一、Docker介绍

问题

问题1：

某IT部门要上线一个项目。常规操作，直接去线上服务器，拷贝一个tomcat，然后改端口号，然后部署应用到webapps文件夹下，重启就好。

一个服务器上可能会部署多个应用服务。如果某个应用出现问题，CPU100%，可能这个服务器上的其他应用也会出现问题。

对于一个大型应用拆分为几十个微服务，分别交由不同的团队开发，不同团队之间水平参差不齐。如果还采用这种部署方式，你的应用可能会因为另一个团队的应用发生意外。因部署在了同一台服务器上，导致全部出现问题。

问题2：

开发和线上代码（同一套代码）问题。开发阶段部署一套软件环境，测试人员在开发中测试没有问题，运维进行部署。但是正式部署到服务器时，发生了问题(启动参数、环境问题、漏配了参数)等意外。

问题3：

随着微服务技术的兴起，一个大的应用需要拆分成多个微服务。多个微服务的生成，就会面临庞大系统的部署效率，开发协同效率问题。然后通过服务的拆分，数据的读写分离、分库分表等方式重新架构，而且这种方式如果要做的彻底，需要花费大量人力物力。可能需要部署很多个服务器。

问题4：

持续的软件版本发布/测试项目。到线上环境的集成

什么是docker

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（类似 iPhone 的 app）,更重要的是容器性能开销极低。

为什么用docker

1、 简化程序：

Docker 让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，便可以实现虚拟化。Docker改变了虚拟化的方式，使开发者可以直接将自己的成果放入Docker中进行管理。方便快捷已经是 Docker的最大优势，过去需要用数天乃至数周的 任务，在Docker容器的处理下，只需要数秒就能完成。

2、 **避免选择恐惧症：** 如果你有选择恐惧症，还是资深患者。Docker 帮你 打包你的纠结！比如 Docker 镜像；Docker 镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

3、 **节省开支** 一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件，Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

4、 **持续交付和部署**

对开发和运维（DevOps）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。使用 Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。开发人员可以通过 Dockerfile 来进行镜像构建，并结合 持续集成(Continuous Integration) 系统进行集成测试，而运维人员则可以直接在生产环境中快速部署该镜像，甚至结合 持续部署(Continuous Delivery/Deployment) 系统进行自动部署。而且使用 Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署该镜像

5、 **更轻松的迁移**

由于 Docker 确保了执行环境的一致性，使得应用的迁移更加容易。Docker 可以在很多平台上运行，无论是物理机、虚拟机、公有云、私有云，甚至是笔记本，其运行结果是一致的。因此用户可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况

Docker的应用场景

Web 应用的自动化打包和发布。

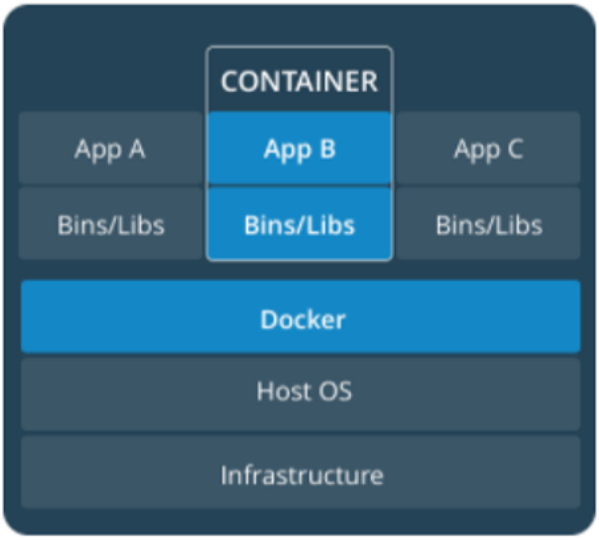
自动化测试和持续集成、发布。

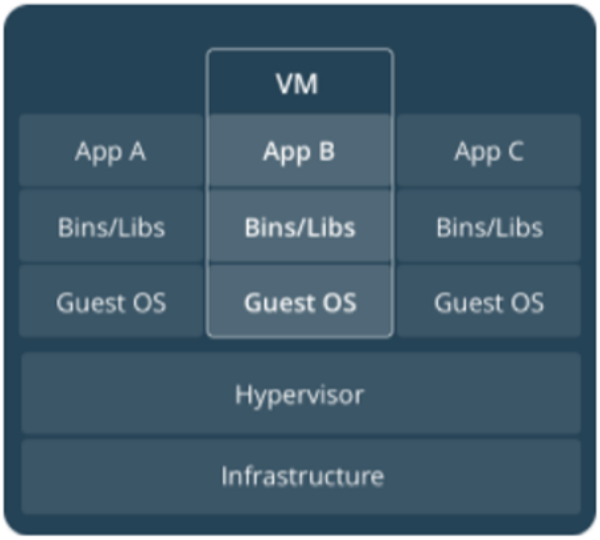
在服务型环境中部署和调整数据库或其他的后台应用。

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。

```
IaaS: (Infrastructure-as-a-Service) (基础设施即服务)
PaaS: (Platform as a Service) (平台即服务)
SaaS: (Software-as-a-Service) (软件即服务)
```

