Docker

简介与概述

1.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。
2. Docker 的主要目标是‘build，ship and run any app，anywhere’，也就是说通过对应用程序组件的封装，分发，部署，运行等生命周期的管理。使用户的 app（可以是一个 web 应用程序或者数据库应用等）**及其运行环境能够做到‘一次封装，到处运行’。**
3.Linux 容器技术的出现解决了这个问题。而 docker 就是基于他的基础上发展过来的。将应用运行到 docker 容器上面，而 docker 容器在任何操作系统上都是一致的，这就是实现跨平台跨服务器。只需要一次配置好环境，换到别的机子上就可以一键部署好，大大简化了操作。
4. 容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。
5.Docker 从 17.03 版本之后分为 CE（Community Edition: 社区版） 和 EE（Enterprise Edition: 企业版），我们用社区版就可以了。

Docker 实际上就是一个虚拟化轻量级 linux 服务器，可以解决我们在开发环境中运行配置问题。

为什么需要使用 docker

Docker：虚拟化容器技术
Docker 主要解决我们开发环境配置迁移的问题。

1.我们现在开发了一个 javaweb 项目，需要依赖很多环境配置 比如：Tomcat、JDK 环境 、Nginx、Redis 环境等。
2.本地需要安装这些环境 Tomcat、JDK 环境、Nginx、Redis 环境等，在打 war 包给运维部署在 linux 服务器，运维人员也需要在 linux 服务器上安装 Tomcat、JDK 环境、Nginx、Redis 环境。
3.但是有时候可能会发生这些问题：我在本地运行环境没有问题，但是打包到 Linux 服务器运行总是遇到很多错误，大多数由于一些版本冲突影响。
4.所以在这时候我们就可以使用 docker 部署和安装软件就非常方便，直接将该 springboot 项目制作成一个镜像文件，镜像文件中包含 jdk 版本 tomcat 版本信息 直接部署 linux 即可，减少依赖冲突概率。

看看 linux 安装 mysql
https://blog.csdn.net/qq_42097051/article/details/113726893 在不同的 linux 内核中安装

Mysql 很容易发生版本冲突的问题。
在对比 docker 安装 mysql

docker pull mysql:5.7 docker create --name mysql3308 -e MYSQL_ROOT_PASSWORD=root -p 3308:3306 mysql:5.7

Docker 最终解决了运行环境配置中的问题。----镜像文件底层封装好了

Springboot 核心思想----

使用 docker 的好处

1. 简化配置 安装创建非常的方便
2. 代码流水线（Code Pipeline）管理 传统项目部署可能需要经过很多环节，
容易产生版本的依赖冲突问题，Docker 给应用提供了一个从开发到上线均一致的环境，让代码的流水线变得简单不少
3. Devops 开发与运维一体化减少沟通的成本 （docker 或者是 k8s 实现）
4. 虚拟技术 快速部署
5. 弹性扩容

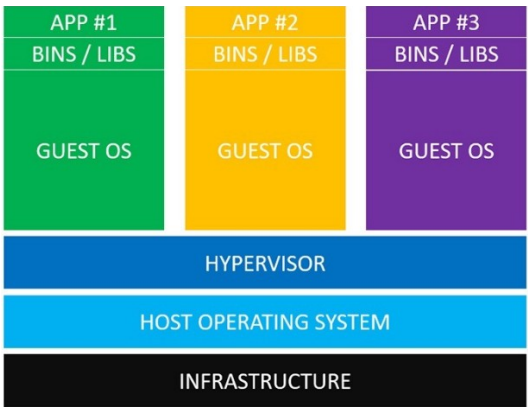
应用场景

1.Web 应用的自动化打包和发布。
2.自动化测试和持续集成、发布。

3.在服务型环境中部署和调整数据库或其他的后台应用。
4.从头编译或者扩展现有的 OpenShift 或 Cloud Foundry 平台来搭建自己的 PaaS 环境。

容器与虚拟机区别

什么是虚拟机：在一台物理机器上，利用虚拟化技术，虚拟出来多个操作系统，每个操作系统之间是隔离的。



从下到上理解上图：

最下面的一层就是物理机，可以是服务器，设置是一台个人电脑；

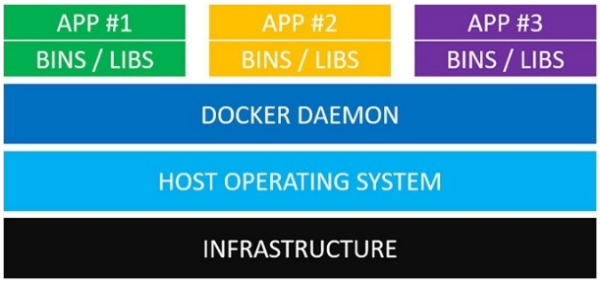
电脑上需要安装操作系统，比如我们安装了 win10 的操作系统；
再往上就是虚拟机软件了，比如我们常用的 VirtualBox、VMWare，它们的作用是模拟计算机硬件；

继续向上，就是虚拟机模拟出来的操作系统了；

在虚拟的操作系统中，安装所需的软件、组件等。比如我们需要在虚拟操作系统中安装 JDK、Tomcat 等；

最后就是具体的应用了，例如部署到 Tomcat 中。

Docker ：Docker 是开源的应用容器引擎



依然从下往上看：
最下面两层，概念同上。
往上，可以看做 Docker 容器的管理器。
依赖和应用都被打包成了 Docker 镜像。例如，JDK、Tomcat、应用都被打包在了一起，运行在 Docker 容器里，容器和容器间是隔离的。

Docker 和虚拟机的区别

1.从两者的架构图上看，**虚拟机是在硬件级别进行虚拟化，模拟硬件搭建操作系统；而 Docker 是在操作系统的层面虚拟化，复用操作系统，运行 Docker 容器**。
2.Docker 的速度很快，秒级，而虚拟机的速度通常要按分钟计算。
3.Docker 所用的资源更少，性能更高。同样一个物理机器，Docker 运行的镜像数量远多于虚拟机的数量。
4.虚拟机实现了操作系统之间的隔离，Docker 是进程之间的隔离，虚拟机隔离级别更高、安全性方面也更强。
5.虚拟机和 Docker 各有优势，不存在谁替代掉谁的问题，很多企业都采用物理机上做虚拟机，虚拟机中跑 Docker 的方式。

特性	容器	虚拟机
启动速度	秒级	分钟级别
硬盘使用	一般为 MB	一般 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个
隔离性	完全隔离	完全隔离

