概述

简介

官网：https://www.docker.com/

文档：https://docs.docker.com/docker-for-windows/

命令：https://docs.docker.com/engine/reference/run/

仓库：https://hub.docker.com/

Docker 是一个开源的应用容器引擎。

Docker 的思想来自于集装箱，彼此之间隔离。

Docker 通过隔离机制，可以将服务器利用到极致。

Docker 容器完全使用沙箱机制，相互之间不会有任何接口。

结构

- **镜像**（image）：Docker 镜像好比一个模板，可以用来创建**容器**（container），一个镜像可以创建多个容器。
- **容器**（container）：容器可以理解为一个微型的系统。
- **仓库**（repository）：存放镜像的地方。

底层原理

Docker 是一个 **Client-Server** 结构的系统。

Docker 的守护进程运行在主机上，通过 Socket 从客户端访问。

Docker Server 接受 **Docker-Client** 的指令。

和虚拟机的区别

Docker所使用的**容器化技术**本质上属于**虚拟化技术**。

提到虚拟化技术，最有名的就是**虚拟机技术**。

> 虚拟机原理示意图

它有以下明显的缺点：

- **资源占用多。**启动虚拟机非常占内存，对电脑资源有不小的占用。
- **冗余步骤多。**启动虚拟机后，还需要进行一些步骤才能进入系统，效率比较低。
- **启动很慢。**由于虚拟机是虚拟化一整个系统，其启动时间会比较缓慢，一般都需要几分钟。

> 容器化原理示意图

它不是模拟的完整操作系统，而是基于操作系统封装成了一个个小的运行环境。

> 区别

	传统虚拟机	**Docker**
虚拟内容	硬件 + 完整的操作系统 + 软件	APP + LIB
大小	笨重，通常几个 G	轻便几个 M 或 KB
启动速度	慢，分钟级	快，秒级

> Docker 为什么比 VM 快

Docker 有着比 VM 更少的抽象层。

Docker 主要用的是宿主机的内核，而 VM 需要 **Guest OS**。

新建容器的时候，Docker 不需要像 VM 一样重新加载一个操作系统内核，避免了引导的过程。

安装

Docker 是用 Go 语言开发的。

Go 语言基于 **C** 和 **C++**，所以需要先安装 **C** 和 **C++** 环境。

安装 **C** 环境

```
yum -y install gcc
```

安装 **C++** 环境

```
yum -y install gcc-c++
```

> 3、环境准备

根据 Docker 官网的建议，需要再安装一些环境。

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

> 4、设置镜像仓库

这里官网建议我们安装国外的镜像仓库。

由于对国外的网络限制，国内安装的话会非常慢，并很有可能失败。

推荐国内用户安装阿里云镜像。

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

> 5、更新 yum 软件包索引

这里官网建议我们更新 yum 软件包索引：

```
yum makecache fast
```

> 6、安装 Docker

做完前面的准备工作，就可以开始安装 Docker 了

```
yum -y install docker-ce docker-ce-cli containerd.io
```

这里的 `docker-ce` 指的是 Docker 的社区版。

如果是需要企业版的话就改为 `docker-ee`，不过企业版是收费的，而且功能也略有不同。

官方更推荐社区版，所以这里我们安装 `docker-ce`。

命令

基本命令

帮助

docker —help

Docker 的帮助命令是一个**万能命令**，可以用来查看 Docker 的**所有命令**。

> 语法

```
docker [命令] --help
```

> 查看镜像相关命令

```
[root@qzs ~]# docker images --help
Usage:  docker images [OPTIONS] [REPOSITORY[:TAG]]
List images
Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show image IDs
```

> 查看容器相关命令

```
[root@qzs ~]# docker ps --help
Usage:  docker ps [OPTIONS]
List containers
Options:
  -a, --all             Show all containers (default shows just running)
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print containers using a Go template
  -n, --last int        Show n last created containers (includes all states)
(default -1)
  -l, --latest          Show the latest created container (includes all states)
      --no-trunc        Don't truncate output
  -q, --quiet           Only display container IDs
  -s, --size            Display total file sizes
```

基本信息

docker version

使用 `docker version` 命令可以查看 Docker 的基本信息。

```
[root@qzs ~]# docker version
Client: Docker Engine - Community
 Version:           20.10.11
 API version:       1.41
 Go version:        go1.16.9
 Git commit:        dea9396
 Built:             Thu Nov 18 00:38:53 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true
Server: Docker Engine - Community
 Engine:
  Version:          20.10.11
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.9
  Git commit:       847da18
  Built:            Thu Nov 18 00:37:17 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.12
  GitCommit:        7b11cfaabd73bb80907dd23182b9347b4245eb5d
 runc:
  Version:          1.0.2
  GitCommit:        v1.0.2-0-g52b36a2
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

系统信息

docker info

使用 `docker info` 命令可以查看 Docker 的系统信息。

```
[root@qzs ~]# docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Build with BuildKit (Docker Inc., v0.6.3-docker)
  scan: Docker Scan (Docker Inc., v0.9.0)
Server:
 Containers: 1
  Running: 0
  Paused: 0
  Stopped: 1
 Images: 1
 Server Version: 20.10.11
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7b11cfaabd73bb80907dd23182b9347b4245eb5d
 runc version: v1.0.2-0-g52b36a2
 init version: de40ad0
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 3.10.0-957.21.3.el7.x86_64
 Operating System: CentOS Linux 7 (Core)
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 1.694GiB
 Name: sail
 ID: 2GYU:CVF3:LSAL:J37H:TGME:AOCA:BROB:PHCD:ZPN2:BQCG:GXHB:7Q3L
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Registry Mirrors:
  https://l6b75cm0.mirror.aliyuncs.com/
 Live Restore Enabled: false
```

镜像命令

查看所有镜像

docker images

可以使用 `docker images` 命令查看所有本地主机上的镜像。

> 该命令等价于 `docker image ls`。
>
> 语法

```
docker images [参数] [镜像[:标签]]
```

命令后加上 [镜像[:标签]] 可以对镜像进行过滤。

> 参数

- `-a`：显示所有镜像。
- `-q`：只显示 ID。

> 显示所有镜像

```
docker images
docker image ls
docker images -a
```

> 只显示镜像的 ID

```
docker image -q
```

> 显示所有镜像的 ID

```
docker images -aq
```

> 对镜像进行过滤

```
docker images 镜像名:版本号
```

> 结果分析

- **REPOSITORY**：镜像名（镜像仓库源）。
- **TAG**：镜像的标签。
- **IMAGE ID**：镜像的 ID。
- **CREATED**：镜像的创建时间。
- **SIZE**：镜像的大小。

搜索镜像

docker search

> 语法

```
docker search [参数]
```

> 参数

- `-f` / `--filter`：根据过滤条件搜索。

> 搜索仓库中的镜像，等价于网页搜索。

```
docker search mysql
```

> 搜索出 Stars 大于 3000 的

```
docker search mysql -f=STARS=3000
```

拉取镜像

docker pull

> 语法

```
docker pull [参数] 镜像名[:标签]
```

> 如果不输入标签，默认拉取最新版镜像。
>
> 参数

- `-a`：拉取镜像的所有标签。
- `-q`：抑制详细输出。

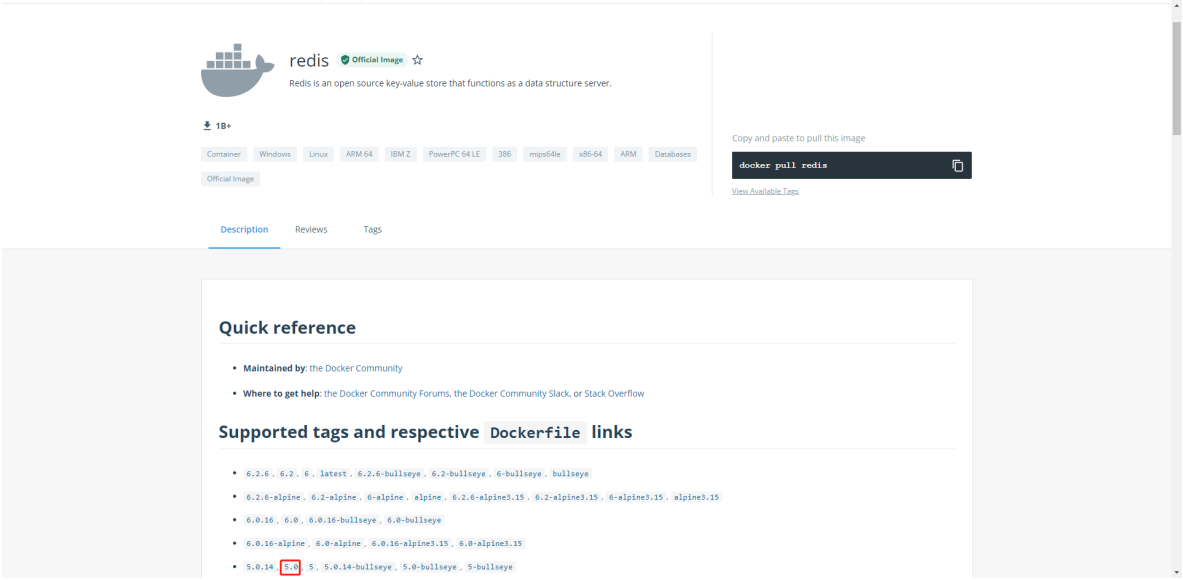
> 默认下载最新版

```
docker pull redis
#输出信息
Using default tag: latest # 默认最新版标签
latest: Pulling from library/redis
e5ae68f74026: Pull complete # 分层下载，docker image的核心：联合文件系统
37c4354629da: Pull complete
b065b1b1fa0f: Pull complete
6954d19bb2e5: Pull complete
6333f8baaf7c: Pull complete
f9772c8a44e7: Pull complete
Digest: sha256:2f502d27c3e9b54295f1c591b3970340d02f8a5824402c8179dcd20d4076b796
#防伪签名
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest # 真实地址，docker pull redis 等价于 docker pull
docker.io/library/redis:latest
```

> 指定版本下载

```
docker pull redis:5.0
#输出信息
5.0: Pulling from library/redis
e5ae68f74026: Already exists # 分层镜像的好处，与之前的版本共用一部分，不用再重复下载
37c4354629da: Already exists
b065b1b1fa0f: Already exists
99ab464ba8bb: Pull complete
eb5bbe3179d2: Pull complete
2067794f93b6: Pull complete
Digest: sha256:310f81701011175dc868e833d73f539282dd18510ca35d6f7b63c4d33ab4f54e
Status: Downloaded newer image for redis:5.0
docker.io/library/redis:5.0
```

版本来自于官网，版本库 https://hub.docker.com/_/redis



> 这里输入的标签必须是官网版本库存在的标签，否则无法拉取镜像。

删除镜像

docker rmi

> 语法

```
docker rmi [参数] 镜像 [镜像...]
```

> 参数

- `-f`：强制删除。

> 查看现存镜像

```
docker images
```

> 删除一个。可以通过名称，也可以指定 ID，`-f` 表示强制删除。

```
docker rmi -f 镜像id
```

> 删除多个。用空格分隔 ID。

```
docker rmi -f 镜像a的id 镜像b的id
```

> 删除所有。先用 `docker images -aq` 查询出所有镜像，再使用 `docker rmi -f` 递归删除。

```
docker rmi -f $(docker images -aq)
```

运行镜像

docker run

> 语法

```
docker run [参数] 镜像名
```

> 参数

- `--name`：指定容器的名称，如果正在运行该名称的容器，会报错。
- `--rm`：用完即删除，通常用来测试。
- `-d`：后台方式运行。
- `-it`：使用交互方式运行，可以进入容器查看内容。
- `-e`：指定运行环境。
- `-p`：随机指定端口。
- `-p`：指定容器的端口，如：

  ```
  -p 8080:8080
  ```

  。还可以有以下写法：
  - `-p ip:主机端口:容器端口`
  - `-p 主机端口:容器端口`
  - `-p 容器端口`

> 运行 centos 镜像

```
docker run -it centos /bin/bash
#输出信息
Unable to find image 'centos:latest' locally # 检索本地镜像，发现没有该镜像，则去仓库中搜索。
latest: Pulling from library/centos # 开始从仓库中拉取
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
[root@81c83ea42dc0 /]# ls # 由于是以交互方式运行，且进入 /bin/bash 中，此时的路径即为 centos 容器中的 /bin/bash
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
```

> 由此可以看出，容器就是一个微型的 Linux 系统。它只保留了最核心的功能和最基本的命令，方便进行操作。

容器命令

查看容器

docker ps

> 语法

```
docker ps [参数]
```

> 参数

- `-a`：查看所有容器（包括正在运行的和已经停止的）。
- `-n`：显示最近创建的容器，设置显示个数。
- `-q`：只显示容器的编号。

> 查看正在运行的容器

```
 docker ps
# 输出内容
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS
PORTS     NAMES
1aaf76d85b9e   centos    "/bin/bash"   About a minute ago   Up About a minute
          intelligent_proskuriakova
```

> 查看所有容器

```
docker ps -a
```

> 显示最近创建的 2 个容器

```
docker ps -a -n=2
```

> 只显示容器的 ID

```
docker ps -aq
```

退出容器

exit

进入容器后，可以使用 `exit` 退出

```
#先运行一个容器
docker run -it centos /bin/bash
#退出容器
exit
```

由此可见，这样退出后容器也会停止。

Ctrl + P + Q

如果想退出后容器不停止，可以使用 `Ctrl + P + Q` 快捷键退出。

```
#运行一个容器
docker run -it centos /bin/bash
#使用ctrl + P + Q 快捷键退出后 在查看运行的容器
docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS
NAMES
1aaf76d85b9e   centos    "/bin/bash"   8 seconds ago    Up 8 seconds
intelligent_proskuriakova
```

删除容器

docker rm

> 语法

```
docker rm [参数] 容器 [容器...]
```

> 参数

- `-f`：强制删除。

> 删除指定容器（不能删除正在运行的容器）

```
#假设 1aaf76d85b9e这个容器正在运行，进行删除
docker rm 1aaf76d85b9e
#提示你需要停止后在删除
Error response from daemon: You cannot remove a running container
1aaf76d85b9ee5002411c1ea390fca05819f19dc400e85127731d37455cb0acc. Stop the
container before attempting removal or force remove
```

> 强制删除指定容器

```
docker rm -f 容器id
```

> 删除所有容器。先使用 `docker ps -aq` 获取所有容器的 ID，再调用 `docker rm -f` 递归删除。

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

> 删除所有容器。使用管道符 `|` 获取 Docker 相关的所有容器 ID 并使用 `docker rm -f` 删除。

```
 docker ps -a -q|xargs docker rm -f
```

启动容器

docker start

> 运行关闭的容器

```
docker start 容器id
```

关闭容器

docker stop

> 关闭运行的容器

```
docker stop 容器id
```

重启容器

docker restart

> 重启关闭的容器

```
docker restart 容器id
```

杀掉容器

docker kill

> 杀掉运行的容器

```
docker kill 容器id
```

常用命令

启动 Docker

systemctl start docker

在服务器关机或者重启后，是需要重新启动 Docker 的。 命令如下：

```
systemctl start docker
```

查看日志

docker logs

> 语法

```
docker logs [参数] 容器
```

> 参数

- `-f`：日志流动输出。
- `-t`：展示时间戳。
- `--tail`：从日志末尾显示的行数。

> 为模拟日志输出效果，我们先编写一段脚本

```
while true;do echo sail;sleep 3;done
```

以上脚本实现的效果为：每隔 3 秒输出字符串 sail。

> 以脚本启动容器

```
#启动容器
docker run -d centos /bin/sh -c "while true;do echo sail;sleep 3;done"
#查看该容器的日志信息
docker logs -f -t --tail 10 容器id
2021-12-10T03:01:28.607288480Z sail
2021-12-10T03:01:31.609334595Z sail
2021-12-10T03:01:34.611361943Z sail
2021-12-10T03:01:37.613461457Z sail
2021-12-10T03:01:40.615619089Z sail
2021-12-10T03:01:43.617595572Z sail
```

可以看到，按此命令会看到容器最后 10 条日志，且每隔 3 秒滚动输出一条日志。

后台启动

docker run -d

> 语法

