

1.Docker 简介

1.1. 什么是虚拟化

在计算机中，虚拟化（英语：Virtualization）是一种资源管理技术，是将计算机的各种实体资源，如服务器、网络、内存及存储等，予以抽象、转换后呈现出来，打破实体结构间的不可切割的障碍，使用户可以比原本的组态更好的方式来应用这些资源。这些资源的新虚拟部份是不受现有资源的架设方式，地域或物理组态所限制。一般所指的虚拟化资源包括计算能力和资料存储。在实际的生产环境中，虚拟化技术主要用来解决高性能的物理硬件产能过剩和老的旧的硬件产能过低的重组重用，透明化底层物理硬件，从而最大化的利用物理硬件对资源充分利用虚拟化技术种类很多，例如：软件虚拟化、硬件虚拟化、内存虚拟化、网络虚拟化(vip)、桌面虚拟化、服务虚拟化、虚拟机等等。

1.2. 什么是 Docker

Docker 是一个开源项目，诞生于 2013 年初，最初是 dotCloud 公司内部的一个业余项目。它基于 Google 公司推出的 Go 语言实现。项目后来加入了 Linux 基金会，遵从了 Apache 2.0 协议，项目代码在 GitHub 上进行维护。



Docker 自开源后受到广泛的关注和讨论，以至于 dotCloud 公司后来都改名为 DockerInc。Redhat 已经在其 RHEL6.5 中集中支持 Docker；Google 也在其 PaaS 产品中广泛应用。Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案。Docker 的基础是 Linux 容器（LXC）等技术。在 LXC 的基础上 Docker 进行了进一步的封装，让用户不需要去关心容器的管理，使得操作更为简便。用户操作 Docker 的容器就像操作一个快速轻量级的虚拟机一样简单。

为什么选择 Docker？

（1）上手快。

用户只需要几分钟，就可以把自己的程序“Docker 化”。Docker 依赖于“写时复制”（copy-on-write）模型，使修改应用程序也非常迅速，可以说达到“随心所欲，代码即改”的境界。随后，就可以创建容器来运行应用程序了。大多数 Docker 容器只需要不到 1 秒中即可启动。由于去除了管理程序的开销，Docker 容器拥有很高的性能，同时同一台宿主机中也可以运行更多的容器，使用户尽可能的充分利用系统资源。

（2）职责的逻辑分类

使用 Docker，开发人员只需要关心容器中运行的应用程序，而运维人员只需要关心如何管理容器。Docker 设计的目的是要加强开发人员写代码的开发环境与应用程序要部署的生产环境一致性。从而降低那种“开发时一切正常，肯定是运维的问题（测试环境都是正常的，上线后出了问题就归结为肯定是运维的问题）”

(3) 快速高效的开发生命周期

Docker 的目标之一就是缩短代码从开发、测试到部署、上线运行的周期，让你的应用程序具备可移植性，易于构建，并易于协作。（通俗一点说，Docker就像一个盒子，里面可以装很多物件，如果需要这些物件的可以直接将该大盒子拿走，而不需要从该盒子中一件件的取。）

