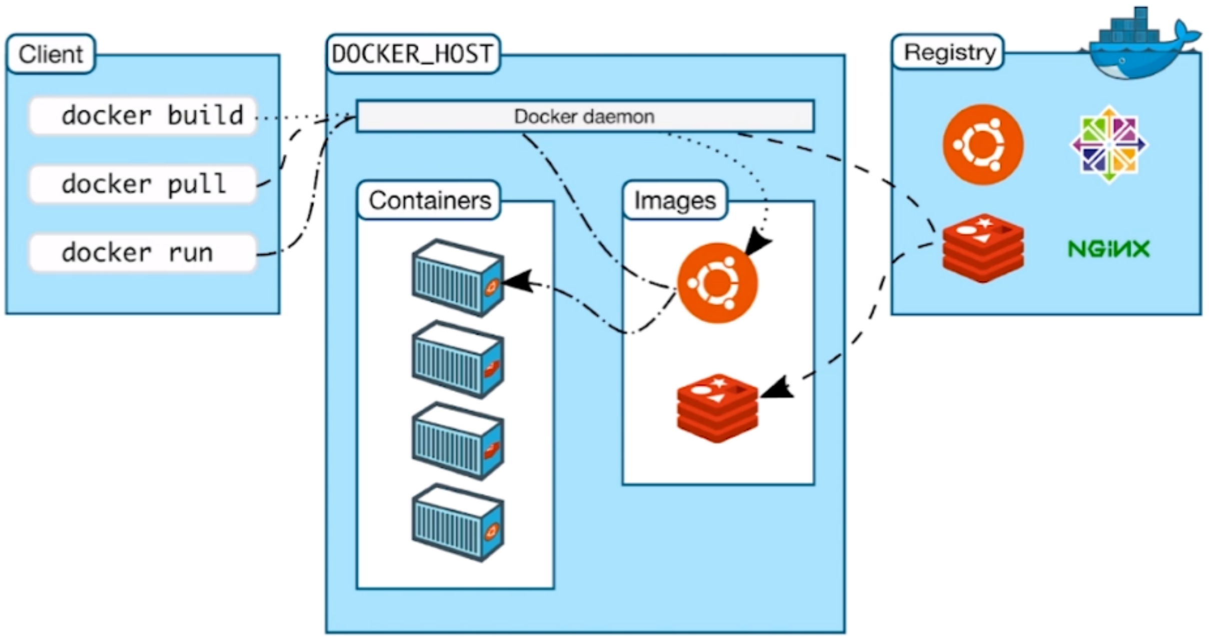
Docker学习

1、Docker安装

1.1 docker的基本组成

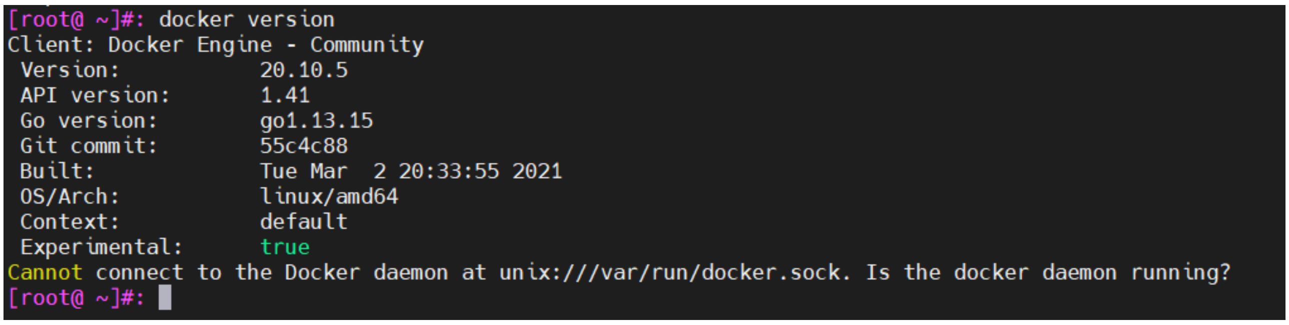


1.2 查看linux的版本

```
# 查看linux版本
[root@ ~]#: uname -r
3.10.0-514.26.2.el7.x86_64
```

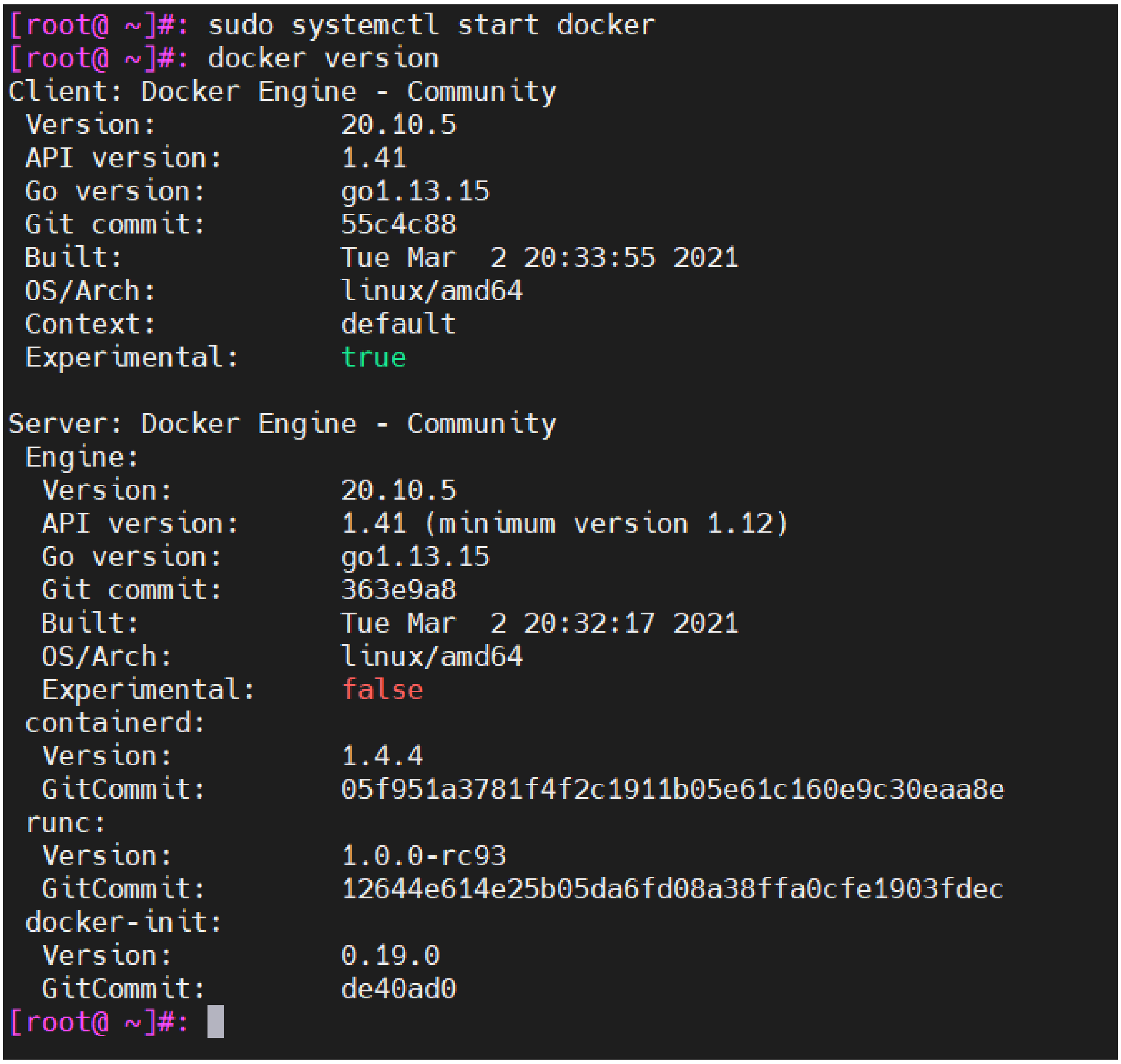
1.3 根据官方文档进行安装

```
# 1、uninstall old versions
sudo yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine

# 2、Install using the repository
sudo yum install -y yum-utils

# 3、set up the stable repository
sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo # 默认是国外的镜像

sudo yum-config-manager \
    --add-repo \
    http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
# 推荐使用阿里云镜像

# 升级yum包索引
yum makecache fast

# 4、安装docker ce是社区版，ee是企业版
sudo yum install docker-ce docker-ce-cli containerd.io

# 也可以指定版本实现，具体看官方文档
```



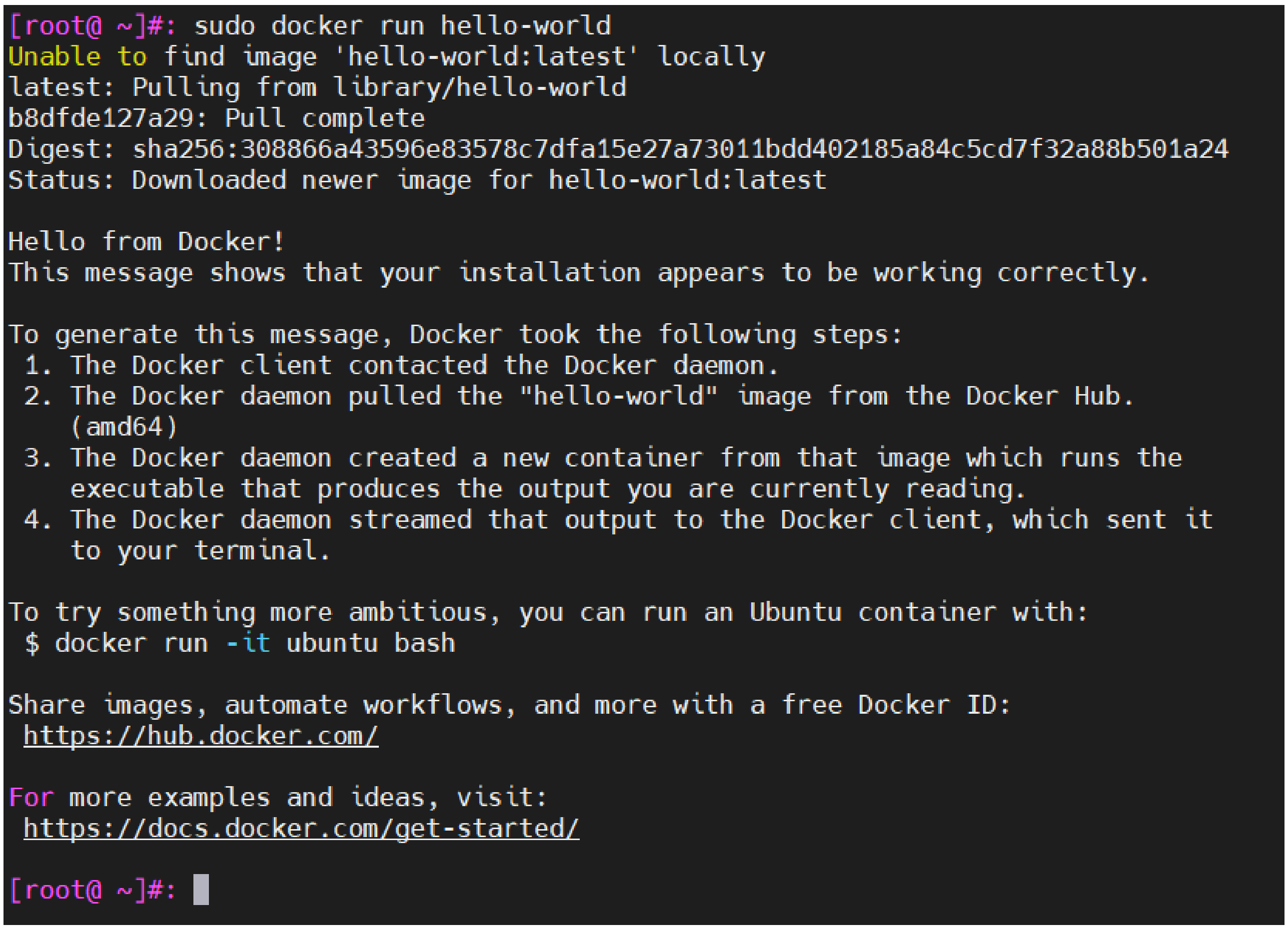
```
[root@ ~]#: docker version
Client: Docker Engine - Community
 Version:           20.10.5
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        55c4c88
 Built:             Tue Mar  2 20:33:55 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
[root@ ~]#:
```

```
# 5、启动docker
sudo systemctl start docker
```



```
[root@ ~]#: sudo systemctl start docker
[root@ ~]#: docker version
Client: Docker Engine - Community
 Version:           20.10.5
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        55c4c88
 Built:             Tue Mar  2 20:33:55 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.5
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       363e9a8
  Built:            Tue Mar  2 20:32:17 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.4
  GitCommit:        05f951a3781f4f2c1911b05e61c160e9c30eaa8e
 runc:
  Version:          1.0.0-rc93
  GitCommit:        12644e614e25b05da6fd08a38ffa0cfe1903fdec
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
[root@ ~]#:
```

```
# 6、运行hello world镜像
sudo docker run hello-world
```



