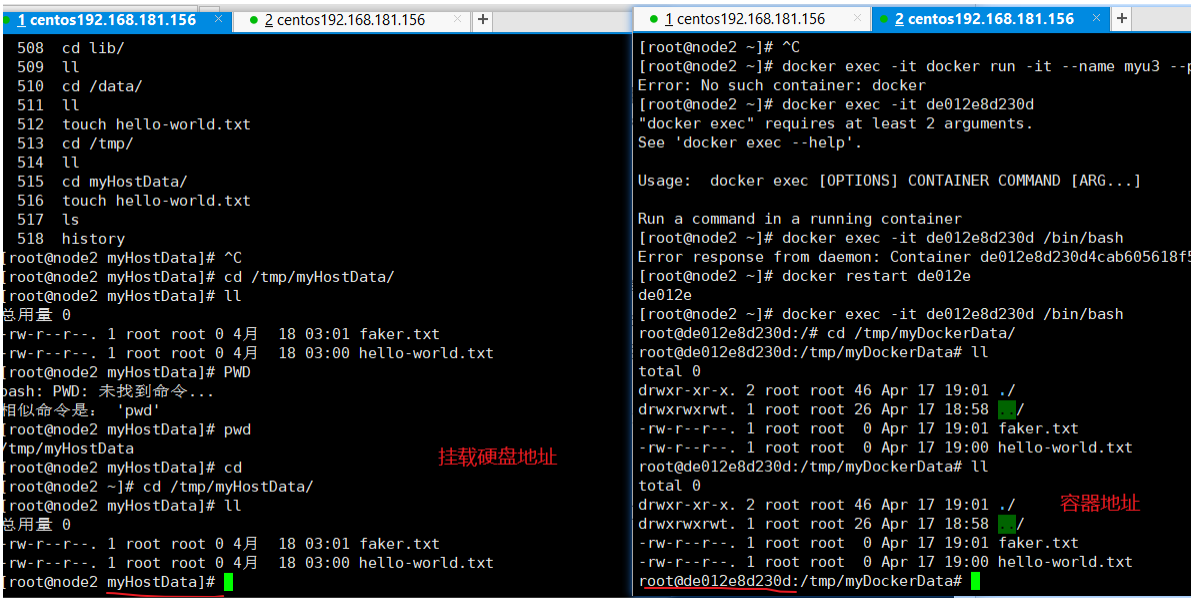
命令

```
docker run -it --name myu3 --privileged=true -v
/tmp/myHostData:/tmp/myDockerData ubuntu /bin/bash
```

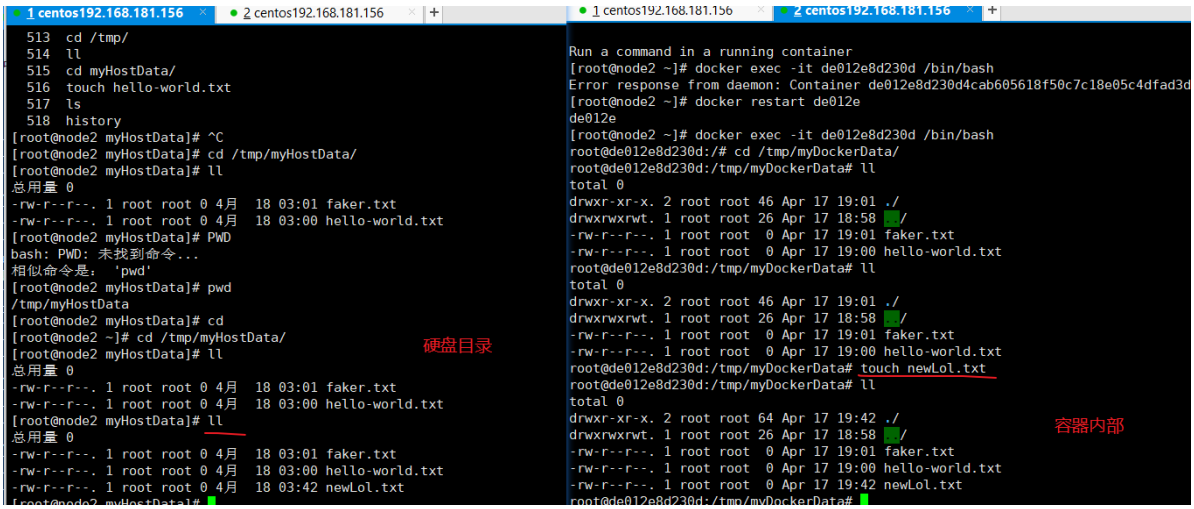
进入我们的容器内部

```
docker exec -it  myu3 /bin/bash

cd  /tmp/myDockerData
```