(4) 鼓励使用面向服务的架构

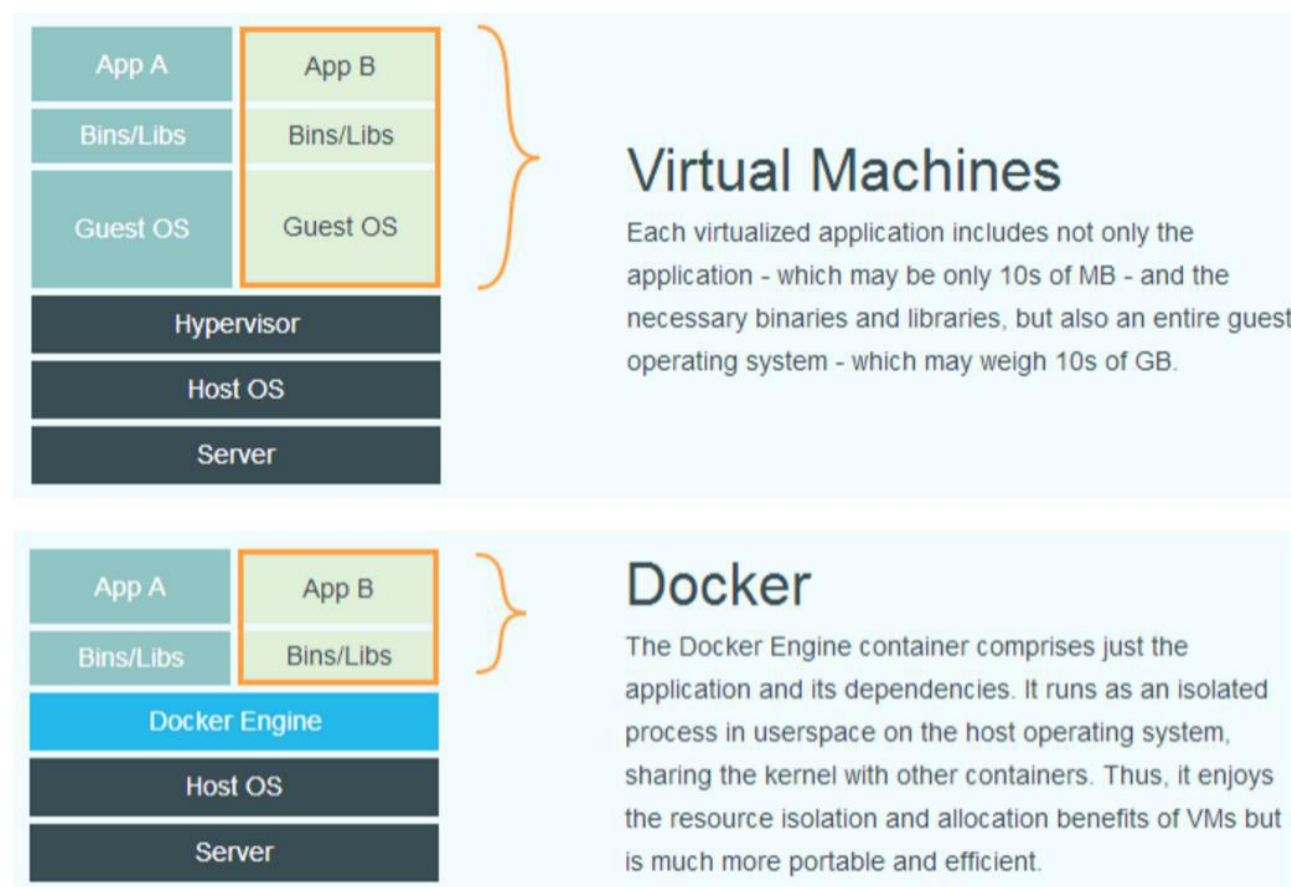
Docker 还鼓励面向服务的体系结构和微服务架构。Docker 推荐单个容器只运行一个应用程序或进程，这样就形成了一个分布式的应用程序模型，在这种模型下，应用程序或者服务都可以表示为一系列内部互联的容器，从而使分布式部署应用程序，扩展或调试应用程序都变得非常简单，同时也提高了程序的自省性。（当然，可以在一个容器中运行多个应用程序）

Docker实质上是一款容器技术框架，将所有的软件“容器化”到docker中，解决传统的开发环境与部署环境一致问题Docker将所有的应用软件变为docker容器，运行在Docker客户端中，对外映射访问端口，使得外部程序访问应用软件时只需要访问docker中对应的容器，开发环境变化时，只需要将docker中的容器“搬运到”另一种环境即可，不存在环境不一致问题