```
docker run -d 镜像
```

> 后台启动镜像

```
docker run -d centos
```

> 使用 `docker run -d` 启动，也并不能保证容器一定能在后台运行，如果没有前台使用，容器启动后发现自己没有提供服务，会立刻停止。

前面的 `docker run -d centos /bin/sh -c "while true;do echo sail;sleep 3;done"` 命令，由于启动后运行了脚本打印日志，即提供了服务，所以不会停止。

查看容器信息

docker inspect

> 语法

```
docker inspect 容器
```

> 示例

```
docker inspect 容器id
```

> 不管容器是否运行，都可以使用该命令查看。

进入正在运行的容器

容器是一个微型的 Linux 系统，我们通常需要进入容器进行操作。

docker exec

使用 `docker exec` 可以进入容器并开启一个新的终端，可以在里面操作。

> 语法

```
docker exec [参数] 容器 路径
```

> 参数

- `-d`：后台运行。
- `-it`：交互模式进入。

```
docker exec -it 容器id /bin/bash
```

这种进入方式是单独开了一个新进程的方式。

docker attach

使用 `docker attach` 会进入容器正在执行的终端，不会启动新的进程。

> 语法

```
docker attach 容器id
```

这种进入方式没有开启新的进程（`/bin/bash` 是 centos 容器的默认终端）。

从容器内拷贝文件到主机

> 进入容器，创建一个文件

```
#进入容器
docker attach 96ed3fe3e7f1
#进入home目录
cd /home
#创建一个文件
touch test.java
#查看文件
ls
#test.java
#退出容器
exit
```

> 退出容器后，不管容器是否启动，都可以复制容器中的文件到主机上

```
#进入主机上的home目录
cd /home
#复制
docker cp 容器id:/home/test.java /home
#查看home目录下的文件
ls
admin  f2  f3  sail  test.java
```

> 这种方式是一个手动过程，很不方便，推荐使用**数据卷技术**，可以实现自动同步主机和容器的目录。

查看Docker内存占用

docker stats

> 语法

```
docker stats [参数] [容器...]
```

> 参数

- `-a`：查看所有容器的内存占用（默认只展示运行的容器）。

```
docker stats -a
```

镜像原理

联合文件系统

联合文件系统（UnionFS）是 Docker 的核心，也是 Docker 得以极致精简的保证。

以拉取 redis 镜像为例

> 先拉取最新版镜像

```
docker pull redis
Using default tag: latest # 默认最新版标签
latest: Pulling from library/redis
e5ae68f74026: Pull complete # 分层下载，docker image的核心：联合文件系统
37c4354629da: Pull complete
b065b1b1fa0f: Pull complete
6954d19bb2e5: Pull complete
6333f8baaf7c: Pull complete
f9772c8a44e7: Pull complete
Digest: sha256:2f502d27c3e9b54295f1c591b3970340d02f8a5824402c8179dcd20d4076b796
#防伪签名
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest # 真实地址，docker pull redis 等价于 docker pull
docker.io/library/redis:latest
```

> 再拉取指定版镜像

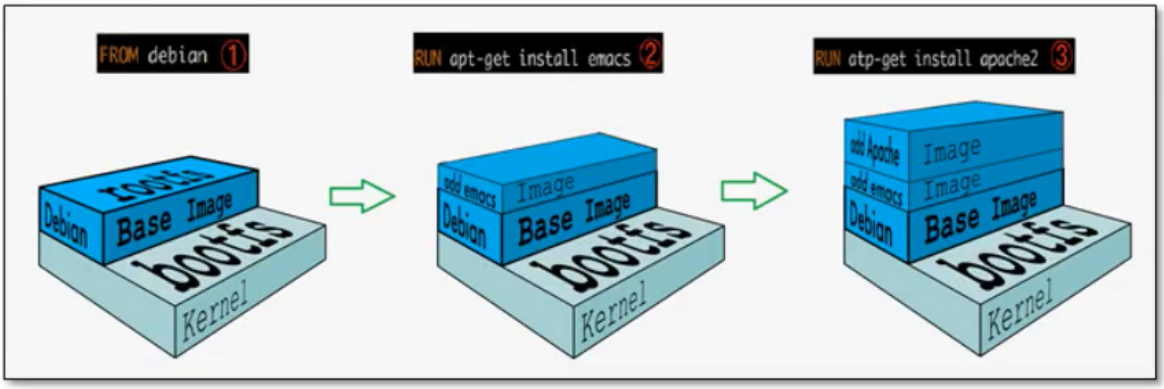
```
docker pull redis:5.0
5.0: Pulling from library/redis
e5ae68f74026: Already exists # 分层镜像的好处，与之前的版本共用一部分，不用再重复下载
37c4354629da: Already exists
b065b1b1fa0f: Already exists
99ab464ba8bb: Pull complete
eb5bbe3179d2: Pull complete
2067794f93b6: Pull complete
Digest: sha256:310f81701011175dc868e833d73f539282dd18510ca35d6f7b63c4d33ab4f54e
Status: Downloaded newer image for redis:5.0
docker.io/library/redis:5.0
```

由此可见，redis 镜像一共 6 层，由于之前拉取了默认的最新版 **redis** 镜像，再拉取 **redis:5.0** 时，有 3 层是可以复用的，所以只下载了不能复用的 3 层。

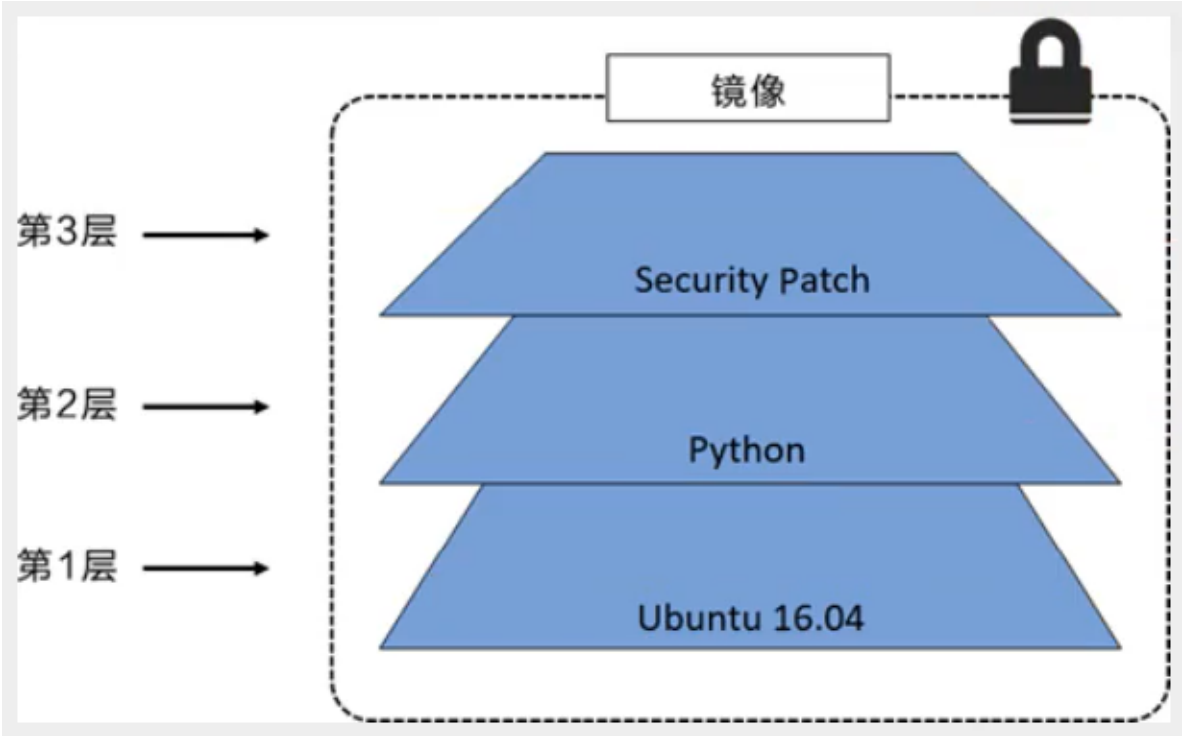
> 这样既能提高下载速度，也能极大节省磁盘占用和资源消耗。

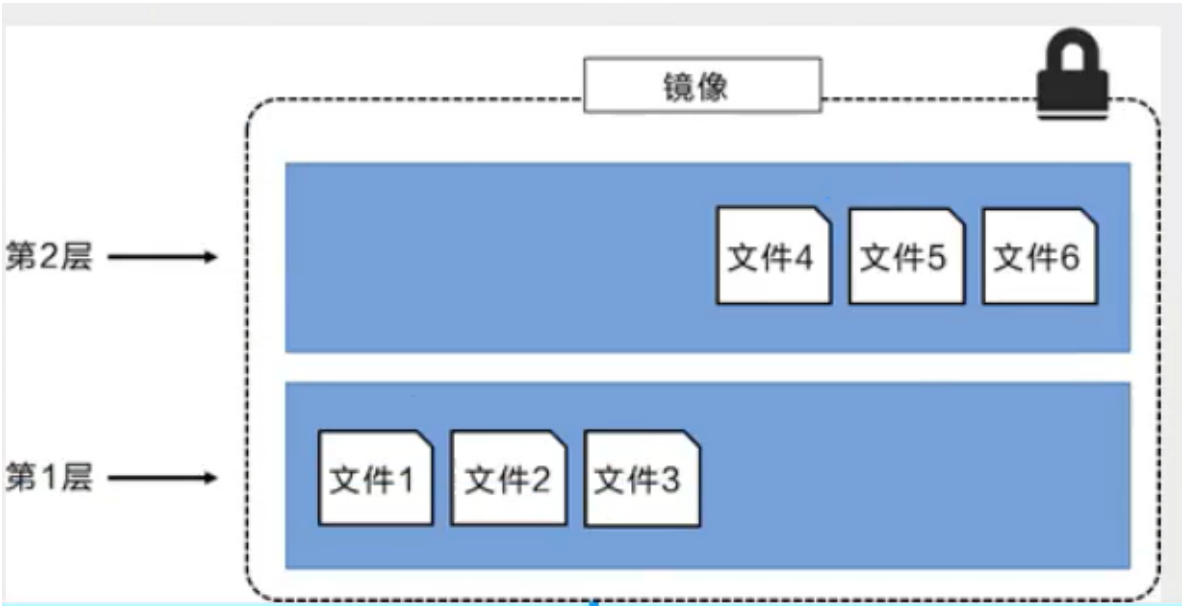
分层镜像

Docker 使用联合文件系统对镜像做了分层，如下图所示：



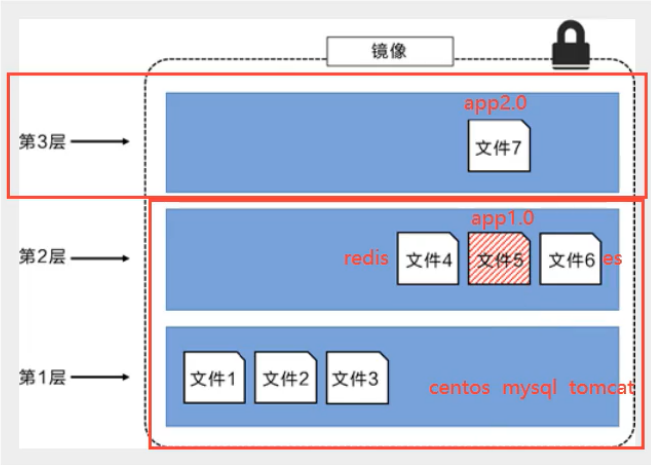
- **bootfs**（boot file system）：启动文件系统。
- **rootfs**：root file system：基础文件系统。





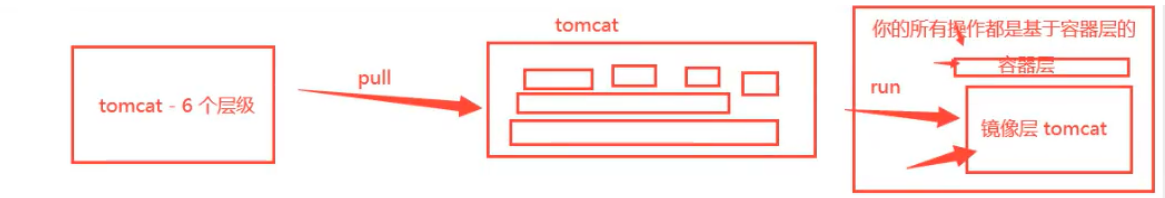
上图中的镜像层跟之前图中的略有区别，主要目的是便于展示文件。

下图中展示了一个稍微复杂的三层镜像，在外部看来整个镜像只有 6 个文件，这是因为最上层中的文件 7 是文件 5 的一个更新版本。



Docker 镜像都是**只读**的，当容器启动时，一个新的可写层被加到镜像的顶部。

这一层就是我们通常说的**容器层**，容器层之下的都叫**镜像层**。



自定义镜像

我们日常拉取的镜像是由别人制作再提交到仓库的。

我们自己也是可以制作镜像并提交的，使用 `docker commit` 命令。

提交镜像

docker commit

> 语法

```
docker commit [参数] 容器 [仓库[:标签]]
```

> 参数

- `-a`：作者信息。一般为 **作者名字<邮箱>**。
- `-c`：将 **Dockerfile** 指令应用于创建的映像。
- `-m`：注释信息。
- `-p`：提交期间暂停容器（默认）。

> 提交自定义镜像

> 将这个容器创建为一个自定义的镜像并提交到仓库中。

```
docker commit -a="提交人信息" -m="注释信息" 要提交的容器 镜像名:版本号
```

数据卷

由来

Docker 是将应用和环境打包成一个镜像。

这样，数据就不应该保存在容器中，否则容器删除，数据就会丢失，有着非常大的风险。

为此，容器和主机之间需要有一个数据共享技术，使得在 Docker 容器中产生的数据能够同步到本地。

这就是**数据卷**技术。其本质上是一个**目录挂载**，将容器内的目录挂载到主机上。

使用

命令方式

> 语法

```
docker run -v 主机目录:容器目录
```

> 查看主机 `/home` 目录。

```
[root@qzs ~] ls /home
admin  f2  f3  sail  test.java
```

> 以交互模式启动 centos 镜像。

```
[root@qzs ~]# docker run -it -v /home/ceshi:/home centos /bin/bash
[root@ec95646b1a4c /]#
```

> 新开一个窗口查看容器详情。

Mounts 下的 **Source** 即为设置的主机目录、**Destination** 即为设置的容器目录，他们已经绑定在了一起。

> 在主机中查看 `/home`。

```
[root@qzs /]# cd /home
[root@qzs home]# ls
admin  ceshi  f2  f3  sail  test.java
```

主机上的 `/home` 下已经有了 `ceshi` 目录。说明容器一经启动，就会在主机生成对应的挂载目录。

> 在容器中的 `/home` 下新建一个文件。

```
[root@ec95646b1a4c /]# cd /home
[root@ec95646b1a4c home]# ls
[root@ec95646b1a4c home]# touch test.java
[root@ec95646b1a4c home]# ls
test.java
```

> 查看主机的 `ceshi` 目录。

```
[root@qzs home]# cd ceshi
[root@qzs ceshi]# ls
test.java
```

此时主机中的 `ceshi` 目录下也有了这个文件。

> 关闭容器。

```
[root@ec95646b1a4c home]# exit
exit
```

> 修改主机中 `/home/ceshi/test.java` 文件的内容。

```
[root@qzs ceshi]# vim test.java
# 此处编辑文件过程省略
[root@qzs ceshi]# cat test.java
hello sail
```

> 重启容器。

```
[root@qzs ~]# docker start ec95646b1a4cec95646b1a4c
[root@qzs ~]# docker ps
CONTAINER ID   IMAGE          COMMAND             CREATED          STATUS
  PORTS           NAME
Sec95646b1a4c   centos         "/bin/bash"          19 minutes ago   Up 5 seconds
                          charming_cartwright
```

> 查看 `/home` 下的文件。

```
[root@qzs ~]# docker exec -it ec95646b1a4c /bin/bash
[root@ec95646b1a4c /]# cd /home
[root@ec95646b1a4c home]# ls
test.java
[root@ec95646b1a4c home]# cat test.java
hello sail
```

此时容器中的文件也更改了。

> 由此可见，数据卷技术实现的是双向同步。

权限设置

在使用命令方式设置卷时，还可以指定权限，以此保证数据安全。

> 参数

- `ro`(readonly)：只读。
- `rw`(readwrite)：可读可写。

> 以数据卷只读权限启动镜像。

