Docker

- Docker基础知识，理解Docker镜像、容器、仓库的概念
- Docker的安装与启动
- Docker镜像与容器的相关命令
- Docker安装软件
- Docker迁移与备份相关命令
- 编写Dockerfile创建容器脚本
- 搭建Docker私有仓库

一、Docker 组件

1. Docker 镜像

- 定义

  镜像是构建 Docker 的基石，用户基于镜像来运行自己的容器。镜像是Docker生命周期中的"构建"部分；也可以理解为镜像是容器的"源代码"；镜像体积很小，非常"便携"，易于分享、存储和更新

- 查看Docker 镜像

```
docker images
```

2. Docker 容器

- 定义

  容器是镜像的一个实例，可以理解为镜像是我们JAVA中的类，容器是类的一个对象

- 启动容器

```
# 第一次启动容器-镜像启动
docker run -id --name=<自定义容器名称> <镜像名称>
# 启动已有容器
docker start <容器名称或容器ID>
```

3. Registry 仓库

- 定义

  Docker 用 Registry 来保存用户构建的镜像，Registry 分为共有和私有，简单的说就是镜像的仓库

二、 Ubuntu - Docker 安装与启动、停止，重启

- 安装Docker

```
# Docker 的旧版本被称为 docker，docker.io 或 docker-engine
sudo apt-get remove docker docker-engine docker.io containerd runc
# 设置Docker仓库
# 更新apt包索引
sudo apt-get update
# 安装apt依赖包，用于通过HTTPS来获取仓库
```

```
sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
# 添加Docker的官方GPG秘钥
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
# 设置稳定版仓库
sudo add-apt-repository \
 "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
# 安装Docker Engine-Community
# 更新 apt 包索引
sudo apt-get update
# 安装最新版的Docker Engine-Community 和 containerd
sudo apt-get install docker-ce docker-ce-cli containerd.io
# 或者安装指定版本的
# 查看所有的Docker版本
# apt-cache madison docker-ce
# 指定Docker版本安装
# sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=
<VERSION_STRING> containerd.io
# 查看Docker是否安装成功
sudo docker run hello-world
# 查看Docker版本
docker -v/version/--version
```

设置Docker ustc的镜像

```
vi /etc/docker/daemon.json
# 输入内容并保存
{
    "registry-mirrors":["http://hub-mirror.c.163.com/"]
}
```

- 启动Docker

```
# 启动Docker服务
systemctl start docker
# 查看Docker服务启动状态
systemctl status docker
# 也可以使用docker命令
docker info
# 开机启动Docker
systemctl enable docker
```

- 停止Docker

```
systemctl stop docker
```

- 重启Docker

```
systemctl restart docker
```

三、常用命令

1. 镜像相关命令

- 查看镜像

  ```
  docker images
  ```

  - REPOSITOY： 镜像名称
  - TAG： 标签-版本
  - IMAGE ID： 镜像ID
  - CREATED： 镜像创建时间
  - SIZE： 镜像大小
- 搜索镜像

  ```
  docker search <镜像名称>
  docker search mysql
  ```

  - NAME： 镜像名称
  - DESCRIPTION： 镜像描述
  - STARS： 星标-类似git的star，受欢迎程度
  - OFFICIAL： 是否官方提供的镜像
  - AUTOMATED： 自动构建，表示该镜像由Docker hub自动构建流程创建
- 拉取镜像

  ```
  # 默认拉取最新镜像
  docker pull <镜像名称>
  # 指定版本拉取镜像
  docker pull <镜像名称>:TAG
  ```

- 删除镜像

  ```
  # 按镜像名称删除
  docker rmi <镜像名称>
  # 按镜像ID删除
  docker rmi <镜像ID>
  # 删除所有镜像
  docker rmi `docker images -q`
  ```

2. 容器相关命令

- 查看容器

```
# 查看启动的容器
docker ps
# 启动与未启动的容器
docker ps -a
# 查看最后一次运行的容器
docker ps -l
# 查看停止的容器
docker ps -f status=exited
```