Docker和虚拟机总结





名词解释:
`infrastructure` （基础服务）硬件 `Host OS` 主机操作系统
`VM` 虚拟机 `Hypervisor` 虚拟层程序

1. **实现原理技术不同** 虚拟机是用来进行硬件资源划分的完美解决方案，利用的是硬件虚拟化技术，如此VT-x、AMD-V会通 过一个 hypervisor 层来实现对资源的彻底隔离。 而容器则是操作系统级别的虚拟化，利用的是内核的 Cgroup 和 Namespace 特性，此功能通过软件 来实现，仅仅是进程本身就可以实现互相隔离，不需要任何辅助。
2. **使用资源方面不同** Docker 容器与主机共享操作系统内核，不同的容器之间可以共享部分系统资源，因此更加轻量级， 消耗的资源更少。 虚拟机会独占分配给自己的资源，不存在资源共享，各个虚拟机之间近乎完全隔离，更加重量级，也 会消耗更多的资源。
3. **应用场景不同** 若需要资源的完全隔离并且不考虑资源的消耗，可以使用虚拟机。 若是想隔离进程并且需要运行大量进程实例，应该选择 Docker 容器。

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 `MB`	一般为 `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

Docker总结

- Docker是世界领先的软件容器平台。

- Docker使用Google公司推出的Go语言进行开发实现，基于Linux内核的cgroup，namespace，以及AUFS类的**UnionFS**等技术，对进程进行封装隔离，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。Docke最初实现是基于LXC。
- Docker能够自动执行重复性任务，例如搭建和配置开发环境，从而解放了开发人员以便他们专注在真正重要的事情上：**构建杰出的软件。**
- 用户可以方便地**创建和使用容器**，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。
- Docker的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现“这段代码在我机器上没问题啊”这类问题；——**一致的运行环境**
- 可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。——更快速的启动时间
- 避免公用的服务器，资源会容易受到其他用户的影响。——**隔离性**
- 善于处理集中爆发的服务器使用压力；——**弹性伸缩，快速扩展**
- 可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。——**迁移方便**
- 使用Docker可以通过定制应用镜像来实现持续集成、持续交付、部署。——**持续交付和部署**

