1、上传至公有仓库

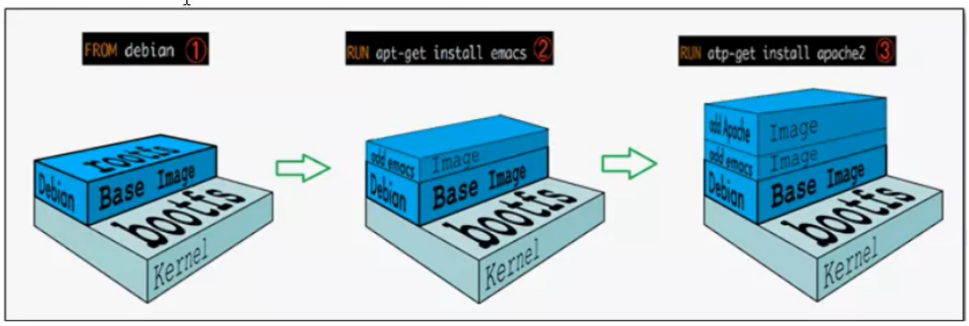
1.1、将容器制作镜像

docker commit -m="提交说明" -a="作者" 容器ID 镜像作者/镜像名:版本号

```
# docker commit -m="提交说明" -a="作者" 容器ID  镜像作者/镜像名:版本号
docker commit -m="we add vim" -a="xxr" 6c8c54982b4e  xxr/myubuntu:1.1

# 查看镜像，注意对比镜像大小
[root@iZf8z3pu4d7ueh3i6pzv5fZ ~]# docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
xxr/myubuntu    1.1       8075463c9ad5   2 minutes ago    174MB
xxr/ubuntu      1.0       d37efa8a6d92   43 minutes ago   77.8MB
ubuntu          latest    2dc39ba059dc   3 weeks ago      77.8MB
```

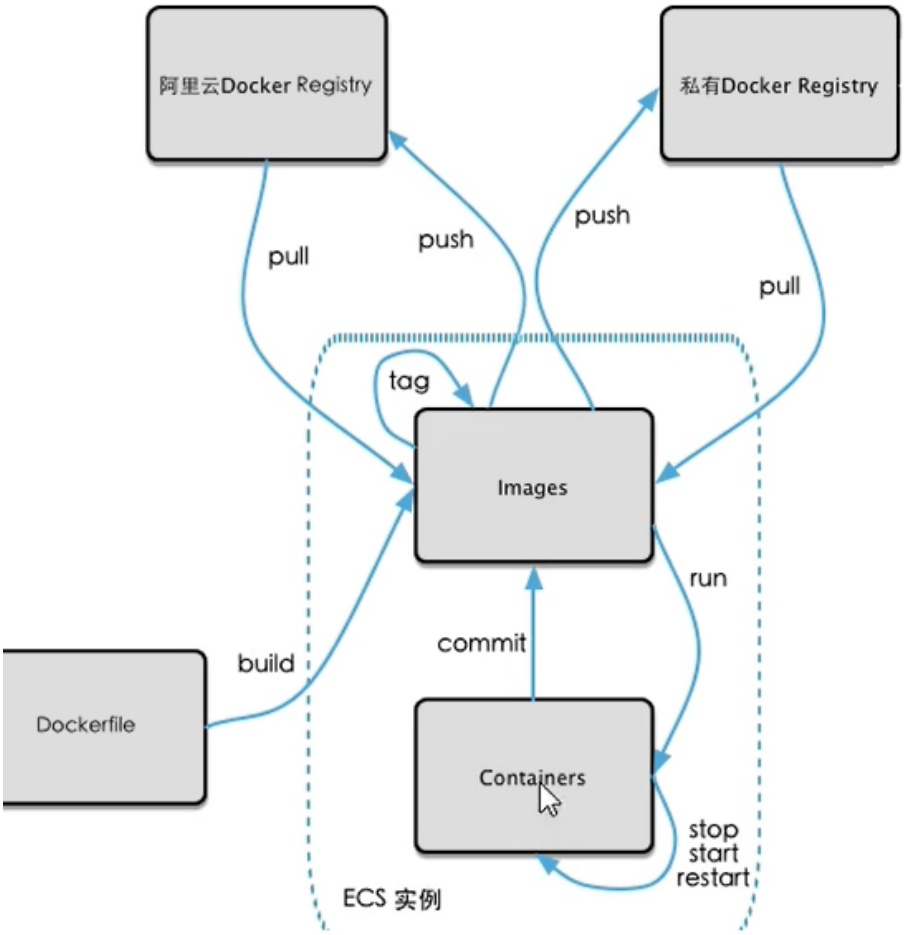
docker中的镜像分层，支持通过扩展现有镜像，创建新的镜像。类似Java继承于一个Base基础类，自己再按需扩展。

