Docker

简介

Docker 是一个开源的应用容器引擎，基于 Go 语言 并遵从Apache2.0协议开源。 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，也可以实现虚拟化。 容器是完全使用沙箱机制，相互之间不会有任何接口,更重要的是容器性能开销极低。 Docker支持将软件编译成一个镜像；然后 在镜像中各种软件做好配置，将镜像发布 出去，其他使用者可以直接使用这个镜像。运行中的这个镜像称为容器，容器启动是 非常快速的。类似windows里面的ghost操 作系统，安装好后什么都有了

为什么使用docker:

由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker 对系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度， 都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。 更快速的启动时间 一致的运行环境 持续交互和部署 更轻松的迁移 更轻松的维护和扩展

Docker引擎：是一个包含以下主要组件的客户端服务器的应用程序

- 一种服务器，他是一个称为守护进程并且长时间运行的程序
- Rest ApI用于指定程序可以用来与守护进程通信的接口，并指示它做什么
- 一个有命令行界面工具的客户端

Docker系统镜像:

docker镜像(Images)：Docker 镜像是用于创建Docker 容器的 模板。 docker容器(Container)：容器是独立运行的一个或一组应用。 docker客户端(Client)：客户端通过命令行或者其他工具使用 Docker API(https://docs.docker.com/reference/api/docker_remote_api) 与Docker 的守护进程通信 docker主机(Host)：一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。 docker仓库(Registry)：Docker 仓库用来保存镜像，可以理解 为代码控制中的代码仓库。Docker Hub(https://hub.docker.com) 提供了庞大的镜像集合供使用

Docker容器:

docker容器是docker运行的实体，容器可以被创建，启动，停止，删除，暂停等 容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的 命名空间。因此容器可以拥有自己的 root 文件系统、自己的网络配置、自己的进程空间，甚至自己的用户 ID 空间。容器内的进程是运行在一个隔离的环境里，使用起来，就好像是在一个独立于宿主的系统下操作一样。容器不应该向其存储层内写入任何数据，容器存储层要保持无状态化。所有的文件写入操作，都应该使用 数据卷（Volume）、或者绑定宿主目录，在这些位置的读写会跳过容器存储层，直接对宿主（或网络存储）发生读写，其性能和稳定性更高。数据卷的生存周期独立于容器，容器消亡，数据卷不会消亡

Docker仓库:

一个 Docker Registry 中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。 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> 的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以 latest 作为默认标签。

ubuntu安装docker:

```
命令
    wget -qO- http://get.docker.com/ |sh
启动：
    sudo service docker start
测试运行
    docker run hello-world
镜像加速：    /etc/docker/daemon.json 在这里加入{ "registry-mirrors":["http://hub-mirror.c.163.com"]}
ubuntu安装：
        1.更换国内软件源，推荐中国科技大学的源，稳定速度快（可选）

            sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
            sudo sed -i 's/archive.ubuntu.com/mirrors.ustc.edu.cn/g'
            /etc/apt/sources.list
            sudo apt update
        2.安装需要的包
            sudo apt install apt-transport-https ca-certificates software-properties-common curl
        3.添加 GPG 密钥，并添加 Docker-ce 软件源，这里还是以中国科技大学的 Docker-ce 源为例
            curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
            sudo add-apt-repository "deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu \
            $添加成功后更新软件包缓存安装设置开机自启动并启动（安装成功后默认已设置并启动，可忽略）测试运行添加当前用户到用户组，可以不用运行（可选）(lsb_release -cs) stable"
        4.添加成功后更新软件包缓存sudo apt update
        5.安装 Docker-ce:::     sudo apt install docker-ce
        6.设置开机自启动并启动 Docker-ce（安装成功后默认已设置并启动，可忽略）
            sudo systemctl enable docker
            sudo systemctl start docker
        7.测试运行sudo docker run hello-world
        8.添加当前用户到 docker 用户组，可以不用 sudo 运行 docker（可选）
            sudo groupadd docker
            sudo usermod -aG docker $USER
        9.测试添加用户组（可选）docker run hello-world
```

ubuntu脚本自动安装:

1.curl -fsSL get.docker.com -o get-docker.sh 2.sh get-docker.sh --mirror Aliyun 或者第二步使用：sudo sh get-docker.sh --mirror AzureChinaCloud 3。测试是否安装成功： docker version

ubuntu安装加速器:

镜像加速: 如果没有则创建这个文件： /etc/docker/daemon.json 在这里加入{ "registry-mirrors":["https://registry.docker-cn.com"]} 重启服务： systemctl restart docker

docker中安装tomcat:

1.命令：docker pull tomcat 下载tomcat 9： docker pull tomcat:9-jre8 2.docker中运行tomcat：需要制定端口 docker run -p 8080:8080 tomcat

docker下载镜像:

1.docker pull ubuntu:16.04 //docker image ls :查看镜像列表列出的是顶级的镜像 // docker ps :查看容器列表

2.运行这个ubuntu容器： docker run -it --rm \ ubuntu:16.04 \ bash 其实上面的一串命令等于：docker run -it --rm ubuntu:16.04 bash 3.说明： it：这是两个参数，一个是 -i：交互式操作，一个是 -t 终端。我们这里打算进入 bash 执行一些命令并查看返回结果，因此我们需要交互式终端。 --rm：这个参数是说容器退出后随之将其删除。默认情况下，为了排障需求，退出的容器并不会立即删除，除非手动 docker rm。我们这里只是随便执行个命令，看看结果，不需要排障和保留结果，因此使用 --rm 可以避免浪费空间。ubuntu:16.04：这是指用 ubuntu:16.04 镜像为基础来启动容器。bash：放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是 bash运行一个容器等于运行一个对象，

dockerfile：

镜像的定制实际上就是定制每一层所添加的配置、文件。如果我们可以把每一层修改、安装、构建、操作的命令都写入一个脚本，用这个脚本来构建、定制镜像，那么之前提及的无法重复的问题、镜像构建透明性的问题、体积的问题就都会解决 dockerfile是一个文本文件，其内包含了一条条指令，每条指令构建一层，因此每条指令都应该描述如何构建 在 /usr/local: 创建docker目录，然后创建一个tomcat的dockerfile目录， from:必须是第一条指令，用于指定基础的镜像 run ： 用于执行命令行命令，由于命令行的强大， 命令有两种格式：shell格式: exec格式： 这里没有使用很多个 RUN 对一一对应不同的命令，而是仅仅使用一个 RUN 指令，并使用 && 将各个所需命令串联起来。将之前的 7 层，简化为了 1 层。这并不是在写 Shell 脚本，而是在定义每一层该如何构建 dockerfile支持shell类后面添加\命令方式换行，以及首行#号进行注释格式，很多人初学 Docker 制作出了很臃肿的镜像的原因之一，就是忘记了每一层构建的最后一定要清理掉无关文件

使用上下文环境构建：

```
    创建一个html文件：
        创建一个Dockerfile 里面书写：from tomcat copy
/usr/local/tomcat/webapps/root
        然后执行一个为：docker build -t myshop .
        里会自动的寻找到Dockerfile
        可以看到打印的执行的语句然后进入 tomcat中会发现已经创建了个index.html
        在Dockerfile 中的使用     构建时
            先声明
            Rrom tomcat
            workdir /usr/local/tomcat/webapps/root/
            run rm -fr *
            copy spring-boot-institute.jar .
            Run  unzip spring-boot-institute.jar
            run rm -fr spring-boot-institute.jar
            workdir /usr/local/tomcat

            运行：docker build -t institute      这里会自动的寻找到Dockerfile
```