二、 Docker架构

简介

Docker 使用客户端-服务器 (C/S) 架构模式，使用远程API来管理和创建Docker容器。

Docker 容器通过 Docker 镜像来创建。

容器与镜像的关系类似于面向对象编程中的对象与类。

Docker 基本概念

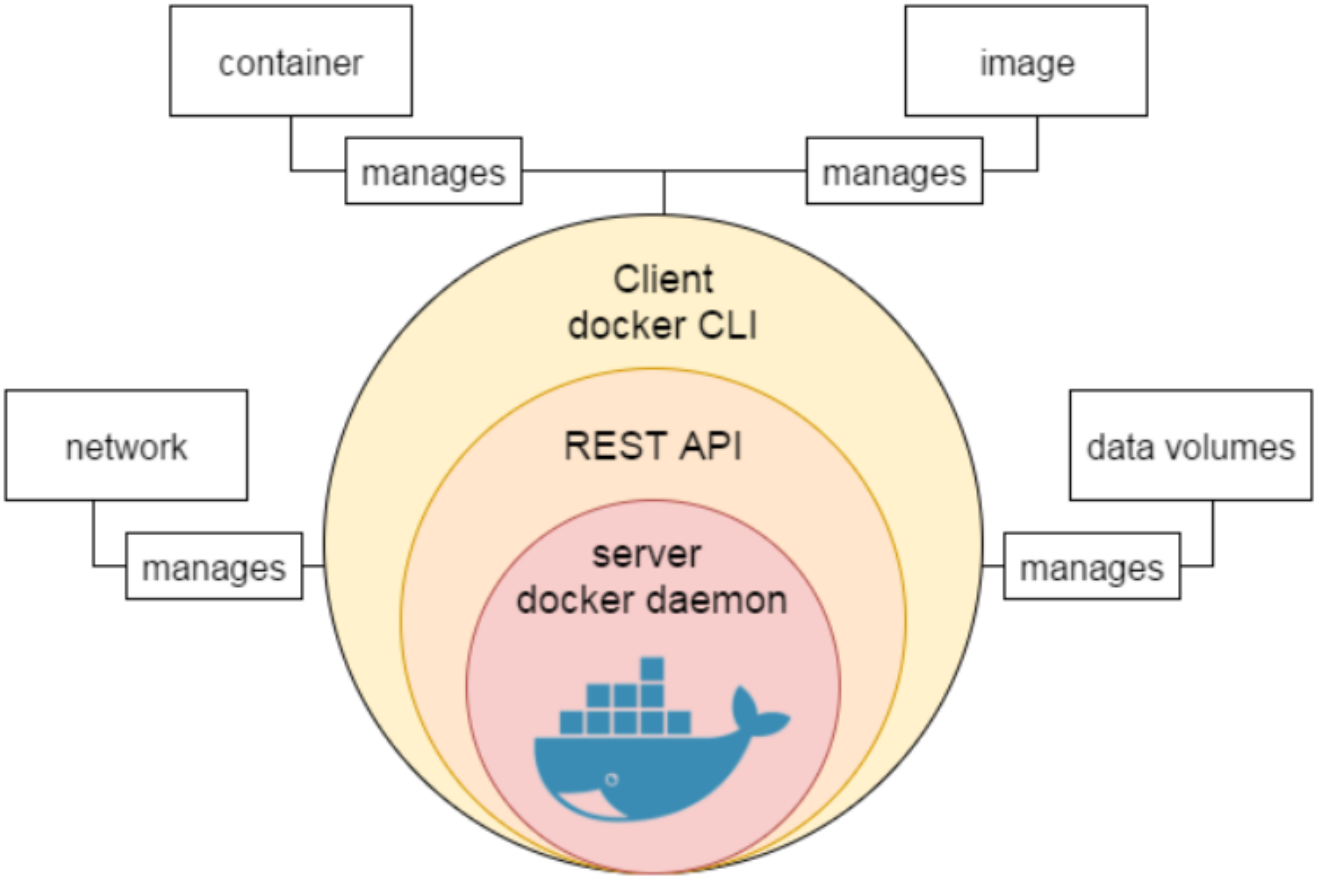
Docker 包括三个基本概念

- 镜像（`Image`）
- 容器（`Container`）
- 仓库（`Repository`）

理解了这三个概念，就理解了 Docker 的整个生命周期。

Docker 引擎

Docker 引擎组件的流程如下图所示：



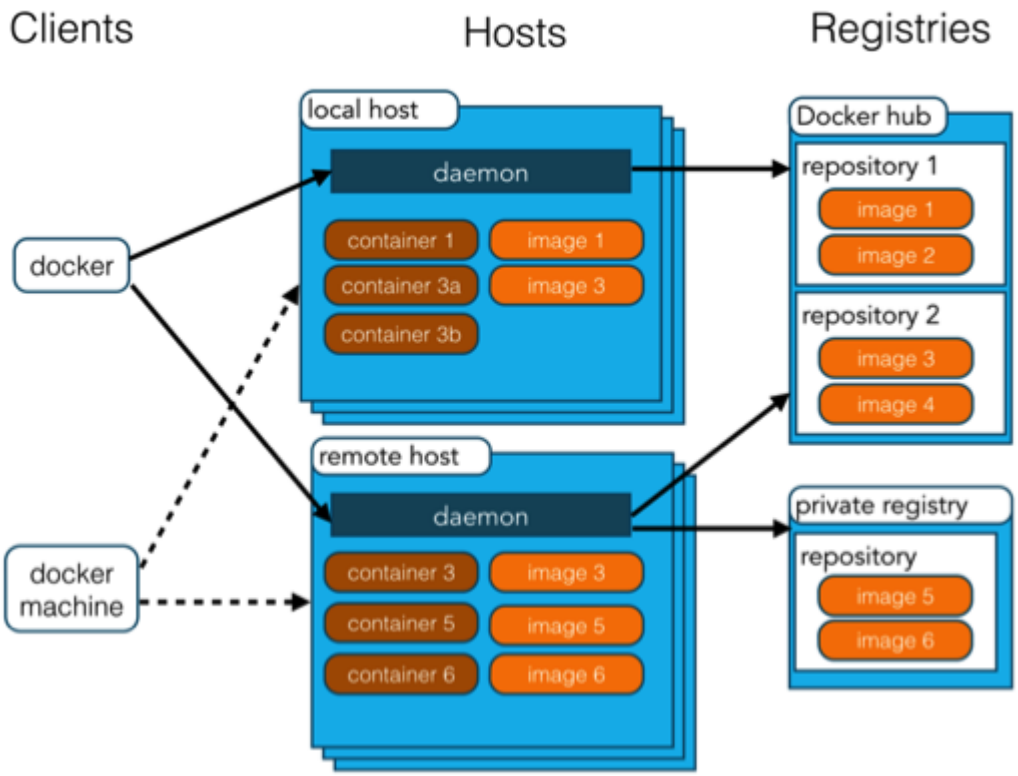
Docker 使用客户端-服务器 (C/S) 架构模式，使用远程API来管理和创建Docker容器。

Docker 容器通过 Docker 镜像来创建。

Docker 架构

容器与镜像的关系类似于面向对象编程中的对象与类。

Docker	面向对象
容器	对象
镜像	类



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)	Docker 客户端通过命令行或者其他工具使用 Docker 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提供了庞大的镜像集合供使用。
Docker Machine	Docker Machine是一个简化Docker安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装Docker，比如VirtualBox、 Digital Ocean、 Microsoft Azure。

Docker 镜像

是一个存入镜像的仓库。通常被部署在互联网服务器或者云端。

Docker Hub(https://hub.docker.com) 提供了庞大的镜像集合供使用。

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、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

分层存储

因为镜像包含操作系统完整的 `root` 文件系统，其体积往往是庞大的，因此在 Docker 设计时，就充分利用 **Union FS** 的技术，将其设计为分层存储的架构。所以严格来说，镜像并非是像一个 ISO 那样的打包文件，镜像只是一个虚拟的概念，其实际体现并非由一个文件组成，而是由一组文件系统组成，或者说，由多层文件系统联合组成。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。比如，删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。

分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像

```
jdk
linux
docker
```

Docker 容器

容器与镜像的关系类似于面向对象编程中的对象与类。

Docker	面向对象
容器	对象
镜像	类

镜像（`Image`）和容器（`Container`）的关系，就像是面向对象程序设计中的 `类` 和 `实例` 一样，镜像是静态的定义，容器是镜像运行时的实体。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。这种特性使得容器封装的应用比直接在宿主运行更加安全。也因为这种隔离的特性，很多人初学 Docker 时常常会混淆容器和虚拟机。

前面讲过镜像使用的是分层存储，容器也是如此。每一个容器运行时，是以镜像为基础层，在其上创建一个当前容器的存储层，我们可以称这个为容器运行时读写而准备的存储层为**容器存储层**。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

按照 Docker 最佳实践的要求，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。因此，使用数据卷后，容器删除或者重新运行之后，数据却不会丢失

三、Docker 仓库

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。

公有 Docker Registry

Docker Registry 公开服务是开放给用户使用、允许用户管理镜像的 Registry 服务。一般这类公开服务允许用户免费上传、下载公开的镜像，并可能提供收费服务供用户管理私有镜像。

最常使用的 Registry 公开服务是官方的 Docker Hub，这也是默认的 Registry，并拥有大量的高质量的官方镜像。除此以外，还有 CoreOS 的 Quay.io，CoreOS 相关的镜像存储在这里；Google 的 Google Container Registry，Kubernetes 的镜像使用的就是这个服务。