Docker 官网

https://docs.docker.com/
https://www.docker.com/

Docker 安装

Docker 要求 CentOS7 系统的内核版本在 3.10 以上 ， 查看本页面的前提条件来验证你的 CentOS 版本是否支持 Docker 。

1、通过 uname -r 命令查看你当前的内核版本

```
uname -r
```

2、 使用 root 权限登录 Centos。确保 yum 包更新到最新。

```
yum -y update
```

```
  nss-sysinit.x86_64 0:3.53.1-3.el7_9                           nss-tools.x86_64 0:3.53.1-3.el7_9
  nss-util.x86_64 0:3.53.1-1.el7_9                              numactl-libs.x86_64 0:2.0.12-5.el7
  openldap.x86_64 0:2.4.44-22.el7                               openssh.x86_64 0:7.4p1-21.el7
  openssh-clients.x86_64 0:7.4p1-21.el7                         openssh-server.x86_64 0:7.4p1-21.el7
  openssl.x86_64 1:1.0.2k-21.el7_9                              openssl-devel.x86_64 1:1.0.2k-21.el7_9
  openssl-libs.x86_64 1:1.0.2k-21.el7_9                         pam.x86_64 0:1.1.8-23.el7
  parted.x86_64 0:3.1-32.el7                                    passwd.x86_64 0:0.79-6.el7
  plymouth.x86_64 0:0.8.9-0.34.20140113.el7.centos              plymouth-core-libs.x86_64 0:0.8.9-0.34.20140
  plymouth-scripts.x86_64 0:0.8.9-0.34.20140113.el7.centos      policycoreutils.x86_64 0:2.5-34.el7
  polkit.x86_64 0:0.112-26.el7                                  postfix.x86_64 2:2.10.1-9.el7
  ppp.x86_64 0:2.4.5-34.el7_7                                   procps-ng.x86_64 0:3.3.10-28.el7
  python.x86_64 0:2.7.5-90.el7                                  python-firewall.noarch 0:0.6.3-12.el7
  python-libs.x86_64 0:2.7.5-90.el7                             python-linux-procfs.noarch 0:0.4.11-4.el7
  python-perf.x86_64 0:3.10.0-1160.15.2.el7                     python-urlgrabber.noarch 0:3.10-10.el7
  readline.x86_64 0:6.2-11.el7                                  rpm.x86_64 0:4.11.3-45.el7
  rpm-build-libs.x86_64 0:4.11.3-45.el7                         rpm-libs.x86_64 0:4.11.3-45.el7
  rpm-python.x86_64 0:4.11.3-45.el7                             rsyslog.x86_64 0:8.24.0-57.el7_9
  ruby.x86_64 0:2.0.0.648-36.el7                                ruby-irb.noarch 0:2.0.0.648-36.el7
  ruby-libs.x86_64 0:2.0.0.648-36.el7                           rubygem-bigdecimal.x86_64 0:1.2.0-36.el7
  rubygem-io-console.x86_64 0:0.4.2-36.el7                      rubygem-json.x86_64 0:1.7.7-36.el7
  rubygem-psych.x86_64 0:2.0.0-36.el7                           rubygem-rdoc.noarch 0:4.0.0-36.el7
  rubygems.noarch 0:2.0.14.1-36.el7                             sed.x86_64 0:4.2.2-7.el7
  selinux-policy.noarch 0:3.13.1-268.el7_9.2                    selinux-policy-targeted.noarch 0:3.13.1-268.
  setup.noarch 0:2.8.71-11.el7                                  shadow-utils.x86_64 2:4.6-5.el7
  shared-mime-info.x86_64 0:1.8-5.el7                           sqlite.x86_64 0:3.7.17-8.el7_7.1
  sudo.x86_64 0:1.8.23-10.el7_9.1                               systemd.x86_64 0:219-78.el7_9.3
  systemd-libs.x86_64 0:219-78.el7_9.3                          systemd-sysv.x86_64 0:219-78.el7_9.3
  tar.x86_64 2:1.26-35.el7                                      teamd.x86_64 0:1.29-3.el7
  tuned.noarch 0:2.11.0-11.el7_9                                tzdata.noarch 0:2021a-1.el7
  unzip.x86_64 0:6.0-21.el7                                     util-linux.x86_64 0:2.23.2-65.el7_9.1
  vim-minimal.x86_64 2:7.4.629-8.el7_9                          wget.x86_64 0:1.14-18.el7_6.1
  wpa_supplicant.x86_64 1:2.6-12.el7                            xfsprogs.x86_64 0:4.5.0-22.el7
  yum.noarch 0:3.4.3-168.el7.centos                             yum-plugin-fastestmirror.noarch 0:1.1.31-54.
  zlib.x86_64 0:1.2.7-19.el7_9                                  zlib-devel.x86_64 0:1.2.7-19.el7_9

替代:
  grub2.x86_64 1:2.02-0.65.el7.centos.2                   grub2-tools.x86_64 1:2.02-0.65.el7.centos.2

完毕!
```

该过程大概需要维持 10 分钟左右

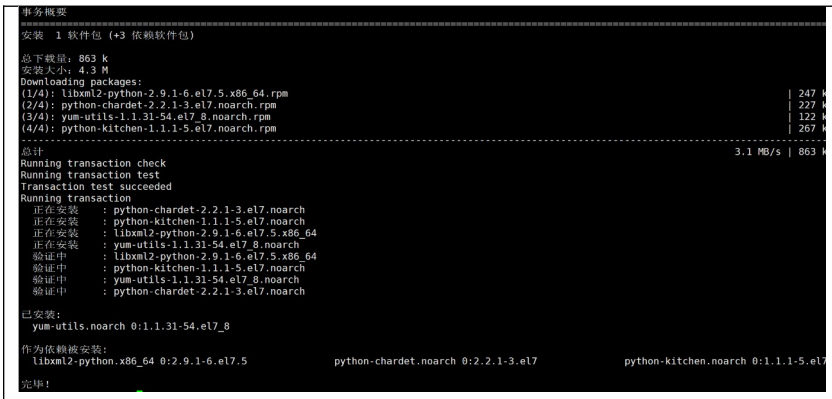
3、 卸载旧版本(如果安装过旧版本的话)

```
yum remove docker docker-common docker-selinux docker-engine
```

```
[root@localhost ~]# ^C
[root@localhost ~]# yum remove docker docker-common docker-selinux docker-engine
已加载插件：fastestmirror
参数 docker 没有匹配
参数 docker-common 没有匹配
参数 docker-selinux 没有匹配
参数 docker-engine 没有匹配
不删除任何软件包
```

4、 安装需要的软件包， yum-util 提供 yum-config-manager 功能，另外两个是 devicemapper 驱动依赖的

```
yum install -y yum-utils device-mapper-persistent-data lvm2
```



5、 设置 yum 源

```
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

6、 可以查看所有仓库中所有 docker 版本，并选择特定版本安装

```
yum list docker-ce --showduplicates | sort -r
```

7、 安装 docker

```
sudo yum install -y docker-ce    #由于 repo 中默认只开启 stable 仓库，故这里安装的是最新稳定版 18.03.1
```

8、 启动并加入开机启动

```
systemctl start docker
systemctl enable docker
```

9、 验证安装是否成功(有 client 和 service 两部分表示 docker 安装启动都成功了)