此时我们的硬盘目录和容器内部目录下面分别有2个相同的文件，faker.txt及hello-world.txt，我们在容器内部添加一个新的文件，文件会同步到我们的硬盘目录下面去



容器卷和主机互通互联成功

4.5 使用Dockerfile定制镜像

Dockerfile是一个文本文件，其包含了一条条的指令，每一条指令构建一层，因此每一条指令的内容就是描述该层如何构建。

以之前的nginx镜像为例，这次我们使用Dockerfile来定制

在一个空白目录中，建立一个文本文件，并命名Dockerfile

```
mkdir  mynginx
cd   mynginx
touch Dockerfile
```

其内容为：

```
FROM nginx
RUN echo '<h2> Hello ,My friend !</h2' /usr/share/nginx/html/index.html
```

FROM指定基础镜像

定制镜像，意为以一个镜像为基础，在其身上进行定制。就像我们之前运行了一个nginx镜像的容器，再进行修改一样，基础镜像是必须指定的。而FROM就是指定**基础镜像**。一个Dockerfile中FROM是必备的指令，并且必须是第一条指令。

RUN执行命令

RUN指令是用来执行命令行指令的。其格式主要有两种：

- shell格式： RUN<命令> ,就像直接在命令行中输入的命令一样。刚才写的Dockerfile中的RUN指令就是这种格式

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

- exec格式： RUN["可执行文件"，"参数1"，"参数2"]

构建镜像

在Dockerfile文件所在目录执行

docker build it nginx:v3 .

```
[root@localhost mynginx]# docker build -t nginx:v3 .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM nginx
 ---> 605c77e624dd
Step 2/2 : RUN echo '<h2> Hello ,My friend !</h2'
/usr/share/nginx/html/index.html
 ---> Running in 92d1daff537a
<h2> Hello ,My friend !</h2 /usr/share/nginx/html/index.html
Removing intermediate container 92d1daff537a
 ---> 749c0aa8cc4b
Successfully built 749c0aa8cc4b
Successfully tagged nginx:v3
```

从命令的输出结果中，我们可以清晰的看到镜像的构建过程。在Step2中，RUN指令启动了一个容器92d1daff537a，并最后提交了这一层749c0aa8cc4b，随后删除了用到的这个容器92d1daff537a

这里使用了docker build命令进行镜像构建，其格式为：

```
docker build  [选项] <上下文路径/URL/>
```

镜像构建上下文(Context)

docker build命令最后有一个 . 。 . 表示当前目录

构建的时候，用户会指定构建对象上下文路径，docker build 命名得知这个路径后，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，展开就会获得构建对象所需的一切文件。

```
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。

4.6 Dockerfile指令

COPY复制文件

- COPY [--chown=:] <源路径>... <目标路径>

ADD更高级的复制文件

ADD指令和COPY的性质基本一致。但是在COPY基础上增加了一些功能。如果<源文件>为一个tar压缩文件的话，压缩格式为gzip、bzip2的情况下，ADD指令将会自动解压缩这个压缩文件到<目标路径>去

```
FROM scratch
ADD ubuntu-xenial-core-cloudimg-amd64-root.tar.gz /
...
```

CMD容器启动命令

CMD命令和RUN很相似，也是两种格式

- shell 格式，CMD<命令>
- exec格式，CMD ["可执行文件", "参数1", "参数2"...]

ENTRYPOINT入口点

ENTRYPOINT的格式和RUN指令格式一样。分为exec和shell格式。

ENTRYPOINT的目的和CMD一样，都是在指定容器启动程序及参数。ENTRYPOINT在运行时也可以替代，不过比CMD要复杂。

ENV设置环境变量

格式有两种

- ENV
- ENV = =

```
ENV VERSION=1.0 DEBUG=on \
    NAME="Happy Feet"
```

VOLUME定义匿名卷

- VOLUME ["<路径1>", "<路径2>"...]

```
VOLUME /data
```

这里的/data目录就会在容器运行时自动挂载为匿名卷，任何向/data写入的信息都不会记录进容器存储层，从而保证了容器存储层的无状态化。当然

```
$ docker run -d -v mydata:/data xxxx
```

在这行命令中，就使用了 `mydata` 这个命名卷挂载到了 `/data` 这个位置，替代了 `Dockerfile` 中定义的匿名卷的挂载配置

4.6使用Dockerfile构建SpringBoot应用镜像

1 编写Dockerfile文件

```
FROM java:8-jre-alpine
MAINTAINER xx"zdwbmw@163.com"
EXPOSE 8080
RUN mkdir -p /usr/local/demo_app/config
VOLUME /tmp
COPY . /usr/local/demo_app
ENV JAVA_OPTS=""
WORKDIR /usr/local/demo_app
ENTRYPOINT java ${JAVA_OPTS} -jar /usr/local/demo_app/mall-tiny.jar
```

2 使用maven打包，将应用jar包及Dockerfile上传到Linux服务器

3 在Linux上构建docker镜像