由于某些原因，在国内访问这些服务可能会比较慢。国内的一些云服务商提供了针对 Docker Hub 的镜像服务（`Registry Mirror`），这些镜像服务被称为**加速器**。常见的有 **阿里云加速器**、**DaoCloud 加速器** 等。使用加速器会直接从国内的地址下载 Docker Hub 的镜像，比直接从 Docker Hub 下载速度会提高很多。

国内也有一些云服务商提供类似于 Docker Hub 的公开服务。比如 **时速云镜像仓库**、**网易云镜像服务**、**DaoCloud 镜像市场**、**阿里云镜像库** 等。

私有 Docker Registry

除了使用公开服务外，用户还可以在本地搭建私有 Docker Registry。Docker 官方提供了 Docker Registry 镜像，可以直接使用做为私有 Registry 服务。

开源的 Docker Registry 镜像只提供了 Docker Registry API 的服务端实现，足以支持 `docker` 命令，不影响使用。但不包含图形界面，以及镜像维护、用户管理、访问控制等高级功能。在官方的商业化版本 Docker Trusted Registry 中，提供了这些高级功能。

除了官方的 Docker Registry 外，还有第三方软件实现了 Docker Registry API，甚至提供了用户界面以及一些高级功能。比如，VMWare Harbor 和 Sonatype Nexus。

四、Docker安装

官方提供的安装教程：

https://docs.docker.com/install/linux/docker-ce/centos/#install-using-the-repository

CentOS安装要求：

Docker支持以下的CentOS版本：

· CentOS 7 (64-bit)

· CentOS 6.5 (64-bit) 或更高的版本

安装

目前，CentOS 仅发行版本中的内核支持 Docker。Docker 运行在 CentOS 7 上，要求系统为64位、系统内核版本为 3.10 以上。Docker 运行在 CentOS-6.5 或更高的版本的 CentOS 上，要求系统为64位、系统内核版本为 2.6.32-431 或者更高版本。

1校验版本

> 命令：uname -r 校验Linux内核版本（3.10以上版本）

```
从 2017 年 3 月开始 docker 在原来的基础上分为两个分支版本：Docker CE 和 Docker EE。
Docker CE 即社区免费版，Docker EE 即企业版，强调安全，但需付费使用。
本文介绍 Docker CE 的安装使用。
```

2移除旧的版本

```
sudo yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-selinux \
                  docker-engine-selinux \
                  docker-engine
```

> 如果 `yum` 报告未安装这些软件包，则可以。

3安装一些必要的系统工具：

> 安装所需的软件包。 `yum-utils` 提供了 `yum-config-manager` 应用，并 `device-mapper-persistent-data` 和 `lvm2` 由需要 `devicemapper` 存储驱动程序。

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

4添加软件源信息

源1：（官方推荐）

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

源2：（阿里云源）

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

5、更新 yum 缓存：

```
sudo yum makecache fast
```

6安装 Docker-CE

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

7启动 Docker 后台服务

```
sudo systemctl start docker
```

8重启 Docker服务

```
sudo systemctl restart docker
```

9安装成功后可通过查看

```
docker version
```

10测试

通过运行hello-world 映像来验证是否正确安装了Docker Engine-Community

```
sudo docker run hello-world
```

卸载

执行以下命令来删除 Docker CE：

```
$ sudo yum remove docker-ce
$ sudo rm -rf /var/lib/docker
```

五、Docker 镜像加速器

鉴于国内网络问题，后续拉取 Docker 镜像十分缓慢，我们可以需要配置加速器来解决

```
    使用Docker 的时候，需要经常从官方获取镜像，但是由于显而易见的网络原因，拉取镜像的过程非
常耗时，严重 影响使用 Docker 的体验。因此 DaoCloud 推出了加速器工具解决这个难题，通过智能
路由和缓存机制，极大提 升了国内网络访问 Docker Hub 的速度，目前已经拥有了广泛的用户群体，
并得到了 Docker 官方的大力推荐。 如果您是在国内的网络环境使用 Docker，那么 Docker 加速器一
定能帮助到您。
```

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：

- **Docker 官方提供的中国 registry mirror**
- **阿里云加速器**
- **DaoCloud 加速器**

Docker官方加速器

我们以 Docker 官方加速器进行介绍

https://docs.docker.com/registry/recipes/mirror/#use-case-the-china-registry-mirror

通过命令查看：在 `/etc/docker/daemon.json` 中写入如下内容（如果文件不存在请新建该文件）

```
{
"registry-mirrors":["https://registry.docker-cn.com"]
}
```

> 注意，一定要保证该文件符合 json 规范，否则 Docker 将不能启动

重启Docker:

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

阿里云加速器

阿里云的镜像源有个加速器，可以加速你获取容器的速度。这个加速器地址是每个人专属的。 网址: https://dev.aliyun.com/

登录阿里云个人帐号。

管理中心--容器镜像服务--镜像加速器

https://cr.console.aliyun.com/cn-hangzhou/instances/mirrors

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://gxeo3yz7.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

检查加速器是否生效

配置加速器之后，如果拉取镜像仍然十分缓慢，请手动检查加速器配置是否生效，

在命令行执行 `docker info`，查看镜像地址是否匹配，如匹配，说明配置成功

```
docker info
```

测试

尝试从docker库中下载镜像：

```
docker pull tomcat
```

通过观察，下载速度明显提升。

查看docker下载的镜像内容：

```
docker images
```

下载指定的镜像版本：

```
docker images tomcat:9-jre11
```

启动tomcat:

```
docker run -p 8080:8080 tomcat
```

六、Docker镜像

镜像是Docker的三大组件之一。

Docker运行容器前需要本地存在对应的镜像，如果本地不存，Docker会从镜像仓库下载。

本节将介绍更多关于镜像的内容，包括：

从仓库获取镜像；

管理本地主机上的镜像；

介绍镜像实现的基本原理;