镜像是从 base 镜像一层一层叠加生成的。每安装一个软件，就在现有镜像的基础上增加一层



1.2、推送镜像到阿里云上

> 类似于git仓库



> 官方操作指南

1. 登录阿里云Docker Registry

```
$ docker login --username=aliyun5599276666 registry.cn-shenzhen.aliyuncs.com
```

用于登录的用户名为阿里云账号全名，密码为开通服务时设置的密码。

您可以在访问凭证页面修改凭证密码。

2. 从Registry中拉取镜像

```
$ docker pull registry.cn-shenzhen.aliyuncs.com/docker_learning_basic/my-ubuntu:
[镜像版本号]
```

3. 将镜像推送到Registry

```
$ docker login --username=aliyun5599276666 registry.cn-shenzhen.aliyuncs.com
$ docker tag [ImageId] registry.cn-
shenzhen.aliyuncs.com/docker_learning_basic/my-ubuntu:[镜像版本号]
$ docker push registry.cn-shenzhen.aliyuncs.com/docker_learning_basic/my-ubuntu:
[镜像版本号]
```

请根据实际镜像信息替换示例中的[ImageId]和[镜像版本号]参数。

4. 选择合适的镜像仓库地址

从ECS推送镜像时，可以选择使用镜像仓库内网地址。推送速度将得到提升并且将不会损耗您的公网流量。

如果您使用的机器位于VPC网络，请使用 registry-vpc.cn-shenzhen.aliyuncs.com 作为Registry的域名登录。

5. 示例

使用"docker tag"命令重命名镜像，并将它通过专有网络地址推送至Registry。

```
$ docker imagesREPOSITORY
  TAG                 IMAGE ID            CREATED             VIRTUAL
SIZEregistry.aliyuncs.com/acs/agent                                    0.7-
dfb6816         37bb9c63c8b2        7 days ago          37.89 MB
$ docker tag 37bb9c63c8b2 registry-vpc.cn-shenzhen.aliyuncs.com/acs/agent:0.7-
dfb6816
```

使用 "docker push" 命令将该镜像推送至远程。

```
$ docker push registry-vpc.cn-shenzhen.aliyuncs.com/acs/agent:0.7-dfb6816
```

1.3、配置镜像加速器

> 链接

容器镜像服务 (aliyun.com)

> 操作指南

1. 安装 / 升级Docker客户端

推荐安装1.10.0以上版本的Docker客户端，参考文档docker-ce

2. 配置镜像加速器

针对Docker客户端版本大于 1.10.0 的用户

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

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

2、搭建私有库

1. 下载registry镜像

```
docker pull registry
```

2. 运行

```
# 这里涉及到容器卷挂载，在后文说明 注意 -v 本主机目录 : 容器的内部的目录位置
docker run -d -p 5000:5000  -v /myregistry/:/tmp/registry --privileged=true
registry
```

3. 制作一个带有ifconfig功能的ubuntu镜像

```
docker run -it --name="myu1" ubuntu /bin/bash
apt-get update
apt-get install net-tools
# 查看容器ID号
docker ps -a
# 提交
docker commit -m=" we add ifconfig " -a="xxr" 6c8c54982b4e xxr/my-ubuntu:1.2
```

4. 查看私库的镜像

```
# curl -XGET http://host:5000/v2/_catalog
curl -XGET http://localhost:5000/v2/_catalog
```

5. 修改镜像tag：公式： docker tag 镜像:Tag Host:Port/Repository:Tag

```
docker tag  xxr/my-ubuntu:1.2  localhost:5000/my-ubuntu:1.2
```

6. 修改配置文件使之支持http推送

```
vim /etc/docker/daemon.json
{
  "registry-mirrors": ["阿里云加速链接"],
  "insecure-registries": ["ip:5000"]
}
```

7. 重启docker和私库容器使配置生效

```
systemctl  restart docker

docker run -d -p 5000:5000  -v /myregistry/:/tmp/registry --privileged=true
registry
```