```
[root@qzs mysql]# docker run -it -v /home/qzs:/home:ro centos /bin/bash
```

> 新建文件测试。

```
[root@02ef70c94920 home]# touch test.java
touch: cannot touch 'test.java': Read-only file system
```

容器内部该目录是没有写入权限的。

> 以数据卷可读可写的权限启动镜像。

```
[root@qzs mysql]# docker run -it -v /home/qzs:/home:rw centos /bin/bash
[root@48678e08f868 /]# cd /home
[root@48678e08f868 home]# touch test.java
[root@48678e08f868 home]# ls
apache-tomcat-9.0.55.tar.gz  jdk-8u301-linux-x64.rpm  test.java
```

> 新建文件测试。

```
[root@qzs mysql]# docker run -it -v /home/sail:/home:rw centos /bin/bash
[root@48678e08f868 /]# cd /home
[root@48678e08f868 home]# touch test.java
[root@48678e08f868 home]# ls
apache-tomcat-9.0.55.tar.gz  jdk-8u301-linux-x64.rpm  test.java
```

容器内部该目录写入是没有问题的。

> 前面我们没有指定权限也可以写入，由此可见，数据卷默认是具有读写权限的。

具名挂载

> 启动镜像时只定义主机卷名称，不指定挂载目录。

```
[root@qzs mysql]# docker run -it -v my-centos:/home centos /bin/bash
[root@3cf74e9e6973 /]#
```

> 查看目前挂载的卷。

```
# 使用 Ctrl + P + Q 不退出容器的情况下回到主机目录。
[root@qzs mysql]# docker volume ls
DRIVER    VOLUME NAME
local     my-centos
```

> 查看卷的详情。

```
[root@qzs mysql]# docker volume inspect my-centos
[
    {
        "CreatedAt": "2021-12-20T16:55:35+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/my-centos/_data",
        "Name": "my-centos",
        "Options": null,
        "Scope": "local"
    }
]
```

卷挂载在 `/var/lib/docker/volumes/卷名/_data` 目录下。

在没有指定主机挂载目录的情况下，会默认挂载到该目录。

> 由于指定了卷名，所以这种方式称为具名挂载。

匿名挂载

> 启动镜像时只指定容器目录。

```
[root@qzs mysql]# docker run -it -v /home centos /bin/bash
```

> 查看目前挂载的卷。

```
[root@qzs mysql]# docker volume ls
DRIVER    VOLUME NAME
local     159830cf55550c9a39e845c1d96aa04cc762005bc0c64d15d5066834b47df940
```

> 查看卷的详情。

```
[root@qzs mysql]# docker volume inspect
159830cf55550c9a39e845c1d96aa04cc762005bc0c64d15d5066834b47df940
[
    {
        "CreatedAt": "2021-12-20T17:05:23+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint":
"/var/lib/docker/volumes/159830cf55550c9a39e845c1d96aa04cc762005bc0c64d15d506683
4b47df940/_data",
        "Name":
"159830cf55550c9a39e845c1d96aa04cc762005bc0c64d15d5066834b47df940",
        "Options": null,
        "Scope": "local"
    }
]
```

卷也是挂载在 `/var/lib/docker/volumes/xxx/_data` 目录下。

在没有指定主机挂载目录的情况下，会默认挂载到该目录。

> 由于没有指定卷名，所以这种方式称为匿名挂载。

只有指定主机目录的情况下会挂载到指定目录，否则都会挂载到默认目录。

实战

mysql数据同步

数据库中的数据极为重要，必须同步到主机，否则将会有非常大的数据丢失风险。

这里以 mysql 镜像为例演示数据同步的过程。

> 启动 mysql 镜像。

```

```

```
[root@qzs ~]# docker run -d -p 3310:3306 -v /home/mysql/conf:/etc/mysql/conf.d -
v /home/mysql/data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 --name mysql01
mysql:5.7
Unable to find image 'mysql:5.7' locally
5.7: Pulling from library/mysql
ffbb094f4f9e: Pull complete
df186527fc46: Pull complete
fa362a6aa7bd: Pull complete
5af7cb1a200e: Pull complete
949da226cc6d: Pull complete
bce007079ee9: Pull complete
eab9f076e5a3: Pull complete
c7b24c3f27af: Pull complete
6fc26ff6705a: Pull complete
bec5cdb5e7f7: Pull complete
6c1cb25f7525: Pull complete
Digest: sha256:d1cc87a3bd5dc07defc837bc9084f748a130606ff41923f46dec1986e0dc828d
Status: Downloaded newer image for mysql:5.7
a016e564d977550e475474556cfd033fb1c731002381bc9f9544c63fccb7f60c
```

其中 `-e` 为环境配置。安装启动 mysql 需要配置密码。

> 使用`docker inspect`查看挂载情况。

已经生成了两个目录的挂载。

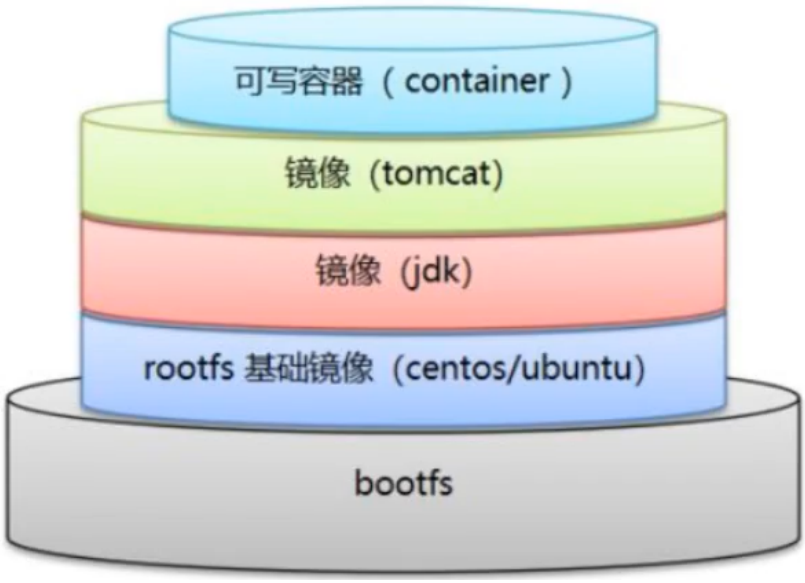
> 查看主机同步的目录。

```
[root@qzs ~]# cd /home
[root@qzs home]# ls
admin  ceshi  f2  f3  mysql  sail  test.java
[root@qzs home]# cd mysql
[root@qzs mysql]# ls
conf  data
```

主机已经同步了容器挂载的目录。

DockerFile

简介



Dockerfile 是用来构建 Docker 镜像的文件，可以理解为**命令参数脚本。**

Dockerfile 是面向开发的，想要打包项目，就要编写 Dockerfile 文件。

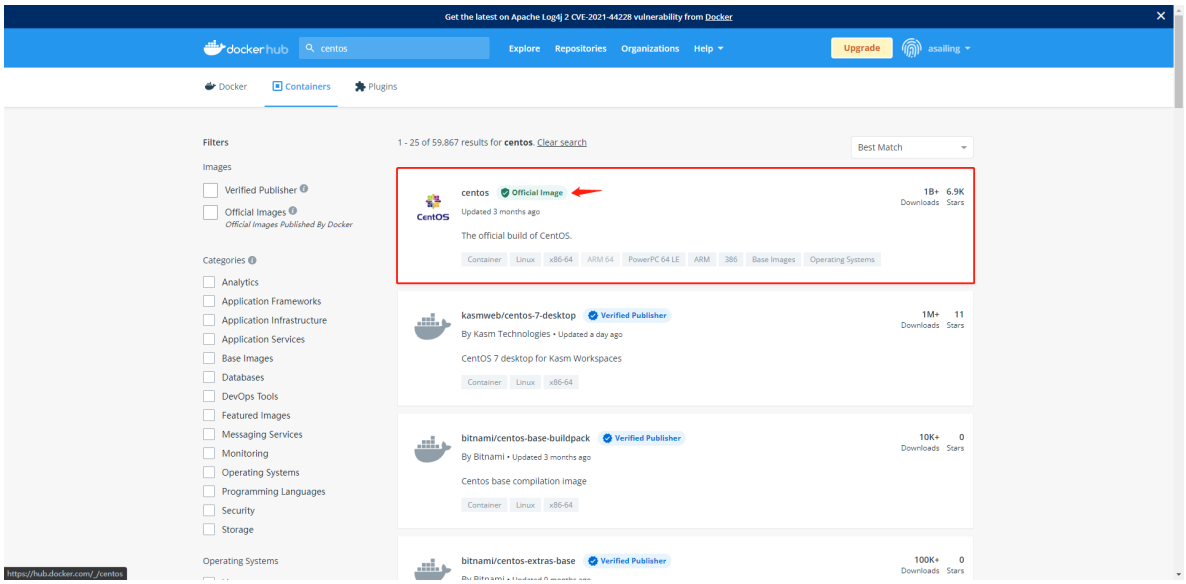
> 由于 Docker 的流行，Docker 镜像逐渐替代 **jar** 或者 **war** 成为企业的交付标准。

官方 Dockerfile

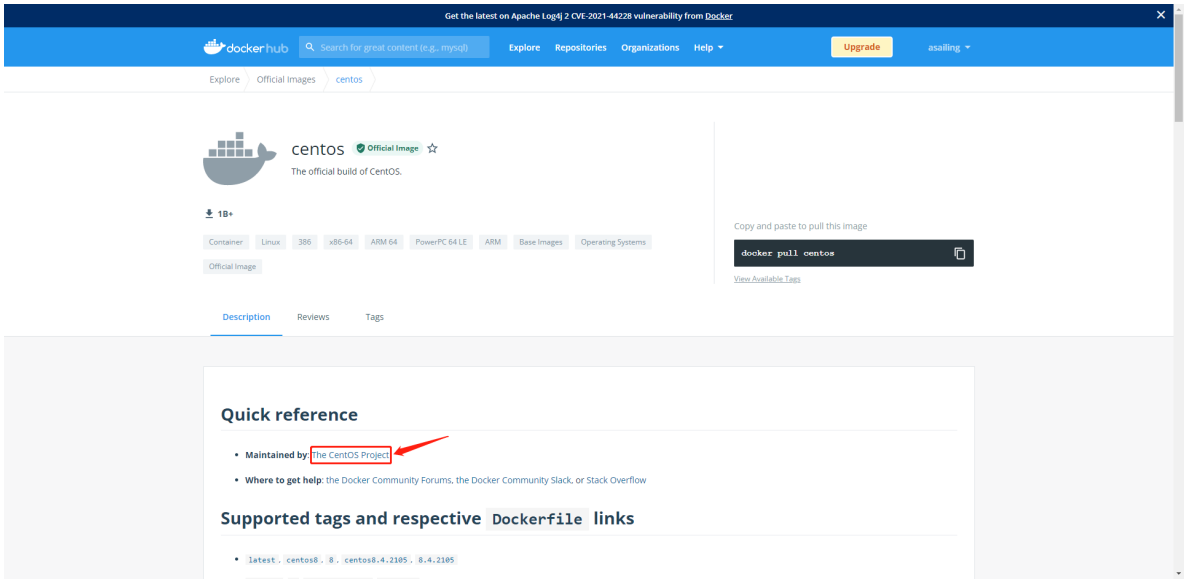
> 首先看一下官方的 Dockerfile。

这里以 centos 的镜像为例。

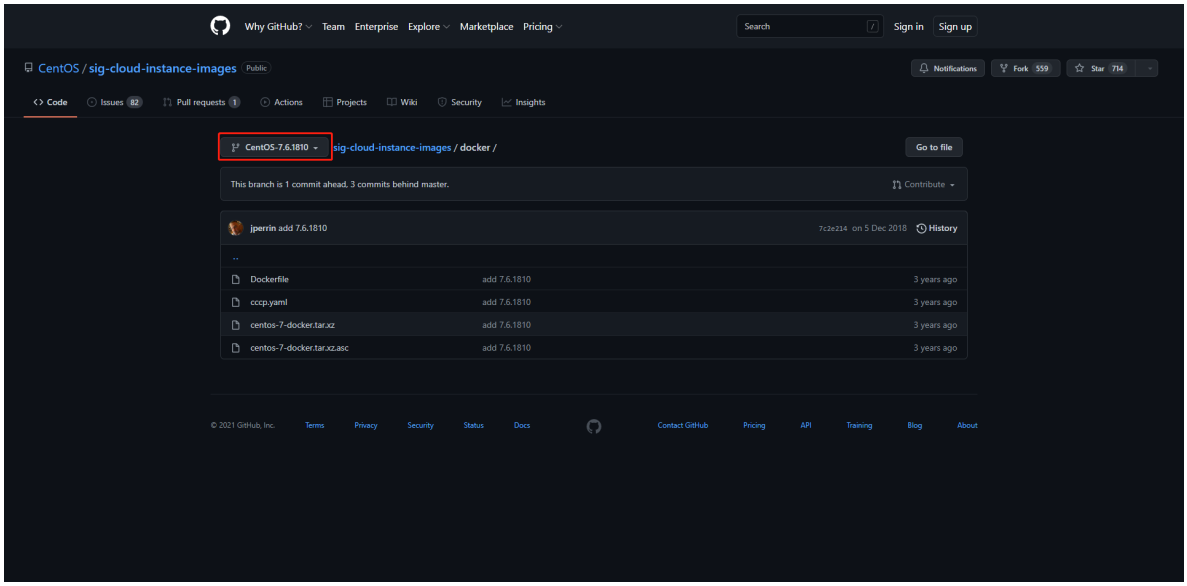
> 在 Docker Hub 搜索 centos 镜像，选择官方镜像。



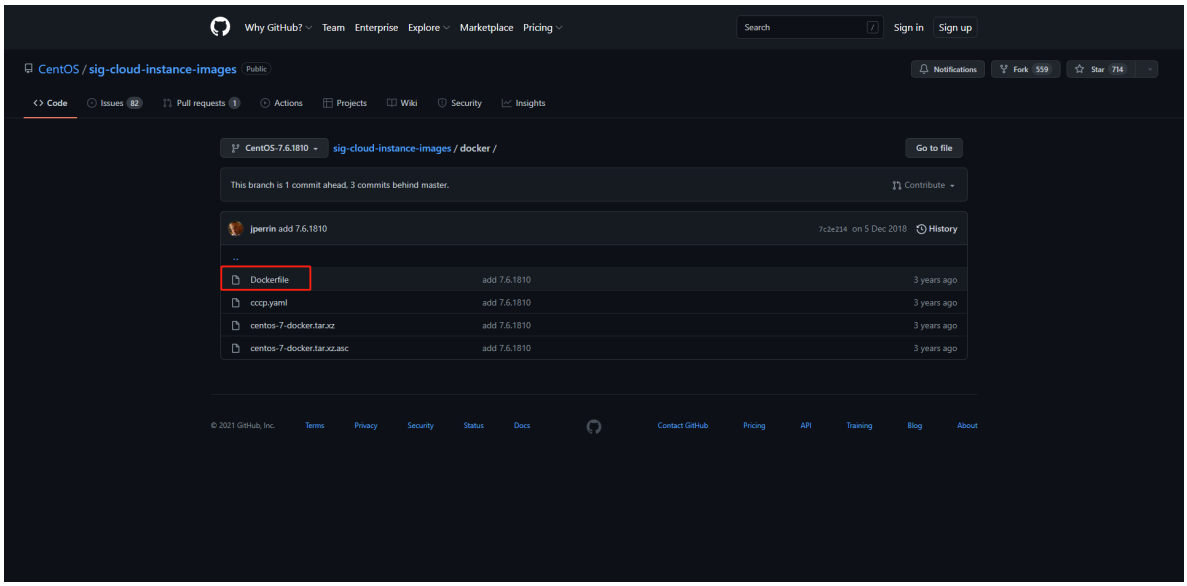
> 点击 **The CentOS Project** 。

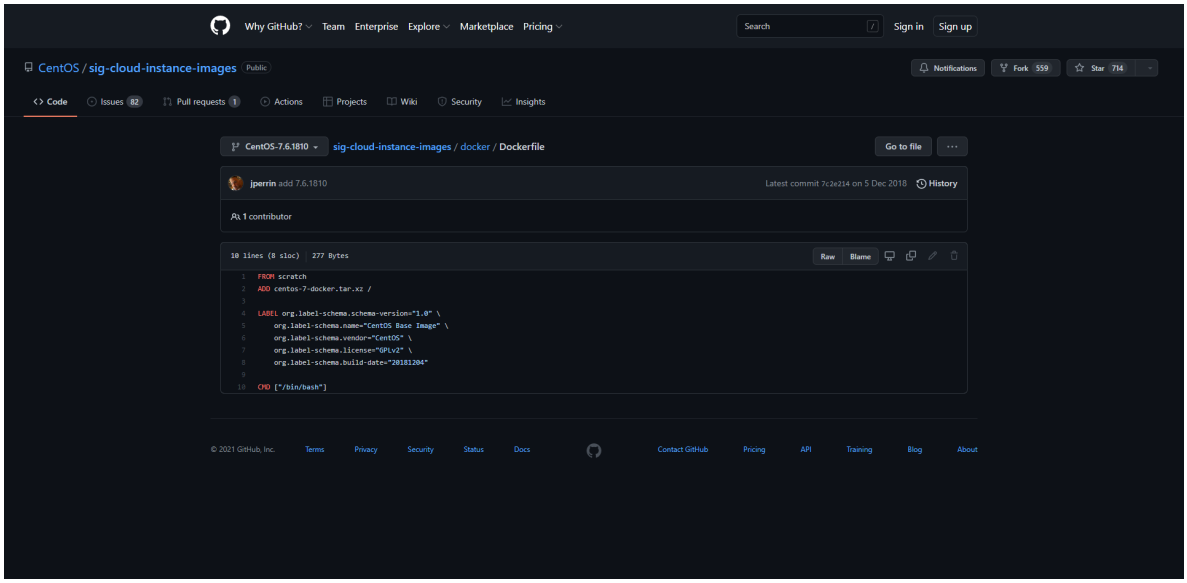


> 选择版本分支。这里以 **CentOS-7.6.1810** 为例。



> 查看 Dockerfile。





> 可以看出官方的 Dockerfile 是很简洁的，这就是 Docker 极致精简的原因。

不过，官方镜像都是基础包，很多功能是没有的，比如 centos 的官方镜像是没有 `vim` 命令的。

像这种基础命令比较常用，如果想要使用可以在容器内安装，但这样很不方便，因为每个新启动的容器都要安装。

我们通常会选择自己搭建镜像，这就要用到 Dockerfile 了。

命令

以上面的 centos 官方镜像的 Dockerfile 为例。