Docker数据卷：

数据卷 是一个可供一个或多个容器使用的特殊目录，它绕过 UFS，可以提供很多有用的特性：

数据卷 可以在容器之间共享和重用

对 数据卷 的修改会立马生效

对 数据卷 的更新，不会影响镜像

数据卷 默认会一直存在，即使容器被删除

注意：数据卷 的使用，类似于 Linux 下对目录或文件进行 mount，镜像中的被指定为挂载点的目录中的文件会隐藏掉，能显示看的是挂载的 数据卷。

创建一个数据卷：docker volume create my-vol 查看所有数据卷：docker volume ls 在主机里查看指定的数据卷：docker volume inspect my-vol 启动一个挂载的数据容器： 在用 docker run 命令的时候，使用 --mount 标记来将 数据卷 挂载到容器里。在一次 docker run 中可以挂载多个 数据卷。 下面创建一个名为 web 的容器，并加载一个 数据卷 到容器的 /webapp 目录。：

```
$ docker run -d -P \
  --name web \
    # -v my-vol:/wepapp \
    --mount source=my-vol,target=/webapp \
    training/webapp \
    python app.py
```

查看数据卷的信息：

在主机里使用以下命令可以查看 web 容器的信息 docker inspect web

删除数据卷：

docker volume rm my-vol 数据卷 是被设计用来持久化数据的，它的生命周期独立于容器，Docker 不会在容器被删除后自动删除 数据卷，并且也不存在垃圾回收这样的机制来处理没有任何容器引用的 数据卷。如果需要在删除容器的同时移除数据卷。可以在删除容器的时候使用 docker rm -v 这个命令。

无主的数据卷的删除用以下命令清理：

docker volume pruse

实例共享数据卷：

1. 在/usr/loca/docker/tomcat下创建一个ROOT目录，并在里面书写一个index.html
2. 然后启动docker容器的tomcat：

   docker run -p 8080:8080 --name tomcat -d -v /usr/local/docker/tomcat/ROOT:/usr/local/tomcat/webapps/ROOT tomcat
3. 通过交互式进入容器中的tomcat：docker exec -it tomcat bash 可以在webapps中的ROOT目录看到index.html，
4. 然后在启动一个tomcat:docker run -p 8081:8080 --name tomcat1 -d -v /usr/local/docker/tomcat/ROOT:/usr/local/tomcat/webapps/ROOT tomcat 这时访问就可以看到两个同时访问了一个数据卷了，这时就可以共享到数据卷了：

Docker构建tomact:

进入docker目录：docker pull tomcat 启动：

```
docker run --name tomcat -p 8080:8080 -v
$PWD/test:/usr/local/tomcat/webapps/test -d tomcat
说明：-p 8080:8080：将容器的8080端口映射到主机的8080端口-v
$PWD/test:/usr/local/tomcat/webapps/test：将主机中当前目录下的test挂载到容器的/test
```

查看容器启动情况：docker ps

Docker构建mysql:

进入docker目录：docker pull mysql:5.7.22 启动运行：docker run -p 3306:3306 --name mysql \ -v /usr/local/docker/mysql/conf:/etc/mysql \ -v /usr/local/docker/mysql/logs:/var/log/mysql \ -v /usr/local/docker/mysql/data:/var/lib/mysql \ -e MYSQL_ROOT_PASSWORD=lkl555 \ -d mysql:5.7.22 说明：-p 3306:3306：将容器的3306端口映射到主机的3306端口 -v /usr/local/docker/mysql/conf:/etc/mysql：将主机当前目录下的 conf 挂载到容器的 /etc/mysql -v /usr/local/docker/mysql/logs:/var/log/mysql：将主机当前目录下的 logs 目录挂载到容器的 /var/log/mysql -v /usr/local/docker/mysql/data:/var/lib/mysql：将主机当前目录下的 data 目录挂载到容器的 /var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456：初始化root用户的密码 查看：docker ps -a 查看刚刚启动的服务

出现的错误:

Error response from daemon: driver failed programming external connectivity on endpoint mysql (c28bcf099d63d5f3b2affd38a033a42d60e3cf7054fc1bd520342b73f6a2987b): Error starting userland proxy: listen tcp 0.0.0.0:3306: bind: address already in use 原因是本地的mysql已经启动了端口不能够进行映射了、 所以停掉本地的mysql服务：sudo service mysql stop 或者使用：docker ps -a 然后使用 docker rm xxxxxxxxx 然后再从新启动

需要进入：docker容器内 docker run -it --rm mysql:5.7.22 bash ls -al 查看安装到容器的哪个位置：whereis mysql 进入、etc/mysql/mysql.conf.d 查看 mysqld.cnf是否有 max_allowd_packet=128M 没有的话则进行追加：echo "max_allowd_packet=128M" >> mysqld.cnf 这样是为了设置mysql可以执行文件的大小。 修改之后退出，进入ubuntu界面，然后重启mysql容器 docker restart mysql 将容器里面的配置文件移到宿主机 mysql/conf 目录下: docker cp mysql:/etc/mysql . 将这些文件移动到conf目录下 注意：这里出现了错误并不能够去copy过去。。。。

docker命令:

运行 docker run --name container-name -d image-name eg:docker run –name myredis –d redis --name：自定义容器名 -d：后台运行 image-name:指定镜像模板 列表 docker ps（查看运行中的容器）；加上-a；可以查看所有容器 //docker image ls :查看镜像列表列出的是顶级的镜像 // docker ps : 查看容器列表

虚悬镜像:

镜像列表中，还可以看到一个特殊的镜像，这个镜像既没有仓库名，也没有标签，均为 。： 原因： 这个镜像原本是有镜像名和标签的，原来为 mongo:3.2，随着官方镜像维护，发布了新版本后，重新 docker pull mongo:3.2 时，mongo:3.2 这个镜像名被转移到了新下载的镜像身上，而旧的镜像上的这个名称则被取消，从而成为了 。除了 docker pull 可能导致这种情况，docker build 也同样可以导致这种现象。 由于新旧镜像同名，旧镜像名称被取消，从而出现仓库名、标签均为 的镜像。

删除虚悬镜像:

$ docker image prune

查看中间层镜像:

docker image ls -a

这些无标签的镜像很多都是中间层镜像，是其它镜像所依赖的镜像。这些无标签镜像不应该删除，否则会导致上层镜像因为依赖丢失而出错。 实际上，这些镜像也没必要删除，因为之前说过，相同的层只会存一遍，而这些镜像是别的镜像的依赖，因此并不会因为它们被列出来而多存了一份，无论如何你也会需要它们 docker image ls 还支持强大的过滤器参数 --filter，或者简写 -f。之前我们已经看到了使用过

滤器来列出虚悬镜像的用法， 它还有更多的用法。比如，我们希望看到在 mongo:3.2 之后建立的镜像

```
$ docker image ls -f since=mongo:3.2
                想看某个位置之前的镜像 ：只需要把 since 换成 before
```

删除指定的镜像：

docker image rm [选项] <镜像1> [<镜像2> ...] 其中，<镜像> 可以是 镜像短 ID、镜像长 ID、镜像名 或者 镜像摘要。

```
停止 docker stop container-name/container-id 停止当前你运行的容器
启动 docker start container-name/container-id 启动容器
删除 docker rm container-id 删除指定容器
端口映射 -p 6379:6379 eg:docker run -d -p 6379:6379 --name myredis
docker.io/redis -p:主机端口(映射到)容器内部的端口
容器日志 docker logs container-name/container-id
```

Docker Compose:

Docker Compose：是docker官方编排的快速，对于容器集群很有利 Compose定位是：定义和运行多个Docker容器的应用，主要是为了解决多个容器间的相互配合来完成某项任务。他允许用户通过docker-compose.yml的定义一组相关连的应用容器为一个项目 服务：一个应用容器，实际上可以包括若干个运行相同镜像的容器实例 项目：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中定义

compose默认管理的是对象是项目，通过子命令对项目快捷的生命周期的管理

安装docker compose:

如果你已经安装了docker那么就先把docker给删除了， 先进行卸载：apt-get autoremove docker-cedocker 然后删除dokcerlist文件：在/etc/apt/sources.list.d

安装docker：

安装docker:sudo curl -fsSL get.docker.com -o get-docker.sh 因为有的aliyun上面没有具体的升级版本，所以把镜像给改写：sh get-docker.sh --mirror AzureChinaCloudcd 这时只需等待即可。。。。。。 检查镜像加速文件是否存在：在/etc/docker/daemon.json中 /etc/docker/daemon.json 在这里加入{ "registry-mirrors":["https://registry.docker-cn.com"]} 检查dockerversion:

然后安装docker-compose看下面的就可以了

安装docker 和docker compose要同时安装，

linux安装docker compose：采用二进制包进行安装：

```
$ sudo curl -L
https://github.com/docker/compose/releases/download/1.22.0/docker-compose-`uname
-s`-`uname -m` > /usr/local/bin/docker-compose
加入可执行的权限：
    $ sudo chmod +x /usr/local/bin/docker-compose
```

卸载docker compose:

$ sudo rm /usr/local/bin/docker-compose

Docker Compose 使用：

模板文件：

在docker 文件夹中的tomcat中的创建 docker-compose.yml

```
version: "3"
services:
  webapp:
    image: examples/web
  ports:
    - "80:80"
  volumes:
    - "/data"
```

注意每个服务都必须通过 image 指令指定镜像或 build 指令（需要 Dockerfile）等来自动构建生成镜像。

运行 compose 项目： docker-compose up

Docker compose的命令的使用：

```
基本的命令的格式的：docker-compose [-f=<arg>...] [options] [COMMAND] [ARGS...]
    命令选项：
        -f, --file FILE 指定使用的 Compose 模板文件，默认为 docker-compose.yml，可以多次指定。
        -p, --project-name NAME 指定项目名称，默认将使用所在目录名称作为项目名。
            --x-networking 使用 Docker 的可拔插网络后端特性
            --x-network-driver DRIVER 指定网络后端的驱动，默认为 bridge
            --verbose 输出更多调试信息。
            -v, --version 打印版本并退出。
        build的命令：
            docker-compose build [options] [SERVICE...]
            option：
                --force-rm 删除构建过程中的临时容器。
                --no-cache 构建镜像过程中不使用 cache（这将加长构建过程）。
                --pull 始终尝试通过 pull 来获取更新版本的镜像。
            config:
                验证文件格式是否正确，若正确则显示配置，若错误则进行错误原因
            down：此命令将会停止 up 命令所启动的容器，并移除网络
            exec：进入指定的容器。
            help：获得一个命令的帮助。
            images：列出 Compose 文件中包含的镜像。
            kill：格式为 docker-compose kill [options] [SERVICE...]。通过发送 SIGKILL 信号来强制停止服务容器。
            rm:格式为 docker-compose rm [options] [SERVICE...]。
            删除所有（停止状态的）服务容器。推荐先执行 docker-compose stop 命令来停止容器。
            -f：强制直接删除
            -v:删除容器所挂载的数据卷
            run:docker-compose run [options] [-p PORT...] [-e KEY=VAL...] SERVICE [COMMAND] [ARGS...]。
            参数：
                -d 后台运行容器。
                --name NAME 为容器指定一个名字。
                --entrypoint CMD 覆盖默认的容器启动指令。
                -e KEY=VAL 设置环境变量值，可多次使用选项来设置多个环境变量。
                -u, --user="" 指定运行容器的用户名或者 uid。
```

```
                --no-deps 不自动启动关联的服务容器。
                --rm 运行命令后自动删除容器，d 模式下将忽略。
                -p, --publish=[] 映射容器端口到本地主机。
                --service-ports 配置服务端口并映射到本地主机。
                -T 不分配伪 tty，意味着依赖 tty 的指令将无法运行。
                start,stop,top,pause,port 和docker中的使用的都一样
前台运行：docker-compose up
后台运行：docker-compose up -d
启动：docker-compose start
停止：docker-compose stop
停止并移除容器：docker-compose down
```