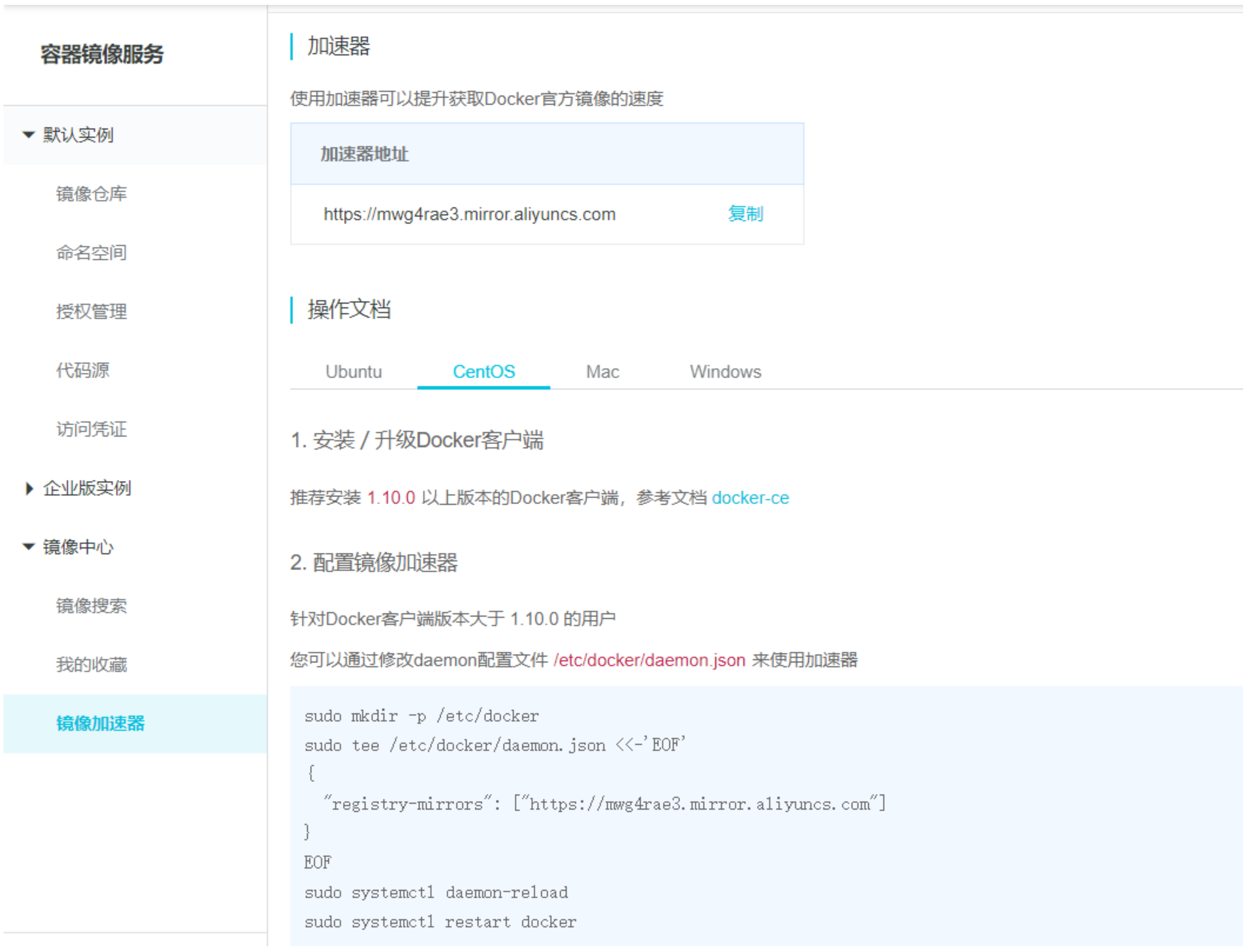
```
[root@ ~]#: sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:308866a43596e83578c7dfa15e27a73011bdd402185a84c5cd7f32a88b501a24
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[root@ ~]#: 
```

```
# 7、 查看下载的镜像
[root@ ~]#: docker images
REPOSITORY     TAG        IMAGE ID       CREATED       SIZE
hello-world    latest     d1165f221234   8 days ago    13.3kB
```

如果要卸载docker

```
# 1、Uninstall the Docker Engine, CLI, and Containerd packages:
sudo yum remove docker-ce docker-ce-cli containerd.io

# 2、Images, containers, volumes, or customized configuration files
on your host are not automatically removed. To delete all images,
containers, and volumes:
sudo rm -rf /var/lib/docker
sudo rm -rf /var/lib/containerd
```

1.4 阿里云镜像加速



加速器

使用加速器可以提升获取Docker官方镜像的速度

加速器地址	
https://mwg4rae3.mirror.aliyuncs.com	复制

操作文档

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

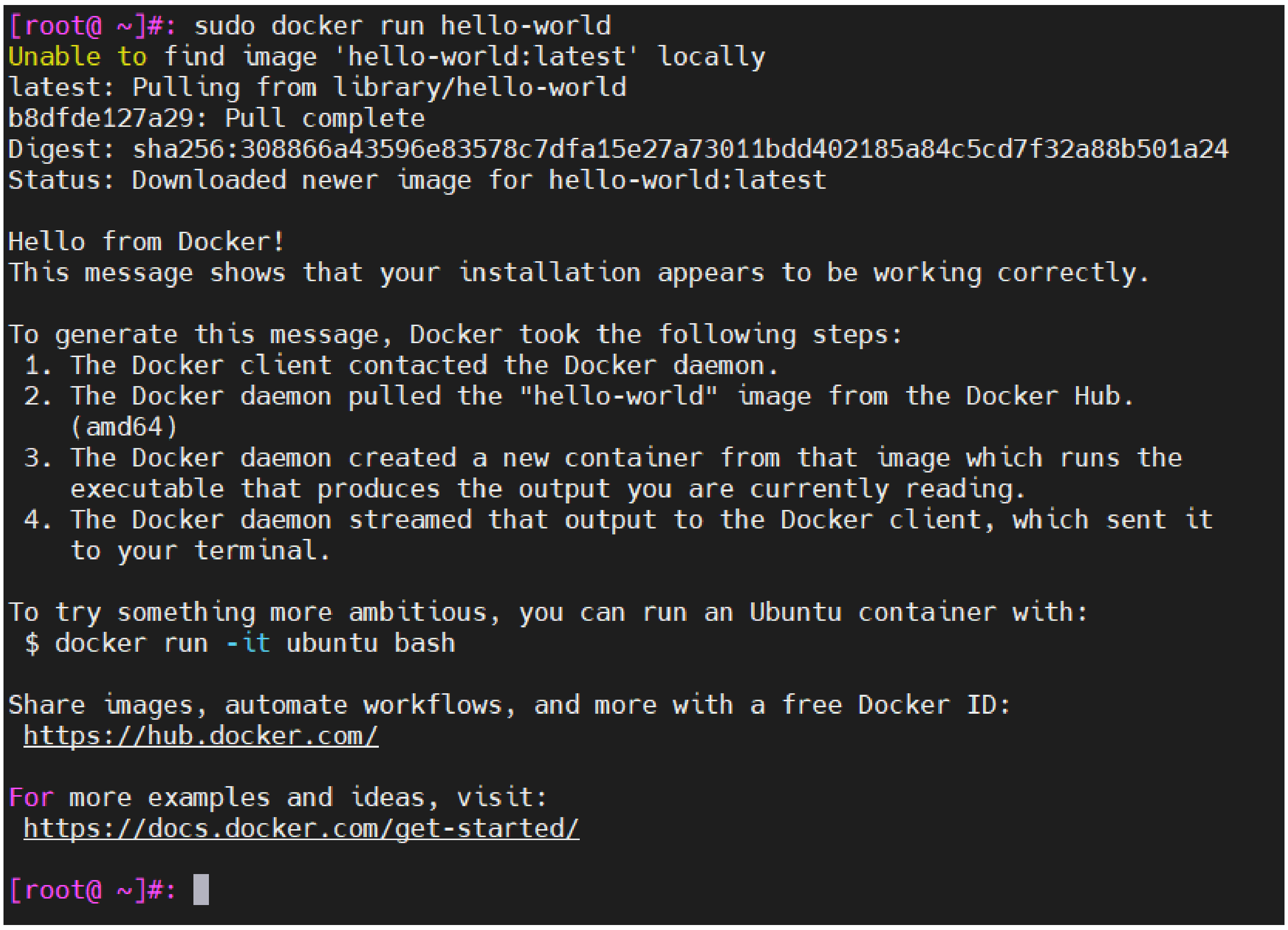
您可以通过修改daemon配置文件 /etc/docker/daemon.json 来使用加速器

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://mwg4rae3.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

在服务器上进行如下配置

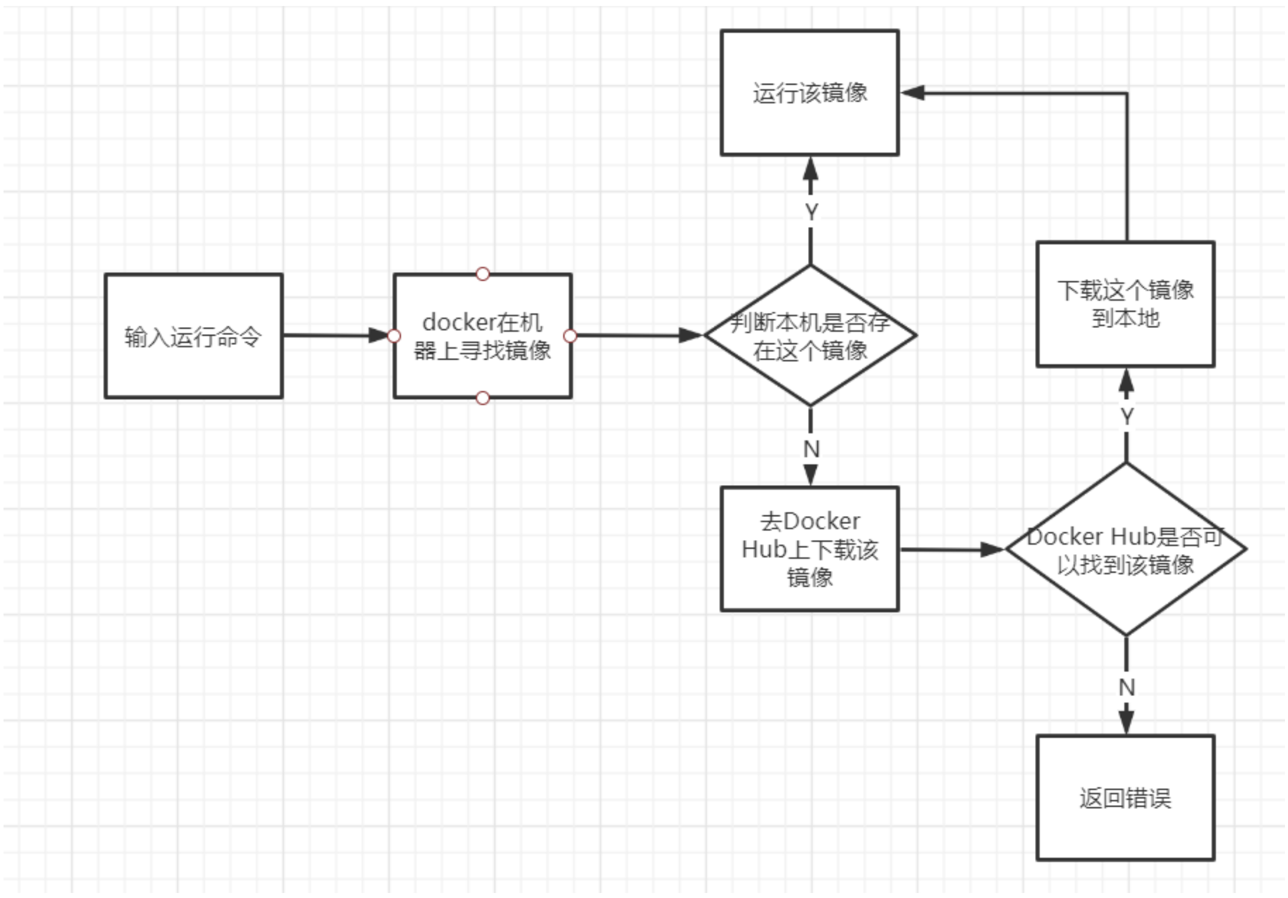
```
sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://mwg4rae3.mirror.aliyuncs.com"]
}
EOF

sudo systemctl daemon-reload

sudo systemctl restart docker
```