1.3. 容器与虚拟机比较

下面的图片比较了 Docker 和传统虚拟化方式的不同之处，可见容器是在操作系统层面上实

现虚拟化，直接复用本地主机的操作系统，而传统方式则是在硬件层面实现。

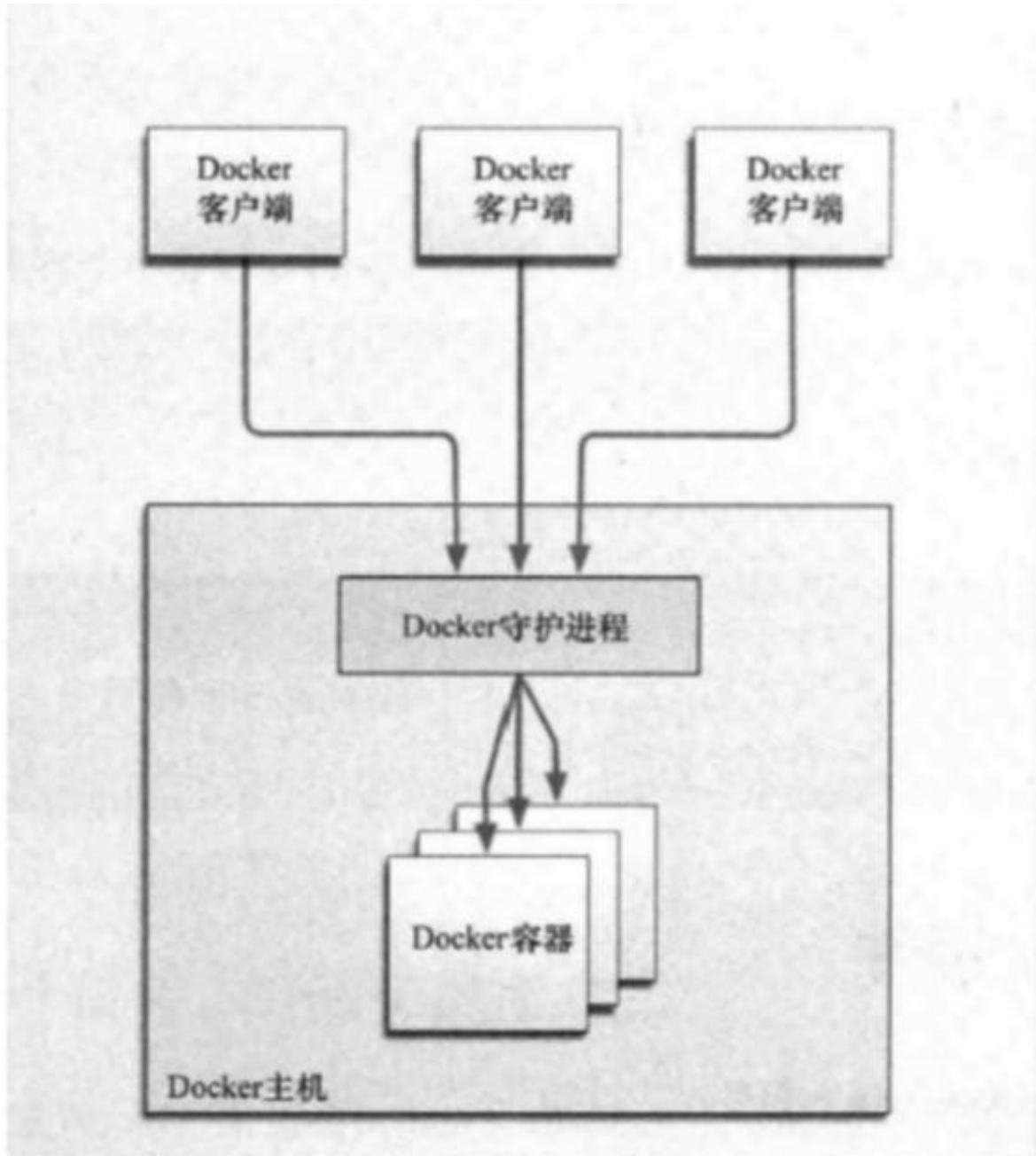


tips：与传统的虚拟机相比，Docker 优势体现为启动速度快、占用体积小。

1.4. Docker 组件

1.4.1. Docker 服务器与客户端

Docker 是一个客户端-服务器 (C/S) 架构程序。Docker 客户端只需要向 Docker服务器或者守护进程发出请求，服务器或者守护进程将完成所有工作并返回结果。Docker提供了一个命令行工具 Docker 以及一整套 RESTful API。你可以在同一台宿主机上运行 Docker守护进程和客户端，也可以从本地的 Docker 客户端连接到运行在另一台宿主机上的远程Docker 守护进程。



1.4.2. Docker 镜像与容器

镜像 是构建 Docker 的基石。用户基于镜像来运行自己的容器。镜像也是 Docker生命周期中的“构建”部分。镜像是基于联合文件系统的一种层式结构，由一系列指令一步一步构建出来。

例如：

添加一个文件；

执行一个命令；

打开一个窗口。

也可以将镜像当作容器的“源代码”。镜像体积很小，非常“便携”，易于分享、存储和更新。

Docker 可以帮助你构建和部署容器，你只需要把自己的应用程序或者服务打包放进容器即可。容器是基于镜像启动起来的，容器中可以运行一个或多个进程。我们可以认为，镜像是 Docker 生命周期中的构建或者打包阶段，而容器则是启动或者执行阶段。容器基于镜像启动，一旦容器启动完成后，我们就可以登录到容器中安装自己需要的软件或者服务。

所以 Docker 容器就是：

一个镜像格式；

一些列标准操作；

一个执行环境。

Docker 借鉴了标准集装箱的概念。标准集装箱将货物运往世界各地，Docker 将这个模型运用到自己的设计中，唯一不同的是：集装箱运输货物，而 Docker 运输软件。和集装箱一样，Docker 在执行上述操作时，并不关心容器中到底装了什么，它不管是web服务器，还是数据库，或者是应用程序服务器什么的。所有的容器都按照相同的方式将内容“装载”进去。Docker 也不关心你要把容器运到何方：我们可以在自己的笔记本中构建容器，上传到 Registry，然后下载到一个物理的或者虚拟的服务器来测试，在把容器部署到具体的主机中。像标准集装箱一样，Docker 容器方便替换，可以叠加，易于分发，并且尽量通用。

1.4.3. Registry（注册中心）

Docker 用 Registry 来保存用户构建的镜像。Registry 分为公共和私有两种。Docker公司运营公共的 Registry 叫做 Docker Hub。用户可以在 Docker Hub注册账号，分享并保存自己的镜像（说明：在 Docker Hub 下载镜像巨慢，可以自己构建私有的 Registry）。

<https://hub.docker.com/>

2.Docker 安装与启动

2.1. 安装 Docker

Docker 官方建议在 Ubuntu 中安装，因为 Docker 是基于 Ubuntu 发布的，而且一般Docker出现的问题 Ubuntu 是最先更新或者打补丁的。在很多版本的 CentOS中是不支持更新最新的一些补丁包的。由于我们学习的环境都使用的是 CentOS，因此这里我们将 Docker 安装到 CentOS上。注意：这里建议安装在 CentOS7.x 以上的版本，在 CentOS6.x的版本中，安装前需要安装其他很多的环境而且 Docker 很多补丁不支持更新。请直接挂载课程配套的 Centos7.x 镜像

(1) yum 包更新到最新

```
sudo yum update
```

(2) 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

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

(3) 设置 yum 源为阿里云

```
sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

(4) 安装 docker