```
docker version
```

Docker 快速入门

Docker 核心名词

镜像文件
容器
仓库

镜像:简单理解为就是一个安装包，里面包含容器所需要运行的的基础文件和配置信息，比如：redis 镜像、mysql 镜像等。
镜像的来源方式:
1. 自己做镜像 比如（自己开发微服务项目）
2. 拉取别人制作好的镜像， 例如 nginx、mysql、redis 等。

容器：容器就是镜像运行的实例，容器状态分为：初创建、运行、停止、暂停、删除， 一个镜像可以创建多个不同的容器。

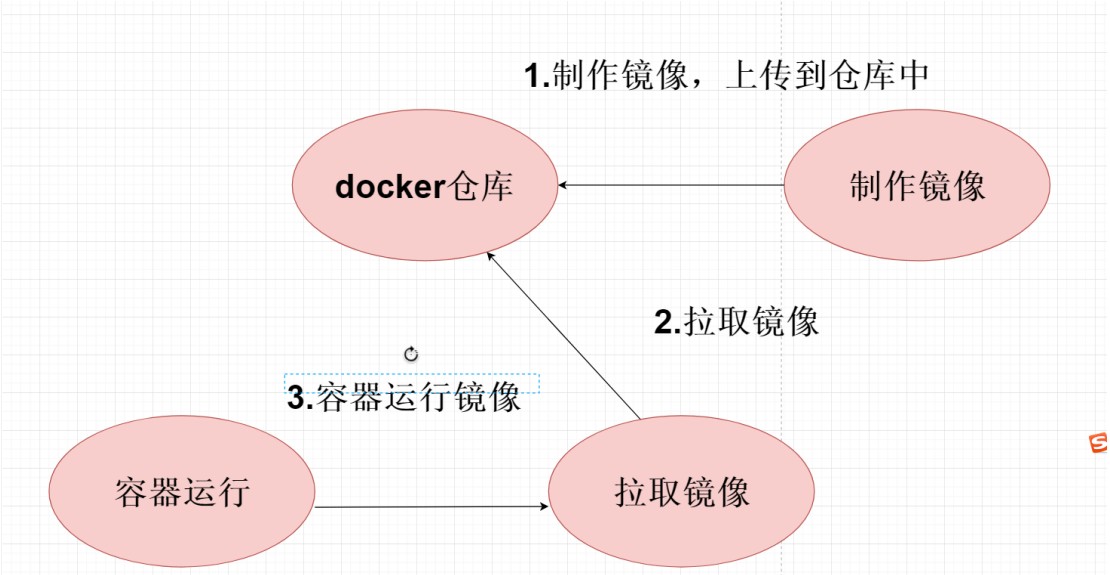
每个镜像文件都有自己独立 ip 信息---轻量级的 linux 服务器 虚拟化

比如：镜像就是类 容器就是实例对象

仓库：仓库可以简单理解为，专门存储镜像文件仓库， 类似于 谷歌手机市场，统一在谷歌手机市场下载开发者的安装包。
Docker 公开仓库地址： Docker hub
https://hub.docker.com/

Docker 官方仓库：https://hub.docker.com/ ----访问比较慢

宿主机：当前 win7 操作系统

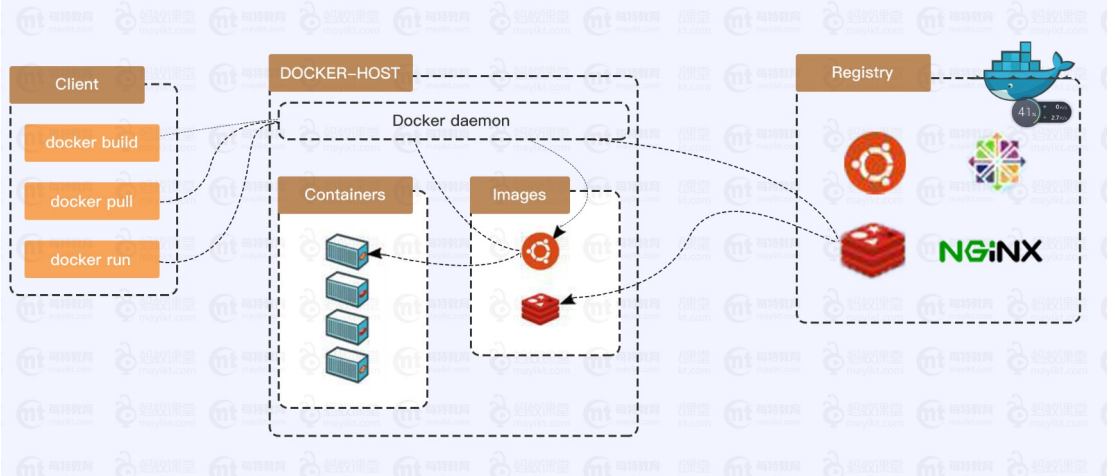


1. 需要制作镜像文件（springboot 项目）------类似于开发者开发安装应用程序打包
2. 需要将我们制作好的镜像文件提交到 docker 仓库中-----开发者将自己的 app 应用程序发布安卓手机助手中。
3. 本地需要拉去我们 docker 仓库中下载镜像文件，在交给我们容器运行---用户从 app 市场中下载安装包运行。

1.在需要制作镜像文件，将该镜像文件发布到 docker 仓库
docker 仓库 dockerhub ----谷歌安卓手机市场　国内加速镜像
阿里云、网易、科大（）----360、小米、华为。
1. 从 docker 仓库下载镜像文件-----用户从手机市场中，下载软件。
2. docker 运行镜像文件----容器---独立 ip 访问信息-----端口号码映射

Docker 下载镜像原理

Docker pull 从远程 docker 官方仓库下载 镜像，到本地，在使用容器运行该镜像。
注意的是：docker 官方镜像仓库地址部署在国外，下载镜像可能比较慢，建议配置国内加速镜像



Docker 加载镜像配置

https://hub.docker.com/search?q=redis&type=image ---在国外访问可能比较慢
国内从 DockerHub 拉取镜像有时会遇到困难，此时可以配置镜像加速器。Docker 官方和国内很多云服务商都提供了国内加速器服务，例如：

科大镜像：https://docker.mirrors.ustc.edu.cn/

网易：https://hub-mirror.c.163.com/
阿里云：https://<你的 ID>.mirror.aliyuncs.com
七牛云加速器：https://reg-mirror.qiniu.com
当配置某一个加速器地址之后，若发现拉取不到镜像，请切换到另一个加速器地址。国内各大云服务商均提供了 Docker 镜像加速服务，建议根据运行 Docker 的云平台选择对应的镜像加速服务。

阿里云加速镜像配置

我的加速镜像：https://66mzqrih.mirror.aliyuncs.com

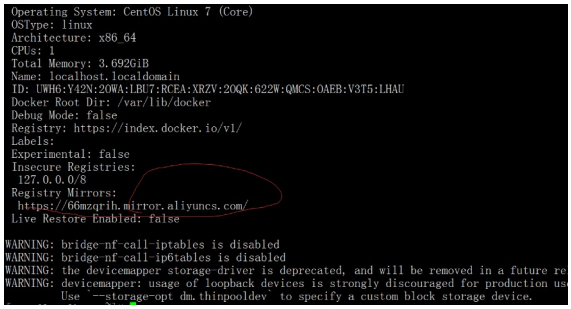
阿里云镜像获取地址：https://cr.console.aliyun.com/cn-hangzhou/instances/mirrors，登陆后，左侧菜单选中镜像加速器就可以看到你的专属地址了：


```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://66mzqrih.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

```
[root@localhost /]# cat /etc/docker/daemon.json
{
  "registry-mirrors": ["https://66mzqrih.mirror.aliyuncs.com"]
}
[root@localhost /]#
```