Docker命令使用

> 获取命令行帮助信息直接在命令行内输入docker 命令后敲回车

Docker 获取镜像

之前提到过，Docker Hub 上有大量的高质量的镜像可以用，这里我们就说一下怎么获取这些镜像。

查找镜像

我们可以从 Docker Hub 网站来搜索镜像，Docker Hub 网址为： https://hub.docker.com/

我们也可以使用 docker search 命令来搜索镜像。比如我们需要一个tomcat的镜像来作为我们的web服务。我们可以通过 docker search 命令搜索tomcat来寻找适合我们的镜像。

```
[root@guoweixin ~]# docker search --help
    Usage: docker search [OPTIONS] TERM
    Search the Docker Hub for images
    Options:
     -f, --filter filter Filter output based on conditions provided
     根据提供的条件过滤器输出
     --format string Pretty-print search using a Go template
     用Go模板打印出漂亮的搜索结果
     --limit int Max number of search results (default 25)
     搜索结果的最大数量（默认值为25）
     --no-trunc Don't truncate output
     不要截断输出
```

```
docker search tomcat
```

获取镜像

从 Docker 镜像仓库获取镜像的命令是 `docker pull` 其命令格式为：

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

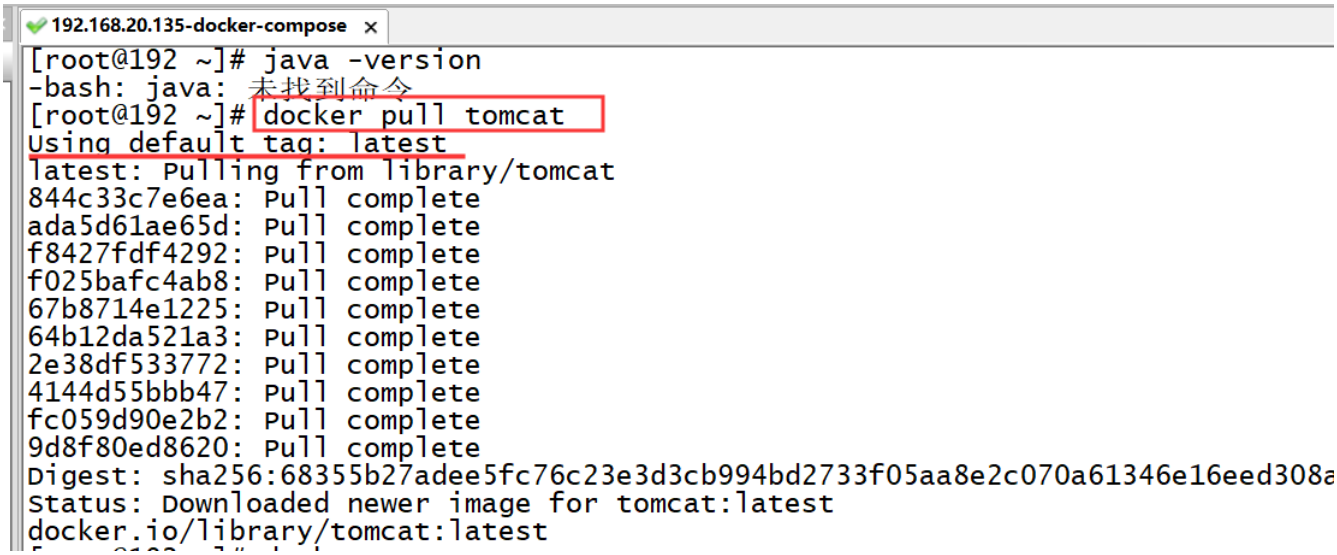
```
具体的选项可以通过 docker pull --help 命令看到，这里我们说一下镜像名称的格式。

- Docker 镜像仓库地址：  地址的格式一般是 <域名/IP>[:端口号]。默认地址是 Docker Hub。

- 仓库名：这里的仓库名是两段式名称，  即 <用户名>/<软件名>。对于 Docker Hub，如果不给出用户名，则默认为 library，也就是官方镜像。
```

我们需要一个tomcat的镜像来作为我们的web服务。通过 docker pull获取镜像

```
docker pull tomcat:版本号   //不写 : 版本号 代表laster版本
```

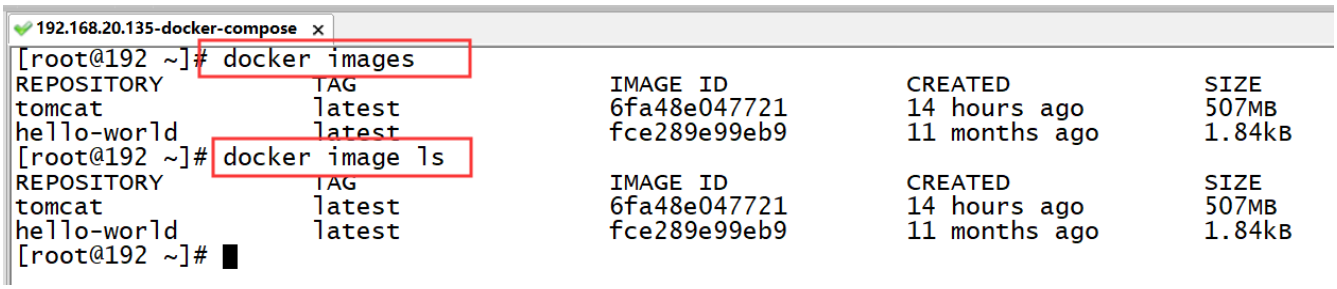


如上图所示。从下载过程中可以看到我们之前提及的分层存储的概念，镜像是由多层存储所构成。下载也是一层层的去下载，并非单一文件。下载过程中给出了每一层的 ID 的前 12 位。并且下载结束后，给出该镜像完整的 sha256 的摘要，以确保下载一致性。
在使用上面命令的时候，你可能会发现，你所看到的层 ID 以及 sha256 的摘要和这里的不一样。这是因为官方镜像是一直在维护的，有任何新的 bug，或者版本更新，都会进行修复再以原来的标签发布，这样可以确保任何使用这个标签的用户可以获得更安全、更稳定的镜像。