1.5 回顾Hello World流程



```
[root@ ~]#: sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:308866a43596e83578c7dfa15e27a73011bdd402185a84c5cd7f32a88b501a24
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[root@ ~]#: 
```

如上图，可以知道运行一个镜像时的大致流程如下：

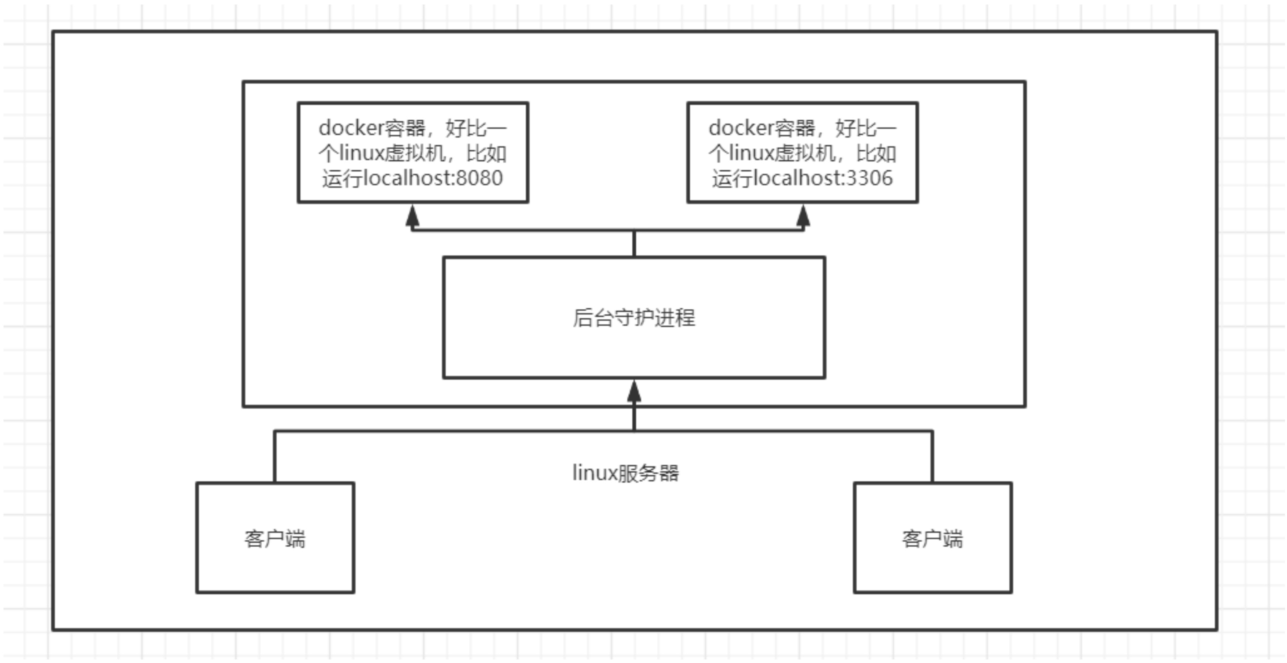


1.6 底层原理

Docker是怎么工作的？

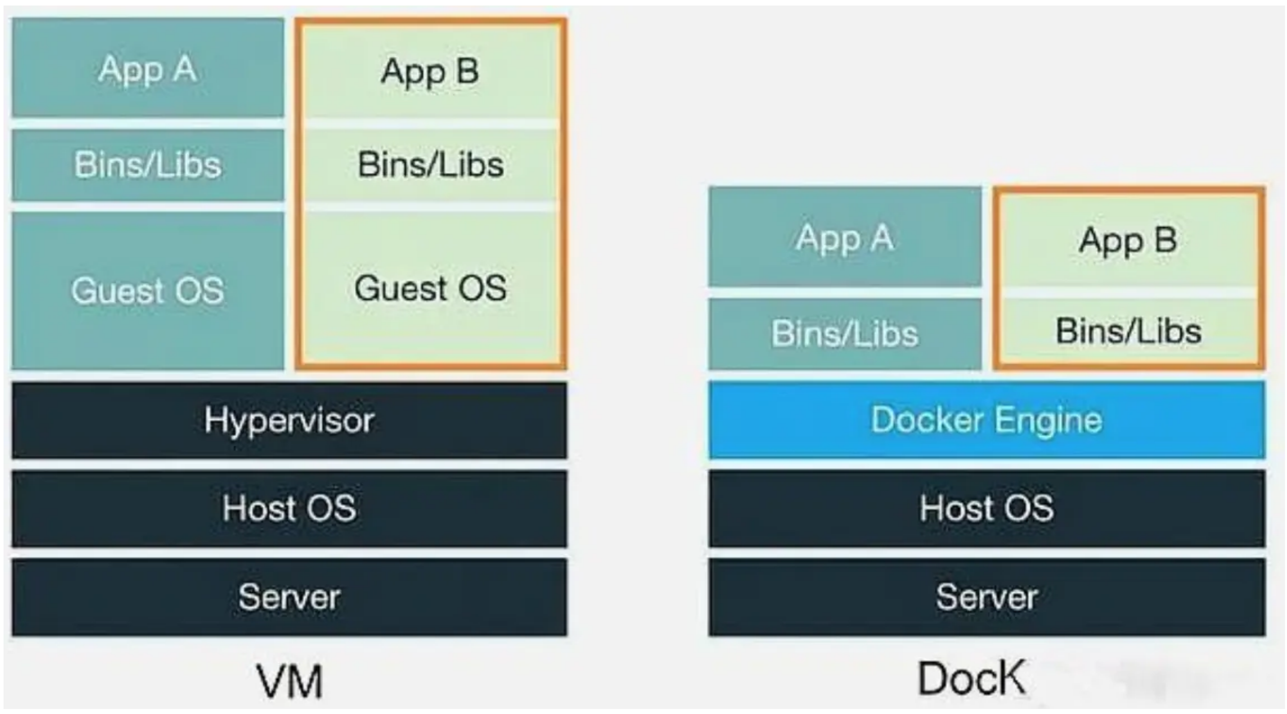
Docker是一个CLient-Server的结构系统，Docker的守护进程运行在主机上。通过Socket从客户端访问。

DockerService接收到DockerClient的命令，就会执行这个命令。

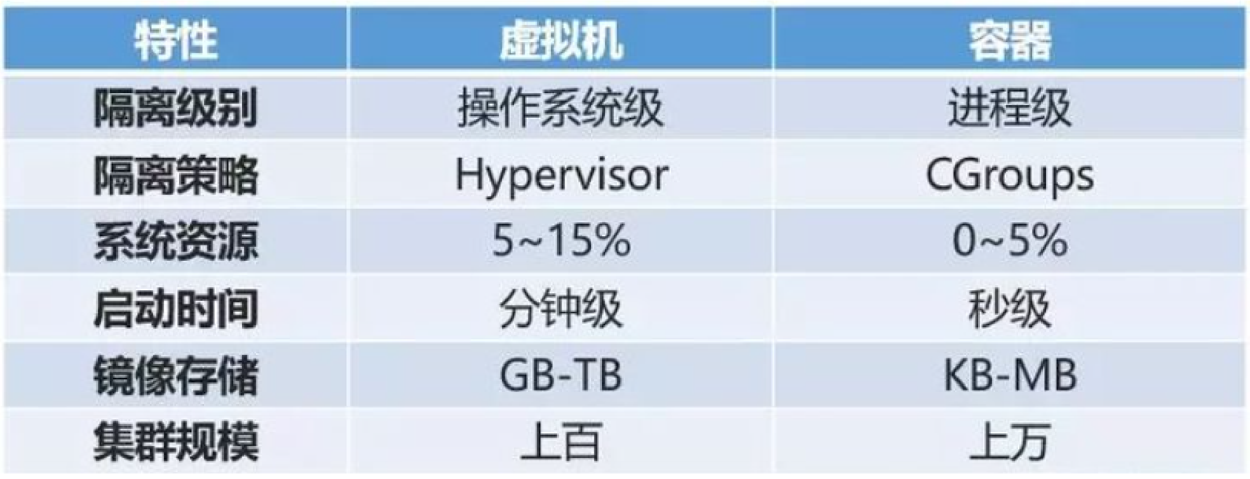

docker容器，好比一个linux虚拟机，比如运行localhost:8080
docker容器，好比一个linux虚拟机，比如运行localhost:3306
后台守护进程
linux服务器
客户端
客户端

Docker为什么比VM快

- Docker有着比VM更少的抽象层。
- Docker利用的是宿主机的内核，VM需要guest os。



所以说，新建一个容器的时候，docker不需要像VM一样重新加载一个操作系统内核，避免引导。虚拟机是加载guest os，而docker是利用宿主机的操作系统，省去了这个复杂的过程。



特性	虚拟机	容器
隔离级别	操作系统级	进程级
隔离策略	Hypervisor	CGroups
系统资源	5~15%	0~5%
启动时间	分钟级	秒级
镜像存储	GB-TB	KB-MB
集群规模	上百	上万

2、Docker的常用命令

2.1 帮助命令

```
docker version   # 显示docker的版本信息
docker info      # 显示docker的系统信息，包括容器和镜像的数量
docker --help    # 帮助命令
```

也可以参考官方的文档命令：https://docs.docker.com/reference/#command-line-interfaces-clis

2.2 镜像命令

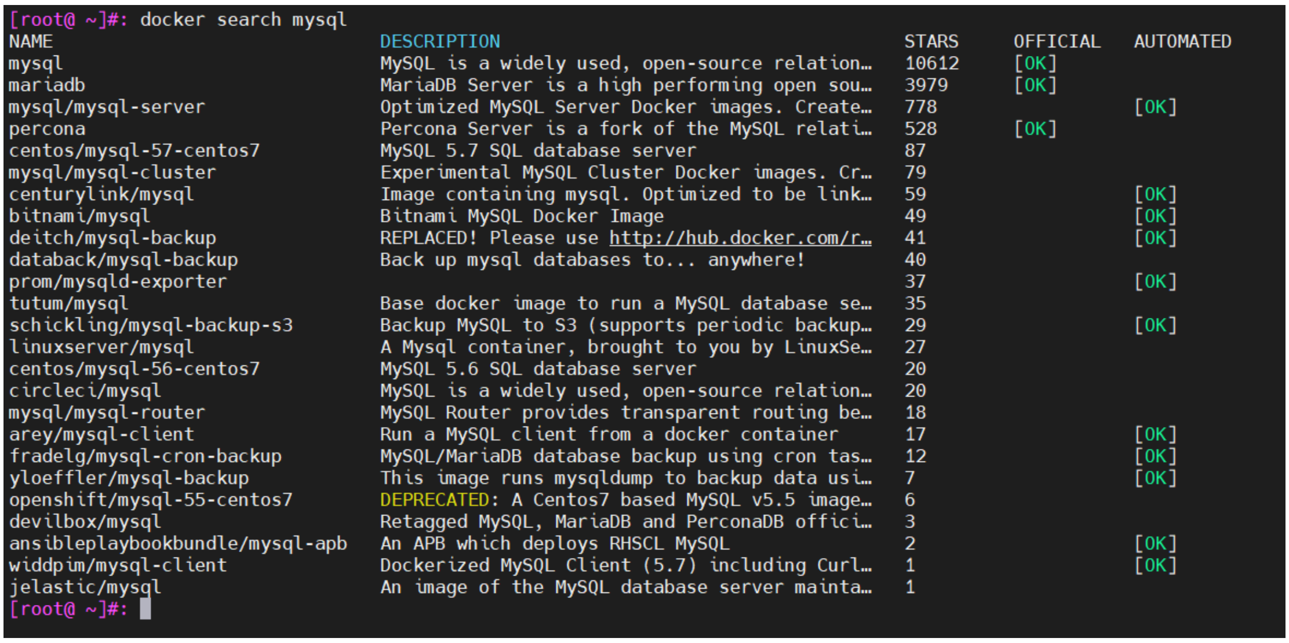
2.2.1 docker images

列出本地所有镜像

```
[root@ ~]#: docker images
REPOSITORY     TAG       IMAGE ID       CREATED       SIZE
hello-world    latest    d1165f221234   9 days ago    13.3kB

# REPOSITORY: 镜像的仓库源
# 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

[root@ ~]#: 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
```

