

# 介绍

---

Docker是一种类似于与虚拟机技术，是一种虚拟容器技术。  
是一个容器运行载体或称之为容器管理引擎。我们把应用程序和配置依赖打包好形成一个可交付的运行环境，这个打包好的运行环境就似乎image镜像文件。  
但与虚拟机技术不同的是，在Docker容器上运行的程序并不是一个完整的操作系统+软件，Docker舍弃了传统虚拟机容器中需要的Hypervisor硬件虚拟化，运行在Docker容器上的程序直接使用实际物理机的硬件资源。因此在CPU，内存利用率上Docker将会在效率上有明显优势。并且Docker的镜像文件也比平常的镜像文件小很多。

## Docker三要素

Docker的组成架构

### 仓库

是集中存放镜像文件的场所

### 镜像

它可以看作是一个轻量级的Linux操作系统+应用程序，但它省去了硬件Hypervisor虚拟化，而又它只能运行在Docker引擎内。

我们把应用程序和配置依赖打包好形成一个可交付的运行环境，这个打包好的运行环境就似乎image镜像文件。

### 容器

我们通过这个镜像文件生成的实例称为容器，image镜像文件可以看作是容器的模板。同一个image文件可以生成多个同时运行的容器实例。就好比class类可以生成多个实例对象。

## 几个Linux命令

---

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.
- 10.

# Docker的安装

在Centos7环境下安装Docker

Docker从1.13版本之后采用时间线的方式作为版本号，分为社区版CE和企业版EE。

社区版是免费提供给个人开发者和小型团体使用的，企业版会提供额外的收费服务，比如经过官方测试认证过的基础设施、容器、插件等。

1. **Docker 要求 CentOS 系统的内核版本高于 3.10，查看本页面的前提条件来验证你的CentOS 版本是否支持 Docker。**

```
$ sudo uname -r
```

2. 使用 `root` 权限登录 Centos。确保 yum 包更新到最

```
$ sudo yum update
```

3. 卸载旧版本(如果安装过旧版本的话)

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

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

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

对于首次安装Docker的用户，需要下载Docker仓库，之后你才能从Docker仓库上更新或下载Docker。

5. 设置Stable镜像仓库

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

执行完毕后，就会产生一个/etc/yum.repos.d/docker-ce.repo文件，会自动设置镜像仓库链接地址，也就是之后的镜像下载更新地址。

通常一个repo文件定义了一个或者多个软件仓库的[细节]

(<http://gcell.yo2.cn/archives/tag/%E7%BB%86%E8%8A%82>)内容，例如我们将从哪里下载需要[安装]

(<http://gcell.yo2.cn/archives/tag/%E5%AE%89%E8%A3%85>)或者升级的软件包，repo文件中的设置内容将被yum读取和应用！

6. 可以查看所有仓库中所有docker版本，并选择特定版本安

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

## 7. 安装docker

```
$ sudo yum install docker-ce #由于repo中默认只开启stable仓库，故这里安装的是最新稳定版17.12.0
$ sudo yum install <FQPN> # 例如：sudo yum install docker-ce-17.12.0.ce
```

## 8. 启动并加入开机启动

```
$ sudo systemctl start docker #启动
$ sudo systemctl enable docker #开机启动
```

## 9. 验证安装是否成功(有client和服务两部分表示docker安装启动都成功了)

```
$ docker version
```

## 10. 配置镜像加速器

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<- 'EOF'
{
  "registry-mirrors": ["https://cdbvzi1l.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