```
FROM scratch
ADD centos-7-docker.tar.xz /
LABEL org.label-schema.schema-version="1.0" \
    org.label-schema.name="CentOS Base Image" \
    org.label-schema.vendor="CentOS" \
    org.label-schema.license="GPLv2" \
    org.label-schema.build-date="20181204"
CMD ["/bin/bash"]
```

Docker Hub 中 99% 的镜像都是从 `FROM scratch` 开始的。

> 规则

- 每个指令都必须是大写字母。
- 按照从上到下顺序执行。
- # 表示注释。
- 每一条指令都会创建一个新的镜像层。

> 解释

- `FROM`：基础镜像，比如 centos。
- `MAINTAINER`：镜像是谁写的。建议以此格式：`姓名<邮箱>`。
- `RUN`：镜像构建时需要运行的命令。
- `ADD`：添加，比如添加一个 tomcat 压缩包。
- `WORKDIR`：镜像的工作目录。
- `VOLUME`：挂载的目录。
- `EXPOSE`：指定暴露端口，跟 -p 一个道理。

- `RUN`：最终要运行的。
- `CMD`：指定这个容器启动的时候要运行的命令，只有最后一个会生效，而且可被替代。
- `ENTRYPOINT`：指定这个容器启动的时候要运行的命令，可以追加命令。
- `ONBUILD`：当构建一个被继承Dockerfile 这个时候运行ONBUILD指定，触发指令。
- `COPY`：将文件拷贝到镜像中。
- `ENV`：构建的时候设置环境变量。

构建镜像

docker build

Dockerfile 编写好后，需要使用 `docker build` 命令运行。

> 语法

```
docker build [参数] 路径 | 网络地址 | -
```

> 参数

- `-f`：指定要使用的Dockerfile路径。
- `-t`：镜像的名字及标签，通常 **name:tag** 或者 **name** 格式；可以在一次构建中为一个镜像设置多个标签。
- `-m`：设置内存最大值。

> Docker 守护进程执行 Dockerfile 中的指令前，首先会对 Dockerfile 进行语法检查，有语法错误时会返回报错信息。

```
Error response from daemon: Unknown instruction: RUNCMD
```

查看构建记录

docker history

> 语法

```
docker history 镜像
```

CMD 与 ENTRYPOINT 区别

CMD 命令演示

> 编写 Dockerfile

```
[root@qzs dockerfile]# vim Dockerfile-cmd-test
[root@qzs dockerfile]# cat Dockerfile-cmd-test
FROM centos
CMD ["ls","-a"]
```

> 构建镜像

```
[root@qzs dockerfile]# docker build -f Dockerfile-cmd-test -t cmdtest .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM centos
 ---> 5d0da3dc9764
Step 2/2 : CMD ["ls","-a"]
 ---> Running in 0a743e929fff
Removing intermediate container 0a743e929fff
 ---> 1683c0790d49
Successfully built 1683c0790d49
Successfully tagged cmdtest:latest
```

```
[root@qzs dockerfile]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
cmdtest             latest    1683c0790d49   13 minutes ago   231MB
```

> 运行镜像

```
[root@qzs dockerfile]# docker run cmdtest
.
..
.dockerenv
bin
dev
etc
home
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

此时 Dockerfile 中编写的命令生效了。

> 追加 `-l` 命令

```
[root@qzs dockerfile]# docker run cmdtest -l
docker: Error response from daemon: OCI runtime create failed:
container_linux.go:380: starting container process caused: exec: "-l":
executable file not found in $PATH: unknown.
ERRO[0000] error waiting for container: context canceled
```

没有达到预期的 `ls -al` 命令。

> `CMD` 是**替换**的方式， `-l` 不是命令，所以报错。

ENTRYPOINT 命令演示

> 编写 Dockerfile

```
[root@qzs dockerfile]# vim Dockerfile-entrypoint-test
[root@qzs dockerfile]# cat Dockerfile-entrypoint-test
FROM centos
ENTRYPOINT ["ls","-a"]
```

> 构建镜像

```
[root@qzs dockerfile]# docker build -f Dockerfile-entrypoint-test -t entrypoint-test .
Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM centos
 ---> 5d0da3dc9764
Step 2/2 : ENTRYPOINT ["ls","-a"]
 ---> Running in a02d55ae0a00
Removing intermediate container a02d55ae0a00
 ---> 795973a0ed43
Successfully built 795973a0ed43
Successfully tagged entrypoint-test:latest
```

> 运行镜像

```
[root@qzs dockerfile]# docker run entrypoint-test
.
..
.dockerenv
bin
dev
etc
home
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

此时 Dockerfile 中编写的命令也生效了。

> 追加 `-l` 命令

```
[root@qzs dockerfile]# docker run entrypoint-test -l
total 56
```

```
drwxr-xr-x   1 root root 4096 Dec 23 06:46 .
drwxr-xr-x   1 root root 4096 Dec 23 06:46 ..
-rwxr-xr-x   1 root root    0 Dec 23 06:46 .dockerenv
lrwxrwxrwx   1 root root    7 Nov  3  2020 bin -> usr/bin
drwxr-xr-x   5 root root  340 Dec 23 06:46 dev
drwxr-xr-x   1 root root 4096 Dec 23 06:46 etc
drwxr-xr-x   2 root root 4096 Nov  3  2020 home
lrwxrwxrwx   1 root root    7 Nov  3  2020 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Nov  3  2020 lib64 -> usr/lib64
drwx------   2 root root 4096 Sep 15 14:17 lost+found
drwxr-xr-x   2 root root 4096 Nov  3  2020 media
drwxr-xr-x   2 root root 4096 Nov  3  2020 mnt
drwxr-xr-x   2 root root 4096 Nov  3  2020 opt
dr-xr-xr-x 106 root root    0 Dec 23 06:46 proc
dr-xr-x---   2 root root 4096 Sep 15 14:17 root
drwxr-xr-x  11 root root 4096 Sep 15 14:17 run
lrwxrwxrwx   1 root root    8 Nov  3  2020 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Nov  3  2020 srv
dr-xr-xr-x  13 root root    0 Oct 31 15:00 sys
drwxrwxrwt   7 root root 4096 Sep 15 14:17 tmp
drwxr-xr-x  12 root root 4096 Sep 15 14:17 usr
drwxr-xr-x  20 root root 4096 Sep 15 14:17 var
```

运行了预期的 `ls -al` 命令。

> `ENTRYPOINT` 是**追加**的方式。

Docker 中许多命令都十分相似，我们需要了解他们的区别，最好的方式就是这样对比测试。

实战

创建包含vim命令的centos镜像

> 编写 Dockerfile

```
[root@qzs dockerfile]# vim Dockerfile-centos-test
[root@qzs dockerfile]# cat Dockerfile-centos-test
FROM centos
MAINTAINER qzs<283408112@qq.com>
ENV MYPATH /usr/local
WORKDIR $MYPATH
RUN yum -y install vim
RUN yum -y install net-tools
EXPOSE 81
CMD echo $MYPATH
CMD echo "---end---"
CMD ["/bin/bash"]
```

> 构建镜像

```
[root@qzs dockerfile]# docker build -f Dockerfile-centos-test -t centos-test .
Sending build context to Docker daemon  5.632kB
Step 1/10 : FROM centos
 ---> 5d0da3dc9764
Step 2/10 : MAINTAINER sail<yifansailing@163.com>
 ---> Running in 8b7340768878
```

```
Removing intermediate container 8b7340768878
 ---> 9616888f3b10
Step 3/10 : ENV MYPATH /usr/local
 ---> Running in 2c73446a56ff
Removing intermediate container 2c73446a56ff
 ---> be89377d4c2c
Step 4/10 : WORKDIR $MYPATH
 ---> Running in db113c4f7cb2
Removing intermediate container db113c4f7cb2
 ---> fb41ece5d944
Step 5/10 : RUN yum -y install vim
 ---> Running in eccee60c0389
CentOS Linux 8 - AppStream                      12 MB/s | 8.4 MB     00:00
CentOS Linux 8 - BaseOS                        1.7 MB/s | 3.6 MB     00:02
CentOS Linux 8 - Extras                         17 kB/s |  10 kB     00:00
Last metadata expiration check: 0:00:01 ago on Sat Dec 25 05:09:40 2021.
Dependencies resolved.
================================================================================
 Package             Arch        Version                   Repository      Size
================================================================================
Installing:
 vim-enhanced        x86_64      2:8.0.1763-16.el8         appstream      1.4 M
Installing dependencies:
 gpm-libs            x86_64      1.20.7-17.el8             appstream       39 k
 vim-common          x86_64      2:8.0.1763-16.el8         appstream      6.3 M
 vim-filesystem      noarch      2:8.0.1763-16.el8         appstream       49 k
 which               x86_64      2.21-16.el8               baseos          49 k
Transaction Summary
================================================================================
Install  5 Packages
Total download size: 7.8 M
Installed size: 30 M
Downloading Packages:
(1/5): gpm-libs-1.20.7-17.el8.x86_64.rpm        582 kB/s |  39 kB     00:00
(2/5): vim-filesystem-8.0.1763-16.el8.noarch.rp 1.2 MB/s |  49 kB     00:00
(3/5): vim-common-8.0.1763-16.el8.x86_64.rpm     40 MB/s | 6.3 MB     00:00
(4/5): vim-enhanced-8.0.1763-16.el8.x86_64.rpm  7.1 MB/s | 1.4 MB     00:00
(5/5): which-2.21-16.el8.x86_64.rpm             252 kB/s |  49 kB     00:00
--------------------------------------------------------------------------------
Total                                           6.4 MB/s | 7.8 MB     00:01
warning: /var/cache/dnf/appstream-02e86d1c976ab532/packages/gpm-libs-1.20.7-
17.el8.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID 8483c65d: NOKEY
CentOS Linux 8 - AppStream                      1.6 MB/s | 1.6 kB     00:00
Importing GPG key 0x8483C65D:
 Userid     : "CentOS (CentOS Official Signing Key) <security@centos.org>"
 Fingerprint: 99DB 70FA E1D7 CE22 7FB6 4882 05B5 55B3 8483 C65D
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : which-2.21-16.el8.x86_64                               1/5
  Installing       : vim-filesystem-2:8.0.1763-16.el8.noarch                2/5
  Installing       : vim-common-2:8.0.1763-16.el8.x86_64                    3/5
  Installing       : gpm-libs-1.20.7-17.el8.x86_64                          4/5
```

```
  Running scriptlet: gpm-libs-1.20.7-17.el8.x86_64                          4/5
  Installing       : vim-enhanced-2:8.0.1763-16.el8.x86_64                  5/5
  Running scriptlet: vim-enhanced-2:8.0.1763-16.el8.x86_64                  5/5
  Running scriptlet: vim-common-2:8.0.1763-16.el8.x86_64                    5/5
  Verifying        : gpm-libs-1.20.7-17.el8.x86_64                          1/5
  Verifying        : vim-common-2:8.0.1763-16.el8.x86_64                    2/5
  Verifying        : vim-enhanced-2:8.0.1763-16.el8.x86_64                  3/5
  Verifying        : vim-filesystem-2:8.0.1763-16.el8.noarch                4/5
  Verifying        : which-2.21-16.el8.x86_64                               5/5
Installed:
  gpm-libs-1.20.7-17.el8.x86_64         vim-common-2:8.0.1763-16.el8.x86_64
  vim-enhanced-2:8.0.1763-16.el8.x86_64 vim-filesystem-2:8.0.1763-16.el8.noarch
  which-2.21-16.el8.x86_64
Complete!
Removing intermediate container eccee60c0389
 ---> 9f54f48660ac
Step 6/10 : RUN yum -y install net-tools
 ---> Running in 6caa7361b001
Last metadata expiration check: 0:00:08 ago on Sat Dec 25 05:09:40 2021.
Dependencies resolved.
================================================================================
 Package         Architecture Version                       Repository    Size
================================================================================
Installing:
 net-tools       x86_64       2.0-0.52.20160912git.el8      baseos       322 k
Transaction Summary
================================================================================
Install  1 Package
Total download size: 322 k
Installed size: 942 k
Downloading Packages:
net-tools-2.0-0.52.20160912git.el8.x86_64.rpm   1.0 MB/s | 322 kB     00:00
--------------------------------------------------------------------------------
Total                                           449 kB/s | 322 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : net-tools-2.0-0.52.20160912git.el8.x86_64              1/1
  Running scriptlet: net-tools-2.0-0.52.20160912git.el8.x86_64              1/1
  Verifying        : net-tools-2.0-0.52.20160912git.el8.x86_64              1/1
Installed:
  net-tools-2.0-0.52.20160912git.el8.x86_64
Complete!
Removing intermediate container 6caa7361b001
 ---> a9431f90fd3f
Step 7/10 : EXPOSE 81
 ---> Running in ad67fa23940a
Removing intermediate container ad67fa23940a
 ---> b5bd21416741
Step 8/10 : CMD echo $MYPATH
 ---> Running in fb1d08538689
Removing intermediate container fb1d08538689
 ---> 5c5def0bbb85
Step 9/10 : CMD echo "---end---"
 ---> Running in a9d955b6b389
```

```
Removing intermediate container a9d955b6b389
 ---> ad95558eb658
Step 10/10 : CMD ["/bin/bash"]
 ---> Running in 190651202e7b
Removing intermediate container 190651202e7b
 ---> d58be7785771
Successfully built d58be7785771
Successfully tagged centos-test:latest
```

> 查看构建的镜像

```
[root@qzs dockerfile]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED              SIZE
centos-test         latest    d58be7785771   About a minute ago   323MB
```

> 查看本地镜像的构建记录