如何查看加速镜像安装成功

输入：docker info



Docker 常用命令

docker --help 帮助命令

docker --version

docker -version

docker images

查看本地 images 镜像缓存

docker images 查看本地镜像文件
docker rmi -f kibana:5.6.9 ---删除镜像文件

REPOSITORY 存储库名称
Tag 镜像的标签 不写版本号码 默认下载最新 latest 镜像
IMAGE ID 镜像 id
CREATED 创建时间
SIZE 大小

docker images -a
docker images -q ---只显示镜像的 id
docker images --digests ---显示镜像的摘要信息
docker images --no-trunc ---显示完整镜像信息

docker rmi tomcat（镜像文件名称）

docker search

docker search mysql
https://hub.docker.com/

```
infinitenature/jdk                  JDK Images for all jdks jabba supports          0
[root@localhost ~]# docker search mysql
NAME                              DESCRIPTION                                     STARS
OMATED
mysql                             MySQL is a widely used, open-source relation…  10468
mariadb                           MariaDB is a community-developed fork of MyS…  3893
mysql/mysql-server                Optimized MySQL Server Docker images. Create…  768
]
percona                           Percona Server is a fork of the MySQL relati…  526
```

docker search -s 30 mysql 列出点赞数超过 30 以上。

latest 表示为最新的镜像文件 mysql8.0 版本

docker pull

latest -----tag 最新版本的镜像文件

docker pull nginx:latest --默认的情况下 下载最新版本的镜像 可以通过
https://hub.docker.com/_/nginx?tab=tags&page=1&ordering=last_updated

```
[root@localhost /]# docker pull nginx:latest
latest: Pulling from library/nginx
a076a628af6f: Pull complete
0732ab25fa22: Pull complete
d7f36f6fe38f: Pull complete
f72584a26f32: Pull complete
7125e4df9063: Pull complete
Digest: sha256:10b8cc432d56da8b61b070f4c7d2543a9ed17c2b23010b43af434fd40e2ca4aa
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

容器管理

查看容器信息

Docker ps 获取到容器 id
docker inspect 1e07cc5cc78d

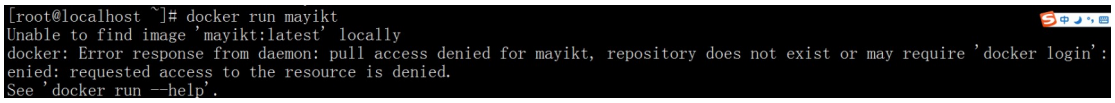
运行容器

docker run

docker run -i（保持容器一直运行）-t（给容器一个伪终端）-d(后台运行，不直接进入容器) --name=tomcat9.2（给启动容器起名字）-p 8080:8080(宿主:docker 容器)tomcat:9.2(启动的容器) 【参数】(加入容器初始化命令)
#通过 -it 启动的容器有两个特点 一创建就进入容器 exit 退出容器 容器就会停止运行 ---交互式容器
#通过 -id 创建的容器 docker exec -it tomcat9.2（--name 起的名称）进入容器 exit 退出容器 容器不会停止运行 ---守护式容器
docker ps 查看正在运行的容器
docker ps -a 查看运行和已经运行关闭大的容器
docker stop tomcat8 关闭容器
docker start tomcat8 启动容器
docker rm tomcat8 删除容器
docker inspect tomcat8 查看容器信息
docker exec 参数 进入容器

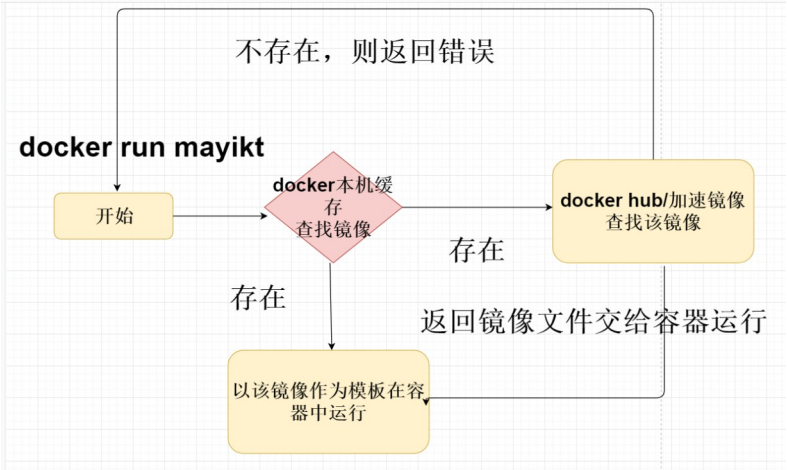
docker run 运行原理

docker run mayikt



简单描述：首先会先从本地获取获取 mayikt 镜像文件，如果本地没有该镜像文件则会去阿里云仓库查找该镜像文件，如果阿里云仓库也没有该镜像文件，则会报错找不到镜像文件。
获取到镜像文件之后直接运行。
详细描述：
1.docker 在本机缓存中 mayikt 镜像文件，如果本地存在该镜像文件

，则以该镜像文件作为模板在容器中运行。
2.如果本地缓存中，没有 mayikt 镜像文件 则会从 dockerhub 或者加速镜像中查找，如果查找不到的话，则返回错误找不到该镜像。
3. 如果能够查找到该镜像，则以该镜像作为模板运行。



每个容器都有自己独立的网络 ip 信息 运行成功 就是一个轻量级 linux 操作系统

启动容器

docker start 容器 id

停止容器

docker stop 容器 id

删除容器

docker rm 容器 id

进入容器中

首先使用下面的命令，查看容器 ID（CONTAINER ID）：
docker ps -a
然后用下面的命令进入容器，就可以使用 bash 命令浏览容器里的文件：
docker exec -it [CONTAINER ID] bash
有的镜像没有 bash 命令，可以用对应的 shell，比如 sh
docker exec -it [CONTAINER ID] sh

Docker 镜像原理

镜像是什么

基于 docker 安装 tomcat 服务器 是否需要配置 jdk 环境变量呢？
docker 安装 tomcat:8 --jdk8 配置环境变量
docker 安装 tomcat:9 --jdk9 配置环境变量
如何封装配置环境依赖的呢？

Dockerfile---文件

Tomcat 100mb
1.依赖于我们 JDK 200mb
2.Linux 服务器 centos 200mb

镜像是一种轻量级、可执行的独立软件包，用来打包软件运行环境和基于运行环境的开发软件，它包含运行某个软件所需的所有内容，包括代码、运行时、库、环境变量和配置文件。

镜像文件的组成通过 Union fs
运行我们 tomcat 镜像文件

tomcat 镜像文件
1. 依赖于我们 JDK
2. Linux 服务器

为什么运行 tomcat 镜像文件，不需要配置 jdk 环境变量。
1. tomcat 镜像文件包含 jdk 依赖镜像 tomcat8-----jdk8 镜像文件
2. 底层 dockerfile -----描述配置 jdk 环境

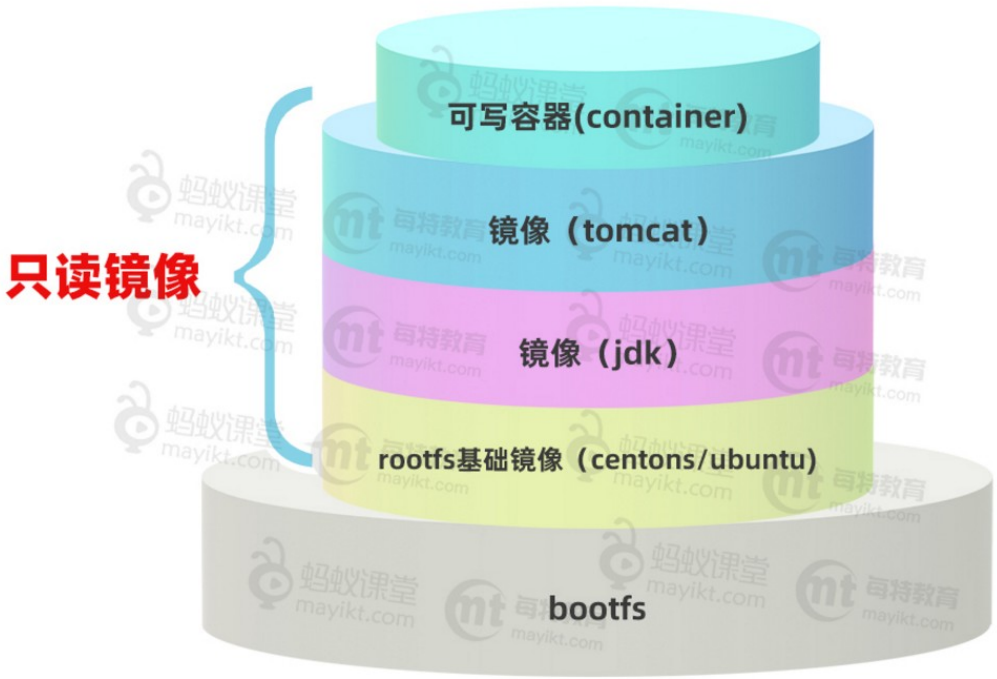
镜像加载的原理

Linux 文件系统由 bootfs 和 rootfs 两部分组成
bootfs：包含 bootloader（引导加载程序）和 kernel（内核）
rootfs： root 文件系统，包含的就是典型 Linux 系统中的/dev，/proc，/bin，/etc 等标准目录和文件
不同的 linux 发行版，bootfs 基本一样，而 rootfs 不同，如 ubuntu，centos 等

Docker 镜像底层实际上是有多个不同的联合文件系统组成的

最底层：bootfs，并使用宿主机的 bootfs-复用
第二层：root 文件系统 rootfs,称为 base image
Union fs
然后再往上可以叠加其他的镜像文件
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，隐藏多层的存在，我们看来只是存在一个文件系统。

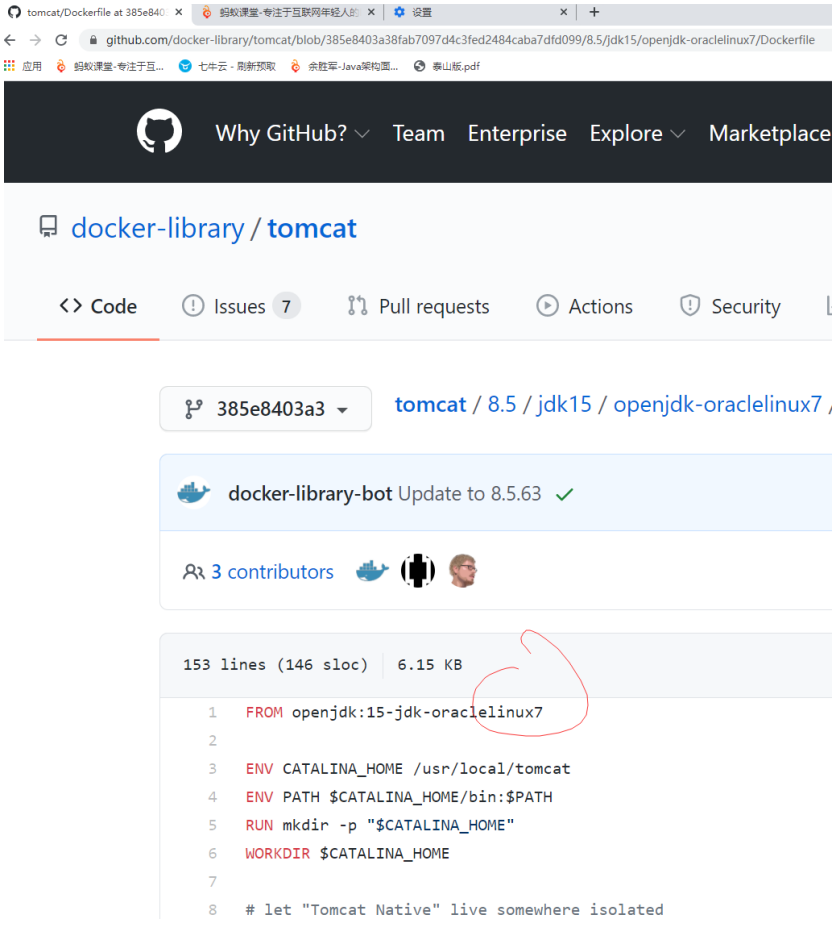
所以当我们安装的 tomcat 镜像大小是 600 多 MB 是因为里面还包含了 jdk 和 centos 的镜像
而 centos 镜像复用了宿主机的 bootfs 下载的只有 rootfs 所以小很多



所以 tomcat>jdk（父镜像）->centos> 所以整个向外暴露就是 600MB

镜像只读 当从一个镜像启动容器时， 所以 docker 会在镜像上面加载一个可读可写的文件系统作为容器运行。

https://hub.docker.com/_/tomcat