解决国内网络延迟

# Docker命令

注意，Docker命令必须在管理员权限下执行

## docker help

查看所有Docker命令及解释

### 语法

docker 指定命令 --help #查询指定帮助命令

```
应用程序 位置 终端 星期四 12:20 JR@localhost/home/JR

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
ansibleplaybookbundle/mysql-apb An APB which deploys RHSC MySQL 0 [OK]
cloudfoundry/cf-mysql-ci Image used in CI of cf-mysql-release 0
[root@localhost JR]# docker run --help

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:
  --add-host list          Add a custom host-to-IP mapping (host:ip)
  -a, --attach list        Attach to STDIN, STDOUT or STDERR
  --blkio-weight uint16    Block IO (relative weight), between 10
                           and 1000, or 0 to disable (default 0)
  --blkio-weight-device list Block IO weight (relative device weight)
                           (default [])
  --cap-add list           Add Linux capabilities
  --cap-drop list          Drop Linux capabilities
  --cgroup-parent string   Optional parent cgroup for the container
  --cidfile string         Write the container ID to the file
  --cpu-period int         Limit CPU CFS (Completely Fair Scheduler)
                           period
  --cpu-quota int          Limit CPU CFS (Completely Fair Scheduler)
                           quota
  --cpu-rt-period int      Limit CPU real-time period in microseconds
  --cpu-rt-runtime int     Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int     CPU shares (relative weight)
  --cpus decimal           Number of CPUs
  --cpuset-cpus string     CPUs in which to allow execution (0-3, 0,1)
  --cpuset-mems string     MEMs in which to allow execution (0-3, 0,1)
  -d, --detach             Run container in background and print
                           container ID
  --detach-keys string     Override the key sequence for detaching a
                           container
  --device list            Add a host device to the container
  --device-cgroup-rule list Add a rule to the cgroup allowed devices list
  --device-read-bps list   Limit read rate (bytes per second) from a
                           device (default [])
  --device-read-iops list  Limit read rate (IO per second) from a
                           device (default [])
  --device-write-bps list  Limit write rate (bytes per second) to a
                           device (default [])
  --device-write-iops list Limit write rate (IO per second) to a
                           device (default [])
```

`docker --help` #查询帮助所有Docker命令

`[root@localhost JR]# docker help`

Usage: `docker [OPTIONS] COMMAND`

A self-sufficient runtime for containers

Options:

- `--config string` Location of client config files (default `"/root/.docker"`)
- `-D, --debug` Enable debug mode
- `-H, --host list` Daemon socket(s) to connect to
- `-l, --log-level string` Set the logging level  
(`"debug"` | `"info"` | `"warn"` | `"error"` | `"fatal"`) (default `"info"`)
- `--tls` Use TLS; implied by `--tlsverify`
- `--tlscacert string` Trust certs signed only by this CA (default `"/root/.docker/ca.pem"`)
- `--tlscert string` Path to TLS certificate file (default `"/root/.docker/cert.pem"`)
- `--tlskey string` Path to TLS key file (default `"/root/.docker/key.pem"`)
- `--tlsverify` Use TLS and verify the remote
- `-v, --version` Print version information and quit

Management Commands:

- `config` Manage Docker configs
- `container` Manage containers
- `image` Manage images
- `network` Manage networks

## docker info

查看当前Docker信息，包括所有镜像文件，当前正在运行容器，关闭容器，以及Docker引擎运行环境...

语法

docker info

```
root@localhost JR] # docker info
Containers: 1
Running: 0
Paused: 0
Stopped: 1
Images: 1
Server Version: 18.06.0-ce
Storage Driver: overlay2
Backing Filesystem: xfs
Supports d_type: true
Native Overlay Diff: true
Logging Driver: json-file
Group Driver: cgroupfs
Plugins:
Volume: local
Network: bridge host macvlan null overlay
Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: d64c661f1d51c48782c9cec8fda7604785f93587
runc version: 69663f0bd4b60df09991c08812a60108003fa340
init version: fec3683
Security Options:
```

docker search

查看仓库服务器内的所有该镜像信息

语法

docker search [OPTIONS]