Docker 列出镜像

要想列出已经下载下来的镜像，可以使用 `docker image ls` 命令。

```
docker images  // docker image ls
```



列表包含了 **仓库名、标签、镜像 ID、创建时间 以及 所占用的空间**。 其中仓库名、标签在之前的基础概念已经介绍过了。**镜像 ID 则是镜像的唯一标识**，一个镜像可以对应多个标签。因此，如果拥有相同的 ID，因为它们对应的是同一个镜像。

> **镜像体积** 如果仔细观察，会注意到，这里标识的所占用空间和在 Docker Hub 上看到的镜像大小不同。这是因为 Docker Hub 中显示的体积是压缩后的体积。在镜像下载和上传过程中镜像是保持着压缩状态的，因此 Docker Hub 所显示的大小是网络传输中更关心的流量大小。而 docker image ls 显示的是镜像下载到本地后，展开的大小，准确说，是展开后的各层所占空间的总和，因为镜像到本地后，查看空间的时候，更关心的是本地磁盘空间占用的大小。 另外一个需要注意的问题是，docker image ls 列表中的镜像体积总和并非是所有镜像实际硬盘消耗。由于 Docker 镜像是多层存储结构，并且可以继承、复用，因此不同镜像可能会因为使用相同的基础镜像，从而拥有共同的层。由于 Docker 使用 Union FS，相同的层只需要保存一份即可，因此实际镜像硬盘占用空间很可能要比这个列表镜像大小的总和要小的多。

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

> 这个镜像原本是有镜像名和标签的，原来为 tomcat:8.0，随着官方镜像维护，发布了新版本后，重新 docker pull tomcat:8.0 时，tomcat:8.0 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。这类无标签镜像也被称为 虚悬镜像(dangling image) 。 一般来说，虚悬镜像已经失去了存在的价值，是可以随意删除的，可以用下面的命令删除：

```
docker image prune
```

Docker 删除本地镜像

语法:

```
docker image rm [选项] <镜像1> [<镜像2>.....]
```

> 用 ID、镜像名、摘要删除镜像 其中，<镜像> 可以是 镜像短 ID、镜像长 ID、镜像名 或者 镜像摘要。

如果要删除本地的镜像，可以使用 docker image rmi / rm 命令

```
docker image rmi 镜像ID  //常用
```

> 要删除镜像必须确认此镜像目前没有被任何容器使用

Docker其它辅助命令

查看本地镜像的 IMAGE ID

```
docker images -q
```

查看一个镜像的制作历程

```
docker history 镜像名称
```

Docker保存镜像

备份本地仓库的镜像

1 用 save 子命令将本地仓库的镜像保存当前目录下

```
docker save -o tomcat.guo.tar 镜像名称
```

2 将本地目录下的镜像备份文件导入到本地 Docker 仓库

```
# 方式一(不输出详细信息)：
[root@localhost ~]# docker load -i tomcat.guo.tar
```

```
# 方式二（输出详细信息）：
[root@localhost ~]# docker load < tomcat.guo.tar
```

```
docker.io/library/tomcat:latest
[root@192 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
tomcat              latest              6fa48e047721        15 hours ago        507MB
hello-world         latest              fce289e99eb9        11 months ago       1.84kB
[root@192 ~]# docker save -o tomcat.guo.tar tomcat
[root@192 ~]# ll
总用量 507032
-rw-------. 1 root root      1417 2月  11 2019 anaconda-ks.cfg
-rw-------. 1 root root 519196160 12月 14 23:01 tomcat.guo.tar
[root@192 ~]# docker load -i tomcat.guo.tar
Loaded image: tomcat:latest
[root@192 ~]#
```

七、Docker 容器

容器是 Docker核心概念。

> 简单的说，容器是独立运行的一个或一组应用，以及它们的运行环境。
>
> 对应的，虚拟机可以理解为模拟运行的一整套操作系统（提供了运行态环境和其他系统环境）和运行在上面的应用。

容器与镜像的关系类似于面向对象编程中的对象与类。

Docker	面向对象
容器	对象
镜像	类

如下将具体介绍如何来管理一个容器，包括创建、启动和停止等

1 查看容器状态

```
docker ps    //查看运行的容器
docker ps -a //查看所有的容器（包含运行和退出）
docker container ls
docker container ls-a
```

2 Docker 启动容器

启动容器有二种方式，一种是基于镜像新建一个容器并启动，一种是将在终止状态（`stopped`）的容器重新启动

docker run 参数 镜像名称:tag 执行的命令 常用参数:

```
-i 保持和 docker 容器内的交互，启动容器时，运行的命令结束后，容器依然存活，没有退出（默认是会退出，即停止的）
-t 为容器的标准输入虚拟一个tty
-d 后台运行容器
--rm 容器在启动后，执行完成命令或程序后就销毁
--name 给容器起一个自定义名称
-p 宿主机：内部端口
```

```
docker run --rm -d --name tomcat1 -p 8080:8080 tomcat
```