```
sudo yum install docker-ce
```

(5) 安装后查看 docker 版本

```
docker -v
```

2.2. 设置阿里云的镜像

阿里云的docker 镜像加速器速度很快。登录并搜索框搜索**容器镜像服务**，找到镜像加速器

三

阿里云

容器镜像服务

▼ 默认实例

镜像仓库

命名空间

授权管理

代码源

访问凭证

▶ 企业版实例

▼ 镜像中心

镜像搜索

我的收藏

镜像加速器

镜像加速器

加速器

使用加速器可以提升获取Docker官方镜像的速度

加速器地址

复制

操作文档

UbuntuCentOSMacWindows

1. 安装 / 升级Docker客户端

推荐安装 1.10.0 以上版本的Docker客户端，参考文档 [docker-ce](#)

2. 配置镜像加速器

针对Docker客户端版本大于 1.10.0 的用户

您可以通过修改daemon配置文件 `/etc/docker/daemon.json` 来使用加速器

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["自己的镜像地址"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

执行下面命令编辑该文件：

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["自己的加速器"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

2.3. Docker 启动与停止

systemctl 命令是系统服务管理器指令

启动 docker:

```
systemctl start docker
```

停止 docker:

```
systemctl stop docker
```

重启 docker:

```
systemctl restart docker
```

查看 docker 状态:

```
systemctl status docker
```

开机启动:

```
systemctl enable docker
```

如果不成功 sudo -i 切换到root用户执行即可

查看 docker 概要信息

```
docker info
```

查看 docker 帮助文档

```
docker info
```

3.Docker 常用命令

3.1. 镜像相关命令

3.1.1. 查看镜像

```
docker images
```

REPOSITORY: 镜像名称

TAG: 镜像标签

IMAGE ID: 镜像 ID

CREATED：镜像的创建日期（不是获取该镜像的日期）

SIZE：镜像大小

这些镜像都是存储在 Docker 宿主机的 /var/lib/docker 目录下

3.1.2. 搜索镜像

如果你需要从网络中查找需要的镜像，可以通过以下命令搜索

```
docker search 镜像名称
```

NAME：仓库名称

DESCRIPTION：镜像描述

STARS：用户评价，反应一个镜像的受欢迎程度

OFFICIAL：是否官方

AUTOMATED：自动构建，表示该镜像由 Docker Hub 自动构建流程创建的

3.1.3. 拉取镜像

拉取镜像就是从中央仓库中下载镜像到本地

```
docker pull 镜像名称
```

例如，我要下载 centos7 镜像

```
docker pull centos:7
```

3.1.4. 删除镜像

按镜像 ID 删除镜像或者安装镜像名称删除镜像

```
docker rmi 镜像ID  
docker rmi 镜像名称
```

3.2. 容器相关命令

3.2.1. 查看容器

查看正在运行的容器

```
docker ps
```


查看所有容器

```
docker ps -a
```

查看最后一次运行的容器

```
docker ps -l
```

3.2.2. 创建与启动容器

创建容器常用的参数说明：

创建容器命令：docker run

-i：表示运行容器

-t：表示容器启动后会进入其命令行。加入这两个参数后，容器创建就能登录进去。即分配一个伪终端。

--name：为创建的容器命名。

-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上的目录），可以使用多个 -v 做多个目录或文件映射。注意：最好做目录映射，在宿主机上做修改，然后共享到容器上。

-d：在 run 后面加上 -d

参数，则会创建一个守护式容器在后台运行（这样创建容器后不会自动登录容器，如果只加 -i -t 两个参数，创建后就会自动进去容器）。

-p：表示端口映射，前者是宿主机端口，后者是容器内的映射端口。可以使用多个 -p 做多个端口映射

（1）交互式方式创建容器

交互式容器创建好后会进入容器，

```
docker run -it --name=容器名称 镜像名称:标签 /bin/bash
docker run -it --name=mycentos01 centos:7 /bin/bash
```

这时我们通过 ps 命令查看，发现可以看到启动的容器，状态为启动状态退出当前容器

```
exit
```

（2）守护式方式创建容器：

```
docker run -id --name=容器名称 镜像名称:标签
docker run -id --name=mycentos02 centos:7
```

（3）登录守护式容器方式：

```
docker exec -it 容器名称（或者容器 ID） /bin/bash
docker exec -it mycentos02 /bin/bash
```

3.2.3. 停止与启动容器

停止容器：

```
docker stop 容器名称（或者容器 ID）
docker stop mycentos02
```

启动容器：

```
docker start 容器名称（或者容器 ID）
docker start mycentos02
```

3.2.4. 文件拷贝

如果我们需要将文件拷贝到容器内可以使用 cp 命令 docker cp 需要拷贝的文件或目录 容器名称:容器目录

```
docker cp 需要拷贝的文件或目录 容器名称:容器目录
docker cp /root/docker.txt mycentos02:/root/docker.txt
```

也可以将文件从容器内拷贝出来