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
mysql	MySQL is a widely used, open-source relation...	6782	[OK]	
mariadb	MariaDB is a community-developed fork of MyS...	2163	[OK]	
mysql/mysql-server	Optimized MySQL Server Docker images. Create...	497		[OK]
percona	Percona Server is a fork of the MySQL relati...	363	[OK]	
zabbix/zabbix-server-mysql	Zabbix Server with MySQL database support	117		[OK]
hypriot/rpi-mysql	RPi-compatible Docker Image with Mysql	93		
zabbix/zabbix-web-nginx-mysql	Zabbix frontend based on Nginx web-server wi...	63		[OK]
centurylink/mysql	Image containing mysql. Optimized to be link...	60		[OK]
landi internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadmin-mysql-5	43		[OK]
centos/mysql-57-centos7	MySQL 5.7 SQL database server	38		
mysql/mysql-cluster	Experimental MySQL Cluster Docker images. Cr...	33		
tutum/mysql	Base docker image to run a MySQL database se...	32		
schickling/mysql-backup-s3	Backup MySQL to S3 (supports periodic backup...	21		[OK]
bitnami/mysql	Bitnami MySQL Docker Image	16		[OK]
zabbix/zabbix-proxy-mysql	Zabbix proxy with MySQL database support	15		[OK]
linuxserver/mysql	A Mysql container, brought to you by LinuxSe...	14		
centos/mysql-56-centos7	MySQL 5.6 SQL database server	9		
openshift/mysql-55-centos7	DEPRECATED: A Centos7 based MySQL v5.5 image...	6		
circlerci/mysql	MySQL is a widely used, open-source relation...	6		
mysql/mysql-router	MySQL Router provides transparent routing be...	3		
openzipkin/zipkin-mysql	Mirror of https://quay.io/repository/openzip...	1		
jetlastic/mysql	An image of the MySQL database server mainta...	1		
cloudposse/mysql	Improved 'mysql' service with support for 'm...	0		[OK]
ansibleplaybookbundle/mysql-apb	An APB which deploys RHSCl MySQL	0		[OK]
cloudfoundry/cf-mysql-ci	Image used in CI of cf-mysql-release	0		

标记

- 1.
- 2.
- 3.

#### OPTIONS (可加参数)

1. --no-trunc: 显示完整的镜像信息

```
docker search --no-trunc 镜像名
```

2. -s: 列出收藏数不小于指定值的镜像

```
docker search -s 500 镜像名
```

3. -automated: 只列出automated build类型的镜像

```
docker search -automated 镜像名
```

4. 可追加参数

```
docker search -s -automated 镜像名
```

## docker login | logout

登录或退出到一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方的Docker Hub

#### 语法

```
docker login [OPTIONS] [SERVER]
```

#### OPTIONS (可加参数)

- 1.
- 2.

#### 案例

```
docker login -u -p registry.cn-jr.aliyuncs.com
```

备注：登录阿里云docker远程仓库

## docker tag

标记本地镜像，将其归入某一仓库

## 语法

```
docker tag 镜像ID 仓库名[:版本号]
```

## 案例

```
docker tag xxx registry.cn-jr.aliyuncs.com/仓库名: 版本号
```

# docker pull

## 下载镜像

## 语法

- **不指定TAG（默认下载latest最新版本）**

```
docker pull mysql
```

- **指定TAG（指定镜像版本）**

```
docker pull mysql:TAG
```

# docker push

将本地的镜像上传到镜像仓库，前提是先登录镜像仓库，并将上传的镜像打上tag

## 语法

```
docker push [OPTIONS] 仓库名[:版本] #可不加版本，默认为latest
```

## OPTIONS（可加参数）

1.

# docker images

查看本地仓库下所有的镜像文件

## 语法

```
docker images [OPTIONS]
```

```
[root@localhost JR]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest             2cb0d9787c4d       5 weeks ago        1.85kB
[root@localhost JR]#
```

标记

- 1.
- 2.
- 3.
- 4.

OPTIONS (可加参数)

1. -a :列出本地所有镜像

```
docker images -a
```

2. -q :只显示镜像ID

```
docker images -q
```

3. --digests: 显示镜像的摘要信息

```
docker images --digests
```

4. --no-trunc: 显示完整的镜像信息

```
docker images --no-trunc
```

5. 可追加参数, 如:

```
[root@localhost JR]# docker images -aq
2cb0d9787c4d
-
[root@localhost JR]# docker images -a --no-trunc
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest             sha256:2cb0d9787c4dd17ef9eb03e512923bc4db10add1
90d3f84af63b744e353a9b34    5 weeks ago        1.85kB
```

## docker rmi

删除镜像

语法

- 删除单个镜像

```
docker rmi 镜像ID
```



- 删除多个镜像

```
docker rmi 镜像名1: TAG 镜像名2: TAG
```

- 动态删除多个容器

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

#### 可加参数

- -f

```
docker rmi -f 镜像ID
```

强制删除，即使存在正在运行的该镜像容器

## docker run

新建并启动一个容器

#### 语法

```
docker run [OPTIONS] 镜像ID [COMMAND] [ARG...]
```

OPTIONS (可加参数)