```
[root@localhost ~]# docker pull tomcat:9
9: Pulling from library/tomcat
b9a857cbf04d: Already exists
d557ee20540b: Already exists
3b9ca4f00c2e: Already exists
667fd949ed93: Already exists
661d3b55f657: Already exists
511ef4338a0b: Pull complete
a56db448fefe: Downloading [>
00612a99c7dc: Download complete
326f9601c512: Waiting
c547db74f1e1: Waiting
```

Docker Commit

主要作用：根据当前容器制作为镜像文件

流程：
1. 从 docker hub 中下载一个 tomcat8 镜像文件；
2. 运行 tomcat8 镜像文件 在 tomcatwebapps 目录中新增 mayikt 文件夹 index.html
3. 将当前容器内容根据模板制作为镜像文件

docker commit 提交容器副本使之成为一个新的镜像
命令：docker commit -m="提交的描述信息" -a="作者" 容器 ID 要创建的目标镜像名:[标签名]

1. 安装一个 tomcat8
docker run -p 8081:8080 tomcat:8
2. docker exec -it 3a06b4c779a8 bash
3. cd webapps
4. mkdir mayikt
5. touch index.html
6. echo "mayikt" >>index.html

docker commit -m="提交的描述信息" -a="作者" 容器 ID 要创建的目标镜像名:[标签名]

1.根据当前容器作为模板制作为镜像文件
docker commit -m="mayikt tomcat" -a="mayikt" 3a06b4c779a8 mayikt-tomcat:1.0
2.在以当前自己制作的镜像文件运行
docker run -p 8088:8080 mayikt-tomcat:1.0