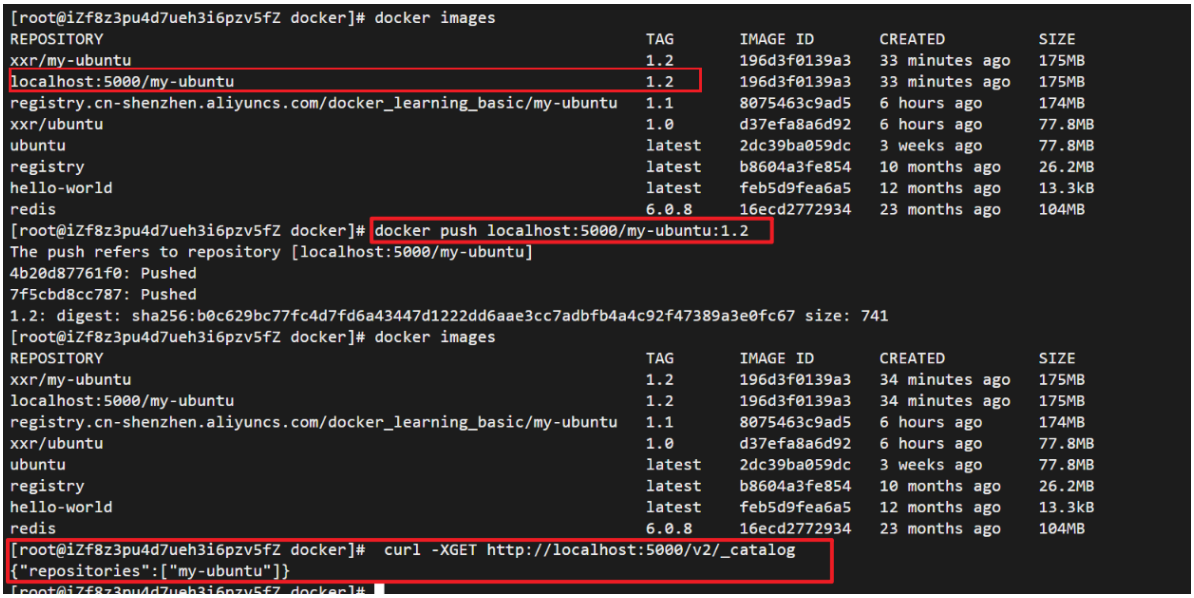
8. 推送至私库

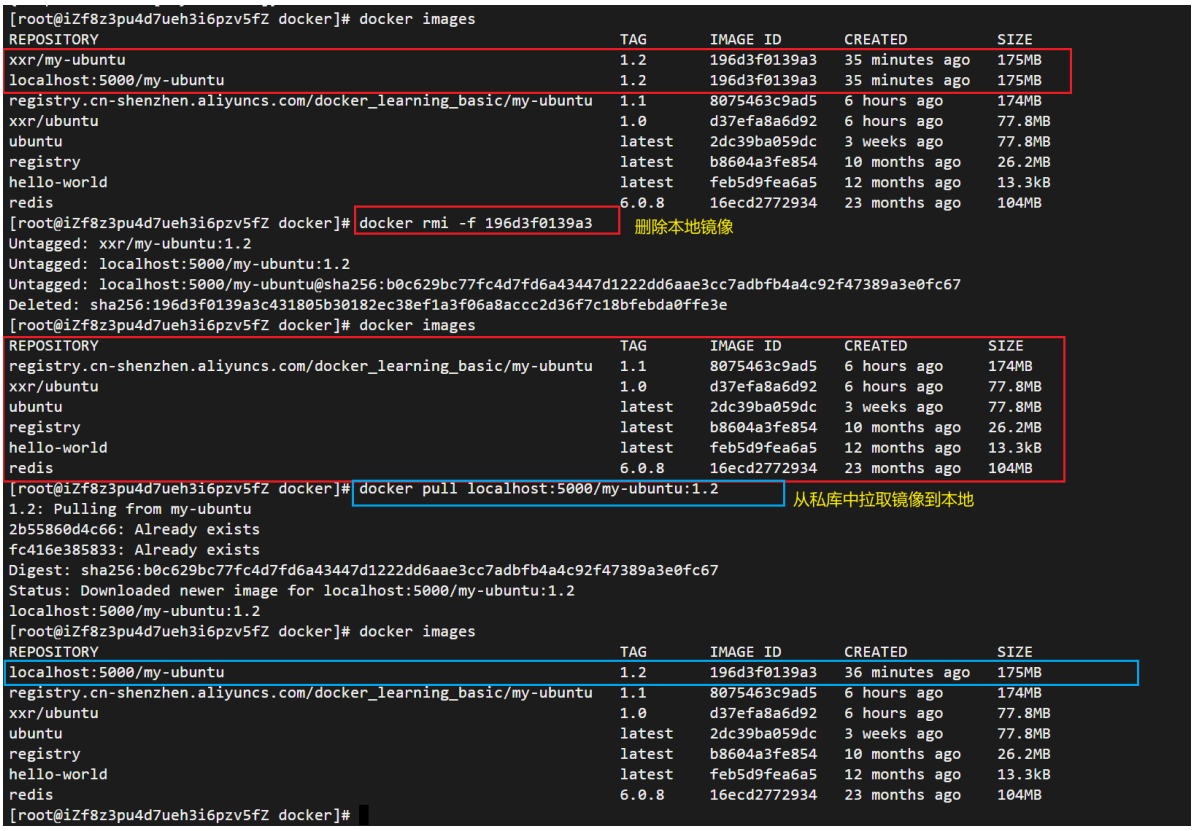
```
docker push ip:port/镜像名:版本号
# 再次验证私库是否添加了该镜像
 curl -XGET http://localhost:5000/v2/_catalog
```