- 创建与启动容器

```
docker run [ -i -t -v -d -p --name ] <镜像名称>:TAG /bin/bash
```

  - **-i**： 表示运行容器
  - -t： 表示容器启动后会进入其命令行，-it 容器创建就能登入容器
  - -v： 表示目录映射关系 [前者是宿主机目录:后者是映射到宿主机的目录]，可以使用多个 -v 进行目录映射
  - **-d**： 表示以创建一个守护式容器在后台运行，不会登入容器
  - **-p**： 表示端口映射关系[前者是宿主机端口:后者是容器内映射端口]，可以使用多个 -p 做多个端口映射
  - **--name**： 为创建的容器命名

1. 交互式方式创建容器

```
docker run -it --name=<自定义名称> <镜像名称>:TAG /bin/bash
# 退出交互，这时容器将会退出停止
exit
```

2. 守护式方式创建容器

```
docker run -id --name=<自定义名称> <镜像名称>:TAG
# 登录容器
docker exec -it <自定义名称> /bin/bash
# 退出，但是这时容器不会停止
exit
```

3. 停止容器与启动容器

```
# 停止容器
docker stop <容器名称或容器ID>
# 启动容器
docker start <容器名称或容器ID>
```

4. 文件拷贝

```
# 将文件拷贝到容器内
docker cp <需要拷贝的文件或目录> 容器名称:容器目录
# 将容器内文件拷贝出来
docker cp 容器名称:容器目录 <需要拷贝的文件或目录(可以改名)>
```

5. 目录挂载(-v)

```
# 命令
docker run -id --name:<容器名称> -v <宿主机目录>:<容器目录> <镜像名称>
# docker run -id --name:myN -v /usr/local/html:/usr/local/myhtml nginx
```

6. 查看容器的IP地址

```
# 查看容器的各种数据
docker inspect <容器名称或容器ID>
# 直接输出容器IP地址
docker inspect --format='{{.NetworkSettings.IPAddress}}' <容器名称或容器ID>
```

7. 删除容器

```
# 想要删除一个容器，想要先停止容器
docker rm <容器名称或容器ID>
```

3. 仓库相关命令

- 下载仓库镜像

```
docker pull registry
```

- 启动仓库容器

```
docker run -id --name=registry -p 5000:5000 registry
```

- daemon.json添加对私有镜像的信任

```
{"insecure-registries":["192.168.2.113:5000"]}
```

- 对需要提交到私有仓库的镜像打tag

```
# 模板命令
docker tag <镜像名称> <自定义tag名称>
# 举例
docker tag jdk1.8 192.168.2.113:5000/jdk1.8
```

- 推送镜像到私有仓库

```
docker push <自定义tag名称>
```

四、应用部署

1. MySQL 部署

- 拉取镜像

```
docker pull mysql:latest
```

- 创建容器

```
docker run -id --name=mysql01 -p 33306:3306 -e
MYSQL_ROOT_PASSWORD=123456 mysql
```

  - -p 表示端口映射
  - -e 表示添加环境变量 MYSQL_ROOT_PASSWORD 是root用户的密码
- 进入mysql容器

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

- 登录mysql

```
mysql -u -root -p
```

- 使用NAVICAT远程登录mysql

2. Tomcat 部署

- 拉取镜像

```
docker pull tomcat
```

- 创建tomcat容器

```
docker run -id --name=mytomcat -p 9000:8080 -v
/usr/local/webapps:/usr/local/tomcat/webapps tomcat
```

- 部署web应用

```
# 上传文件到服务器
alt+p
# 命令,目录的话加上 -r
put d:\demo.war
```

- 访问应用

```
http://192.168.2.113:9000/xxx
```

3. Nginx 部署

- 拉去镜像

```
docker pull nginx
```

- 创建Ngnix容器

```
docker run -id --name=myNginx -p 80:80 nginx
```

- 创建myindex.html并将myindex替换掉容器里面的index.html

```
# 写内容
vi myindex.html
# 使用cp拷贝到容器中
docker cp myindex.html myngx:/usr/share/nginx/html/index/html
```