```
docker build -t mall-tiny:1.0 .
```

```
drwxr-xr-x. 2 root root    6 4月   2 10:28 桌面
[root@localhost ~]# cd /data/
[root@localhost data]# ll
总用量 0
drwxr-xr-x. 2 root root 45 4月  15 13:32 mall-tiny
drwxr-xr-x. 2 root root 24 4月  15 12:21 mynginx
[root@localhost data]# cd ml
-bash: cd: ml: 没有那个文件或目录
[root@localhost data]# cd mall-tiny/
[root@localhost mall-tiny]# ll
总用量 59160
-rw-r--r--. 1 root root      257 4月  15 13:32 Dockerfile
-rw-r--r--. 1 root root 60572689 4月  15 13:23 mall-tiny.jar
[root@localhost mall-tiny]# docker images
REPOSITORY           TAG            IMAGE ID       CREATED          SIZE
mall-tiny            1              106bf7aec2e1   41 minutes ago   168MB
nginx                v3             749c0aa8cc4b   2 hours ago      141MB
nginx                v2             9c0185ce74fd   5 hours ago      141MB
nginx                latest         605c77e624dd   3 months ago     141MB
mysql                latest         3218b38490ce   3 months ago     516MB
portainer/portainer  latest         580c0e4e98b0   13 months ago    79.1MB
java                 8-jre-alpine   fdc893b19a14   5 years ago      108MB
[root@localhost mall-tiny]#
```

启动我们的镜像

```
docker run \
-p 7789:8080 \
--name mall-tiny \
--rm \
-e JAVA_OPTS='-server -Xmx1024m -Xms1024m' \
-d \
mall-tiny:1
```

五、Docker Compose

Docker Compose是Docker官方编排,负责快速的部署分布式应用。

Compose项目是Dokcer官方的开源项目，负责对Docker容器快速编排。其定位是定义和运行多个Docker容器的应用。在日常工作中，经常会碰到需要多个容器相互配合来完成某项任务的情况。例如要实现一个 Web 项目，除了 Web 服务容器本身，往往还需要再加上后端的数据库服务容器，甚至还包括负载均衡容器等。

Compose允许用户通过一个单独的docker-compose.yml模板文件来定义一组相关联的应用容器为一个项目。

Compose中有两个重要的概念：

- 服务(service)：一个应用的容器，实际上可以包括若干个运行相同镜像的容器实例
- 项目(project)：由一组关联的应用容器组成的一个完整业务单元，在docker-compose.yml文件中定义。

Compose的默认管理对象是项目，通过子命令对项目中的一组容器进行便捷地生命周期管理。

5.1安装

Compose支持Linux、macOS、Windows三大平台

pip安装这种方式是将 Compose 当作一个 Python 应用来从 pip 源中安装。

```
1、安装pip

yum -y install epel-release

yum install python3-pip

pip3 install --upgrade pip

2、安装docker-compose
```

```
pip3 install docker-compose

3、查看版本

docker-compose version
```

可以看到类似如下输出，说明安装成功

5.2使用

术语

- 服务(service) :一个应用容器，实际上可以运行多个相同的镜像实例
- 项目(project)：由一组关联的应用容器组成的一个完整业务单元

可见，一个项目可以由多个服务(容器)关联而成，Compose面向项目进行管理

场景

最常见的是web网站，该项目包含web应用和缓存。下面我们用 `Python` 来建立一个能够记录页面访问次数的 web 网站。

web应用

新建文件夹，在该目录中编写 `app.py` 文件

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! 该页面已被访问 {} 次。\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

Dockerfile

编写Dockerfile文件，内容为

```
FROM python:3.6-alpine
ADD . /code
WORKDIR /code
RUN pip install redis flask
CMD ["python", "app.py"]
```

docker-compose.yml

编写docker-compose.yml文件，这个是Compose使用的主模板文件

5.3使用docker-compose编排SpringBoot服务

我们使用docker-compose编排一个springboot的服务，服务配置文件见下

application.yml

```
spring:
  datasource:
    username: root
    password: woaini
    url: jdbc:mysql://uni_mysql:3306/swls?useUnicode=true&characterEncoding=UTF-
8
    driver-class-name: com.mysql.cj.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource
  #thymeleaf start
  thymeleaf:
    mode: HTML5
    encoding: UTF-8
    content-type: text/html
    #开发时关闭缓存,不然没法看到实时页面
    cache: false
  redis:
    host: uni_redis # redis 的主机IP名
    port: 6379
    username: root
mybatis:
#  config-location: classpath:mybatis/mybatis-config.xml
  mapper-locations: classpath:mybatis/mapper/*.xml
```

- 1 准备工作，使用maven打包，复制我们的jar包到linux下的一个目录中