练习1：

运行一个tomcat镜像容器，

```
docker run -i-t  --name tomcat1 tomcat
```

3 查看正在运行的容器

```
docker ps
```

> CONTAINER ID 容器ID IMAGE 容器依赖的镜像 COMMAND 启动容器时执行的命令或程序 CREATED 容器启动时到现在的相隔时间 STATUS 容器状态 PORTS 宿主机到容器的端口映射 注意: 当运行一个容器的时候，没有用参数--name去指定容器名时，Docker会从自己的名称库中随机给这个容器起一个名字

4 查看所有容器

```
docker ps -a
```

5 停止容器

> 通过docker ps 找到容器id

```
docker stop 9be696a0c283  //停止正在运行容器（或Ctrl+c）
docker container stop tomcat1//停止正运行容器(ID或Names)
```

6 启动已终止容器

> 通过docker ps 找到容器id

```
docker start 容器名/容器 ID
```

7 重启已关闭容器

```
docker restart 9be696a0c283//启动容器（根据ID或NAMES）
```

8 关闭和删除容器

```
docker rm 容器ID
```

练习2：

docker方式完整的启动tomcat服务器

```
docker run --rm -d --name tomcat1 -p 8080:8080 tomcat
```

在浏览器上访问 http://10.12.19.13:8080

9 Docker 守护态运行

> 更多的时候，需要让 Docker 在后台运行而不是直接把执行命令的结果输出在当前宿主机下。此时，可以通过添加 -d 参数来实现
>
> 如果不使用 -d 参数运行容器:将会在当前宿主机运行。

如果使用了 -d 参数运行容器。此时容器会在后台运行并不会把输出的结果 (STDOUT) 打印到宿主机上面(输出结果可以用 **docker logs**查看)。

```
查看控制台结果：
docker logs tomcat1//(ID或Names)
```

10 Docker 进入容器

某些时候需要进入容器进行操作，使用 docker exec 命令

> -i -t 参数 docker exec 后边可以跟多个参数，这里主要说明 -i -t 参数。 只用 -i 参数时，由于没有分配伪终端，界面没有我们熟悉的 Linux 命令提示符，但命令执行结果仍然可以返回。 当 -i -t 参数一起使用时，则可以看到我们熟悉的 Linux 命令提示符。

```
docker exec -it 容器ID (Names) bash
```

示例：

进入容器，对默认的tomcat进行页面修改，然后再访问查看效果。

> 注意默认容器内linux包是最小安装。只拥有最基本的命令
>
> exit，不会导致容器的停止

```
docker exec -it tomcat1 bash //进入容器名称叫tomcat1
echo 'qfnj-weixin'>>index.jsp //对容器内的index.jsp进行字符串追加
```

11 在宿主机和容器之间交换文件

在宿主机和容器之间相互COPY文件 cp的用法如下

```
docker cp [OPTIONS] CONTAINER:PATH LOCALPATH    //容器中 复制到 宿主机
docker cp [OPTIONS] LOCALPATH|- CONTAINER:PATH  //宿主机 复制到 容器中
```

```
宿主机复制一个图片到容器中：将png图片复制到了容器指定目录下
docker cp guoweixin.png tomcat2:/usr/local/tomcat/webapps/ROOT
```

```
将容器内的index.jsp 复制出来，修改再复制回去
docker cp tomcat2:/usr/local/tomcat/webapps/ROOT/index.jsp /root
```

八、Docker 查看日志

Docker查看日志

```
docker logs 容器名称/ID
```

```
docker logs -f -t --since="2018-12-1" --tail=10 qfjy_exam
```

> --since : 此参数指定了输出日志开始日期，即只输出指定日期之后的日志。 -f : 查看实时日志 -t : 查看日志产生的日期 -tail=10 : 查看最后的10条日志 qfjy_exam : 容器名称

```
docker logs -f --tail=10 容器名称
```

九、作业案例

根据前面学的知识点内容，将**qfnj** 项目页面，用docker的方式部署到tomcat服务器上。

要求集群的方式部署3台。(8080/8081/8082)

十、Docker 数据卷

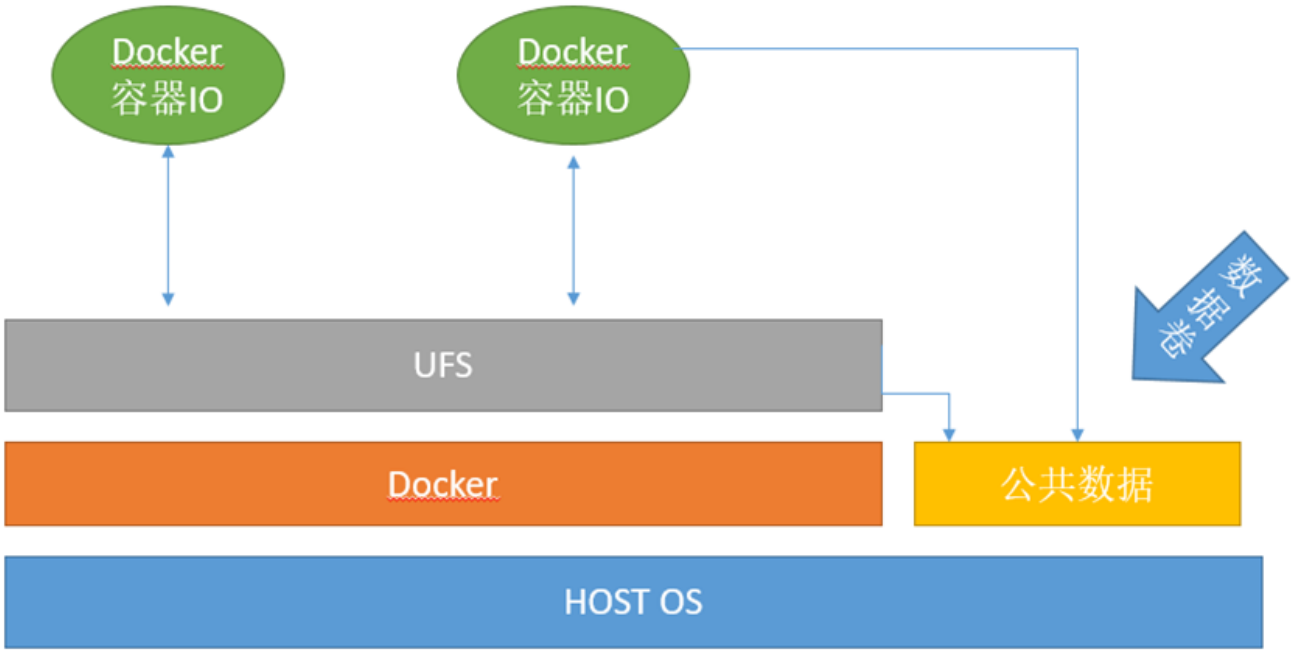
问题：通过镜像创建一个容器。容器一旦被销毁，则容器内的数据将一并被删除。但有些情况下，通过服务器上传的图片出会丢失。容器中的数据不是持久化状态的。