- 访问地址 192.138.2.113:80

4. Redis 部署

- 拉去镜像

```
docker pull redis
```

- 创建容器

```
docker run -id --name=myredis -p 6379:6379 redis
```

五、迁移与备份

1. 容器保存为镜像

```
# 命令
docker commit <容器名称> <自定义镜像名称>
# 将myngx容器保存为镜像myngx_i
docker commit myngx myngx_i
```

2. 镜像备份

```
# 将镜像保存为 tar 文件命令
docker save -o <自定义tar文件名称> <镜像名称>
# 将myngx_i镜像保存为myngx.tar文件
docker save -o myngx.tar myngx_i
```

- -o： 输出的文件

3. 镜像恢复与迁移

```
# 恢复命令
docker load -i <tar镜像文件>
# demo
docker load -i myngx.tar
```

- -i： 输入的文件

六、Dockerfile

1. Dockerfile是什么?

Dockerfile是由一系列命令和参数构成的脚本，这些命令应用于基础镜像并最终创建一个新的镜像。

1. 对于开发人员，可以为开发团队提供一个完全一致的开发环境

2. 对于测试人员，可以直接拿开发时所构建的镜像或者通过Dockerfile文件构建一个新的镜像开始工作
3. 对于运维人员，在部署时，可以实现应用的无缝移植

2. 常用命令

命令	作用
FROM image_name:tag	定义了使用哪个基础镜像启动构建流程
MAINTAINER user_name	声明镜像的创建者
ENV key value	设置环境变量(可以写多条)
RUN command	是Dockerfile的核心部分(可以写多条)
ADD source_dir/file dest_dir/file	将宿主机的文件复制到容器内，如果是一个压缩文件，将会复制后自动解压
COPY source_dir/file	和ADD相似，但是如果有压缩文件并不能解压
WORKDIR path_dir	设置工作目录

3. 使用脚本创建镜像

- 创建Dockerfile文件

```
vi Dockerfile
```

- 输入内容

```
FROM centos
MAINTAINER shadow
WORKDIR /usr
RUN mkdir /usr/local/java
ADD jdk-8u212-linux-x64.tar.gz /usr/local/java

ENV JAVA_HOME /usr/local/java/jdk1.8.0_212
ENV JRE_HOME $JAVA_HOME/jre
ENV CLASSPATH
$JAVA_HOME/bin/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib:$CLASSPATH
ENV PATH $JAVA_HOME/bin:$PATH
```

- 执行构建脚本

```
docker build -t='jdk1.8' .
```

- 构建完成后查看镜像

```
docker images
```

七、Docker 私有仓库

1. 私有仓库搭建与配置

- 拉取私有仓库镜像

```
docker pull registry
```

- 启动私有仓库容器

```
docker run -id --name=registry -p 5000:5000 registry
```

- 打开浏览器输入地址http://192.168.2.113:5000/v2/_catalog看见如下内容表示私有仓库搭建成功并且内容为空

```
{"repositories":[]}
```

- 修改daemon.json

```
vi /etc/docker/daemon.json
```

添加内容，用于让docker信任私有仓库地址

```
{"insecure-registries":["192.168.2.113:5000"]}
```

- 重启docker

```
systemctl restart docker
```

- 重新启动私有仓库

```
docker start registry
```

2. 上传镜像到私有仓库

- 标记此镜像为私有仓库的镜像

```
# 打标记
docker tag jdk1.8 192.168.2.113:5000/jdk1.8
# 查看镜像
docker images
```

- 上传标记的镜像

```
docker push 192.168.2.113:5000/jdk1.8
```

- 上传完成后查看 http://192.168.2.113:5000/v2/_catalog

3. 另外一台服务器使用私有仓库

- 下载docker
- 配置daemon.json

```
{
       "registry-mirrors":["http://hub-mirror.c.163.com/"],
       "insecure-registries":["192.168.2.115:5000"]

}
```

- 下载镜像

```
docker pull 192.168.2.115:5000/jdk1.8
```