[^--expose=[]]: 开放一个端口或一组端口

我们在使用Docker的时候，经常可能需要连接到其他的容器，比如：web服务需要连接数据库。按照往常的做法，需要先启动数据库的容器，映射出端口来，然后配置好客户端的容器，再去访问。其实针对这种场景，Docker提供了-link 参数来满足。

1. ip:主机（宿主）端口:容器端口
2. ip::容器端口
3. 主机（宿主）端口:容器端口
4. 容器端口

#### COMMAND

```
docker run -it centos /bin/bash
```

启动后在容器内执行/bin/bash

#### 注意

1. Docker容器后台运行，必须有一个前台进程，否则就会自动退出

## docker create

创建一个新的容器但不启动它

语法

```
docker create [OPTIONS] 镜像ID [执行命令] [启动参数]
```

OPTIONS (可加参数)

如同docker run

## docker ps

查看docker创建的容器，类似于linux中的ps。不加-a，默认列出当前正在运行的容器进程信息。

语法

```
docker ps [OPTIONS]
```

OPTIONS (可加参数)

```
docker ps -f before=镜像ID #before过滤器在容器9c3527ed70ce之前创建的容器
```

```
docker ps -f since=镜像ID #since过滤器显示在给定容器id或名称之后创建的容器
```

```
docker ps -n 5
```

## docker start

启动已经被停止的容器

语法

```
docker start [OPTIONS] 容器ID [启动参数...]
```

## docker restart

重新启动容器

语法

```
docker restart [OPTIONS] 容器ID [启动参数...]
```

## docker stop

停止一个正在运行的容器，正常关闭

语法

```
docker stop [OPTIONS] 容器ID [启动参数...]
```

## docker kill

直接杀死容器进程，相当于直接拔掉电源

语法

```
docker kill [OPTIONS] CONTAINER [启动参数...]
```

## docker pause

暂停容器中的所有进程

语法

- 暂停单个容器

```
docker pause [OPTIONS] 容器ID
```

- 暂停多个容器

```
docker pause [OPTIONS] 容器ID...
```

- 动态暂停多个容器

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

## docker unpause

恢复容器中的所有进程

#### 语法

- 恢复单个容器

```
docker pause [OPTIONS] 容器ID
```

- 恢复多个容器

```
docker pause [OPTIONS] 容器ID...
```

- 动态恢复多个容器

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

## 退出当前交互式容器

- exit  
退出并关闭容器
- ctrl+P+Q  
退出并保持容器以后台进行

## docker rm

删除一个或多个容器

#### 语法

- 删除单个容器

```
docker rm [OPTIONS] 容器ID
```

- 删除多个容器

```
docker rm [OPTIONS] 容器ID...
```

- 动态删除多个容器

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

OPTIONS (可加参数)

- 1.
- 2.

## docker logs

获取容器的日志，也就是容器的echo输出

语法

```
docker logs [OPTIONS] 容器ID
```

OPTIONS (可加参数)

```
docker logs --since="2016-07-01" --tail=10 mynginx
```

备注：查看容器mynginx从2016年7月1日后的最新10条日志

## docker top

查看容器内正在运行的进程，类似于Linux的top

语法

```
docker top 容器ID
```

## docker inspect

查看容器内部细节，包括容器ID、创建时间、运行命令、运行参数、容器数据卷映射...

语法

```
docker inspect [OPTIONS] 容器ID
```

OPTIONS (可加参数)

- 1.
- 2.
- 3.

## docker attach

进入正在运行的容器

语法

```
docker attach [OPTIONS] 容器ID
```

## docker exec

进入正在运行的容器后执行参数命令

语法

```
docker exec [OPTIONS] 容器ID 执行命令 [ARG...]
```

- 进入centos容器并查看当前目录结构，返回结果退出

```
docker exec centos ls -l
```

- 进入centos容器启动命令终端，不退出

```
docker exec centos /bin/bash
```

OPTIONS (可加参数)

- 1.
- 2.
- 3.