```
[root@qzs dockerfile]# docker history centos-test
IMAGE          CREATED         CREATED BY
SIZE      COMMENT
d58be7785771   5 minutes ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]
0B
ad95558eb658   5 minutes ago   /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "echo…
0B
5c5def0bbb85   5 minutes ago   /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "echo…
0B
b5bd21416741   5 minutes ago   /bin/sh -c #(nop)  EXPOSE 81
0B
a9431f90fd3f   5 minutes ago   /bin/sh -c yum -y install net-tools
27.3MB
9f54f48660ac   5 minutes ago   /bin/sh -c yum -y install vim
64.8MB
fb41ece5d944   5 minutes ago   /bin/sh -c #(nop) WORKDIR /usr/local
0B
be89377d4c2c   5 minutes ago   /bin/sh -c #(nop)  ENV MYPATH=/usr/local
0B
9616888f3b10   5 minutes ago   /bin/sh -c #(nop)  MAINTAINER sail<yifansail…
0B
5d0da3dc9764   3 months ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]
0B
<missing>      3 months ago    /bin/sh -c #(nop)  LABEL org.label-schema.sc…
0B
<missing>      3 months ago    /bin/sh -c #(nop) ADD file:805cb5e15fb6e0bb0…
231MB
```

> 运行测试

```
[root@qzs ~]# docker run -it centos-test
[root@530551bc2162 local]# pwd
/usr/local
[root@530551bc2162 local]# vim test.java
[root@530551bc2162 local]#
```

默认的工作目录正是 Dockerfile 中设置的 `/usr/local`，且可以使用 `vim` 命令了。

自定义tomcat环境镜像

> 编写 Dockerfile

```
FROM centos
MAINTAINER qzs<283408112@qq.com>
COPY readme.txt /usr/local/readme.txt
ENV MYPATH /usr/local/
WORKDIR $MYPATH
ADD jdk-8u301-linux-x64.tar.gz $MYPATH
ADD apache-tomcat-9.0.55.tar.gz $MYPATH
RUN yum -y install vim
ENV JAVA_HOME $MYPATH/jdk1.8.0_301-amd64
ENV CLASSPATH $JAVA_HOME/lib/
ENV CATALINA_HOME $MYPATH/apache-tomcat-9.0.55
ENV CATALINA_BASH $MYPATH/apache-tomcat-9.0.55
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/bin:$CATALINA_HOME/lib
EXPOSE 8080
CMD $CATALINA_HOME/bin/startup.sh && tail -F $CATALINA_HOME/logs/catalina.out
```

其中的 **readme.txt** 一般作为镜像说明文件，可以在里面编写镜像的信息。

> 构建镜像

```
[root@qzs tomcat]# docker build -t tomcat-test .
Sending build context to Docker daemon  157.1MB
Step 1/15 : FROM centos
 ---> 5d0da3dc9764
Step 2/15 : MAINTAINER sail<yifansailing@163.com>
 ---> Using cache
 ---> 9616888f3b10
Step 3/15 : COPY readme.txt /usr/local/readme.txt
 ---> da792df641f8
Step 4/15 : ENV MYPATH /usr/local/
 ---> Running in e4a5b13decd7
Removing intermediate container e4a5b13decd7
 ---> 7b1e6970b4b3
Step 5/15 : WORKDIR $MYPATH
 ---> Running in 835dabd080dd
Removing intermediate container 835dabd080dd
 ---> 7be17b1556ee
Step 6/15 : ADD jdk-8u301-linux-x64.tar.gz $MYPATH
 ---> 480721043fda
Step 7/15 : ADD apache-tomcat-9.0.55.tar.gz $MYPATH
 ---> c7bfa13bfcd1
Step 8/15 : RUN yum -y install vim
 ---> Running in 85532523d784
CentOS Linux 8 - AppStream                      9.0 MB/s | 8.4 MB     00:00
CentOS Linux 8 - BaseOS                         5.6 MB/s | 3.6 MB     00:00
CentOS Linux 8 - Extras                          20 kB/s |  10 kB     00:00
Dependencies resolved.
================================================================================
 Package             Arch        Version                   Repository      Size
================================================================================
Installing:
 vim-enhanced        x86_64      2:8.0.1763-16.el8         appstream      1.4 M
Installing dependencies:
```

```
 gpm-libs            x86_64      1.20.7-17.el8             appstream       39 k
 vim-common          x86_64      2:8.0.1763-16.el8         appstream      6.3 M
 vim-filesystem      noarch      2:8.0.1763-16.el8         appstream       49 k
 which               x86_64      2.21-16.el8               baseos          49 k
Transaction Summary
================================================================================
Install  5 Packages
Total download size: 7.8 M
Installed size: 30 M
Downloading Packages:
(1/5): gpm-libs-1.20.7-17.el8.x86_64.rpm        973 kB/s |  39 kB     00:00
(2/5): vim-filesystem-8.0.1763-16.el8.noarch.rp 726 kB/s |  49 kB     00:00
(3/5): vim-enhanced-8.0.1763-16.el8.x86_64.rpm   10 MB/s | 1.4 MB     00:00
(4/5): which-2.21-16.el8.x86_64.rpm             901 kB/s |  49 kB     00:00
(5/5): vim-common-8.0.1763-16.el8.x86_64.rpm     27 MB/s | 6.3 MB     00:00
--------------------------------------------------------------------------------
Total                                            6.6 MB/s | 7.8 MB     00:01
warning: /var/cache/dnf/appstream-02e86d1c976ab532/packages/gpm-libs-1.20.7-
17.el8.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID 8483c65d: NOKEY
CentOS Linux 8 - AppStream                       1.6 MB/s | 1.6 kB     00:00
Importing GPG key 0x8483C65D:
 Userid     : "CentOS (CentOS Official Signing Key) <security@centos.org>"
 Fingerprint: 99DB 70FA E1D7 CE22 7FB6 4882 05B5 55B3 8483 C65D
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : which-2.21-16.el8.x86_64                               1/5
  Installing       : vim-filesystem-2:8.0.1763-16.el8.noarch                2/5
  Installing       : vim-common-2:8.0.1763-16.el8.x86_64                    3/5
  Installing       : gpm-libs-1.20.7-17.el8.x86_64                          4/5
  Running scriptlet: gpm-libs-1.20.7-17.el8.x86_64                          4/5
  Installing       : vim-enhanced-2:8.0.1763-16.el8.x86_64                  5/5
  Running scriptlet: vim-enhanced-2:8.0.1763-16.el8.x86_64                  5/5
  Running scriptlet: vim-common-2:8.0.1763-16.el8.x86_64                    5/5
  Verifying        : gpm-libs-1.20.7-17.el8.x86_64                          1/5
  Verifying        : vim-common-2:8.0.1763-16.el8.x86_64                    2/5
  Verifying        : vim-enhanced-2:8.0.1763-16.el8.x86_64                  3/5
  Verifying        : vim-filesystem-2:8.0.1763-16.el8.noarch                4/5
  Verifying        : which-2.21-16.el8.x86_64                               5/5
Installed:
  gpm-libs-1.20.7-17.el8.x86_64         vim-common-2:8.0.1763-16.el8.x86_64
  vim-enhanced-2:8.0.1763-16.el8.x86_64 vim-filesystem-2:8.0.1763-16.el8.noarch
  which-2.21-16.el8.x86_64
Complete!
Removing intermediate container 85532523d784
 ---> e091ece0364d
Step 9/15 : ENV JAVA_HOME $MYPATH/jdk1.8.0_301-amd64
 ---> Running in 473066cf57f4
Removing intermediate container 473066cf57f4
 ---> 0a8963a2c1ab
Step 10/15 : ENV CLASSPATH $JAVA_HOME/lib/
 ---> Running in 78a2cb9b06cd
Removing intermediate container 78a2cb9b06cd
```

```
 ---> 3dd34a2857b4
Step 11/15 : ENV CATALINA_HOME $MYPATH/apache-tomcat-9.0.55
 ---> Running in 4ca540479e3d
Removing intermediate container 4ca540479e3d
 ---> fa38f4581510
Step 12/15 : ENV CATALINA_BASH $MYPATH/apache-tomcat-9.0.55
 ---> Running in 31dc5b38478c
Removing intermediate container 31dc5b38478c
 ---> 8ae919106bf6
Step 13/15 : ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/bin:$CATALINA_HOME/lib
 ---> Running in d3fe1f81fab7
Removing intermediate container d3fe1f81fab7
 ---> dd8b07b2adfd
Step 14/15 : EXPOSE 8080
 ---> Running in 1f1601f2dcc2
Removing intermediate container 1f1601f2dcc2
 ---> 9078648b7a2e
Step 15/15 : CMD $CATALINA_HOME/bin/startup.sh && tail -F
$CATALINA_HOME/logs/catalina.out
 ---> Running in 6a3b2aefaf44
Removing intermediate container 6a3b2aefaf44
 ---> 23a538c107a0
Successfully built 23a538c107a0
Successfully tagged tomcat-test:latest
```

> 查看镜像

```
[root@qzs tomcat]# docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
tomcat-test       latest    23a538c107a0   25 minutes ago   673MB
```

> 启动镜像

```
[root@qzs tomcat]# docker run -d -p 8080:8080 --name sail-tomcat -v
/home/sail/tomcat/webapps:/usr/local/apache-tomcat-9.0.55/webapps -v
/home/sail/tomcat/logs:/usr/local/apache-tomcat-9.0.55/logs tomcat-test
9d391e13efdc495206429dbdb0392180a7bd3a4750cbc1419c31c80cd69c6b7b
[root@sail tomcat]#
```

启动时将 tomcat 的 **webapps** 和 **logs** 目录都挂载到了本机。

> 查看挂载目录

```
[root@qzs tomcat]# ls /home/sail/tomcat
logs  webapps
```

这里找到了挂载到本机的两个目录，说明挂载成功了。

> 进入容器

```
[root@qzs tomcat]# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS
       PORTS                    NAMES
9d391e13efdc   tomcat-test   "/bin/sh -c '$CATALI…"   24 minutes ago   Up 24
minutes   0.0.0.0:8080->8080/tcp   sail-tomcat

[root@sail tomcat]# docker exec -it 9d391e13efdc /bin/bash
[root@9d391e13efdc local]# ls
apache-tomcat-9.0.55  bin  etc    games  include    jdk1.8.0_301  lib  lib64
libexec  readme.txt  sbin  share  src


[root@9d391e13efdc local]# cd apache-tomcat-9.0.55/
[root@9d391e13efdc apache-tomcat-9.0.55]# ls
BUILDING.txt  CONTRIBUTING.md  LICENSE    NOTICE    README.md  RELEASE-NOTES
RUNNING.txt  bin  conf  lib  logs  temp  webapps  work
```

jdk 和 readme.txt 都是具备了的，且 tomcat 目录下的文件也是完整的。

> 查看挂载文件

这里以 logs 为例，我们先进入 tomcat 容器中的 logs 文件夹查看日志内容。

```
[root@9d391e13efdc apache-tomcat-9.0.55]# cd logs
[root@9d391e13efdc logs]# ls
catalina.out
[root@9d391e13efdc logs]# cat catalina.out
/usr/local//apache-tomcat-9.0.55/bin/catalina.sh: line 504:
/usr/local//jdk1.8.0_301-amd64/bin/java: No such file or directory
```

然后再退出查看主机上挂载的 logs 文件夹。

```
[root@9d391e13efdc logs]# exit
exit
[root@qzs tomcat]# cd /home/sail/tomcat/logs
[root@qzs logs]# ls
catalina.out
[root@qzs logs]# cat catalina.out
/usr/local//apache-tomcat-9.0.55/bin/catalina.sh: line 504:
/usr/local//jdk1.8.0_301-amd64/bin/java: No such file or directory
```

两个地方 logs 下的文件内容一致，说明挂载成功。

发布镜像到 Docker Hub

> 注册账号

如果没有 Docker Hub 账号，先注册账号：https://hub.docker.com/

> 登录 Docker Hub 账号

```
[root@qzs logs]# docker login -u 账号
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

发布镜像

docker push

> 直接发布镜像

```
[root@qzs logs]# docker push centos-test
Using default tag: latest
The push refers to repository [docker.io/library/centos-test]
de70c523870b: Preparing
909db45c4bc4: Preparing
74ddd0ec08fa: Preparing
denied: requested access to the resource is denied
```

访问资源被拒绝了。拒绝的原因是我们没有带标签，默认的 latest 标签是不能被识别的。

指定镜像标签

docker tag

我们可以使用 `docker tag` 命令给镜像加一个标签。

> 必须以 `账号名/镜像名:标签` 的格式命令才能提交。

```
[root@qzs logs]# docker tag d58be7785771 账号/centos:1.0
[root@qzs logs]# docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
asailing/centos   1.0       d58be7785771   29 hours ago    323MB
centos-test       latest    d58be7785771   29 hours ago    323MB
```

此时会多出一个相同 ID 但是标签和名字不同的镜像。

> 再次发布镜像

```
[root@qzs logs]# docker push asailing/centos:1.0
The push refers to repository [docker.io/asailing/centos]
de70c523870b: Pushed
909db45c4bc4: Pushed
74ddd0ec08fa: Pushed
1.0: digest:
sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626b10f0f1 size:
953
```

这样就能发布成功了。且可以发现，**镜像的发布也是分层发布的。**

配置国内镜像站

由于对国外网络的限制，发布镜像到 DockerHub 是比较缓慢的。

这里可以使用配置 **Docker 国内镜像站**的方式实现加速。

运行以下命令即可：

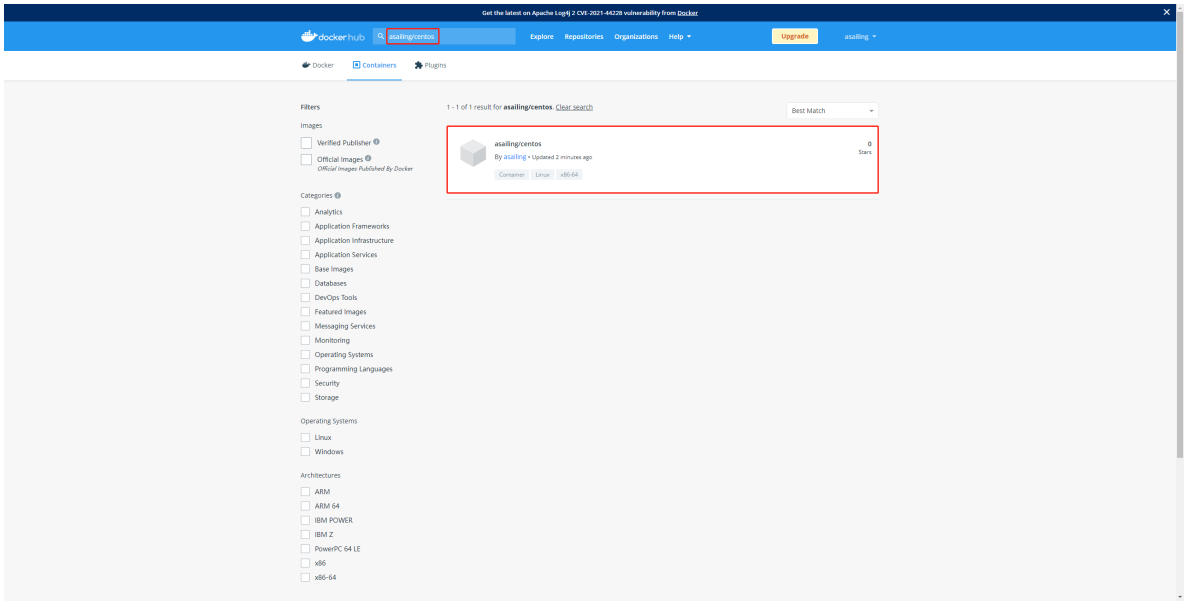
```
[root@qzs ~]# curl -sSL https://get.daocloud.io/daotools/set_mirror.sh | sh -s http://f1361db2.m.daocloud.io
docker version >= 1.12
{
  "registry-mirrors": ["http://f1361db2.m.daocloud.io"]
}
Success.
You need to restart docker to take effect: sudo systemctl restart docker


[root@qzs ~]# systemctl restart docker
[root@qzs ~]#
```