2.2.2 docker search

搜索镜像



```
[root@ ~]#: docker search mysql
NAME                              DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
mysql                             MySQL is a widely used, open-source relation…   10612     [OK]
mariadb                           MariaDB Server is a high performing open sou…   3979      [OK]
mysql/mysql-server                Optimized MySQL Server Docker images. Create…   778                  [OK]
percona                           Percona Server is a fork of the MySQL relati…   528       [OK]
centos/mysql-57-centos7           MySQL 5.7 SQL database server                   87
mysql/mysql-cluster               Experimental MySQL Cluster Docker images. Cr…   79
centurylink/mysql                 Image containing mysql. Optimized to be link…   59                   [OK]
bitnami/mysql                     Bitnami MySQL Docker Image                      49                   [OK]
deitch/mysql-backup               REPLACED! Please use http://hub.docker.com/r…   41                   [OK]
databack/mysql-backup             Back up mysql databases to... anywhere!         40
prom/mysqld-exporter                                                              37                   [OK]
tutum/mysql                       Base docker image to run a MySQL database se…   35
schickling/mysql-backup-s3        Backup MySQL to S3 (supports periodic backup…   29                   [OK]
linuxserver/mysql                 A Mysql container, brought to you by LinuxSe…   27
centos/mysql-56-centos7           MySQL 5.6 SQL database server                   20
circleci/mysql                    MySQL is a widely used, open-source relation…   20
mysql/mysql-router                MySQL Router provides transparent routing be…   18
arey/mysql-client                 Run a MySQL client from a docker container      17                   [OK]
fradelg/mysql-cron-backup         MySQL/MariaDB database backup using cron tas…   12                   [OK]
yloeffler/mysql-backup            This image runs mysqldump to backup data usi…   7                    [OK]
openshift/mysql-55-centos7        DEPRECATED: A Centos7 based MySQL v5.5 image…   6
devilbox/mysql                    Retagged MySQL, MariaDB and PerconaDB offici…   3
ansibleplaybookbundle/mysql-apb   An APB which deploys RHSCL MySQL                2                    [OK]
widdpim/mysql-client              Dockerized MySQL Client (5.7) including Curl…   1                    [OK]
jelastic/mysql                    An image of the MySQL database server mainta…   1
[root@ ~]#:
```

```
[root@ ~]#: docker search --help

Usage:  docker search [OPTIONS] TERM

```

```
Search the Docker Hub for images

Options:
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print search using a Go template
      --limit int       Max number of search results (default 25)
      --no-trunc        Don't truncate output

# 搜索start数量大于5000的镜像
[root@ ~]#: docker search mysql --filter=STARS=5000
NAME      DESCRIPTION                                     STARS
OFFICIAL    AUTOMATED
mysql     MySQL is a widely used, open-source relation…   10612
[OK]
```

2.2.3 docker pull

下载镜像

```
[root@ ~]#: docker pull --help

Usage:  docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Pull an image or a repository from a registry

Options:
  -a, --all-tags                Download all tagged images in the
repository
      --disable-content-trust   Skip image verification (default
true)
      --platform string         Set platform if server is multi-
platform capable
  -q, --quiet                   Suppress verbose output

[root@ ~]#: docker pull mysql
Using default tag: latest   # 不知道版本的话，默认版本是最新版latest
latest: Pulling from library/mysql
a076a628af6f: Pull complete  # 分层下载，docker image的核心，联合文件系统
f6c208f3f991: Pull complete
88a9455a9165: Pull complete
406c9b8427c6: Pull complete
```

```
7c88599c0b25: Pull complete
25b5c6debdaf: Pull complete
43a5816f1617: Pull complete
1a8c919e89bf: Pull complete
9f3cf4bd1a07: Pull complete
80539cea118d: Pull complete
201b3cad54ce: Pull complete
944ba37e1c06: Pull complete
Digest:
sha256:feada149cb8ff54eade1336da7c1d080c4a1c7ed82b5e320efb5beebed85
ae8c  # 签名
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest  # 镜像的真实地址，等价于docker pull
docker.io/library/mysql:latest

[root@ ~]#: docker pull mysql:5.7
5.7: Pulling from library/mysql   # 指定5.7版本下载
a076a628af6f: Already exists  # 可以共用，已经下载的就不会再下载
f6c208f3f991: Already exists
88a9455a9165: Already exists
406c9b8427c6: Already exists
7c88599c0b25: Already exists
25b5c6debdaf: Already exists
43a5816f1617: Already exists
1831ac1245f4: Pull complete
37677b8c1f79: Pull complete
27e4ac3b0f6e: Pull complete
7227baa8c445: Pull complete
Digest:
sha256:b3d1eff023f698cd433695c9506171f0d08a8f92a0c8063c1a4d9db9a558
08df
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7

[root@ ~]#: docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
hello-world   latest    d1165f221234   9 days ago    13.3kB
mysql         5.7       a70d36bc331a   7 weeks ago   449MB
mysql         latest    c8562eaf9d81   7 weeks ago   546MB
```

2.2.4 docker rmi

删除镜像

```
[root@ ~]#: docker rmi --help

Usage:  docker rmi [OPTIONS] IMAGE [IMAGE...]

Remove one or more images

Options:
  -f, --force      Force removal of the image
      --no-prune   Do not delete untagged parents

[root@ ~]#: docker rmi -f IMAGE ID   # 删除指定id的镜像
[root@ ~]#: docker rmi -f IMAGE ID IMAGE ID IMAGE ID    # 删除多个指定id的镜像
[root@ ~]#: docker rmi -f $(docker images -aq)   # 删除所有的镜像
```

2.3 容器命令

有了镜像，才可以创建容器。下载一个centos的镜像进行学习。

```
docker pull centos
```

2.3.1 新建容器并启动

```
docker run [可选参数] image

# 参数说明
--name = "name"  # 容器的名字  tomcat01，tomcat02，用来区分容器
-d               # 后台方式运行
-it              # 使用交互方式运行，进入容器查看内容
-p               # 指定容器的端口
    -p ip:主机端口:容器端口
    -p 主机端口:容器端口（常用）
    -p 容器端口
    容器端口
-P  # 随机指定端口

# 以交互方式启动容器
```

```
[root@ ~]#: docker run -it centos /bin/bash
[root@ca6ba4820f8f /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc
root  run  sbin  srv  sys  tmp  usr  var

# 退出容器
[root@ca6ba4820f8f /]# exit
exit
```

2.3.2 列出所有运行的容器

```
docker ps    # 列出当前正在运行的容器
-a   # 列出所有容器，包括正在运行的和曾经运行的
-n=?  # 显示最近创建的?个容器
-q    # 只显示容器的编号

[root@ ~]#: docker ps
CONTAINER ID   IMAGE     COMMAND    CREATED    STATUS     PORTS
NAMES
[root@ ~]#: docker ps -a
CONTAINER ID   IMAGE          COMMAND        CREATED          STATUS
                         PORTS      NAMES
27d4179559d6   centos         "/bin/bash"   2 minutes ago    Exited
(0) About a minute ago             beautiful_bardeen
a3924479b852   centos         "/bin/bash"   4 minutes ago    Exited
(0) 3 minutes ago                  condescending_colden
ca6ba4820f8f   centos         "/bin/bash"   4 minutes ago    Exited
(0) 4 minutes ago                  angry_jones
9806ba877da7   d1165f221234   "/hello"      3 days ago       Exited
(0) 3 days ago                     nervous_faraday
8c7ad6faa5d4   d1165f221234   "/hello"      3 days ago       Exited
(0) 3 days ago                     nostalgic_pasteur
```

2.3.3 退出容器

```
exit    # 容器停止并退出
Ctrl + P + Q    # 容器不停止退出
```

2.3.4 删除容器

```
docker rm 容器id    # 删除某个容器，但不能删除正在运行中的容器，强制删除使用rm
-f
docker rm -f $(docker ps -aq)    # 删除所有容器
```

2.3.5 启动和停止容器

```
docker start 容器id     # 启动容器
docker restart 容器id   # 重启容器
docker stop 容器id      # 停止容器
docker kill 容器id      # 强制停止当前容器
```

2.4 其他常用命令

2.4.1 后台启动容器

```
docker run -d 镜像名称

[root@ ~]#: docker run -d centos
f66e178b45ff7ca8a504a814841e57763bae1beedaa6433da004dfc9a9c747a5
[root@ ~]#: docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
NAMES

# 使用后台方式运行容器后，docker ps发现并没有正在运行的容器。原因是docker容器使
用后台运行，就必须要有一个前台进程，不然docker发现没有应用，就会自动停止。
```

2.4.2 查看日志

```
# 后台启动容器，同时每秒打印一次CodeTiger
[root@ ~]#: docker run -d centos /bin/bash -c "while true;do echo
CodeTiger;sleep 1;done"
da4eab79a9c46486dc9bec10e384aca941ad0501cc3f64aaca80aed138a1f66f
[root@ ~]#: docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED
STATUS         PORTS     NAMES
da4eab79a9c4   centos    "/bin/bash -c 'while…"   4 seconds ago
Up 3 seconds             condescending_clarke
```

```
# 显示日志
[root@ ~]#: docker logs -tf --tail 10 da4eab79a9c4
2021-03-18T13:14:09.160931683Z CodeTiger
2021-03-18T13:14:10.162581267Z CodeTiger
2021-03-18T13:14:11.164348373Z CodeTiger
2021-03-18T13:14:12.166027616Z CodeTiger
2021-03-18T13:14:13.167761562Z CodeTiger
2021-03-18T13:14:14.169477290Z CodeTiger
2021-03-18T13:14:15.171136110Z CodeTiger
2021-03-18T13:14:16.172875315Z CodeTiger
2021-03-18T13:14:17.174586497Z CodeTiger
2021-03-18T13:14:18.176378839Z CodeTiger
2021-03-18T13:14:19.177982611Z CodeTiger
2021-03-18T13:14:20.179750901Z CodeTiger
2021-03-18T13:14:21.181485704Z CodeTiger
```

2.4.3 查看容器中进程信息

```
[root@ ~]#: docker top 7efd96c256a1
UID              PID              PPID                    C
STIME                  TTY   root         7488            7455
     0                       21:18                ?     root         7558
          7488                 0                        21:19
  ?
```