```
exit
[root@localhost ~]# docker commit -m="mayikt tomcat" -a="mayikt"  3a06b4c779a8 mayikt-tomcat:1.0
```

Docker 数据卷

基本的概念

数据卷就是宿主机上的一个文件或目录
当容器目录和数据卷（宿主机）目录绑定，双方修改会立即同步操作

一个数据卷可以被多个容器同时挂载
数据卷作用：容器数据的持久化 外部机器和容器间接通信 容器之间数据交换
使用 -v 命令。

数据卷添加的方式

容器内与宿主机实现数据的共享
数据卷--添加两种方式
1. 直接命令形式添加 docker run -it -v 宿主机绝对路径目录:容器内目录 镜像文件名称
2. Dockerfile 方式添加

安装 Nginx 实现负载均衡

挂载 nginx html 文件

https://hub.docker.com/search?q=nginx&type=image
1. 创建挂载目录
mkdir -p /data/nginx/{conf,conf.d,html,logs}

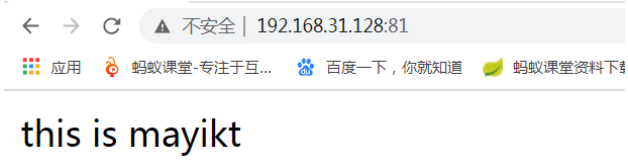
```
[root@localhost nginx]# pwd
/data/nginx
[root@localhost nginx]# ls
conf  conf.d  html  logs
[root@localhost nginx]#
```

2. 启动 docker 容器
docker run --name nginx81 -d -p 81:80 -v /data/nginx/html:/usr/share/nginx/html nginx

-v /data/nginx/html 虚拟机目录 --挂载 容器目录 /usr/share/nginx/html
上传一个 html 放入到 /data/nginx/html

docker run --name nginx81 -d -p 81:80 -v /data/nginx/html:/usr/share/nginx/html nginx

-v /data/nginx/html: linux 虚拟机目录
/usr/share/nginx/html 容器中 html 目录



nginx .conf 文件和日志文件

docker run --name nginx81 -d -p 81:80 -v /data/nginx/html:/usr/share/nginx/html \
-v /data/nginx/conf/nginx.conf:/etc/nginx/nginx.conf \

-v /data/nginx/logs:/var/log/nginx nginx

\反斜杠 表示换行
/usr/share/nginx/html
/usr/share/nginx/conf
/usr/share/nginx/log

Docker 实战部署软件

安装 Tomcat 服务器

docker run -p 8081:8080   tomcat:8

-p 8081 :8080   容器外部（linux 虚拟机访问端口 8081）:8080（容器端口号）
docker ps 获取 tomcat 正在运行的容器 id 进入到中
docker exec -it 1210e05f1a59 bash

docker run -p 8081:8080   tomcat:8

-p 8081:8080 8081(linux 虚拟机访问的端口号):8080(容器内部中端口号)

docker run -p 8081:8080 -d   tomcat:8 后台启动 ---每次运行都会创建一个新的容器

docker run --name mayikt-tomcat -p 8081:8080 -d   tomcat:8

--name: 指定容器名称
-p:指定容器端口号
-d:指定容器后台运行
docker run --name mayikt-tomcat   tomcat

docker run --name mayikt-tomcat1 -p 8081:8080 tomcat
8081（容器外部或者 linux 虚拟机访问的端口号 宿主机）
8080 容器内部的端口号
docker run --name mayikt-tomcat2022 -p 8081:8080 -d   tomcat:8
-d 后台启动
前台启动与后台启动的区别
前台启动会打印启动日志信息
后台启动不会打印启动日志信息

安装 Nginx 实现静态服务

Docker run 运行容器

--name nginx-test：容器名称。
-p 8080:80 端口进行映射，将本地 8080 端口映射到容器内部的 80 端口。
-d nginx： 设置容器在在后台一直运行。

docker ps --- 正在运行的容器
docker ps -a 显示所有的容器 包括为运行的容器
docker ps 容器 id

docker run --name nginx-mayikt -p 8080:80 nginx 默认前台启动
docker run --name nginx-mayikt -p 8080:80 -d nginx 后台启动方式

前台与后台启动区别：

前台启动：会展示容器启动的日志信息-----
后台启动：不会展示启动日志信息

8080:80 8080 虚拟机本地端口 ---浏览器访问 80 容器内部端口
Elk+kafka---
systemctl stop firewalld

安装 MySQL5.7

1.查询 mysql 版本

```
docker search mysql
```

2.下载 MySQL5.7 版本

```
docker pull mysql:5.7  (这里选择的是第一个 mysql 镜像， :5.7 选择的 5.7 版本)
```

3.等待下载完成、创建 MySQL 容器

```
docker create --name mysql3308 -e MYSQL_ROOT_PASSWORD=root -p 3308:3306 mysql:5.7
```

创建容器名称为 mysql3308，密码为 root

1. 启动容器
   docker start mysql3308
2. 进入到容器
   docker exec -it mysql3308 bash
3. mysql 连接
   mysql -uroot –p

Docker 运行底层原理

1. 首先启动 docker systemctl start docker
2. Docker 是一个 CS 架构的系统，docker 守护进程运行在主机上，让后通过 socket 连接从客户端访问，守护进程从客户端接收命令管理运行在主机上的容器。

ps aux | grep 'docker'

网站“bs”CS

数据卷 volumes-from

容器间传递共享数据 volumes-from

启动容器报错了如何解决？

先改为前台启动如果没有问题的情况下，在修改为后台启动。

容器与容器局域网

DockerFile 解析

一个镜像文件到底是如何创建？

1. dockerfile 描述出镜像文件需要的一些依赖配置和环境变量 执行命令
2. 将我们 dockerfile 文件打包成一个镜像文件
3. 直接使用我们的容器运行到该镜像文件。

1. 需要手动编写一个 dockerfile 文件
2. 将该 dockerfile docker build 自定义成一个镜像文件
3. docker run 运行容器

Centos 镜像文件

docker run -it centos

15 lines (13 sloc) | 516 Bytes

```
1  FROM scratch
2  ADD centos-7-x86_64-docker.tar.xz /
3
4  LABEL \
5      org.label-schema.schema-version="1.0" \
6      org.label-schema.name="CentOS Base Image" \
7      org.label-schema.vendor="CentOS" \
8      org.label-schema.license="GPLv2" \
9      org.label-schema.build-date="20201113" \
10     org.opencontainers.image.title="CentOS Base Image" \
11     org.opencontainers.image.vendor="CentOS" \
12     org.opencontainers.image.licenses="GPL-2.0-only" \
13     org.opencontainers.image.created="2020-11-13 00:00:00+00:00"
14
15 CMD ["/bin/bash"]
```

https://github.com/CentOS/sig-cloud-instance-images/blob/

b2d195220e1c5b181427c3172829c23ab9cd27eb/docker/Dockerfile

DockerFile 编写规范

A.#描述注释
B.指令必须要大写，后面至少需要带至少一个参数;
C.指令是按照从上到下，顺序执行；

DockerFile 指令

1. FROM 指定父镜像: 基于哪个镜像 image 构建 指定基础镜像，必须为第一个命令
2. MAINTAINER :维护者
3. RUN: 容器创建的时候执行一段命令 构建镜像时执行的命令
4. ADD: 将本地文件添加到容器中，tar 类型文件会自动解压(网络压缩资源不会被解压)，可以访问网络资源，类似 wget
5. COPY:功能类似 ADD，但是是不会自动解压文件，也不能访问网络资源
6. CMD:构建容器后调用，也就是在容器启动时才进行调用。 .sh 执行文件
7. ENV: 设置环境变量
8. EXPOSE: 指定于外界交互的端口
9. VOLUME 用于指定持久化目录
10. WORKDIR 设置进入容器时的路径 默认访问的目录

Tomcat-----jdk 环境

Tomcat docker File：
https://github.com/docker-library/tomcat/blob/385e8403a38fab7097d4c3fed2484caba7dfd099/8.5/jdk8/openjdk-slim-buster/Dockerfile

https://github.com/docker-library/redis/blob/231905d0841f52ee4f3a5b8b42d62cd6d14a1a93/6.2/Dock

进入 tomcat 容器----/data
redis 容器/data
/

DockerFile 案例

Base 镜像(scratch) docker hub 中的镜像都是通过 base 镜像中安装和配置需要的软件构建的。

构建自己 centos 镜像

docker run -it centos

1. 需求定制修改 centos 根目录；
2. 实现支持 vim 插件；

yum -y install vim

1. 需要自己制作一个 dockerfile 文件
2. 继承 docker hub 中的 centos
3. 在 docker hubcentos 上加入以下两个功能

A.进入容器中 默认访问目录/usr
B.实现支持 vim 插件

需要将该 dockerfile 文件打包成一个镜像文件 交给我们容器执行

https://github.com/CentOS/sig-cloud-instance-images/blob/b2d195220e1c5b181427c3172829c23ab9cd27eb/docker/Dockerfile

定制 CentOS 镜像