Docker compose实战tomcat:

```
version: '3.1'
services:
    tomcat:
        restart: always
        image: tomcat
        container_name: tomcat
        ports:
            - 8080:8080
        volumes:
            -
/usr/local/docker/tomcat/webapps/test:/usr/local/tomcat/webapps/test
        environment:
            TZ: Asia/Shanghai
```

Docker compose 实战mysql5：

```
version: '3.1'
services:
    mysql:
        restart: always
        image: mysql:5.7.22
        container_name: mysql
        ports:
            - - 3306:3306
        environment:
            TZ: Asia/Shanghai
            MYSQL_ROOT_PASSWORD: 123456
        command:
            --character-set-server=utf8mb4
            --collation-server=utf8mb4_general_ci
            --explicit_defaults_for_timestamp=true
            --lower_case_table_names=1
            --max_allowed_packet=128M
            --sql-
mode="STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,NO_ZERO_DAT
E,NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO"
        volumes:
                - mysql-data:/var/lib/mysq
        volumes:
                mysql-data:
```

就是为了简化docker的使用： 直接运行 docker-compose.yml中的就可以使用。运行即可以 docker-compose up

Docker-Registry:

Docker 私服:

cd /usr/local/docker 创建 registry目录 使用docker compose命令直接使用:

```
version: '3.1'
services:
    registry:
        image: registry
        restart: always
        container_name: registry
        ports:
            - 5000:5000
        volumes:
            - /usr/local/docker/registry/data:/var/lib/registry
```

只要安装之后就可以了，这里只是服务端，同样我们需要客户端测试：http://ip:5000/v2/ 我们在开发时构建服务端之后，在客户端构建一个服务之后上传到这个docker私服中饭 有其他人使用的话直接在这个地方拉取： 同时需要在客户端配置：/etc/docker/daemon.json

```
daemon.json中配置：
{
    "registry-mirrors": [
                "https://registry.docker-cn.com"
    ],
    "insecure-registries": [
                "ip:5000"
    ]
}
```

之后重启就可以：sudo systemctl daemon-reload sudo systemctl restart docker

测试镜像上传:

拉取一个镜像 docker pull nginx

```
            ## 查看全部镜像
            docker images

            ## 标记本地镜像并指向目标仓库（ip:port/image_name:tag，该格式为标记版本号）
            docker tag nginx 192.168.75.133:5000/nginx

            ## 提交镜像到仓库
            docker push 192.168.75.133:5000/nginx
            查看全部镜像：
                    curl -XGET http://192.168.75.133:5000/v2/_catalog
部署Docker Registry WebUI
            私服安装成功后就可以使用 docker 命令行工具对 registry 做各种操作了。然而不太方便的地方是不能直观的查看 registry 中的资源情况。如果可以使用 UI 工具管理镜像就更好了。这里介绍两个 Docker Registry WebUI 工具
```

```
            docker-registry-frontend
            docker-registry-web
```

在客服端中docker-compose中使用这个：

```
version: '3.1'
services:
    frontend:
        image: konradkleine/docker-registry-frontend:v2
        ports:
            - 8080:80
        volumes:
            - ./certs/frontend.crt:/etc/apache2/server.crt:ro
            - ./certs/frontend.key:/etc/apache2/server.key:ro
        environment:
            - ENV_DOCKER_REGISTRY_HOST=192.168.75.133
            - ENV_DOCKER_REGISTRY_PORT=5000
```