2.4.4 查看镜像的元数据

```
[root@ ~]#: docker inspect 7efd96c256a1
[
    {
        "Id":
"7efd96c256a1b5ab1ce03364043e92f4782ab3779dbec6c29c042e7ca0fe70be",
        "Created": "2021-03-18T13:18:56.635919677Z",
        "Path": "/bin/bash",
        "Args": [
            "-c",
            "while true;do echo CodeTiger;sleep 1;done"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 7488,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2021-03-18T13:18:56.987106052Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:300e315adb2f96afe5f0b2780b87f28ae95231fe3bdd1e16b9ba6063077
28f55",
        "ResolvConfPath":
"/var/lib/docker/containers/7efd96c256a1b5ab1ce03364043e92f4782ab37
79dbec6c29c042e7ca0fe70be/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/7efd96c256a1b5ab1ce03364043e92f4782ab37
79dbec6c29c042e7ca0fe70be/hostname",
        "HostsPath":
"/var/lib/docker/containers/7efd96c256a1b5ab1ce03364043e92f4782ab37
79dbec6c29c042e7ca0fe70be/hosts",
        "LogPath":
"/var/lib/docker/containers/7efd96c256a1b5ab1ce03364043e92f4782ab37
79dbec6c29c042e7ca0fe70be/7efd96c256a1b5ab1ce03364043e92f4782ab3779
dbec6c29c042e7ca0fe70be-json.log",
        "Name": "/frosty_goldwasser",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "CgroupnsMode": "host",
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "private",
            "Cgroup": "",
            "Links": null,
            "OomScoreAdj": 0,
            "PidMode": "",
            "Privileged": false,
            "PublishAllPorts": false,
            "ReadonlyRootfs": false,
            "SecurityOpt": null,
            "UTSMode": "",
            "UsernsMode": "",
            "ShmSize": 67108864,
            "Runtime": "runc",
            "ConsoleSize": [
                0,
                0
            ],
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": [],
            "BlkioDeviceReadBps": null,
            "BlkioDeviceWriteBps": null,
            "BlkioDeviceReadIOps": null,
            "BlkioDeviceWriteIOps": null,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DeviceCgroupRules": null,
            "DeviceRequests": null,
            "KernelMemory": 0,
            "KernelMemoryTCP": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": false,
            "PidsLimit": null,
            "Ulimits": null,
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0,
            "MaskedPaths": [
                "/proc/asound",
                "/proc/acpi",
                "/proc/kcore",
                "/proc/keys",
                "/proc/latency_stats",
                "/proc/timer_list",
                "/proc/timer_stats",
                "/proc/sched_debug",
                "/proc/scsi",
                "/sys/firmware"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/26b5063329fc677502adff59ba9f01ff9de86a115
570d5df9bac08786f8bd120-
init/diff:/var/lib/docker/overlay2/25cdc25ed53f1d4ad8a5a7c76f8771e5
ca13bb7e38ed7c9d6aafd552e217a88b/diff",
                "MergedDir":
"/var/lib/docker/overlay2/26b5063329fc677502adff59ba9f01ff9de86a115
570d5df9bac08786f8bd120/merged",
                "UpperDir":
"/var/lib/docker/overlay2/26b5063329fc677502adff59ba9f01ff9de86a115
570d5df9bac08786f8bd120/diff",
                "WorkDir":
"/var/lib/docker/overlay2/26b5063329fc677502adff59ba9f01ff9de86a115
570d5df9bac08786f8bd120/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "7efd96c256a1",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
"
            ],
            "Cmd": [
                "/bin/bash",
                "-c",
                "while true;do echo CodeTiger;sleep 1;done"
            ],
            "Image": "centos",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "org.label-schema.build-date": "20201204",
                "org.label-schema.license": "GPLv2",
                "org.label-schema.name": "CentOS Base Image",
                "org.label-schema.schema-version": "1.0",
                "org.label-schema.vendor": "CentOS"
            }
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"84e2d81f86f2a4e3829328b0073b6e53144d8eb89d6e7bc552a6d3c8f1154647",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {},
            "SandboxKey": "/var/run/docker/netns/84e2d81f86f2",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"3143e16381a426a1035ad09703212c668bf1e948635d9608a37be442745ded94",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"2718fe2feb075c9b748868e931a886193017c91c66843790ba2d728a2a77e919",
                    "EndpointID":
"3143e16381a426a1035ad09703212c668bf1e948635d9608a37be442745ded94",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

2.4.5 进入当前正在运行的容器

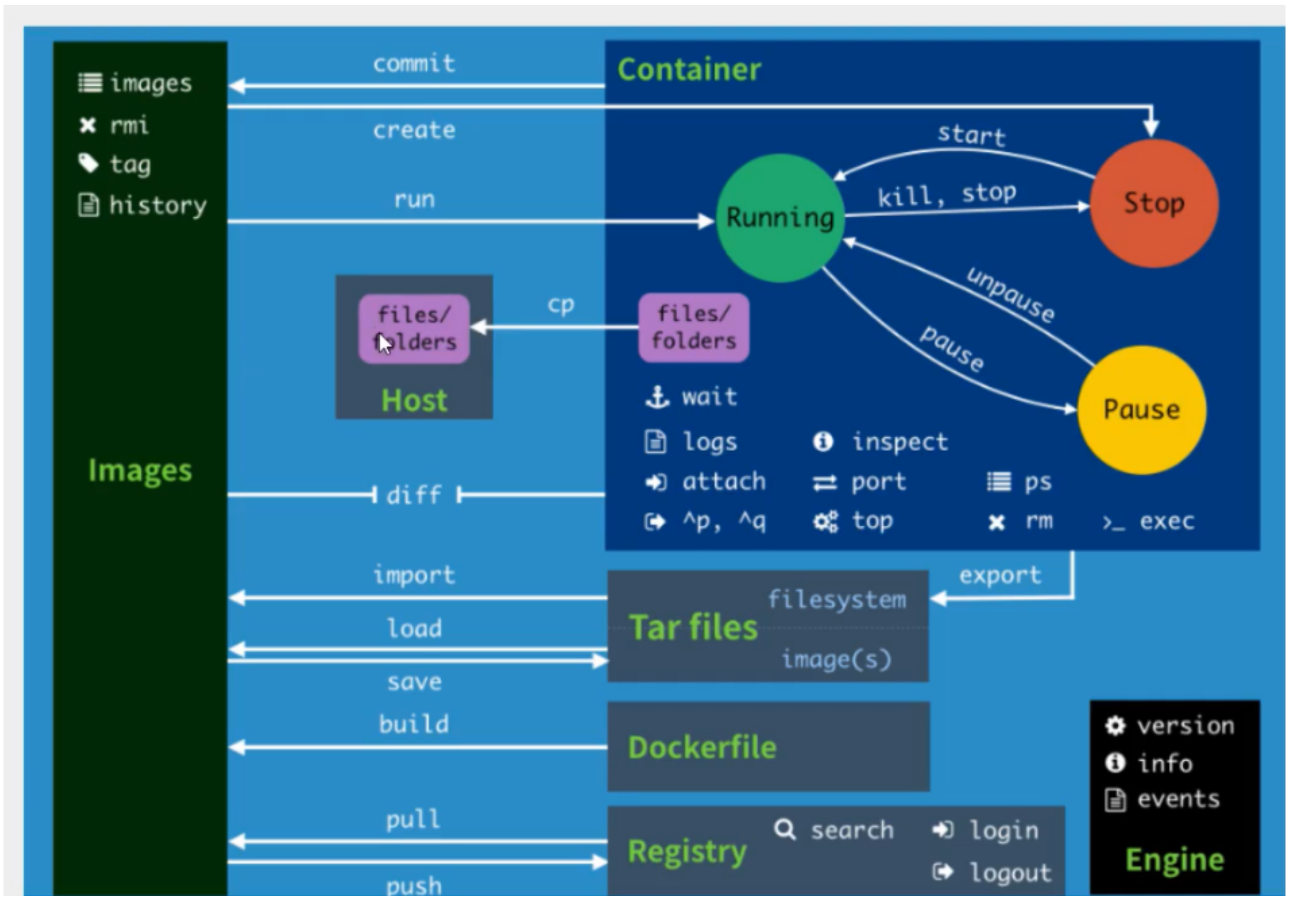
```
# 方式一，至少要有两个参数
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
[root@ ~]#: docker exec -it 2bf59a1d4601 /bin/bash
[root@2bf59a1d4601 /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc
root  run  sbin  srv  sys  tmp  usr  var

# 方式二
docker attach [OPTIONS] CONTAINER
[root@ ~]#: docker attach 2bf59a1d4601

# 方式一进入容器后开启一个新的终端，可以在里面操作（常用）
# 方式二进入容器正在执行的终端，不会启动新的进程。
```

2.4.6 从容器内拷贝文件到主机上

```
docker cp 容器id:容器内路径 目标路径

[root@ ~]#: docker ps
CONTAINER ID   IMAGE     COMMAND    CREATED    STATUS     PORTS
NAMES
[root@ ~]#: docker run -it centos

# ctrl + P + Q退出容器，但保持容器运行
[root@ ~]#: docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED           STATUS
     PORTS     NAMES
880f454bd93d   centos    "/bin/bash"   21 seconds ago    Up 20
seconds             jolly_poitras
[root@ ~]#: ls
```

```
ctags  jenkins  os

# 重新进入容器
[root@ ~]#: docker attach 880f454bd93d
[root@880f454bd93d /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc
root  run  sbin  srv  sys  tmp  usr  var
[root@880f454bd93d /]# cd home
[root@880f454bd93d home]# ls
[root@880f454bd93d home]# vi test.txt
[root@880f454bd93d home]# ls
test.txt
[root@880f454bd93d home]# exit
exit
[root@ ~]#: docker ps
CONTAINER ID   IMAGE      COMMAND    CREATED    STATUS     PORTS
NAMES
[root@ ~]#: docker cp 880f454bd93d:/home/test.txt /root
[root@ ~]#: ls
ctags  jenkins  os  test.txt
[root@ ~]#: vi test.txt

# docker cp是手动拷贝的过程，使用 -v 卷的技术，可以实现自动同步数据。
```

2.4.7 小结



```
  attach      Attach local standard input, output, and error
streams to a running container
  #当前shell下 attach连接指定运行的镜像
  build       Build an image from a Dockerfile # 通过Dockerfile定制镜
像
  commit      Create a new image from a container's changes #提交当前
容器为新的镜像
  cp          Copy files/folders between a container and the local
filesystem #拷贝文件
  create      Create a new container #创建一个新的容器
  diff        Inspect changes to files or directories on a
container's filesystem #查看docker容器的变化
  events      Get real time events from the server # 从服务获取容器实时
时间
  exec        Run a command in a running container # 在运行中的容器上运
行命令
  export      Export a container's filesystem as a tar archive #导出
容器文件系统作为一个tar归档文件[对应import]
  history     Show the history of an image # 展示一个镜像形成历史
  images      List images #列出系统当前的镜像
  import      Import the contents from a tarball to create a
filesystem image #从tar包中导入内容创建一个文件系统镜像
  info        Display system-wide information # 显示全系统信息
  inspect     Return low-level information on Docker objects #查看容
器详细信息
  kill        Kill one or more running containers # kill指定docker容
器
  load        Load an image from a tar archive or STDIN #从一个tar包
或标准输入中加载一个镜像[对应save]
  login       Log in to a Docker registry #
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the
container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to
STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage
statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print
their exit codes
```

2.5 作业

> 实战---部署一个**nginx**

（1）搜索镜像

```
docker search nginx
```

（2）下载镜像

```
docker pull nginx
```

（3）启动容器测试

```
[root@ ~]#: docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    f6d0b4767a6c   2 months ago   133MB
centos        latest    300e315adb2f   3 months ago   209MB

# -d  后台运行
# --name  指定容器的名称
# -p 宿主机端口:容器内部端口
[root@ ~]#: docker run -d --name nginx01 -p 8888:80 nginx
befa8275ea7da23018af54639af8f6bb6984562820ebab2ec1b0bf858198e614
[root@ ~]#: docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED
STATUS         PORTS                    NAMES
befa8275ea7d   nginx     "/docker-entrypoint.…"   3 seconds ago
Up 2 seconds   0.0.0.0:8888->80/tcp     nginx01
[root@ ~]#: curl localhost:8888
```

遇到问题了

```
[root@ ~]#: docker run -d --name nginx01 -p 8888:80 nginx
0db91c62b8568b9fb4071fefe97e87238af68802b523297d78a9793dd9f11e72
docker: Error response from daemon: driver failed programming
external connectivity on endpoint nginx01
(94c83de32bbae1f38b6260638be5882b3a452513ef2352937290d67ec003a1ab):
(iptables failed: iptables --wait -t nat -A DOCKER -p tcp -d 0/0 --
dport 8888 -j DNAT --to-destination 172.17.0.2:80 ! -i docker0:
iptables: No chain/target/match by that name.
 (exit status 1)).
```

重启docker解决

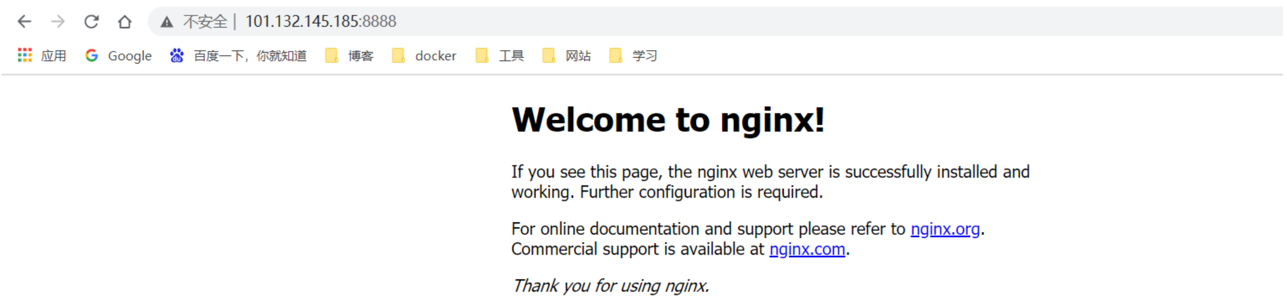
```
[root@ ~]#: systemctl restart docker
```

这里映射到服务器8888端口后，发现通过浏览器无法访问。

首先要在服务器的防火墙开启8888端口