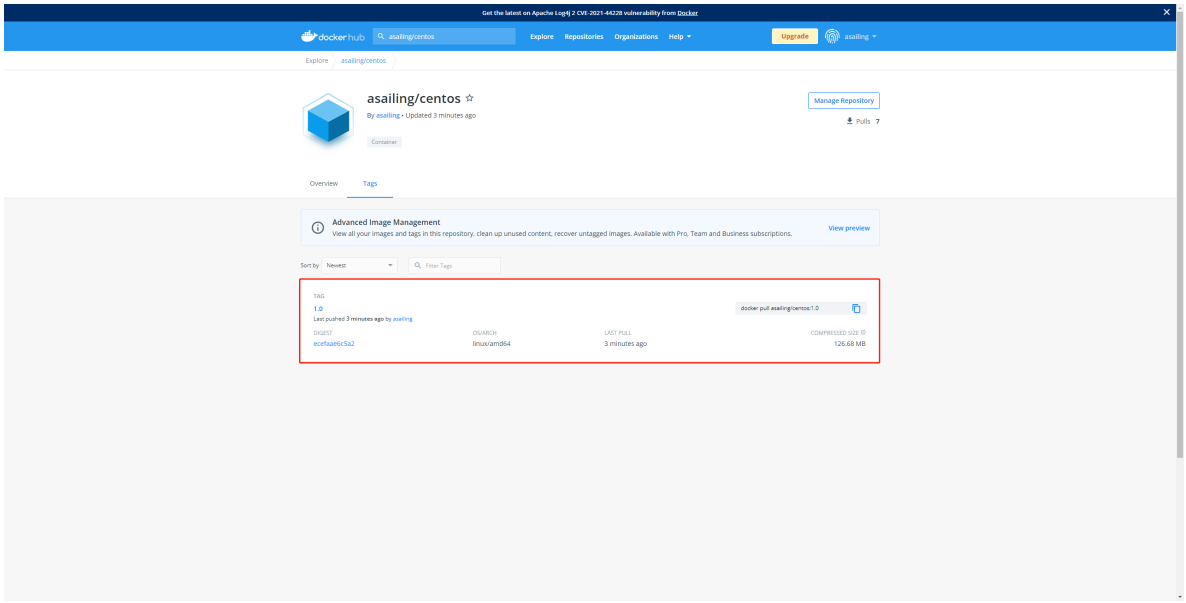
该脚本可以将 `--registry-mirror` 加入到 Docker 配置文件 `/etc/docker/daemon.json` 中。

适用于 **Ubuntu14.04、Debian、CentOS6 、CentOS7、Fedora、Arch Linux、openSUSE Leap 42.1**，其他版本可能有细微不同。

> 去 Docker Hub 上以 **账号名/镜像名** 搜索我们刚发布的镜像，发现是可以搜索到的。



> 查看详情也可以镜像的具体信息。



DIGEST 的值正是刚才发布后返回值 `ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626b10f0f1` 的缩写。

且镜像的大小是小于我们本地镜像的，说明**发布的过程中也会压缩镜像。**

> 拉取我们发布的镜像

```
[root@qzs logs]# docker pull 账号/centos:1.0
1.0: Pulling from asailing/centos
Digest: sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626b10f0f1
Status: Image is up to date for asailing/centos:1.0
docker.io/asailing/centos:1.0
```

无法拉取。原因很简单，因为我们本地存在了同名镜像。

> 我们先删除这个镜像再拉取

```
[root@qzs logs]# docker rmi -f d58be7785771
Untagged: asailing/centos:1.0
Untagged:
asailing/centos@sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626
b10f0f1
Untagged: centos-test:latest
Untagged: sail/centos:1.0
Deleted: sha256:d58be7785771bd95d8016fa5807a486d6c50e195879eddd88cb602172fc51ffe
Deleted: sha256:ad95558eb65801f5871215837558156c5e33ba351b3b52e0a50aac045abb46c1
Deleted: sha256:5c5def0bbb85d8779d02f115c3d072fe9adb1fd07556ee8c5a130823ecf6811d
Deleted: sha256:b5bd21416741daec348f417dbea1b73001e257f1e63a5d2abddabc8554fca611
Deleted: sha256:a9431f90fd3f23387c456ad5b925dbb9531beece3eab825848db99db29c6a1fa
Deleted: sha256:9f54f48660acb350921aefab74769e51fc7917a1e1e730d3df2edd1513517c42
Deleted: sha256:fb41ece5d944c667762945fdf7275a1d267acd92fe9dc56709fc3adaca6f087f
Deleted: sha256:be89377d4c2ccea11308d8196ba53f03985882db015e01ed8b54fc114f4ba058
Deleted: sha256:9616888f3b103230ed5f378af4afc11b7ce7ed3d96653e5bd918c49152bbdf8c

[root@qzs logs]# docker pull 账号/centos:1.0
1.0: Pulling from asailing/centos
a1d0c7532777: Already exists
0594d57f8468: Already exists
9c13f720f33e: Already exists
Digest: sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626b10f0f1
```

```
Status: Downloaded newer image for asailing/centos:1.0
docker.io/asailing/centos:1.0


[root@qzs logs]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
asailing/centos     1.0       d58be7785771   29 hours ago     323MB
```

拉取成功，且大小又恢复到了之前本地的镜像大小，说明**拉取的过程中也会解压镜像。**

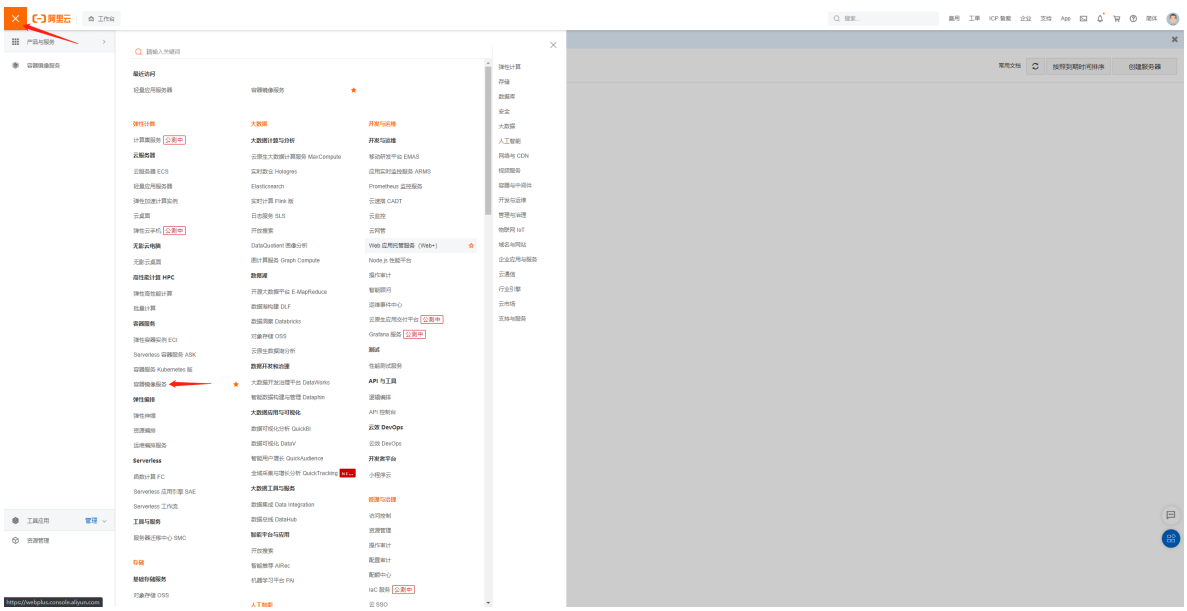
> 启动拉取的镜像

```
[root@qzs logs]# docker run -it asailing/centos:1.0 /bin/bash
[root@168c9e550886 local]# vim test.java
[root@168c9e550886 local]#
```