9. 从私库拉取镜像到本地

```
docker pull ip:port/镜像名:版本号
```

示意图：





docker commit 和 docker tag 区别：commit 是将某容器变成镜像，tag 是将某镜像和某容器镜像云（包含私服）更名对应起来

首先发送命令到docker host根据镜像产生容器的同时于虚拟机的5000端口进映射关联,然后将需要传递的容器进行打包为镜像,再将镜像按照规范进行更名，发布到映射端口,而后拉取

3、容器数据卷的简单使用

3.1、特点

卷就是目录或文件，存在于一个或多个容器中，由Docker挂载到容器，但卷不属于联合文件系统（Union FileSystem），因此能够绕过联合文件系统提供一些用于持续存储或共享数据的特性:。

卷的设计目的就是数据的持久化，完全独立于容器的生存周期，因此Docker不会在容器删除时删除其挂载的数据卷。

数据卷的特点:

- 数据卷可在容器之间共享或重用数据
- 卷中的更改可以直接生效
- 数据卷中的更改不会包含在镜像的更新中
- 数据卷的生命周期一直持续到没有容器使用它为止

3.2、基本使用

-v挂载的文件目录如果不存在会自动创建

Docker容器数据卷详解博客docker数据卷

读写（默认）： docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录:rw 镜像名

```
docker run -it --privileged=true -v /tmp/host_data:/tmp/volume_data --name=u1
ubuntu
# 发现在主机中对/tmp/host_data目录内容的读写和容器的/tmp/volume_data目录内容是互通的，应该
都是指向主机/tmp/host_data，就算容器被下线或摧毁都不会改变主机对应的目录内容，重新上线仍然满足
同步和恢复。
```

只读（ ro）： docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录:ro 镜像名

```
docker run -it --privileged=true -v /tmp/host_data:/tmp/volume_data:ro --name=u2
ubuntu

# 只读
root@00cc13ea8047:/tmp/volume_data# touch u2.txt
touch: cannot touch 'u2.txt': Read-only file system
root@00cc13ea8047:/tmp/volume_data# cat volume_data
hello, i am u1
root@00cc13ea8047:/tmp/volume_data# echo "hello, i am u2" > volume_data
bash: volume_data: Read-only file system
```

容器数据卷中的"继承"关系

```
docker run -it  --privileged=true --volumes-from 父类容器名  --name u2 ubuntu
# 例如让我们的u1继承上可读写的u1，其实就是继承了-v的属性而已，一旦确认，u1与u2其实只是共用一套挂载的参数，不会随任何一方（包括自身）的停止或重启而失效
```

查看：

```
# docker inspect 容器id 就能找到挂载目录
docker inspect 容器ID
```

4、常用软件的下载

4.1、tomcat

```
docker search tomcat
docker pull tomcat
docker images tomcat
# 如果是下载的是最新版的tomcat镜像如10.0.14，那么直接输入host:8080是无法访问的，因为猫的页面在wepapp.list文件夹而不是wepapp文件夹，只要将weppapp.list文件夹覆盖掉wepapp文件夹后，再开放防火墙对应端口的前提下即可访问tomcat猫页面

docker run -d -p 8080:8080 --name t1 tomcat
[root@iZf8z3pu4d7ueh3i6pzv5fZ docker]# docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS          PORTS
                  NAMES
a692f75067ce   tomcat    "catalina.sh run"   2 minutes ago   Up 2 minutes
0.0.0.0:8080->8080/tcp   t1

# 还是下回tomcat8版本~，先删对应的镜像和t1容器吧！
docker rmi -f tomcat
docker rm -f t1

docker pull billygoo/tomcat8-jdk8
docker run -d -p 8080:8080 --name t1 billygoo/tomcat8-jdk8
# or
```

```
docker run -d -p 8080:8080 --name t1 30ef4019761d
```