```
FROM  centos
MAINTAINER mayikt-yushengjun
ENV  MYPATH /usr
WORKDIR $MYPATH
RUN yum -y install vim
EXPOSE 80
CMD /bin/bash
```

Dockerfile→使用 docker 将该 Dockerfile 实现打包成镜像文件→
容器运行该镜像文件。

docker build---将该 Dockerfile 实现打包成镜像文件

将该 dockerfile 文件上传到 linux 服务器中
使用 docker build -f Dockerfile -t mycs:1 .
Dockerfile ------配置文件
mycs----打包镜像文件名称
1 tag 版本号码

```
 which-2.21-12.el8.x86_64

omplete!
emoving intermediate container 86125e742bb1
---> 7615ff284038
tep 6/7 : EXPOSE 80
---> Running in 9037838b19b7
emoving intermediate container 9037838b19b7
---> 2c572699ef20
tep 7/7 : CMD /bin/bash
---> Running in 2425e7e23624
emoving intermediate container 2425e7e23624
---> 07275d557f48
uccessfully built 07275d557f48
uccessfully tagged mycs:1
root@localhost dockerfile]# ^C
root@localhost dockerfile]# ^C
```

docker run -it mycs:1

将 springboot 项目打包部署

1. 基于 docker 原生方式 部署我们的 springboot 项目
Dockerfile
2.dockercompose----- 容器编排技术

springboot 项目----变成镜像文件---容器运行
1. 将我们 springboot 项目---打成一个 jar 包

2.定义 dockerfile 文件-----描述出 springboot 项目 配置依赖和环境变量
JDK
注意：springboot 内置嵌入我们的 tomcat 服务器 所以不需要额外的 tomcat 容器来运行。
原生方式运行我们的 jar 包
Java- jar 指令
2. 需要先将我们外部 jar，拷贝到容器中
3. 容器运行成功执行 java -jar
2.将该 dockerfile 文件打包成镜像文件-

1. 将 springboot 项目打包；
2. 制作 dockerfile 文件；
   A. 继承我们的 jdk 环境
   B. 将我们本地的 jar 包拷贝到容器中
   C. Java -jar

3. 将 dockerfile 文件打包成镜像文件；
4. 运行该镜像文件即可；

将 springboot 项目打包

mvn clean package

制作 dockerfile 文件

```
# 基础镜像使用 java
FROM java:8
# 作者
MAINTAINER www.mayikt.com
# VOLUME 指定了临时文件目录为/tmp。
# 其效果是在主机 /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 mayikt.jar
ADD mayikt-thymeleaf-1.0-SNAPSHOT.jar mayikt.jar
# 运行 jar 包
RUN bash -c 'touch /mayikt.jar'
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/mayikt.jar"]
#暴露 8080 端口
EXPOSE 8080
```

打包成镜像文件

docker build -f Dockerfile -t mayikt-member:1 .
docker build -f Dockerfile -t mayikt-member:1 .

启动容器

docker run -p 8070:8080 mayikt-member:1

访问测试

Docker Compose

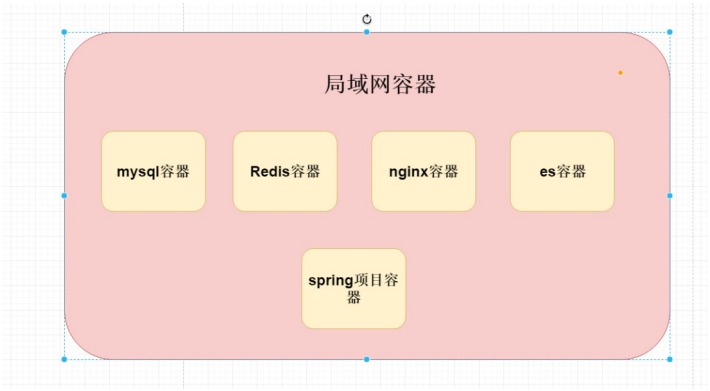
Sit pre prd 环境

为什么需要使用 Docker Compose

Docker Compose 容器编排技术

容器编排技术

1.现在我们有一个 springboot 项目，需要依赖 Redis、mysql5.7、nginx。
如果使用 docker 原生部署的话，则需要安装 Redis、mysql5、nginx 容器，在才可以启动我们 springboot 项目，这样的话部署项目的流程非常复杂，所以需要引入我们的
Docker compose 实现容器编排技术。



基本的概念

Docker-Compose 项目是 **Docker 官方的开源项目**，负责实现对 Docker 容器集群的快速编排。Docker-Compose 将所管理的容器分为三层，分别是工程（project），服务（service）以及容器（container）。
开发一个 springboot 项目---大工程
1. 依赖 mysql
2. 依赖 redis
3. 依赖 zk
等。
需要在 docker-compose.yml 配置项目工程依赖环境配置

Docker-Compose 运行目录下的所有文件（docker-compose.yml，extends 文件或环境变量文件等）组成一个工程，若无特殊指定工程名即为当前目录名。一个工程当中可包含多个服务，每个服务中定义了容器运行的镜像，参数，依赖。一个服务当中可包括多个容器实例 Docker-Compose 并没有解决负载均衡的问题，因此需要借助其它工具实现服务发现及负载均衡。

Docker-Compose 的工程配置文件默认为 docker-compose.yml，可通过环境变量 COMPOSE_FILE 或-f 参数自定义配置文件，其定义了多个有依赖关系的服务及每个服务运行的容器。

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。
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，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。

Compose 项目由 Python 编写，实现上调用了 Docker 服务提供的 API 来对容器进行管理。因此，只要所操作的平台支持 Docker API，就可以在其上利用 Compose 来进行编排管理。

Docker-Compose 分成三层
1.项目层 springboot 项目依赖于我们的 mysql redis、nginx 等 一个项目是由多个容器组成的。
2.服务层 运行一个镜像的实例 ---

Compose 环境安装（离线安装）

1. 访问 docker compose github 官网

2. docker-compose-Linux-x86_64 上传到服务器中，然后执行如下命令将其移动到/usr/local/bin/目录中 并且更名为 docker-compose

mv docker-compose-Linux-x86_64 /usr/local/bin/docker-compose

3. 执行如下命令：添加可执行的权限

sudo chmod +x /usr/local/bin/docker-compose

4. 验证 docker-compose

docker-compose -v

```
[root@localhost usr]# docker-compose -v
docker-compose version 1.24.0, build 0a[illegible]59064
```

Compose 常用命令

docker-compose -h # 查看帮助

docker-compose up # 创建并运行所有容器
docker-compose up -d # 创建并后台运行所有容器
docker-compose -f docker-compose.yml up -d # 指定模板
docker-compose down # 停止并删除容器、网络、卷、镜像。

docker-compose logs # 查看容器输出日志
docker-compose pull # 拉取依赖镜像
dokcer-compose config # 检查配置
dokcer-compose config -q # 检查配置，有问题才有输出

docker-compose restart # 重启服务
docker-compose start # 启动服务
docker-compose stop # 停止服务

Compose 入门案例

流程：

1. **需要定义一个 docker-compose.yml 文件----工程**
2. **需要在 docker-compose 文件配置依赖服务**
3. **docker-compose up 执行该文件**

1. 创建一个 docker-**compose.yml；**
2. **定制 docker-compose 内容；**
3. **运行 docker-compose up ；**