```
-rw-r--r--. 1 root root     1211 4月  18 02:01 docker-compose.yml
-rw-r--r--. 1 root root      441 4月  18 01:07 Dockerfile
-rw-r--r--. 1 root root 54332200 4月  18 02:01 springboot-web-login-simple-0.1.jar
[root@node2 docker_compose_test]#
```

- 2 编写Dockerfile文件

```
# 基础镜像使用java
FROM java:8
# 作者
MAINTAINER uni
# VOLUME 指定临时文件目录为/tmp，在主机/var/lib/docker目录下创建了一个临时文件并链接到容器的/tmp
VOLUME /tmp
# 将jar包添加到容器中并更名为uni.jar
ADD springboot-web-login-simple-0.1.jar uni.jar
# 运行jar包
RUN bash -c 'touch /uni.jar.jar'
ENTRYPOINT ["java","-jar","/uni.jar"]
#暴露8080作为web访问端口
EXPOSE 8080
```

- 3 编写docker-compose文件

```
version: "3"

services:
  microService:
    image: uni:0.1
    container_name: simple_springboot_login
    ports:
      - "8081:8080"
    volumes:
      - /app/microService:/data
    networks:
      - uni_net
    depends_on:
      - uni_redis
      - uni_mysql

  uni_redis:
    image: redis:6.0.16
    container_name: uni_redis
    ports:
      - "6379:6379"
    volumes:
      - /opt/module/docker/uni/redis/redis.conf:/etc/redis/redis.conf
      - /opt/module/docker/uni/redis/data:/data
    networks:
      - uni_net
    command: redis-server /etc/redis/redis.conf

  uni_mysql:
    image: mysql:8
    restart: always
    container_name: uni_mysql
    environment:
      MYSQL_ROOT_PASSWORD: 'woaini'
      MYSQL_ALLOW_EMPTY_PASSWORD: 'no'
    ports:
      - "3306:3306"
    volumes:
```

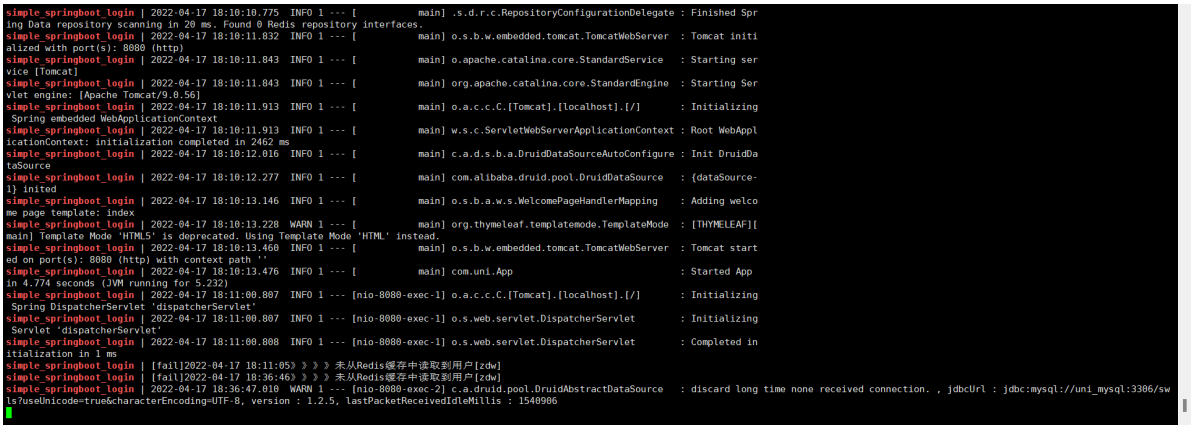
```
      - /opt/module/docker/uni/mysql/db:/var/lib/mysql
      - /opt/module/docker/uni/mysql/conf.d:/etc/mysql/conf.d
      # 配置mysql容器的初始化sql脚本
      - /opt/module/docker/uni/mysql/init:/docker-entrypoint-initdb.d
    networks:
      - uni_net
    command: --default-authentication-plugin=mysql_native_password #解决外部无法访问
networks:
  uni_net:
```

- 4 依次运行下面命令

```
docker build -t uni:0.1 .

docker-compose up -d

docker-compose logs -f | grep simple_springboot_login
```



至此，docker-compose的功能测试完毕。

其他：

参考：

- Docker 学习新手笔记：从入门到放弃
- Docker-从入门到实践
- 使用docker-compose编排服务