```
# 照着复制一行改成8888端口
[root@ ~]#: vi /etc/sysconfig/iptables
# 重启防火墙
[root@ ~]#: service iptables restartd
```

做完上面的这步还不行，还需要去阿里云把服务器的8888端口打开就能正常访问了。



Welcome to nginx!

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

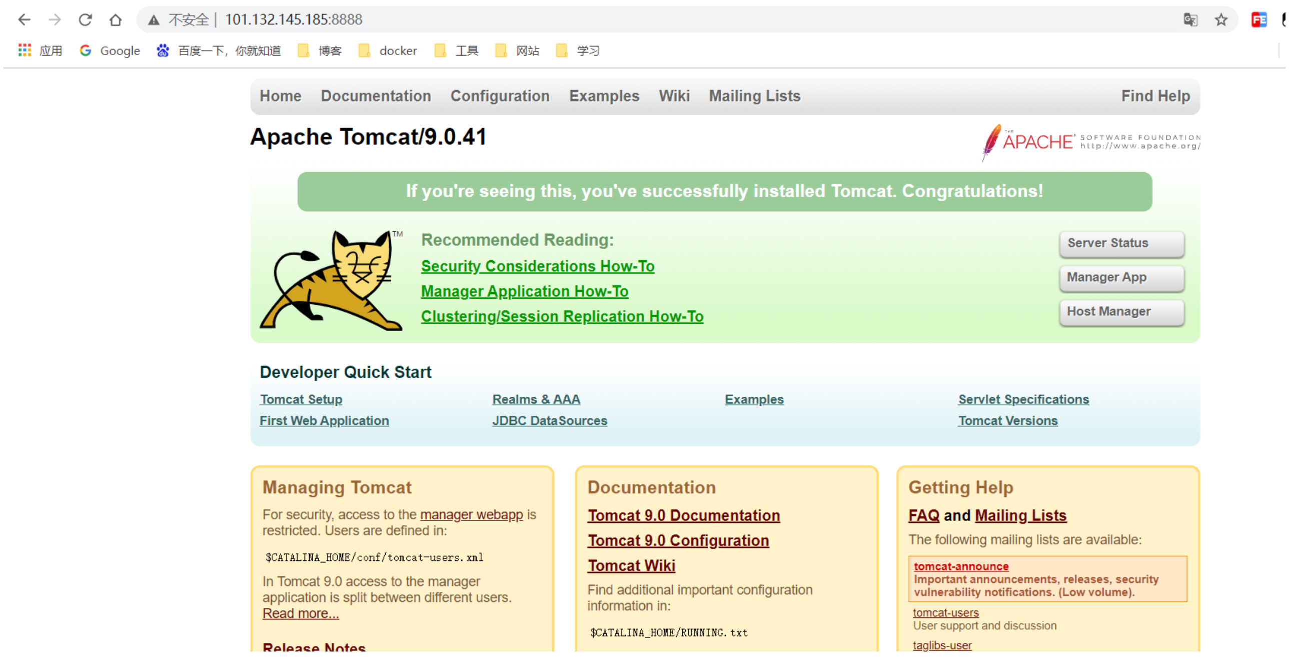
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

实战---部署一个tomcat

```
# 根据官方文档，有下面的用法
docker run -it --rm docker:9.0
-it  # 交互方式运行
--rm  # 用完就是删除镜像，这里启动完tomcat后就会删除镜像，一般用来测试

[root@ ~]#: docker pull tomcat:9.0
[root@ ~]#: docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
tomcat        9.0       040bdb29ab37   2 months ago   649MB
nginx         latest    f6d0b4767a6c   2 months ago   133MB
centos        latest    300e315adb2f   3 months ago   209MB

# 启动tomcat镜像，使用浏览器访问404
[root@ ~]#: docker run -d -p 8888:8080 --name tomcat02 tomcat:9.0

# 进入正在运行的容器，发现tomcat的webapps目录下面是空的，所以访问出现404
[root@ ~]#: docker exec -it 7c1a30039eba /bin/bash
root@7c1a30039eba:/usr/local/tomcat# ls
BUILDING.txt     LICENSE  README.md      RUNNING.txt  conf   logs
         temp      webapps.dist
```

```
CONTRIBUTING.md  NOTICE   RELEASE-NOTES  bin          lib   native-
jni-lib  webapps  work
root@7c1a30039eba:/usr/local/tomcat# cd webapps
root@7c1a30039eba:/usr/local/tomcat/webapps# ls
root@7c1a30039eba:/usr/local/tomcat/webapps#

# 把webapps.dist下面的文件夹全部拷贝到webapps下面，再访问就正常了。
root@7c1a30039eba:/usr/local/tomcat# cd webapps.dist/
root@7c1a30039eba:/usr/local/tomcat/webapps.dist# ls
ROOT  docs  examples  host-manager  manager
root@7c1a30039eba:/usr/local/tomcat/webapps.dist# cd ..
root@7c1a30039eba:/usr/local/tomcat# cp -r webapps.dist/* webapps

# 出现这种现象是因为，阿里云镜像默认是最小的可运行的镜像，其他不必要的都被剔除掉。
```



Apache Tomcat/9.0.41

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:
$CATALINA_HOME/conf/tomcat-users.xml
In Tomcat 9.0 access to the manager application is split between different users.
Read more...
Release Notes

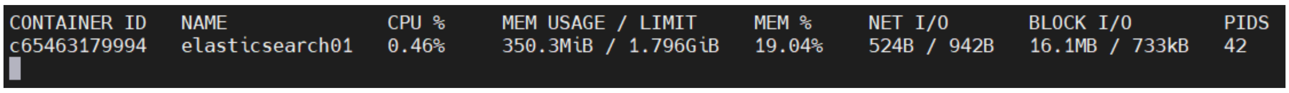
Documentation
Tomcat 9.0 Documentation
Tomcat 9.0 Configuration
Tomcat Wiki
Find additional important configuration information in:
$CATALINA_HOME/RUNNING.txt

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).
tomcat-users
User support and discussion
taglibs-user

实战---部署ES + Kibana

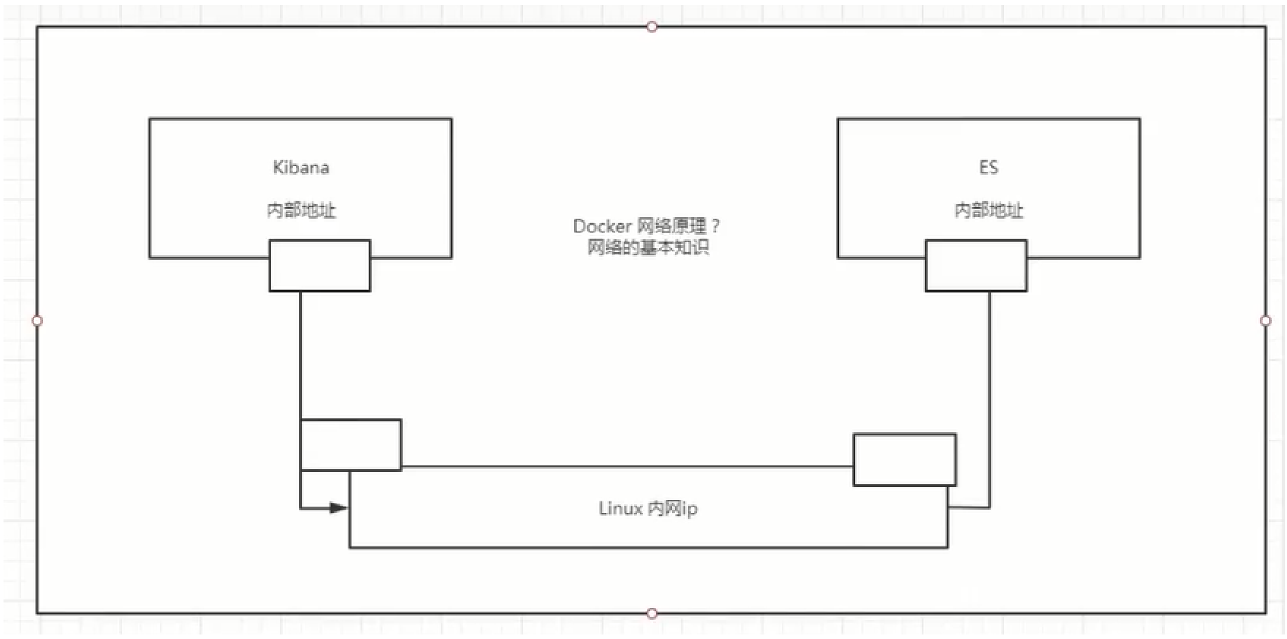
```
# 1、es暴露的端口很多
# 2、es十分耗内存
# 3、es的数据一般要放到安全目录挂载

# 启动elasticsearch   --net somenetwork  网络配置
docker run -d --name elasticsearch01 -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node" elasticsearch:tag

# 按照上面的命令直接启动会很卡，因为服务器只有1核2G，所以需要加上-e参数修改配置文
件，增加内存的限制（限制es运行内存在64M到512M之间）
```

```
docker run -d --name elasticsearch01 -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node" -e ES_JAVA_OPTS="-Xms64m -Xmx512m"
elasticsearch:7.6.2

# 测试启动成功
[root@ ~]#: curl localhost:9200
{
  "name" : "c65463179994",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "MrI19p3BT_-8N6ZKw6QodQ",
  "version" : {
    "number" : "7.6.2",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "ef48eb35cf30adf4db14086e8aabd07ef6fb113f",
    "build_date" : "2020-03-26T06:34:37.794943Z",
    "build_snapshot" : false,
    "lucene_version" : "8.4.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}

[root@ ~]#: docker stats c65463179994
```



```
CONTAINER ID   NAME              CPU %   MEM USAGE / LIMIT     MEM %    NET I/O       BLOCK I/O        PIDS
c65463179994   elasticsearch01   0.46%   350.3MiB / 1.796GiB   19.04%   524B / 942B   16.1MB / 733kB   42
```

使用kibana连接es，网络如何才能连接过去？



3、Docker镜像讲解

3.1 镜像是什么

镜像是一种轻量级的可执行的软件包，用来打包软件运行的环境和基于运行环境开发的软件。它包含软件运行所需要的全部内容，包括代码、依赖库、环境变量、配置文件。

所有的应用，直接打包docker镜像，就可以直接运行。

如何得到镜像：

- 从远程仓库下载
- 朋友拷贝给你
- 自己制作一个镜像 DockerFile

3.2 Docker镜像加载原理

> UnionFS（联合文件系统）

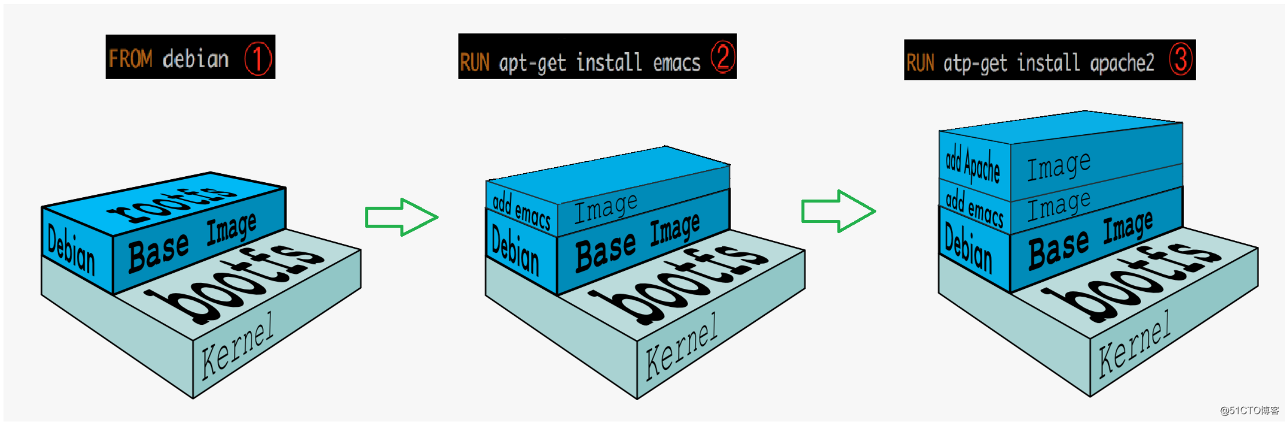
UnionFS（联合文件系统）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下。Union文件系统是Docker镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录。

> Docker镜像加载原来

bootfs(file system) 主要包含bootloader和kernel，boot加载器主要是引导加载内核，linux刚启动时会加载bootfs文件系统，在Docker镜像的最底层是bootfs。这一层与我们典型Linux/Unix系统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就在内存中了，此时内存使用权已有bootfs转交给内核，此时系统会卸载bootfs。

rootfs(root file system)是bootfs之上，包含的就是典型的Linux系统中的/dev、/proc、/bin、/etc等标准目录和文件。
rootfs就是各种不同操作系统发行版，比如Ubuntu,Centos等等。



平时我们安装到虚拟机的centos都是好几个G，在docker里却只有几百兆。对于一个精简的os，rootfs可以很小，只需要包含最基本的命令、工具和程序库就可以了。因为底层直接用Host的kernel，自己只需要提供rootfs就可以了。

由此可见对于Linux的不同发行版，bootfs基本都是一样的，rootfs会有差别，因此不同发行版可以公用bootfs.

虚拟机是分钟级别，容器是秒级！

3.3 分层理解