```
docker cp 容器名称:容器目录 需要拷贝的文件或目录
docker cp mycentos02:/root/docker.txt /root/test.txt
```

3.2.5. 目录挂载

我们可以在创建容器的时候，将宿主机的目录与容器内的目录进行映射，这样我们就可以通过修改宿主机某个目录的文件从而去影响容器。创建容器 添加-v 参数 后边为宿主机目录：

容器目录，例如：

```
docker run -id -v 宿主机目录:容器目录 --name=容器名称 镜像名称:镜像标签名
docker run -id -v /usr/local/myhtml:/usr/local/myhtml --name=mycentos3 centos:7
```

注意：目录挂载必须在创建容器的时候就挂载，不能后面追加挂载

3.2.6. 删除容器

删除指定的容器：

```
docker rm 容器名称（容器 ID）
```

```
docker stop mycentos01
docker rm mycentos01
```

删除容器时不能删除正在运行的容器，必须先把容器停止

4.应用部署

4.1. MySQL 部署

(1) 拉取 mysql 镜像

```
docker pull centos/mysql-57-centos7
```

(2) 创建容器

```
docker run -p 3306:3306 --name mysql -d --restart=always -v /data/mysql/conf:/etc/mysql/conf.d
-v /data/mysql/logs:/logs -v /data/mysql/data:/data -e MYSQL_ROOT_PASSWORD=root
centos/mysql-57-centos7
```

#添加防火墙放行端口

```
firewall-cmd --zone=public --add-port=3306/tcp --permanent
```

-p 代表端口映射，格式为 宿主机映射端口:容器运行端口

-e 代表添加环境变量 **MYSQL_ROOT_PASSWORD** 是 **root** 用户的登陆密码

(3) 远程登录 mysql

连接宿主机的 IP ,指定端口为 3307

测试连接：



4.2. Tomcat 部署

4.2.1 部署Tomcat

(1) 拉取镜像

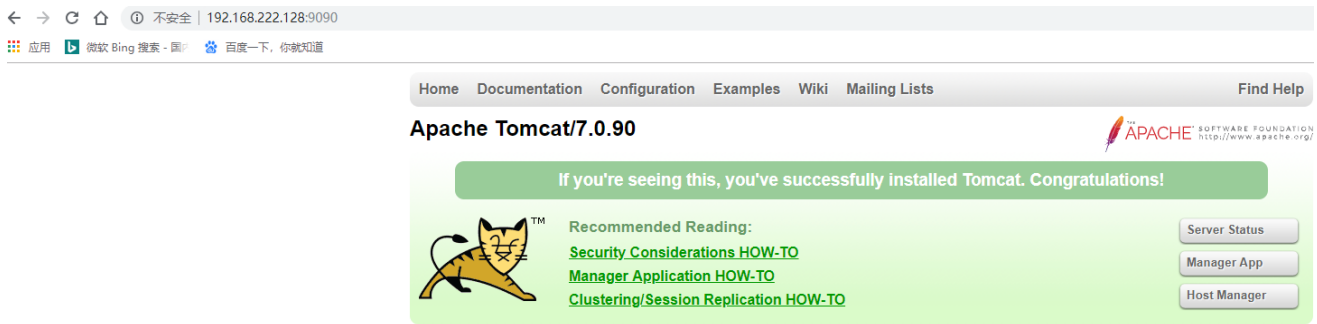
```
docker pull tomcat:7-jre7
```

(2) 创建容器

创建容器 -p 表示地址映射

```
docker run -id --name=mytomcat -p:9090:8080 tomcat:7-jre7
```

测试访问:



4.2.2 部署Tomcat（挂载webapps目录）

1.登录tomcat容器

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

```
[root@localhost ~]# docker exec -it mytomcat /bin/bash
root@513db40e3e29:/usr/local/tomcat# ls
BUILDING.txt  CONTRIBUTING.md  LICENSE  NOTICE  README.md  RELEASE-NOTES  RUNNING.txt  bin  conf  include  lib  logs  native-jni-lib  temp  webapps  work
```

经查询发现webapps目录在容器中的/usr/local/tomcat下

2.停止mytomcat容器

```
docker stop mytomcat
```

3.删除mytomcat容器

```
docker rm mytomcat
```

4.创建mytomcat容器并且配置挂载目录

```
docker run -id --name=mytomcat -p 9090:8080 -v /root/tomcatDir:/usr/local/tomcat/webapps
tomcat:7-jre7
```

创建容器并指定挂载目录

拷贝"HelloKTC.war"到宿主机目录:/root/tomcatDir

访问地址: <http://192.168.222.122:9090/HelloKTC>



HelloKTC~

4.3. Nginx 部署

(1) 拉取镜像


```
docker pull nginx
```

(2) 创建 Nginx 容器

```
docker run -id --name=mynginx -p 90:80 nginx
```

测试访问:

192.168.222.128:90

 百度一下，你就知道

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

登录Nginx容器中发现 `nginx.conf` 配置文件在 `/etc/nginx` 目录下，并查看内部配置发现没有server服务配置，找到了 `include /etc/nginx/conf.d/*.conf`，nginx默认加载该目录下的所有配置文件