## docker cp

用于容器与主机之间的数据拷贝

语法

- 主机拷向容器

```
docker cp [OPTIONS] 主机保存路径 容器ID: 文件路径
```

- 容器拷向主机

```
docker cp [OPTIONS] 容器ID: 文件路径 主机保存路径
```

## docker commit

以容器为模板创建一个新的镜像

#### 语法

```
docker commit [OPTIONS] 容器ID [REPOSITORY[:TAG]]
```

#### OPTIONS (可加参数)

- 1.
- 2.
- 3.
- 4.

#### 实例

```
docker commit -a "runoob.com" -m "my apache" a404c6c174a2 mymysql:v1
```

将容器a404c6c174a2 保存为新的镜像,并添加提交人信息和说明信息。若不指定镜像名:TAG,默认根据容器来创建。

## docker build

命令用于使用 Dockerfile 创建镜像

#### 语法

- 使用当前目录的 Dockerfile 创建镜像

```
docker build [OPTIONS] PATH
```

- 使用URL 的 Dockerfile 创建镜像

```
docker build [OPTIONS] URL
```

- 也可以通过 -f Dockerfile 文件的位置创建镜像

```
docker build [OPTIONS] -f /path/to/a/Dockerfile .
```

.: 表示保存在当前目录下

#### OPTIONS (可加参数)

1. [^--build-arg=[]]: 设置镜像创建时的变量
- 2.
- 3.
- 4.
- 5.

## 镜像的原理

它是一种联合文件系统，也就是类似于文件夹，一个一个子目录嵌套成一个完整的目录。放在镜像当中理解就是，第一层镜像就好似一个根目录，在子下存在多个子镜像。

采用这种架构的好处就是能够**共享资源**。比如：

有多个镜像都从相同的base镜像构建而来，那么宿主机只需要在磁盘上保存一份base镜像。同时在内存中只需要加载一份base镜像，就可以为所有容器服务了。而且镜像每一层都可以被共享。

在使用中有一个现象，当你一开始下载的时候你会觉得特别慢，但到之后下的时候，你就会发觉到很快，就是因为镜像这种架构的存在。

## 容器数据卷

### 介绍

Docker容器产生的数据，如果不同docker commit生成新的镜像，使得数据作为镜像的一部分保存下来。

那么当容器关闭，随即也会将产生的数据删除。

**1.容器数据卷能够将数据进行持久化，将其映射到主机目录上，实现容器资源与主机共享。**

**2.并使之也能够在容器间进行数据共享**

### 实现

-v参数命令创建

- 不带权限（容器可读读写）

```
docker run|create -v /宿主机目录绝对路径: /容器内目录 镜像ID
```

- 带权限（容器只读不能写）

```
docker run|create -v /宿主机目录绝对路径: /容器内目录:ro 镜像ID
```



## 使用Dockerfile创建

### 1. 创建Dockerfile文件

```
FROM centos
VOLUME ["容器数据卷目录1", "容器数据卷目录2"]
CMD echo "finish,-----success"
CMD /bin/bash
```

### 2. 使用docker build命令构建新的镜像

## 查看数据卷是否挂在成功

创建之后就会在宿主机和容器下的系统个创建一个数据卷目录。

可通过docker inspect查看容器完整信息。

## Dockerfile

### 介绍

Dockerfile是用来构建Docker镜像的构建文件，用Java的思维想，就是如果需要创建一个类的实例，首先必须要有一个类的class声明，这个class默认的会继承一个Object类，之后在根据开发者的需求再往class中添加各种属性，最终通过new语法创建出了一个实例对象。

在镜像层面上理解的话，同样都属于构建文件，也就是所有的镜像是起源于一个基础镜像，之后根据开发者的需求在往环境添加工具，比如说jdk、mysql、tomcat...。最终通过build命令创建一个可使用的镜像。

### build构建过程

1. docker从基础镜像运行一个容器
2. 执行一条执行并对容器做出修改
3. 执行类似docker commit的操作提交一个新的镜像层
4. docker 再基于刚提交的容器运行一个新容器
5. 从第二部开始重新对新容器执行dockerfile中的下一条指令，直到所有指令都执行完成

### Dockerfile保留字指令

下面说到的命令的执行，这些命令的执行是在类似于linux终端上执行的，比如：ls（列出当前所有目录）、pwd（当前所在命令）、yum...（软件包操作命令）等...