4.2、mysql

快速创建并运行容器：

```
docker pull mysql:8.0.30
# 初始化密码为 123456 ，在后台运行
# docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -p 主机端口:容器端口 -d mysql:tag
docker run --name test-mysql -e MYSQL_ROOT_PASSWORD=123456 -p 3306:3306 -d mysql:8.0.30
# 进入容器，前台交互
docker exec -it test-mysql /bin/bash
mysql -uroot -p
enter password: 123456
mysql> ... # 进行sql操作
```

两个问题：

1. 乱码

   SHOW VARIABLES LIKE 'character%';

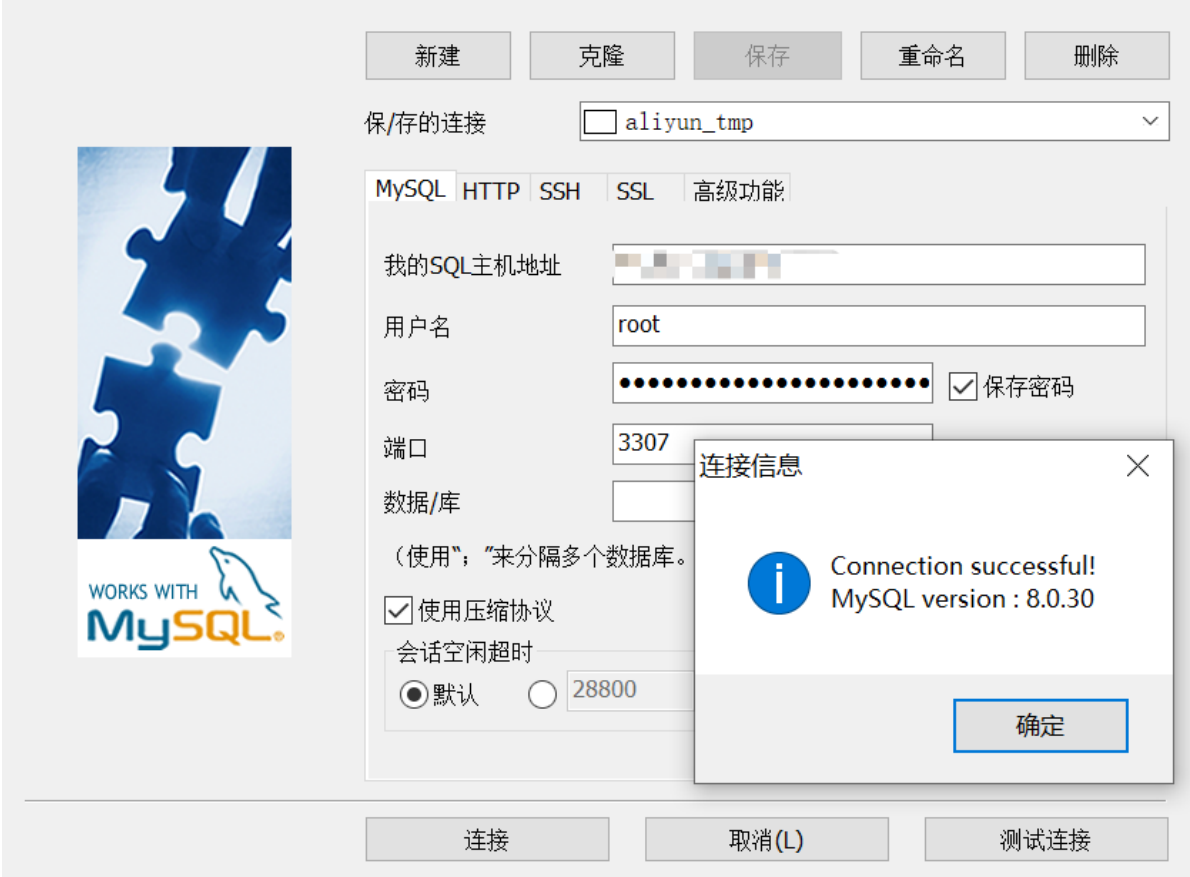
2. 数据备份

> 应用**容器数据卷**解决上述两个问题

```
# 小细节，宿主机端口3306改为3307提高主机安全，设置多个挂载点保证数据可恢复，在云服务器控制台放行3307端口即可
docker run -d -p 3307:3306 --privileged=true -v /tmp/mysql/log:/var/log/mysql -v /tmp/mysql/data:/var/lib/mysql -v /tmp/mysql/conf:/etc/mysql/conf.d -e MYSQL_ROOT_PASSWORD=YOUR-PASSWORD --name test-mysql mysql:8.0.30

# 格式化如下：
docker run -d -p 3307:3306 --privileged=true
-v /tmp/mysql/log:/var/log/mysql
-v /tmp/mysql/data:/var/lib/mysql
-v /tmp/mysql/conf:/etc/mysql/conf.d
-e MYSQL_ROOT_PASSWORD=YOUR-PASSWORD
--name test-mysql mysql:8.0.30


# 在主机下
vim /tmp/mysql/conf/my.cnf
# 写入如下内容
[client]
default_character_set=utf8
[mysqld]
```

```
collation_server = utf8_general_ci
character_set_server = utf8


# 重启test-mysql容器并用exec前台交互式进入
docker restart test-mysql # docker ps | grep test-mysql验证运行中
docker exec -it test-mysql /bin/bnash #前台交互式
cat /etc/mysql/conf.d/my.cnf   # 验证容器数据卷正常工作，my.cnf文件数据"同步"成功


# 在test-mysql容器中登录mysql
mysql -uroot -p
enter password:
mysql>

# 远程连接mysql的相关配置修改
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'YOUR-PASSWORD';

flush privileges;
```

连接成功：



注意：

这种利用容器数据卷方式只能恢复已经落地到数据库本地文件的数据，如果数据还在运行的，需要借助binlog进行恢复

4.3、redis

```
# 直接pull最新的redis镜像
docker pull redis
# 创建redis容器实例，注意容器的数据卷挂载位置
docker run --restart=always --log-opt max-size=100m --log-opt max-file=2 -p
6380:6379 --name=test-redis -v