`vim` 命令也是可以使用的，镜像发布成功。

发布镜像到阿里云镜像仓库

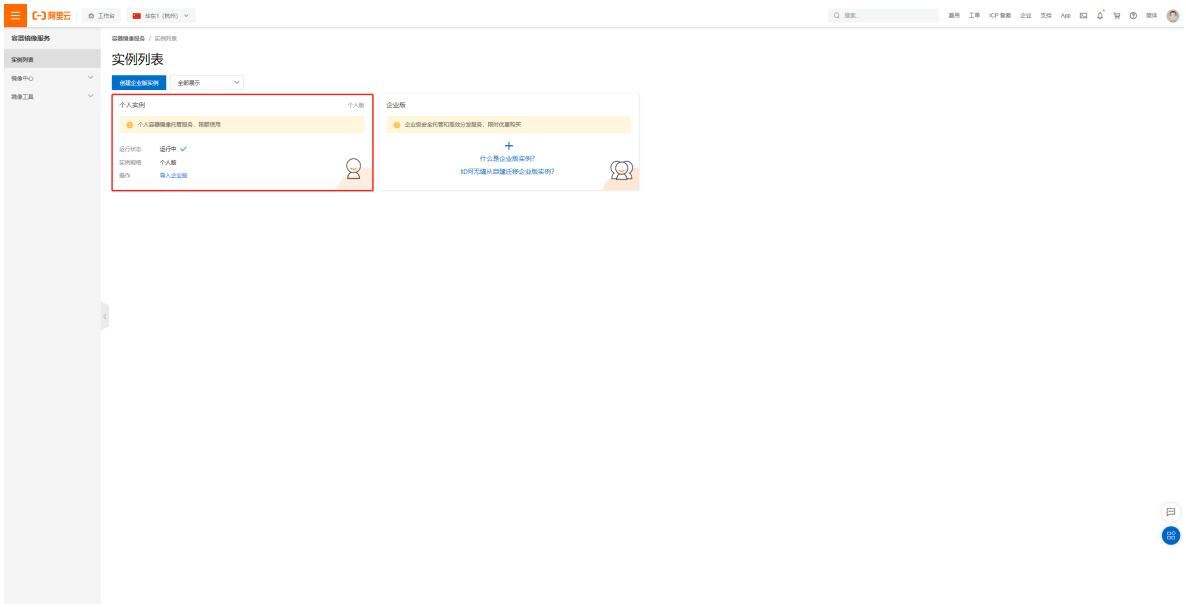
> 登录阿里云，点击**我的阿里云**



创建实例

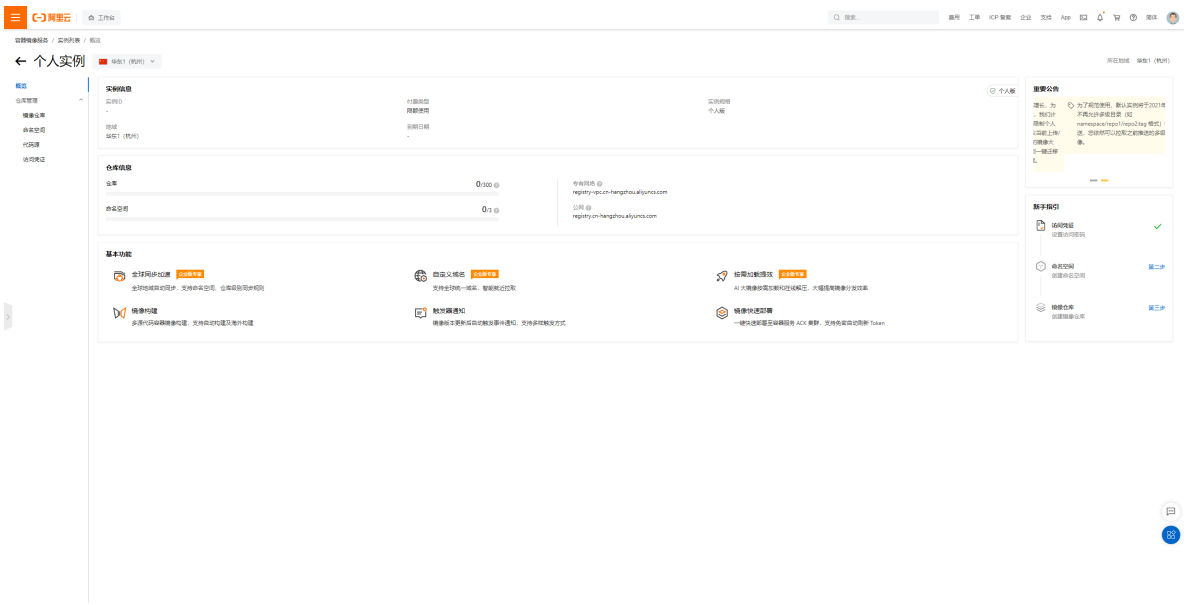
这里以创建个人版实例为例。

我这里已经创建好了，如果没有创建点击创建即可。

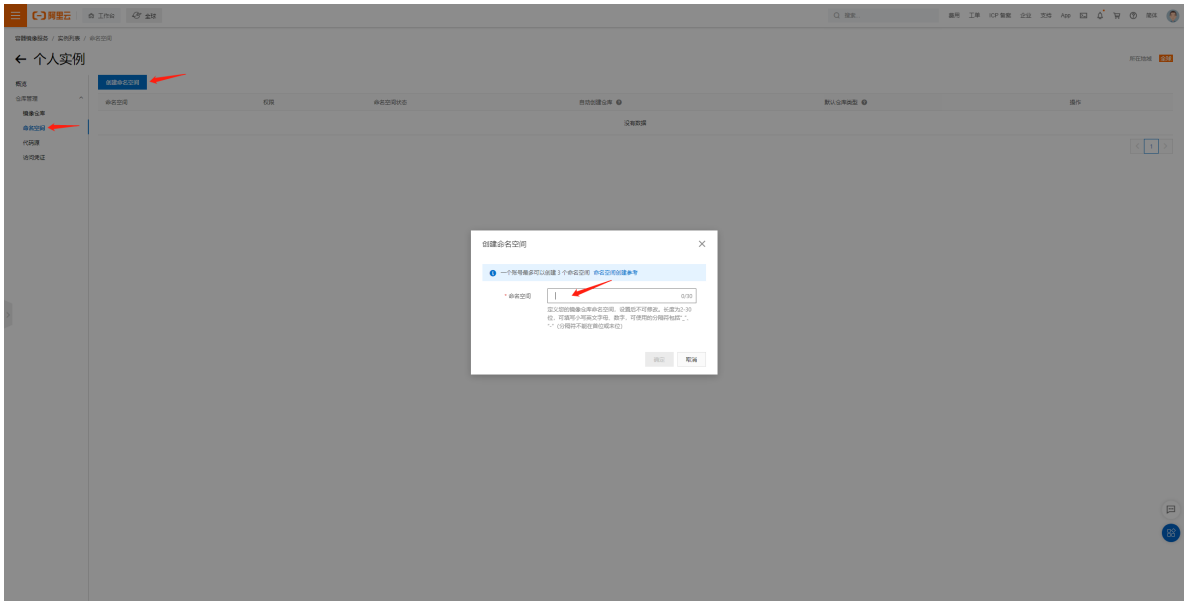


进入镜像仓库

创建好个人实例后，点击进入。

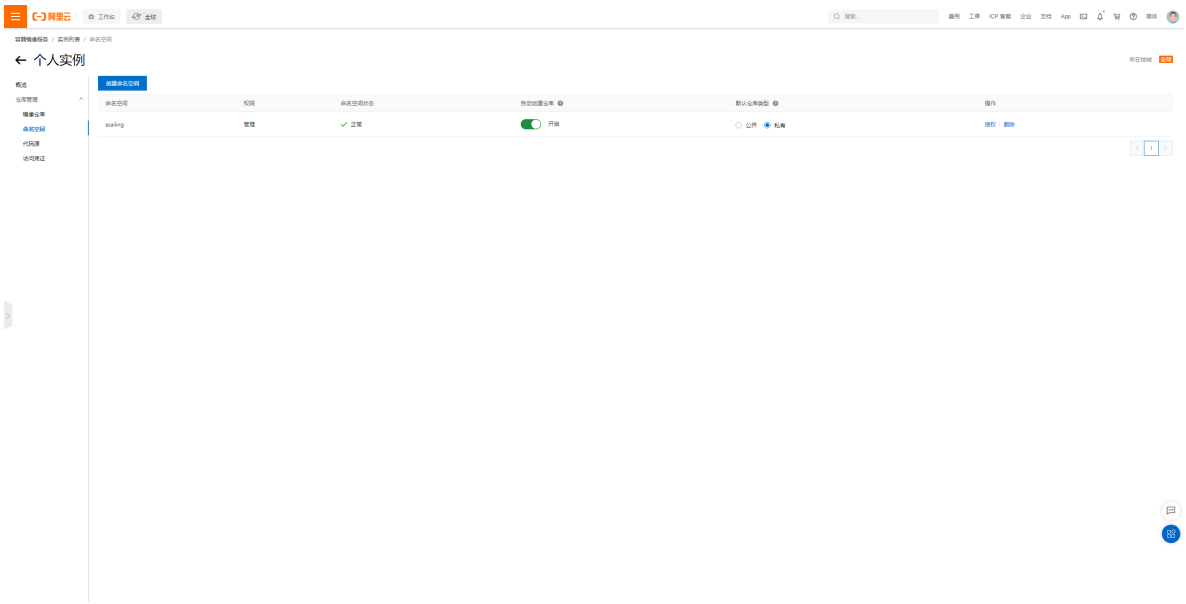


创建命名空间

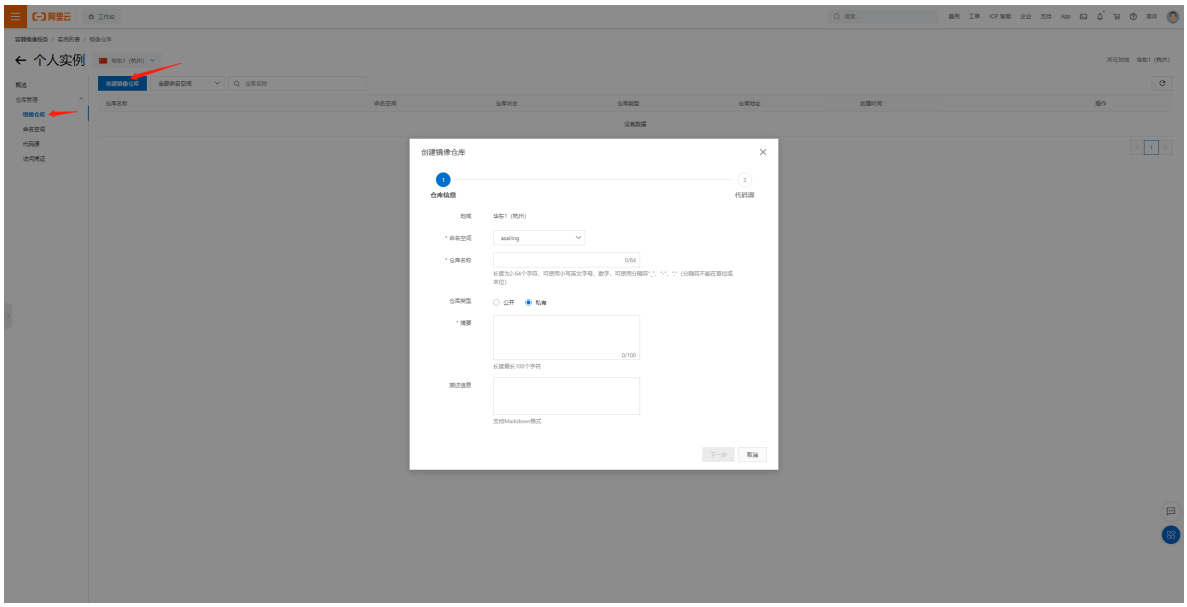


一个账号只能创建 3 个命名空间，需要谨慎创建。

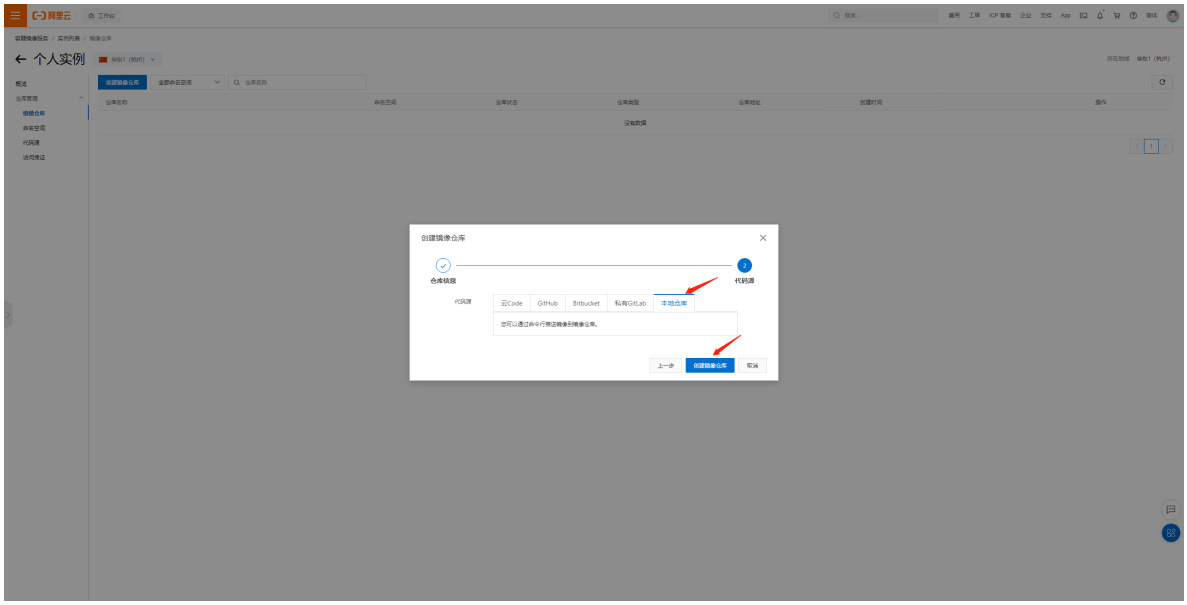
创建好后就是这样。

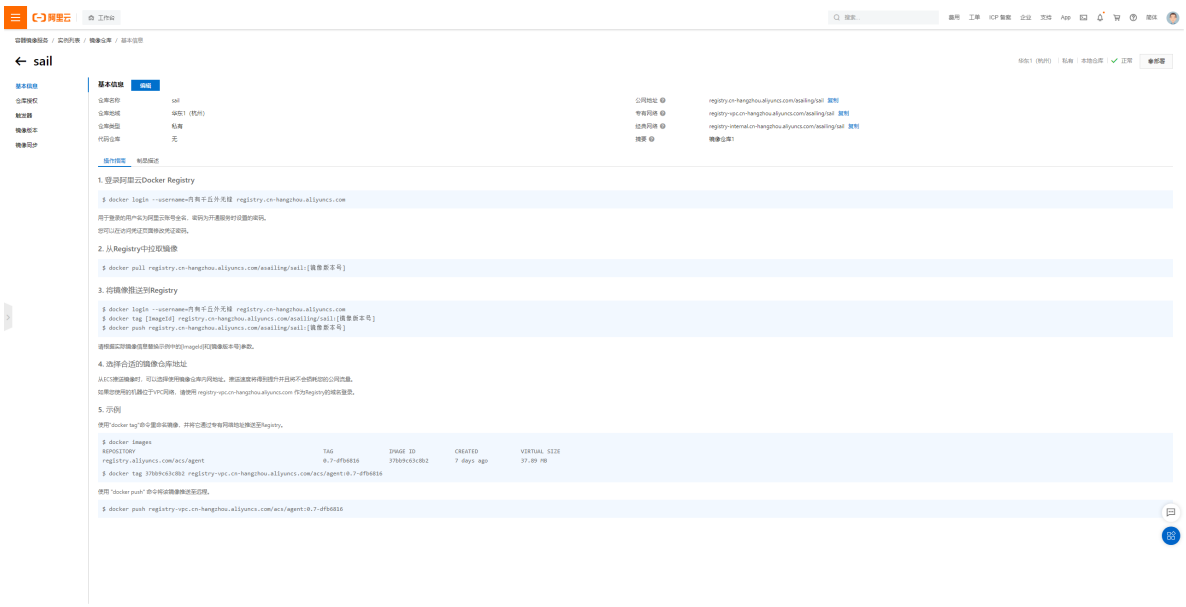


> 创建镜像仓库



> 点击下一步，创建本地仓库





至此，我们就创建好了阿里云的镜像仓库，具体的操作步骤上图也写得非常清楚。

> 退出登录的账号

如果之前登录了 Docker Hub 账号或者其他阿里云账号，先退出账号。

```
[root@qzs logs]# docker logout
Removing login credentials for https://index.docker.io/v1/
```

> 登录阿里云账号

```
[root@qzs logs]# docker login --username=用户名 registry.cn-hangzhou.aliyuncs.com
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

> 设置镜像标签

```
[root@qzs logs]# docker tag d58be7785771 registry.cn-
hangzhou.aliyuncs.com/asailing/qzs:1.0
[root@qzs logs]# docker images
REPOSITORY                                          TAG       IMAGE ID
CREATED        SIZE
asailing/centos                                     1.0       d58be7785771   32
hours ago   323MB
registry.cn-hangzhou.aliyuncs.com/asailing/sail   1.0       d58be7785771   32
hours ago   323MB
```

> 提交镜像

```
[root@qzs logs]# docker push registry.cn-hangzhou.aliyuncs.com/asailing/qzs:1.0
The push refers to repository [registry.cn-hangzhou.aliyuncs.com/asailing/qzs]
de70c523870b: Pushed
909db45c4bc4: Pushed
74ddd0ec08fa: Pushed
1.0: digest:
sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626b10f0f1 size:
953
```

> 查看提交的镜像

提交的镜像可以在这里查看。

> 拉取镜像

先删除本地镜像， 再拉取测试。

```
[root@qzs logs]# docker rmi -f d58be7785771
Untagged: asailing/centos:1.0
Untagged:
asailing/centos@sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626
b10f0f1
Untagged: registry.cn-hangzhou.aliyuncs.com/asailing/sail/centos:1.0
Untagged: registry.cn-hangzhou.aliyuncs.com/asailing/sail:1.0
Untagged: registry.cn-
hangzhou.aliyuncs.com/asailing/sail@sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa
0160e33a0bf3eb4ab626b10f0f1
Deleted: sha256:d58be7785771bd95d8016fa5807a486d6c50e195879eddd88cb602172fc51ffe


[root@qzs logs]# docker pull registry.cn-hangzhou.aliyuncs.com/asailing/sail:1.0
1.0: Pulling from asailing/sail
a1d0c7532777: Already exists
0594d57f8468: Already exists
9c13f720f33e: Already exists
Digest: sha256:ecefaae6c5a2cab84693175ea3b18d0d0a7aa0160e33a0bf3eb4ab626b10f0f1
Status: Downloaded newer image for registry.cn-
hangzhou.aliyuncs.com/asailing/sail:1.0
registry.cn-hangzhou.aliyuncs.com/asailing/sail:1.0

[root@qzs logs]# docker images
REPOSITORY                                          TAG       IMAGE ID
CREATED        SIZE
```

```
registry.cn-hangzhou.aliyuncs.com/asailing/sail   1.0       d58be7785771   32
hours ago   323MB
```

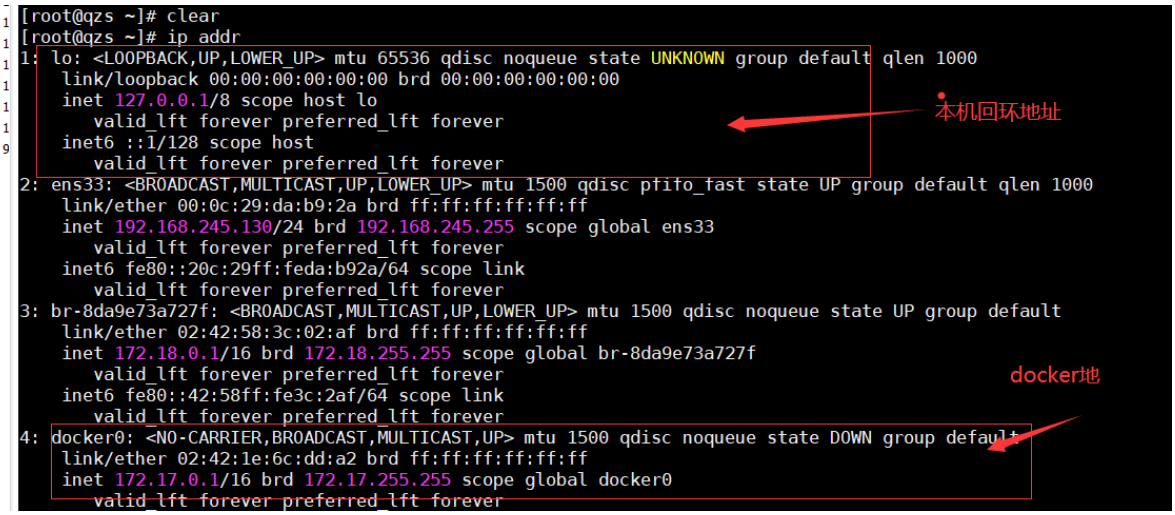
> 启动镜像

```
[root@qzs logs]# docker run -it registry.cn-
hangzhou.aliyuncs.com/asailing/sail:1.0
[root@c612099a94a2 local]# vim test.java
[root@c612099a94a2 local]#
```

`vim` 命令可以使用，说明提交镜像成功。

Docker网络

理解Docker0



> docker是如何处理容器访问网络的?

```
#1.启动一个tomcat容器
docker run -d -P --name tomcat01 tomcat
#2.查看容器的内部网络地址
  #第一种docker exec -it tomact ip addr
  #第二种 先进入容器 在使用ip addr
root@8c32956f7370:/usr/local/tomcat# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
36: eth0@if37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever

#3. ping 容器ip
[root@qzs /]# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
```

```
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.986 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.102 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.062 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.076 ms
--- 172.17.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.062/0.259/0.986/0.363 ms
```

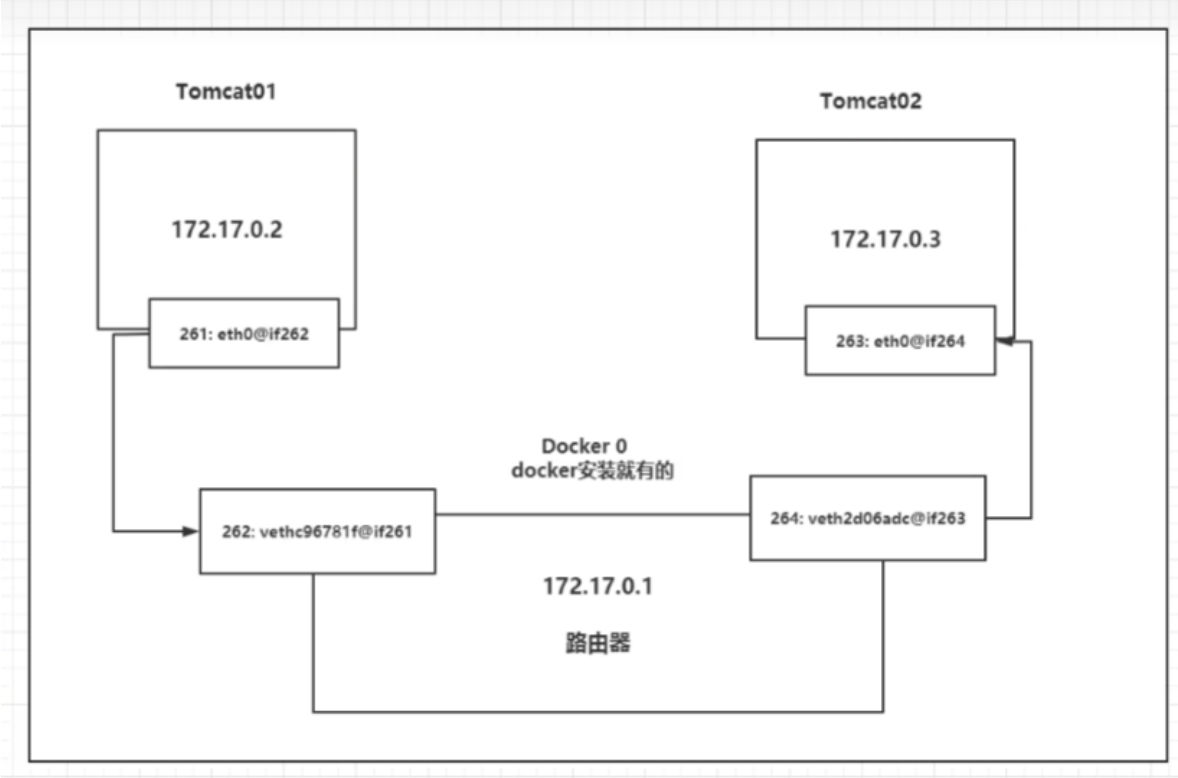
> 原理

1.每启动一个docker容器，docker就会给docker容器分配一个ip，我们只要安装了docker，就会有一个网卡docker0

docker0是桥接模式，使用的技术是evth-pair技术