```

root@6b06e191966a:/etc/nginx# ls
conf.d fastcgi_params koi-utf koi-win mime.types modules nginx.conf scgi_params uwsgi_params win-utf
root@6b06e191966a:/etc/nginx# cd nginx.conf
bash: cd: nginx.conf: Not a directory
root@6b06e191966a:/etc/nginx# ls
conf.d fastcgi_params koi-utf koi-win mime.types modules nginx.conf scgi_params uwsgi_params win-utf
root@6b06e191966a:/etc/nginx# cat nginx.conf

user nginx;
worker_processes 1;

error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on;
    #tcp_nopush on;

    keepalive_timeout 65;

    #gzip on;

    include /etc/nginx/conf.d/*.conf;
}
root@6b06e191966a:/etc/nginx#

```

进入 `/etc/nginx/conf.d` 目录，查看 `default.conf` 配置文件，发现nginx默认代理 `/etc/share/nginx/html` 目录

```

root@6b06e191966a:/etc/nginx/conf.d# ls
default.conf
root@6b06e191966a:/etc/nginx/conf.d# cat default.conf
server {
    listen      80;
    server_name localhost;

    #charset koi8-r;
    #access_log /var/log/nginx/host.access.log  main;

    location / {
        root    /usr/share/nginx/html;
        index   index.html index.htm;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root    /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ /\.php$ {
    #    proxy_pass http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ /\.php$ {
    #    root    /usr/share/nginx/html;
    #    fastcgi_pass 127.0.0.1:9000;
    #    include fastcgi.conf;
    #}
}

```

4.3.1 Nginx目录挂载

停止Nginx容器

```
docker stop mynginx
```

删除mynginx容器

```
docker rm mynginx
```

创建Nginx容器并且实现目录挂载

将KTC静态资源考入到宿主机中

```
docker run -id --name=mynginx -p 90:80 -v /root/page:/usr/share/nginx/html nginx:latest
```


测试访问

4.4. Redis 部署

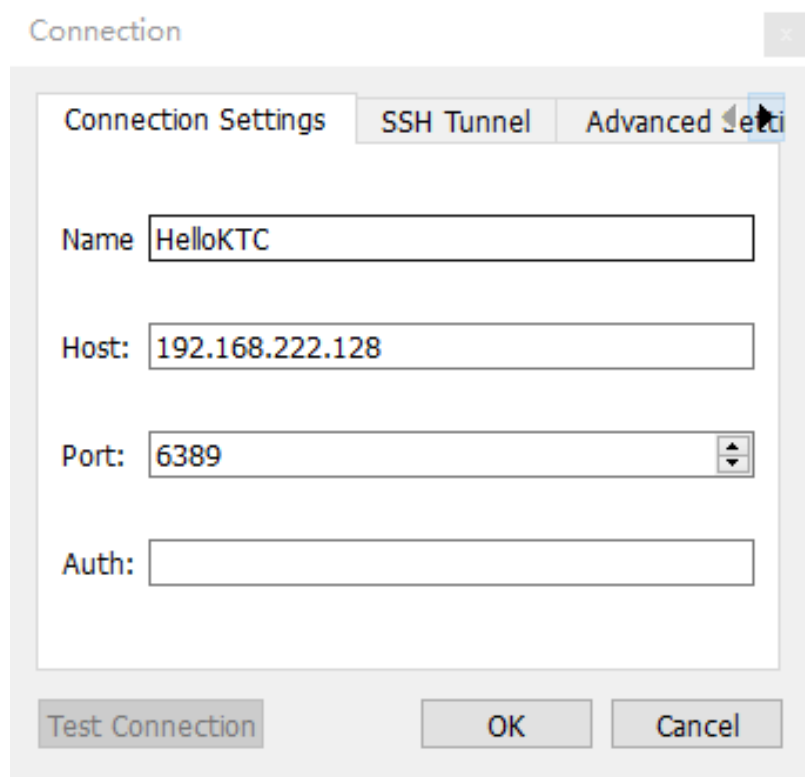
(1) 拉取镜像

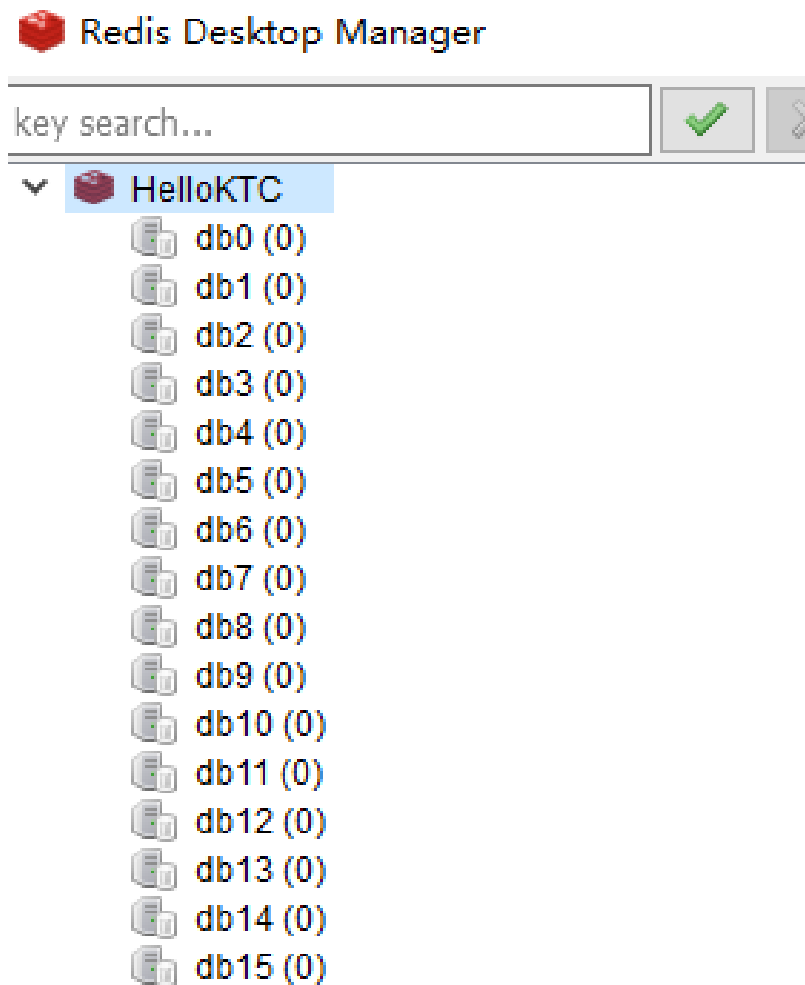
```
docker pull redis
```

(2) 创建容器