> 那有没有一种独立于容器、提供持久化并能服务于多个容器的东西呢?

什么是数据卷

数据卷： 是一个可供一个或多个容器使用的特殊目录

> 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



为什么需要数据卷?

> 这得从 docker 容器的文件系统说起。出于效率等一系列原因，docker 容器的文件系统在宿主机上存在的方式很复杂，这会带来下面几个问题：
>
> · 不能在宿主机上很方便地访问容器中的文件。
>
> · 无法在多个容器之间共享数据。
>
> · 当容器删除时，容器中产生的数据将丢失。

为了解决这些问题，docker 引入了数据卷(volume) 机制。数据卷是存在于一个或多个容器中的特定文件或文件夹，这个文件或文件夹以独立于 docker 文件系统的形式存在于宿主机中。

数据卷的最大特定是：其生存周期独立于容器的生存周期。

> 使用数据卷的最佳场景
>
> · 在多个容器之间共享数据，多个容器可以同时以只读或者读写的方式挂载同一个数据卷，从而共享数据卷中的数据。
>
> · 当宿主机不能保证一定存在某个目录或一些固定路径的文件时，使用数据卷可以规避这种限制带来的问题。
>
> · 当你想把容器中的数据存储在宿主机之外的地方时，比如远程主机上或云存储上。
>
> · 当你需要把容器数据在不同的宿主机之间备份、恢复或迁移时，数据卷是很好的选择。

数据卷应用

```
#1 创建数据卷
docker volume create 数据卷名称
# 创建数据卷之后，默认会存放到目录：  /var/lib/docker/volume/数据卷名称/_data目录下
```

```
#2 查看数据卷
docker volume inspect 数据卷名称
#3 查看全部数据卷信息
docker volume ls
```

```
#4 删除数据卷
docker volume rm 数据卷名称
```

```
#5 应用数据卷
#5.1 当你映射数据卷时，如果数据卷不存在，Docker会帮你自动创建
docker run  -v 数据卷名称:容器内路径 镜像ID
#5.2 直接指定一个路径作为数据卷的存储位置
docker run -v 路径:容器内部的路径  镜像ID
```

案例1

数据卷方式

创建数据卷，将项目案例放入数据卷中，启动Tomcat。来访问并进行查看(5.1)

> docker volume create vol_qfnj #创建数据卷
>
> docker run -it --name tomcat-8080 -d -p 8080:8080 **-v vol_qfnj:/usr/local/tomcat/webapps/** tomcat #运行容器

路径方式

将项目案例放入指定路径中，-v数据卷。启动Tomcat。来访问并进行查看(5.2)

> docker run -d -p 8080:8080 --name tomcat-8080 -v /opt/volumn_exam:/usr/local/tomcat/webapps tomcat

案例2

qfnj公共的文件夹资源静态资源

创建数据卷

```
docker volume create vol-qfnj   //qfnj公共的文件夹资源静态资源
```

查看所有的 数据卷

```
docker volume ls
```

将 qfnj 文件夹 放入宿主机目录下。

```
cp qfnj  /usr/local/docker/   //复制到 宿主机 /usr/local/docker 下
```

分别根据镜像tomcat 启动两个容器，并挂上数据卷。查看效果：

```
docker run --rm -d --name tomcat-8080 -p 8080:8080
-v /usr/local/docker/qfnj/:/usr/local/tomcat/webapps/qfnj tomcat
```

```
docker run --rm -d --name tomcat-8081 -p 8081:8080 -v
/usr/local/docker/qfnj/:/usr/local/tomcat/webapps/qfnj tomcat
```

> -v /usr/local/docker/qfnj/:/usr/local/tomcat/webapps/qfnj tomcat
>
> -v 数据卷参数。
>
> 将宿主机 /usr/local/docker/qfnj/ 文件内的内容信息
>
> 挂载在容器 /usr/local/tomcat/webapps/qfnj 目录下

Centos防火墙端口

开放8080端口（如下命令只针对Centos7以上）

查看已经开放的端口：

```
firewall-cmd --list-ports
```

开启端口:

```
firewall-cmd --zone=public --add-port=8080/tcp --permanent
```

关闭端口：

```
firewall-cmd --permanent --zone=public --remove-port=8080/tcp
```

开启防火墙：

```
systemctl start firewalld
```

重启防火墙:

```
firewall-cmd --reload #重启
firewall systemctl stop firewalld.service #停止
firewall systemctl disable firewalld.service #禁止firewall开机启动
```