```
[root@ ~]#: docker image inspect nginx:latest
[
    {
        "Id":
"sha256:f6d0b4767a6c466c178bf718f99bea0d3742b26679081e52dbf8e0c7c4c
42d74",
        "RepoTags": [
            "nginx:latest"
        ],
        "RepoDigests": [

 "nginx@sha256:10b8cc432d56da8b61b070f4c7d2543a9ed17c2b23010b43af43
4fd40e2ca4aa"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2021-01-12T10:17:41.649267496Z",
        "Container":
"faa742a137cfbf261402d359c09203c3fd894fa49689e4f4952a657ea80d9107",
        "ContainerConfig": {
            "Hostname": "faa742a137cf",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
",
                "NGINX_VERSION=1.19.6",
                "NJS_VERSION=0.5.0",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"nginx\" \"-g\" \"daemon off;\"]"
            ],
            "Image":
"sha256:a5531bdc09faaa444e565e6f9ec98ed4474970ed6fdd5db6b8b255534b2
20689",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": [
                "/docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers <docker-
maint@nginx.com>"
            },
            "StopSignal": "SIGQUIT"
        },
        "DockerVersion": "19.03.12",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
",
                "NGINX_VERSION=1.19.6",
                "NJS_VERSION=0.5.0",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
            "Image":
"sha256:a5531bdc09faaa444e565e6f9ec98ed4474970ed6fdd5db6b8b255534b2
20689",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": [
                "/docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers <docker-
maint@nginx.com>"
            },
            "StopSignal": "SIGQUIT"
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 132951424,
        "VirtualSize": 132951424,
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/e1c8267af463488b12de060a222683a5d829aa299
45c420e3b3f3491541f97a9/diff:/var/lib/docker/overlay2/82c9a075fecc9
4616d441120396b56dcc441b1c52a9660298afa9b6379ea7fe1/diff:/var/lib/d
ocker/overlay2/06587896b2e0d3401dccf6362dabba6ea235522ac11a25934dcb
96c8380a8c19/diff:/var/lib/docker/overlay2/0c995d62818ad5e415fd881c
c1bd5ceb03fd9406d7dd623cfef684c7732f6cbf/diff",
                "MergedDir":
"/var/lib/docker/overlay2/71c8debfe1eaed94a111fa2002686ba5c8d7afac3
e459a05efe9ef3aa162fb33/merged",
                "UpperDir":
"/var/lib/docker/overlay2/71c8debfe1eaed94a111fa2002686ba5c8d7afac3
e459a05efe9ef3aa162fb33/diff",
                "WorkDir":
"/var/lib/docker/overlay2/71c8debfe1eaed94a111fa2002686ba5c8d7afac3
e459a05efe9ef3aa162fb33/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [

 "sha256:cb42413394c4059335228c137fe884ff3ab8946a014014309676c25e3a
c86864",
 "sha256:1c91bf69a08b515a1f9c36893d01bd3123d896b38b082e7c21b4b7cc70
23525a",

 "sha256:56bc37de0858bc2a5c94db9d69b85b4ded4e0d03684bb44da77e0fe93a
829292",

 "sha256:3e5288f7a70f526d6bceb54b3568d13c72952936cebfe28ddcb3386fe3
a236ba",

 "sha256:85fcec7ef3efbf3b4e76a0f5fb8ea14eca6a6c7cbc0c52a1d401ad5548
a29ba5"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        }
    }
]
```

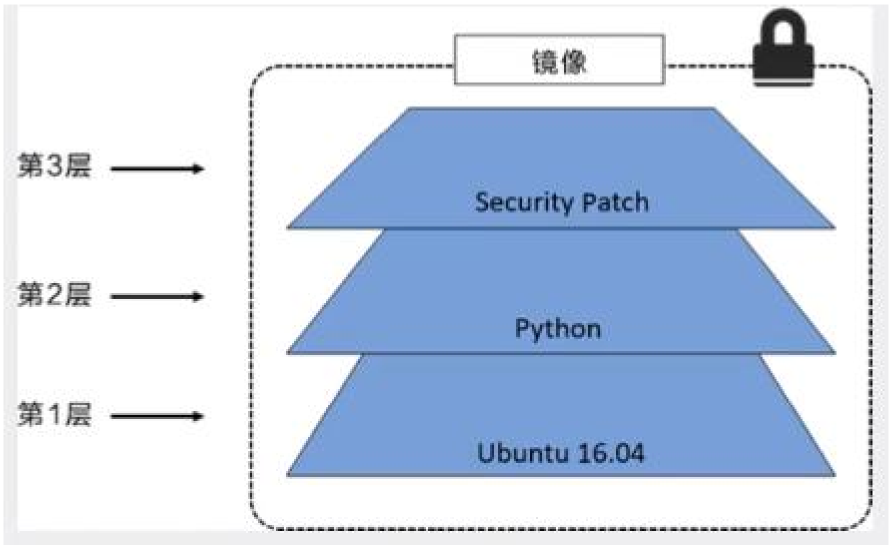
上面可以查看nginx镜像的元数据，它的分层结构如下

```
"Layers": [

 "sha256:cb42413394c4059335228c137fe884ff3ab8946a014014309676c25e3a
c86864",

 "sha256:1c91bf69a08b515a1f9c36893d01bd3123d896b38b082e7c21b4b7cc70
23525a",

 "sha256:56bc37de0858bc2a5c94db9d69b85b4ded4e0d03684bb44da77e0fe93a
829292",

 "sha256:3e5288f7a70f526d6bceb54b3568d13c72952936cebfe28ddcb3386fe3
a236ba",

 "sha256:85fcec7ef3efbf3b4e76a0f5fb8ea14eca6a6c7cbc0c52a1d401ad5548
a29ba5"
            ]
```

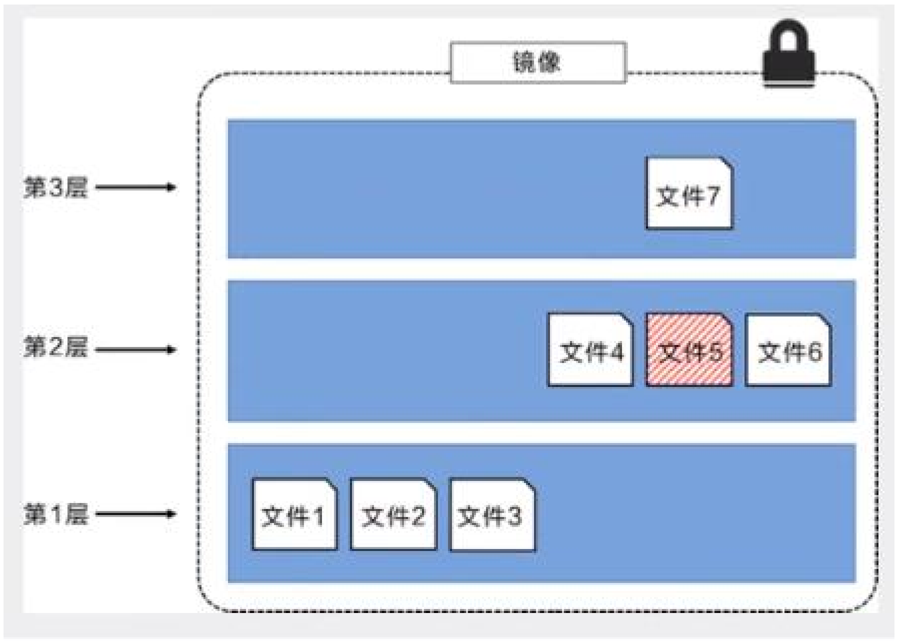
每次pull命令下载镜像时，都可以看到是分层下载的，比如一个redis7包含7层（Layer），当下载redis8的时候可能只需要下载最新的三层，redis7用的层不需要重新下载。

所有的Docker镜像都起始于一个基础镜像层，当进行修改或增加新的内容时，就会在当前镜像层之上，创建新的镜像层。

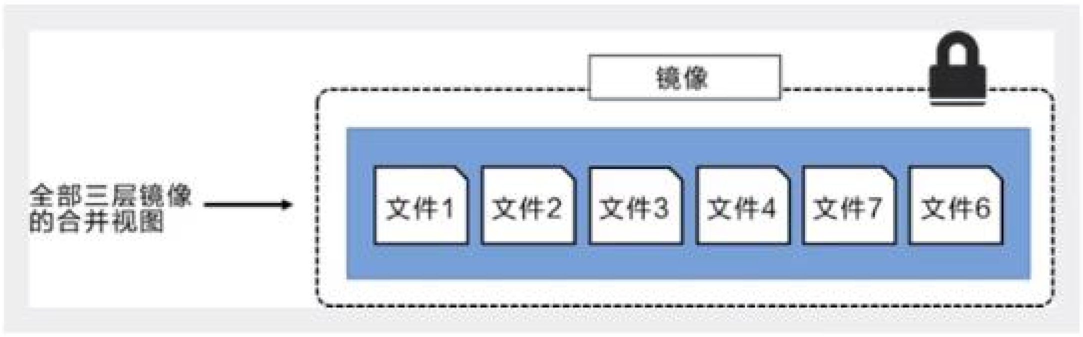
比如基于ubuntu16.04创建一个新的镜像，这就是镜像的第一层，如果在该镜像中添加python包，就会在基础镜像之上创建第二个镜像层，如果继续添加一个安全补丁，就会创建第三个镜像层。



在添加额外的镜像层的同时，镜像始终保持是当前所有镜像的组合。



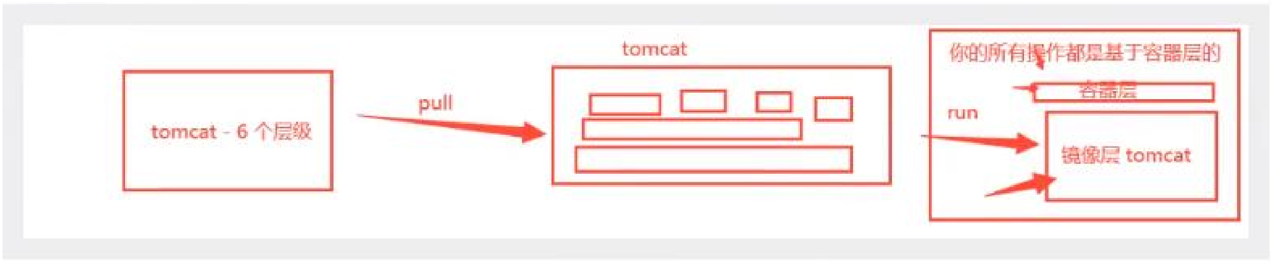
上图中文件7是对文件5的修改，在打包镜像的时候，上层镜像层中的文件覆盖了底层镜像层中的文件。这样就使得文件的更新版本作为一个新镜像层添加到镜像当中。



比如文件1是mysql，文件2是redis，文件3是tomcat，文件4是maven，文件6是jdk，文件5是我们之间的app1，文件7是app2，那么第一层的三个文件和第二层的两个文件，都是可以被其他镜像共用的。

docker镜像都是只读的，当容器启动时，一个新的可写层被加载到镜像的顶部。这一层就是我们通常说的容器层，容器之下的都叫镜像层。

Docker 通过存储引擎（新版本采用快照机制）的方式来实现镜像层堆栈，并保证多镜像层对外展示为统一的文件系统。

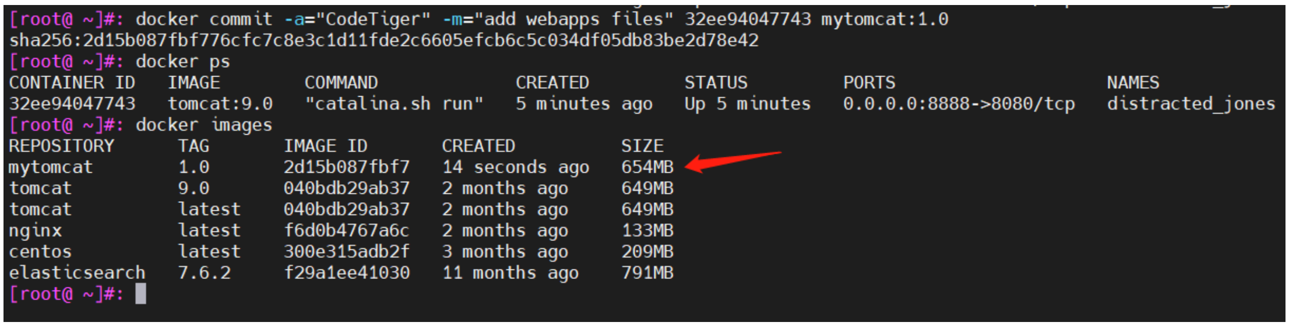


3.4 commit镜像