```
docker run -id --name=myredis -p 6389:6379 redis
```

测试访问：





4.5 rabbitmq部署

```
[testhadoop@sz-145-centos101 ~]$ docker search rabbitmq
```

下载镜像（有时候网络问题超时，多尝试几次即可。我这里选择的是可以访问web管理界面的tag）

```
sudo docker pull rabbitmq
```

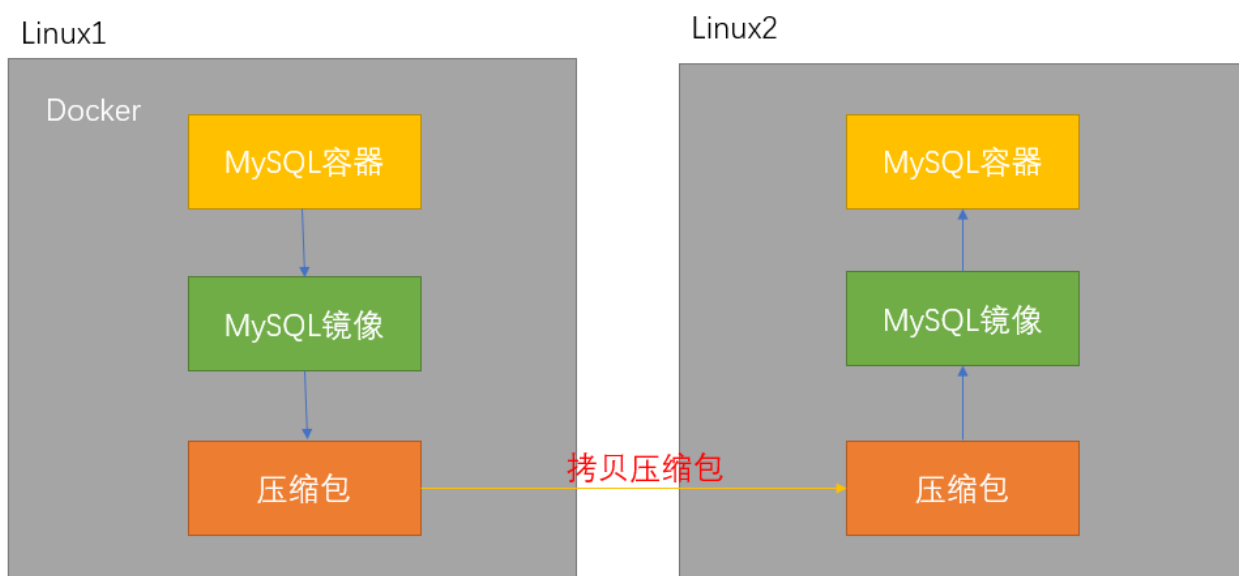
创建容器并运行（15672是管理界面的端口，5672是服务的端口。这里顺便将管理系统的用户名和密码设置为admin admin）

```
docker run -id --name rabbitmq -e RABBITMQ_DEFAULT_USER=guest -e RABBITMQ_DEFAULT_PASS=guest -p 15672:15672 -p 5672:5672 rabbitmq
```

5.迁移与备份

在Docker中实现软件（容器/数据）的迁移非常简单，而且数据不会丢失，效率非常高，速度非常快首先需要将容器变为镜像，然后将镜像打包成.tar文件，最终把.tar文件传递给其他用户其他用户拿到.tar文件后，将文件解压成镜像，最后通过镜像创建容器，就会复制出一份一模一样的容器（应用程序）

Docker迁移原理：



5.1. 容器提交为镜像

我们可以通过以下命令将容器提交为镜像