```
#容器带来网卡，都是一对对的
#evth-pair 就是一对的虚拟设备接口，它们都是成对出现的，一段连着协议，一段彼此相连
#正因为有这个特性，evth-pair充当一个桥梁，连接各种虚拟网络设备的
```

容器与容器之间可以相互ping通的

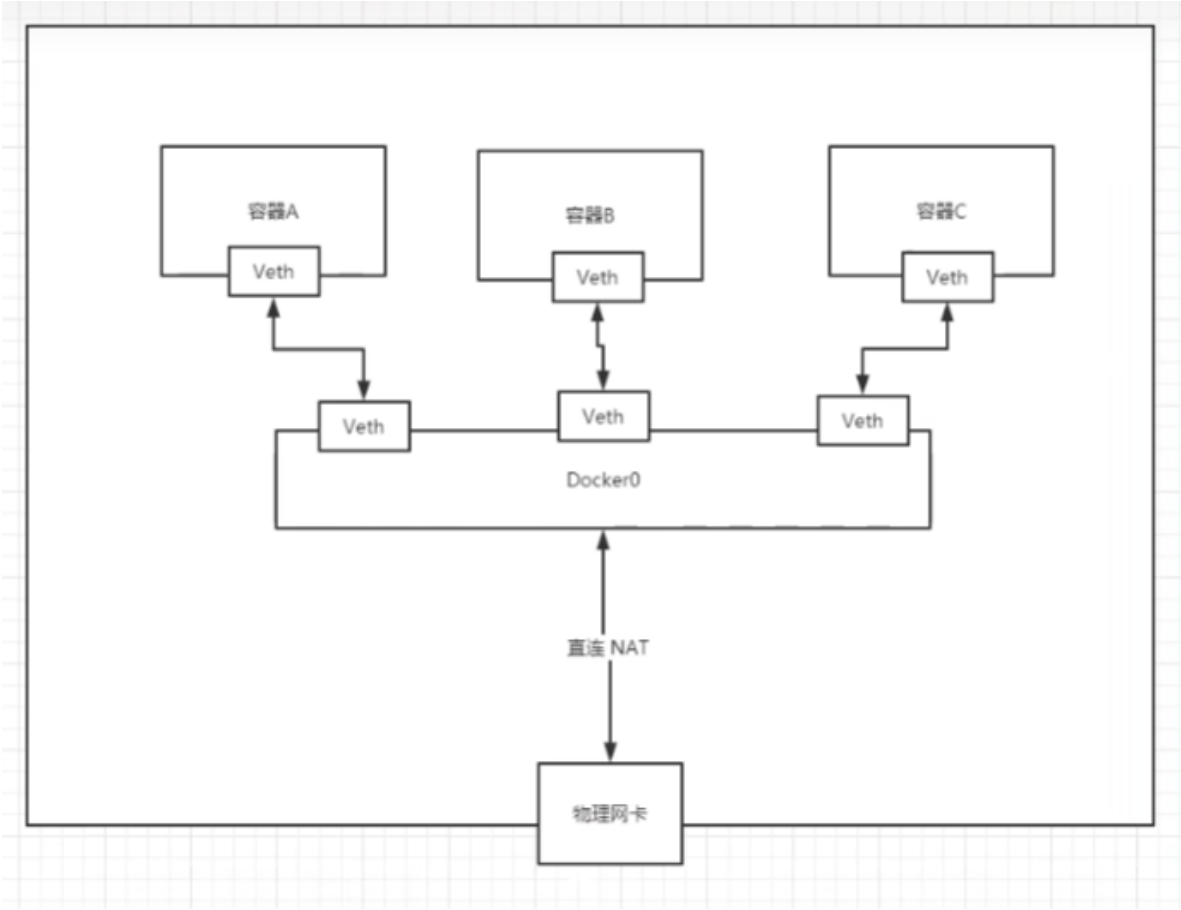


结论：tomcat01 与 tomcat02是公用的一个路由器 docker0

所有的容器不指定网络的情况下，都是docker0路由，docker会给我们容器分配一个默认的可用IP

> 小结

Docker使用的是Linux的桥接



Docker中的所有网络接口都是虚拟的，虚拟的转发效率高、

只要容器删除，对应的网桥就没了

--link

> 场景问题: 每次重新启动一个容器 容器ip变化，那么使用ip去访问就十分麻烦，有没有可以提供访问服务名来访问容器这样不管ip怎么变只要服务名不变就可以正常访问该容器

```
# 1. 通过服务名ping不通；如何解决？
docker exec -it tomcat02 ping tomcat01
ping: tomcat01: Name or service not known
#2.通过--link可以解决网络连接问题。
docker run -d -P --name tomcat02 --link tomcat01 tomcat:7.0
docker exec -it tomcat03 ping tomcat02
#3.反向可以ping通吗？（不可以）
docker exec -it tomcat02 ping tomcat03
ping: tomcat03: Name or service not known
```

探究：docker network inspect networkID (docker network ls可以查看networkID)

其实这个tomcat03就是在本地配置了tomcat02的配置

```
#1.查看
exec -it tomcat03 cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.3      tomcat02 20398a94efa7
172.17.0.4      2393eecb870e
```

本质探究：—link 就是我们在hosts配置中增加了一个”172.17.0.3 tomcat02 20398a94efa7”
我们现在玩Docker已经不建议使用—link了！
自定义网络！不适用docker0！
docker0问题：他不支持容器名连接访问！

自定义网络

查看所有的docker网络

```
docker network ls
```

网络模式

bridge：桥接 docker（默认，自己创建也使用bridge桥接模式）
none：不配置网络
host：和主机共享网络
container：容器网络连通！（用的少！局限很大）

```
# 我们直接启动的命令--net bridge（这个就是我们的docker0）；默认带上这个参数的，以下两种启动方式效果一致。
docker run -d -P --name tomcat01 tomcat
docker run -d -P --name tomcato1 --het bridge tomcat
```

创建自定义网络

- docker network creat
  - `-d`：网络模式
  - `--subnet`：子网
  - `--gateway`：网关

```
# docker0特点：默认，域名不能访问，--1ink可以打通连接！
# 我们可以自定义一个网络！
# docker network
docker network create --driver bridge --subnet 192.168.0.0/16 --gateway
192.168.0.1 mynet
# 查看创建的网络
docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
7254ffccbdf5   bridge    bridge    local
45610891738f   host      host      local
266acd66473c   mynet     bridge    local
7795cbc2686c   none      null      local
```

```
docker network inspect 801fbbe1b38c
[
    {
        "Name": "mynet",
        "Id":
"801fbbe1b38c81b12ce90aa9139561b5843dca64b4b17718b6e2622369f9be67",
        "Created": "2021-12-30T17:39:43.100705632+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.0.0/16",
                    "Gateway": "192.168.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

> 以下配置可以看出，自定义网络创建完成。

```
"Config": [
    {
        "Subnet": "192.168.0.0/16",
        "Gateway": "192.168.0.1"
    }
]
```

启动容器

```
#启动两个容器测试：
docker run -d -P --name tomcat-net-01 --net mynet tomcat:7.0
docker run -d -P --name tomcat-net-02 --net mynet tomcat:7.0
```

连接测试

```
# 不使用--link，ping名字也可以ping通。tomcat-net-01 ping tomcat-net-02可以ping通
docker exec -it tomcat-net-01 ping tomcat-net-02
PING tomcat-net-02 (192.168.0.3) 56(84) bytes of data.
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=2 ttl=64 time=0.056 ms
# 不使用--link，ping名字也可以ping通。tomcat-net-02 ping tomcat-net-01可以ping通
docker exec -it tomcat-net-02 ping tomcat-net-01
PING tomcat-net-01 (192.168.0.2) 56(84) bytes of data.
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=1 ttl=64 time=0.050 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=2 ttl=64 time=0.056 ms
```

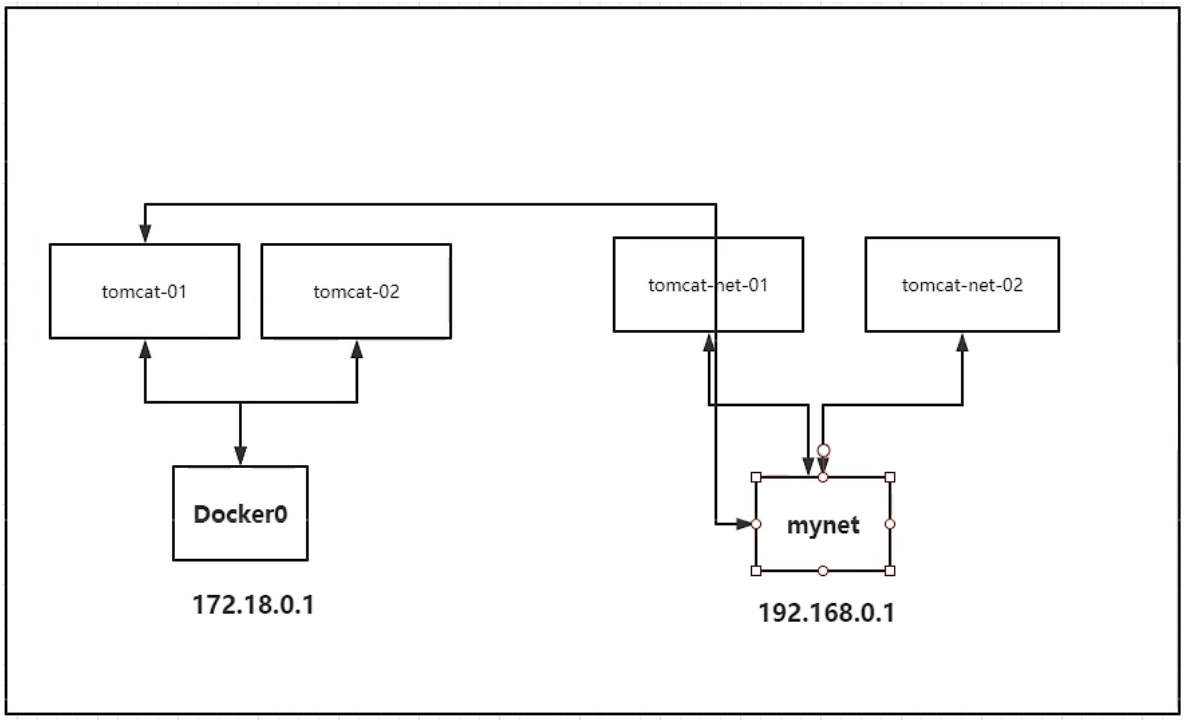
能够连通说明不同容器处于同一网络下。

> 这种方式可以实现不同集群使用不同的网络，保证集群网络的安全。

我们自定义的网络docker都已经帮我们维护好了对应的关系，**推荐我们平时这样使用网络**！
redis -不同的集群使用不同的网络，保证集群是安全和健康的
mysql -不同的集群使用不同的网络，保证集群是安全和健康的

网络连通

```
# tomcat01在docker0网络下，tomcat-net-01在mynet网络下；
# tomcat01 ping tomcat-net-01是ping不通的
docker exec -it tomcat01 ping tomcat-net-01
ping: tomcat-net-01: Name or service not known
```




```
#1.查看docker 网络
docker network --help
Usage:  docker network COMMAND
Manage networks
Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
```

```
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks
Run 'docker network COMMAND --help' for more information on a command.
```

使用`docker network connect`实现一个容器链接到另一个网段。

```
#1.建立连接
docker network connect mynet tomcat02-net
#2.查看docker网络
docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
f3eeb014197a   bridge    bridge    local
28d77e958643   host      host      local
801fbbe1b38c   mynet     bridge    local
c3ff850e96f0   none      null      local
#3.查看 容器所在网络的网络配置
docker network inspect 801fbbe1b38c
[
    {
        "Name": "mynet",
        "Id":
"801fbbe1b38c81b12ce90aa9139561b5843dca64b4b17718b6e2622369f9be67",
        "Created": "2021-12-30T17:39:43.100705632+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.0.0/16",
                    "Gateway": "192.168.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "5a02cd4172daccc5073907ee6b063560687db5ffdd5041b18fd3ff1055a8984c":
{
                "Name": "tomcat02-net",
                "EndpointID":
"e6503138bcb91a7693576e324df75d1dff594f1c5aa3e083978o2c38133eb0e9",
                "MacAddress": "02:42:c0:a8:00:05",
                "IPv4Address": "192.168.0.5/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
```

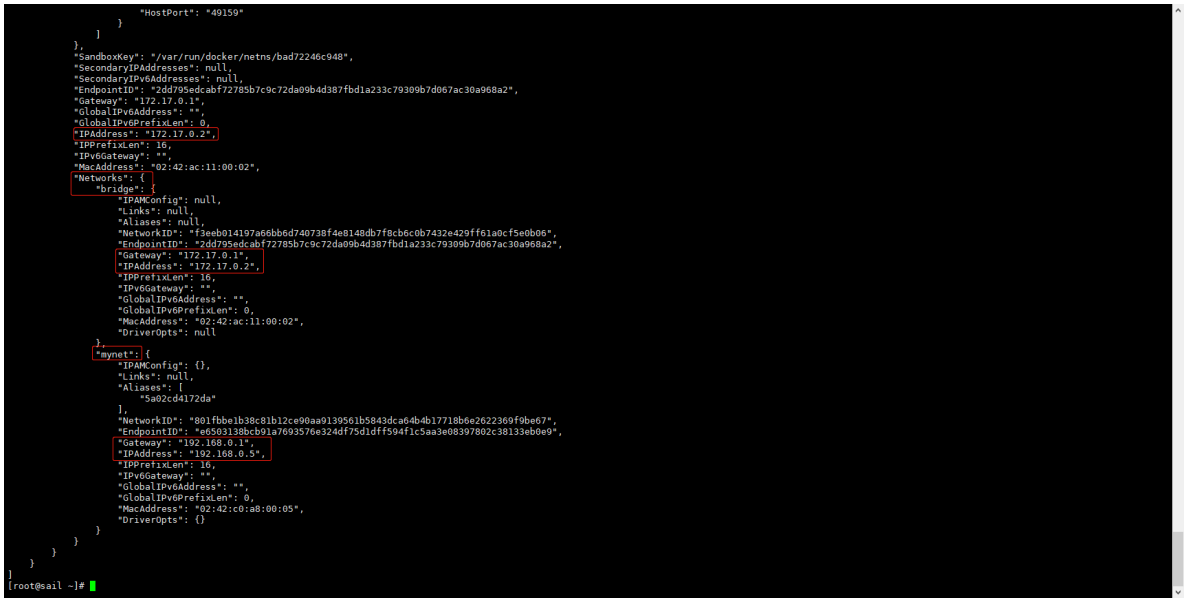
```
            "Labels": {}
        }
]
```

这样也可以实现容器链接到自定义网络。

> 查看容器详情

```
#1.查看运行中的容器id
docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED             STATUS
    PORTS                      NAMES
5a02cd4172da   tomcat    "catalina.sh run"   5 minutes ago       Up 5 minutes
    0.0.0.0:49159->8080/tcp   tomcat02-net

#2.查看容器的网络配置
docker inspect 5a02cd4172da
```



> 测试连接

```
docker exec -it tomcat02 ping tomcat02-net
PING tomcat02-net (192.168.0.2) 56(84) bytes of data.
64 bytes from tomcat02-net.mynet (192.168.0.2): icmp_seq=1 ttl=64 time=0.121 ms
64 bytes from tomcat02-net.mynet (192.168.0.2): icmp_seq=2 ttl=64 time=0.064 ms
^C
--- tomcat02-net ping statistics ---
```

网络连通成功。