```
docker commit  # 提交容器成为一个新的副本

# 和git命令类似
docker commit -m="提交的描述信息" -a="作者" 容器id 目标镜像名:[TAG]
```

```
# 1、启动tomcat
[root@ ~]#: docker run -d -p 8888:8080 tomcat:9.0
# 2、发现这个默认的tomcat，webapps目录为空，官方默认是没有的。
# 3、从webapps.dist把文件拷贝到webapps下面。
# 4、将我们修改过的容器通过commit提交为一个镜像。
[root@ ~]#: docker commit -a="CodeTiger" -m="add webapps files"
32ee94047743 mytomcat:1.0
sha256:2d15b087fbf776cfc7c8e3c1d11fde2c6605efcb6c5c034df05db83be2d7
8e42
```

```
[root@ ~]#: docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
mytomcat        1.0       2d15b087fbf7   14 seconds ago   654MB
tomcat          9.0       040bdb29ab37   2 months ago     649MB
tomcat          latest    040bdb29ab37   2 months ago     649MB
nginx           latest    f6d0b4767a6c   2 months ago     133MB
centos          latest    300e315adb2f   3 months ago     209MB
elasticsearch   7.6.2     f29a1ee41030   11 months ago    791MB
```



```
[root@ ~]#: docker commit -a="CodeTiger" -m="add webapps files" 32ee94047743 mytomcat:1.0
sha256:2d15b087fbf776cfc7c8e3c1d11fde2c6605efcb6c5c034df05db83be2d78e42
[root@ ~]#: docker ps
CONTAINER ID   IMAGE        COMMAND             CREATED         STATUS         PORTS                    NAMES
32ee94047743   tomcat:9.0   "catalina.sh run"   5 minutes ago   Up 5 minutes   0.0.0.0:8888->8080/tcp   distracted_jones
[root@ ~]#: docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
mytomcat        1.0       2d15b087fbf7   14 seconds ago   654MB
tomcat          9.0       040bdb29ab37   2 months ago     649MB
tomcat          latest    040bdb29ab37   2 months ago     649MB
nginx           latest    f6d0b4767a6c   2 months ago     133MB
centos          latest    300e315adb2f   3 months ago     209MB
elasticsearch   7.6.2     f29a1ee41030   11 months ago    791MB
[root@ ~]#:
```

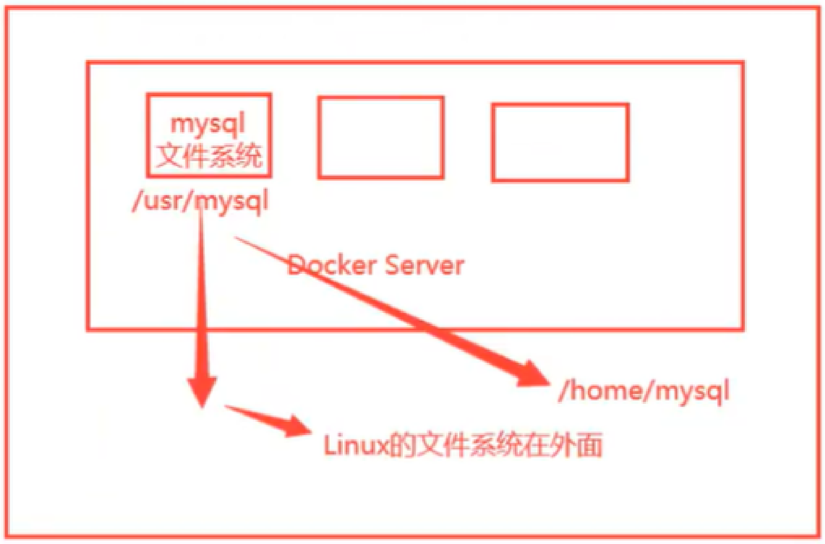
4、容器数据卷

4.1 什么是容器数据卷

将应用和数据打包成一个镜像后，数据都在容器中，那么如果容器被删除了，数据都会丢失？

需求：数据可以持久化

容器之间有一个数据共享的技术，docker容器中产生的数据，可以同步到本地，这就是**卷技术**。其实就是将容器内的目录，挂载到Linux上。



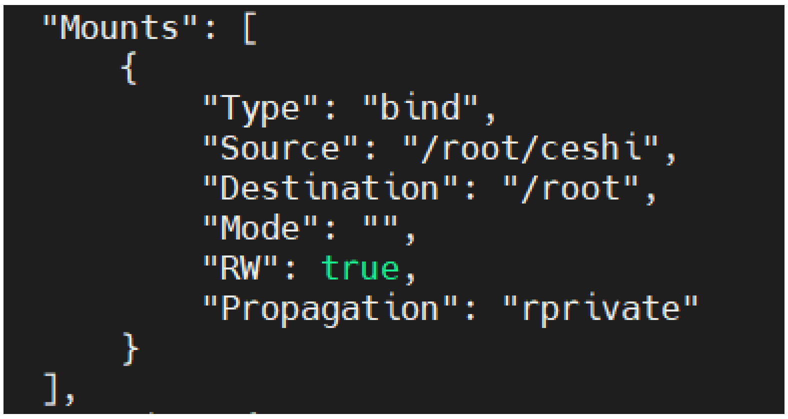
总结一句话：**容器的持久化和同步操作，容器间也是可以数据共享的。**

4.2 使用数据卷

> 直接使用命令来挂载 -v

```
docker run -it -v 主机目录:容器内目录
```

```
# 启动容器，可以发现在主机上的root文件夹下会自动创建一个ceshi目录
[root@ ~]#: docker run -it -v /root/ceshi:/root centos

# 查看镜像的元数据信息，可以发现挂载信息
[root@ ~]#: docker inspect d245ea321c52
```



```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/root/ceshi",
        "Destination": "/root",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

如果你在容器内`/root`目录下新建、删除和修改文件，都会自动同步到主机的`/root/ceshi`目录下，反之也是这样。

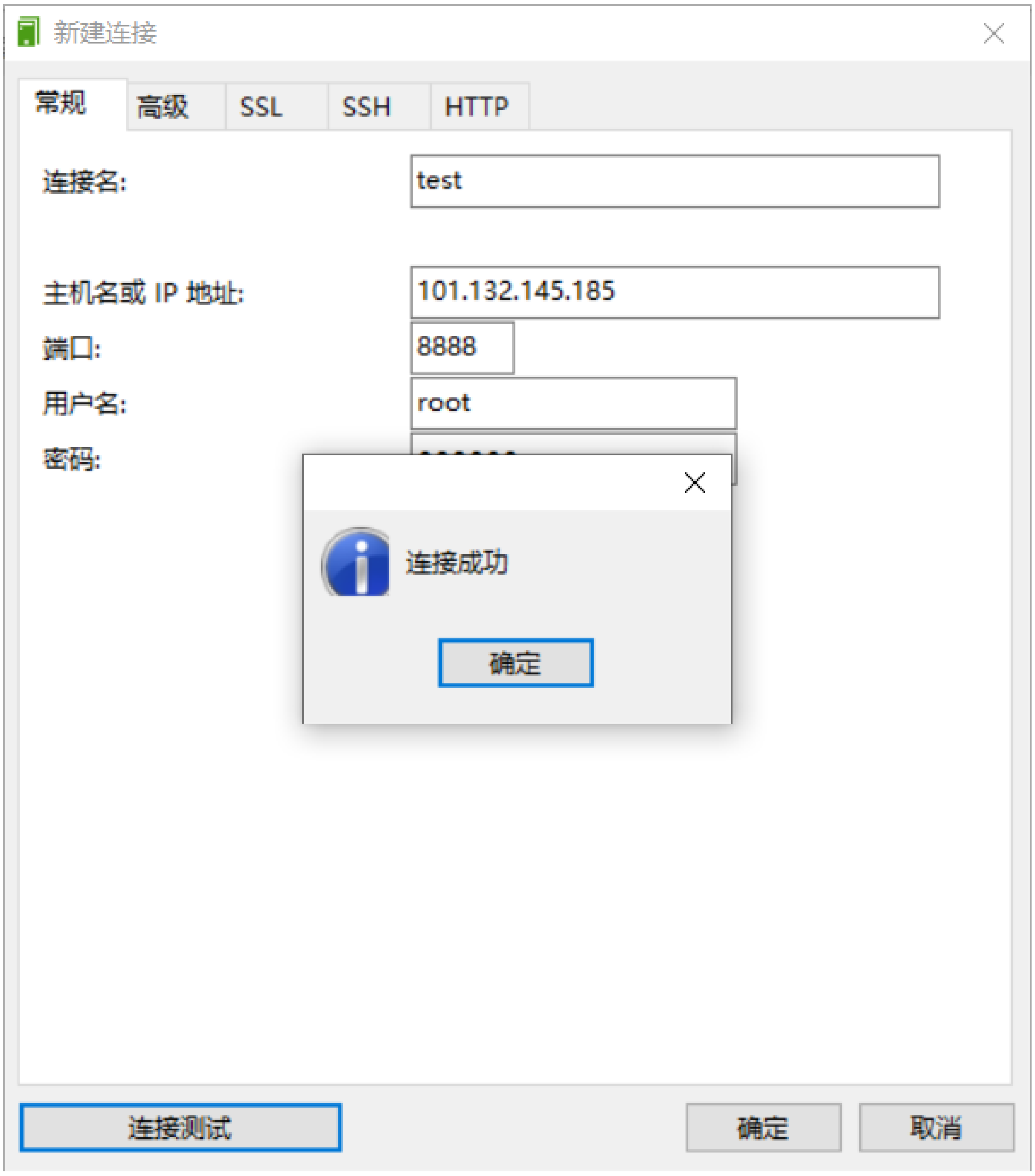
如果停止容器，在主机上对`/root/ceshi`进行操作，也会自动同步到容器内。

需要占用两倍的存储。

如果容器删除了，主机上的文件不会被删除。

4.3 实战---安装MySQL

```
# 下载mysql
[root@ ~]#: docker pull mysql:5.7

# $ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-
pw -d mysql:tag
# 上面是文档中的启动命令，因为mysql在启动的时候需要设置密码

# 启动mysql，并且挂载相应的目录到主机
[root@ ~]#: docker run -d -p 8888:3306 --name mysql01 -e
MYSQL_ROOT_PASSWORD=123456 -v /root/mysql/conf:/etc/mysql/conf.d -v
/root/mysql/data:/var/lib/mysql mysql:5.7
9ae2bbcf9698d020d0c757e8f24d106d230dbe52eee25e8b1372852855519411
[root@ ~]#: docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED
STATUS         PORTS                              NAMES
9ae2bbcf9698   mysql:5.7   "docker-entrypoint.s…"   3 seconds ago
Up 3 seconds   33060/tcp, 0.0.0.0:8888->3306/tcp   mysql01
```


新建连接

常规 高级 SSL SSH HTTP

连接名: test

主机名或 IP 地址: 101.132.145.185

端口: 8888

用户名: root

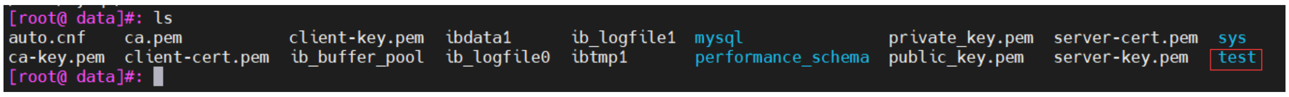
密码: ••••••

连接成功

使用navicat测试，可以发现能够连接成功，说明容器正常启动。

```
[root@ data]#: ls
auto.cnf     ca.pem           client-key.pem  ibdata1
ib_logfile1  mysql                private_key.pem  server-cert.pem
sys
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile0  ibtmp1
    performance_schema  public_key.pem   server-key.pem
[root@ data]#: pwd
/root/mysql/data
```

主机上`/root/mysql/data`已经有数据了。新建数据库`test`，在主机上也会多出相应的文件夹。



```
[root@ data]#: ls
auto.cnf    ca.pem           client-key.pem  ibdata1      ib_logfile1  mysql               private_key.pem  server-cert.pem  sys
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile0  ibtmp1       performance_schema  public_key.pem   server-key.pem   test
[root@ data]#:
```

删除该容器，主机上的数据也不会丢失。

4.4 具名挂载和匿名挂载