```
version: '3.0'
services:
  tomcat: ##服务名称
     image: tomcat:8 #镜像文件名称
     ports:
    - 8080:8080
```

Compose 模板文件

```
version: '3.0'
services:
  tomcat80: ##服务名称
     #container_name: tomcat8080 指定容器名称
     image: tomcat:8 #镜像文件名称 run images
    ports:  ###端口号的映射 -p
    - 8080:8080
    volumes: ## 数据源 宿主机与容器数据共享  -v
```

```
      - /usr/tomcat/webapps:/usr/local/tomcat/webapps
    networks:  ###定义网络的桥
      - mayikt


  tomcat81: ##服务名称
    #container_name: tomcat8080 指定容器名称
    image: tomcat:8 #镜像文件名称
    ports:  ###端口号的映射
      - 8081:8080
    volumes: ## 数据源 宿主机与容器数据共享
      - /usr/tomcat/webapps:/usr/local/tomcat/webapps
    networks:
      - mayikt
networks:  ## 定义服务的桥
  mayikt:
```

Compose 常用命令

docker-compose ps 列出项目中所有的容器
docker-compose stop 停止 docker-compose
docker-compose logs 查看容器中日志信息
docker-compose pull 拉取服务依赖的镜像

Compose 常用配置

Image 镜像名称；
Build 根据 docker file 打包 成镜像；
Context 指定 docker file 文件位置；
Commond 使用 command 可以覆盖容器启动后默认执行的命令；
Container_name 容器名称；
depends_on 指定依赖那个服务；
Ports 映射的端口号；
extra_hosts 会在/etc/hosts 文件中添加一些记录；

Volumes 持久化目录；
volumes_from 从另外一个容器挂在数据卷；
Dns 设置 dns

Compose 部署 springboot 项目

定义 Compose 文件

```
version: "3.0"
services:
  mysql: # mysql 服务
    image: mysql:5.7
   command: --default-authentication-plugin=mysql_native_password #解决外部无法访问
    ports:
     - "3306:3306"      #容器端口映射到宿主机的端口
    environment:
     MYSQL_ROOT_PASSWORD: 'root'
     MYSQL_ALLOW_EMPTY_PASSWORD: 'no'
     MYSQL_DATABASE: 'mayikt'
     MYSQL_USER: 'mayikt'
     MYSQL_PASSWORD: 'mayikt'
   networks:
     - mayikt_web
  mayikt-web: #自己单独的 springboot 项目
    hostname: mayikt
   build: ./     #需要构建的 Dockerfile 文件
    ports:
     - "38000:8080"      #容器端口映射到宿主机的端口
    depends_on:      #web 服务依赖 mysql 服务，要等 mysql 服务先启动
      - mysql
   networks:
     - mayikt_web
networks:  ## 定义服务的桥
  mayikt_web:
```

Spring 项目配置

```yaml
spring:
  profiles:
    active: prd
  datasource:
    url: jdbc:mysql://mysql:3306/mayikt?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
server:
  ###端口号
  port: 8080
  servlet:
    ##设置springboot项目访问路径
     context-path: /mayikt
```

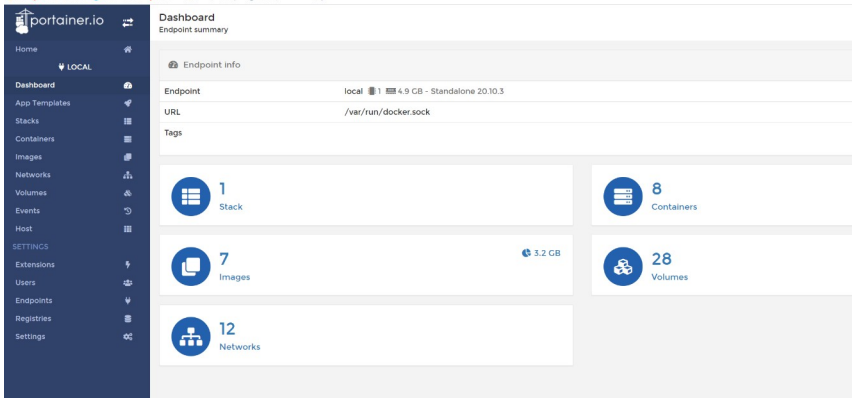
演示效果

http://192.168.163.129:38000/mayikt/insertUser?userName=mayikt&userAge=22

Docker 可视化工具使用

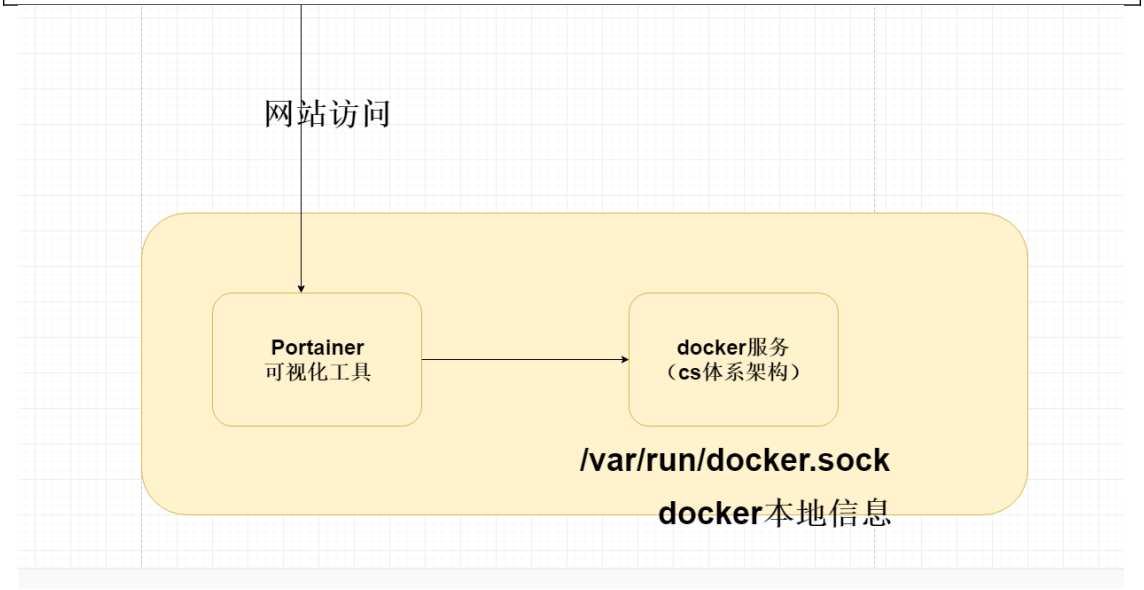
Portainer

Portainer 是一款 Docker 可视化管理工具，允许我们在网页中方便的查看和管理 Docker 容器。要使用 Portainer 很简单，运行下面两条命令即可。这些命令会创建一个 Portainer 专用的卷，然后在 8000 和 9000 端口创建容器并运行。



启动：

```
docker run -d -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```



CentOS Docker 安装

```
docker rm $(docker ps -aq)
```

docker stop $(docker ps -q) & docker rm $(docker ps -aq) ---删除所有的容器

```
docker rmi $(docker images -q)
```

```
docker network ls
```

Linux 关闭防火墙命令

systemctl stop firewalld