- FROM (from)  
基础镜像
- MAINTAINER (maintainer)  
作者
- ENV (env)

环境变量，这个环境变量可以在后续的任何RUN等其他指令中使用，如：

```
ENV    MY_PATH /root
WORKDIR $MY_PATH
```

当设置JDK等的环境变量也需要使用到该指令

- RUN (run)  
**容器构建时**在新构建后运行的容器上执行的命令
- EXPOSE (expose)  
暴露的端口
- WORKDIR (workdir)  
工作目录，也就是进入容器后通过pwd查看到目录
- ADD (add)  
往容器添加文件，若是压缩文件的话，就会解压
- COPY (copy)  
同样也是往容器添加文件，但也是压缩文件的话，不会解压
- VOLUME (volume)  
容器数据卷映射名
- CMD (cmd)  
**镜像构建之后运行的容器启动时**需要执行的命令，Dockerfile中可以存在多条CMD指令，那么最后一条也就是最终构建出来的镜像所运行时所执行的命令。

若容器运行时添加了运行参数命令，优先执行该命令，不执行Dockerfile中的CMD命令

- ENTRYPOINT (entrypoint)  
同样也是**容器启动时**需要执行的命令，与CMD不同的是，

若容器运行时添加了运行参数命令，ENTRYPOINT中的命令同样会得到执行

- ONBUILD (onbuild)  
若通过该Dockerfile构建出来的镜像之后被作为基础镜像所构建，那么当被构建时（docker build）将执行ONBUILD声明的命令

## 案例

构建一个有vim编辑器的centos

- 创建centos\_Dockerfile

```
FROM centos
MAINTAINER JR<1336913542@qq.com>

ENV MY_WORKDIR /usr/local
WORKDIR $MY_WORKDIR

RUN yum -y install vim #下载vim编辑器

EXPOSE 80

CMD echo "BULID SUCCESS-----SUCCESS!!!"
CMD /bin/bash
```

- 使用docker build命令构建新镜像

```
docker build -f /myCentos/centos_Dockerfile -t my_centos . # .:表示当前目录
```

- 结果

```
Complete!
Removing intermediate container be4bd33abe7f
---> 146405e95dd3
Step 6/8 : EXPOSE 80
---> Running in ee8fceb78c30
Removing intermediate container ee8fceb78c30
---> 3c0af312b838
Step 7/8 : CMD echo "BULID SUCCESS-----SUCCESS!!!"
---> Running in 9a033e040c8e
Removing intermediate container 9a033e040c8e
---> 390f8fcf4bb7
Step 8/8 : CMD /bin/bash
---> Running in daf8d49d2e97
Removing intermediate container daf8d49d2e97
---> 414fddf08c94
Successfully built 414fddf08c94
Successfully tagged my_centos:latest
```

## 安装并运行Redis

### 1. 下载Redis镜像

```
docker push redis #默认下载最新版本
```

### 2. 创建并运行Redis容器

```
docker run                                #创建并启动容器
-p 6379:6379                              #容器端口映射
-v /JR/myredis/data:/data                 #映射容器数据卷
-v /JR/myredis/conf/redis.conf:/usr/local/etc/redis/redis.conf
-d                                         #后台运行
redis:tag                                 #指定镜像名
redis-server /usr/local/etc/redis/redis.conf --appendonly yes #执行命令和启动参数
```

## 提示

映射数据卷容器中的data文件是redis中持久化文件，也就是数据从内存持久化到磁盘中的数据，比如set k1 v1，那么最终这些数据保存到磁盘中就是在该目录下。

映射的redis.conf文件的好处是，如果我们需要修改容器内的redis软件的配置，则不需要进入到容器内修改，而是直接在宿主机的映射目录下修改即可。

redis-server是启动容器时的执行命令，表示启动容器后，再容器系统内启动redis服务。

### 3. 进入容器内的redis客户端

```
docker exec           #进入容器内执行命令
--it                  #开启终端运行容器
容器ID                #真正运行的容器ID
redis-cli             #执行命令（开启客户端）
```