/home/redis/myredis/myredis.conf:/etc/redis/redis.conf -v
/home/redis/myredis/data:/data -d redis redis-server /etc/redis/redis.conf --
appendonly yes --requirepass 1=ebd913e4f~a520a5@Wait

# 宿主机下
mkdir -p /home/redis/myredis
mkdir data
vim myredis.conf  #见文末

# 查看启动状态
docker ps -a |grep test-redis
# 查看此容器30分钟之内的日志情况
docker logs --since 30m test-redis

docker exec -it test-redis redis-cli
# 验证密码
auth 1=ebd913e4f~a520a5@Wait
```

`redis.conf`文件如下:

```
# bind 192.168.1.100 10.0.0.1
# bind 127.0.0.1 ::1
#bind 127.0.0.1

protected-mode no

port 6379

tcp-backlog 511

requirepass 1=ebd913e4f~a520a5@Wait

timeout 0

tcp-keepalive 300

daemonize no

supervised no

pidfile /var/run/redis_6379.pid

loglevel notice

logfile ""
```

```
databases 30

always-show-logo yes

save 900 1
save 300 10
save 60 10000

stop-writes-on-bgsave-error yes

rdbcompression yes

rdbchecksum yes

dbfilename dump.rdb

dir ./

replica-serve-stale-data yes

replica-read-only yes

repl-diskless-sync no

repl-disable-tcp-nodelay no

replica-priority 100

lazyfree-lazy-eviction no
lazyfree-lazy-expire no
lazyfree-lazy-server-del no
replica-lazy-flush no

appendonly yes

appendfilename "appendonly.aof"

no-appendfsync-on-rewrite no

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb

aof-load-truncated yes

aof-use-rdb-preamble yes

lua-time-limit 5000

slowlog-max-len 128

notify-keyspace-events ""

hash-max-ziplist-entries 512
hash-max-ziplist-value 64

list-max-ziplist-size -2
```

```
list-compress-depth 0

set-max-intset-entries 512

zset-max-ziplist-entries 128
zset-max-ziplist-value 64

hll-sparse-max-bytes 3000

stream-node-max-bytes 4096
stream-node-max-entries 100

activerehashing yes

hz 10

dynamic-hz yes

aof-rewrite-incremental-fsync yes

rdb-save-incremental-fsync yes
```