```
docker commit 容器名称 镜像名称
docker commit mysql mysql_i
```

```
[root@localhost tomcatDir]# docker images
REPOSITORY          TAG
mysql_i             latest
rancher/server      latest
mongo               latest
gogs/gogs            latest
grafana/grafana     latest
rabbitmq             management
centos/mysql-57-centos7 latest
centos               7
redis                latest
tomcat               7-jre7
nginx                latest
registry             latest
elasticsearch        5.6.8
google/cadvisor      latest
mobz/elasticsearch-head 5
tutum/influxdb       latest
[root@localhost tomcatDir]#
```

5.2. 镜像备份

我们可以通过以下命令将镜像保存为 tar 文件（保存为压缩包时间会有点长）

默认打包地址就在当前目录

```
docker save -o 要打包成压缩包的名称 要打包的镜像名称
docker save -o mysql.tar mysql_i
```

```
[root@localhost ~]# docker save -o mysql.tar mysql_i
[root@localhost ~]# ll
总用量 529100
-rw-----. 1 root root      1257 8月 27 2018 anaconda-ks.cfg
-rw-----. 1 root root 511221248 6月 19 12:05 mysql.tar
-rw-r--r--. 1 root root 30570862 9月 17 2012 nexus.war
drwxr-xr-x. 3 root root      42 6月 19 12:04 tomcatDir
[root@localhost ~]#
```

5.3. 镜像恢复与迁移

首先我们先删除掉 mysql_i 镜像 然后执行此命令进行恢复

```
docker load -i mysql.tar
```

-i 输入的文件

执行后再次查看镜像，可以看到镜像已经恢复

```
[root@localhost ~]# docker rmi mysql_i
Untagged: mysql_i:latest
Deleted: sha256:d9066cad357dbefa9671488a493135f7560fac8dfcebb6968bb45803f7269fe7
Deleted: sha256:2ecca91c547a03846df4509c615647f14114163024b6e22007b659e31501779a
[root@localhost ~]# docker load -i mysql.tar
351cb2939dec: Loading layer [=====] 53.53MB/53.53MB
Loaded image: mysql_i:latest
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mysql_i              latest             d9066cad357d       6 minutes ago      498MB
rancher/server       latest             a0b9e05b2a03       9 months ago       1.08GB
mongo                latest             c1d6c06b5775       9 months ago       381MB
gogs/gogs            latest             bd66c686064b       9 months ago       91.8MB
grafana/grafana      latest             920eb69ade2a       9 months ago       245MB
rabbitmq             management         2888deb59dfc       9 months ago       149MB
```

6. Dockerfile

6.1. 什么 Dockerfile

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了；
- 3、对于运维人员：在部署时，可以实现应用的无缝移植。

6.2. 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，将会在复制后自动解压
copy source_dir/filedest_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

6.3. 使用脚本创建镜像

步骤：

- (1) 下载 jdk-8u171-linux-x64.tar.gz 并上传到服务器（虚拟机）
- (2) 创建文件 vi Dockerfile

```
#依赖镜像名称和 ID

FROM centos:7

#指定镜像创建者信息

MAINTAINER itdfbz

RUN mkdir    /usr/local/jdk

#ADD 是相对路径 jar,把 java 添加到容器中
ADD jdk-8u171-linux-x64.tar.gz /usr/local/jdk/
#配置 java 环境变量

ENV JAVA_HOME /usr/local/jdk/jdk1.8.0_171

ENV JRE_HOME $JAVA_HOME/jre

ENV PATH $JAVA_HOME/bin:$PATH

#切换工作目录

WORKDIR /usr
```

- (4) 执行命令构建镜像

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

tips: 注意后边的空格和点, 不要省略

(5) 查看镜像是否建立完成

```
docker images
```

7.Docker 私有仓库

7.1. 私有仓库搭建与配置

(1) 拉取私有仓库镜像 (此步省略)

```
docker pull registry
```

(2) 启动私有仓库容器

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

(3) 打开浏览器输入地址 http://192.168.222.128:5000/v2/_catalog 看到 {"repositories":[]} 表示私有仓库搭建成功并且内容为空

(4) 修改 daemon.json (/etc/docker/daemon.json)

添加以下内容, 保存退出。

```
{"insecure-registries":["192.168.222.128:5000"]}
```

此步用于让 docker 信任私有仓库地址

(5) 重启 docker 服务

```
systemctl restart docker
```

7.2. 镜像上传至私有仓库

(1) 标记此镜像为私有仓库的镜像

```
docker tag 镜像名称 私服地址:端口/要打包的标签名 (随意)
```

```
docker tag jdk1.8 192.168.222.128:5000/myjdk1.8
```

(2) 上传标记的镜像

```
docker push 私服地址:端口/打包的标签名  
docker push 192.168.222.128:5000/myjdk1.8
```

(3) 删除myjdk1.8镜像

```
docker rmi 192.168.222.128:5000/myjdk1.8
```

(4) 从私服上拉取myjdk1.8镜像

```
docker pull 192.168.222.128:5000/myjdk1.8
```