链滴

Docker 基础与实战，看这一篇就够了

作者：jianzh5

原文链接：https://ld246.com/article/1574648247226

来源网站：链滴

docker 基础

什么是Docker

Docker 使用 Google 公司推出的 Go 语言 进行开发实现，基于 Linux 内核的 cgroup，namespace 以及 AUFS 类的 Union FS 等技术，对进程进行封装隔离，属于 操作系统层面的虚拟化技术。由于隔的进程独立于宿主和其它的隔离的进程，因此也称其为容器。

Docker 在容器的基础上，进行了进一步的封装，从文件系统、网络互联到进程隔离等等，极大的简了容器的创建和维护。使得 Docker 技术比虚拟机技术更为轻便、快捷。

记住最重要的一点，Dokcer实际是宿主机的一个普通的进程，这也是Dokcer与传统虚拟化技术的最不同。

为什么要使用Docker

使用Docker最重要的一点就是Docker能保证运行环境的一致性，不会出现开发、测试、生产由于环配置不一致导致的各种问题，一次配置多次运行。使用Docker，可更快地打包、测试以及部署应用序，并可减少从编写到部署运行代码的周期。

docker 安装

- 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 版本否支持 Docker 。uname -r

```
[root@localhost ~]# uname -r
3.10.0-693.el7.x86_64
[root@localhost ~]#
```

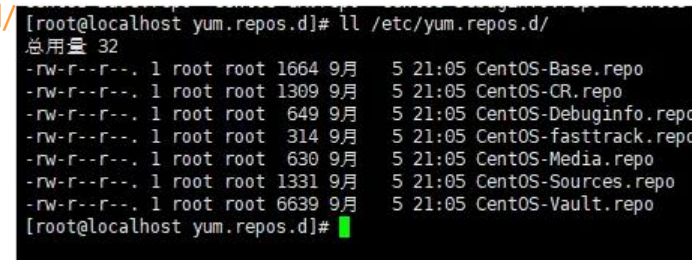
- 更新yum,升级到最新版本　yum update
- 卸载老版本的docker(若有)　yum remove docker　docker-common docker-selinux docker-eng

原文链接：Docker 基础与实战，看这一篇就够了

ne

执行该命令只会卸载Docker本身，而不会删除Docker存储的文件，例如镜像、容器、卷以及网络文等。这些文件保存在/var/lib/docker 目录中，需要手动删除。

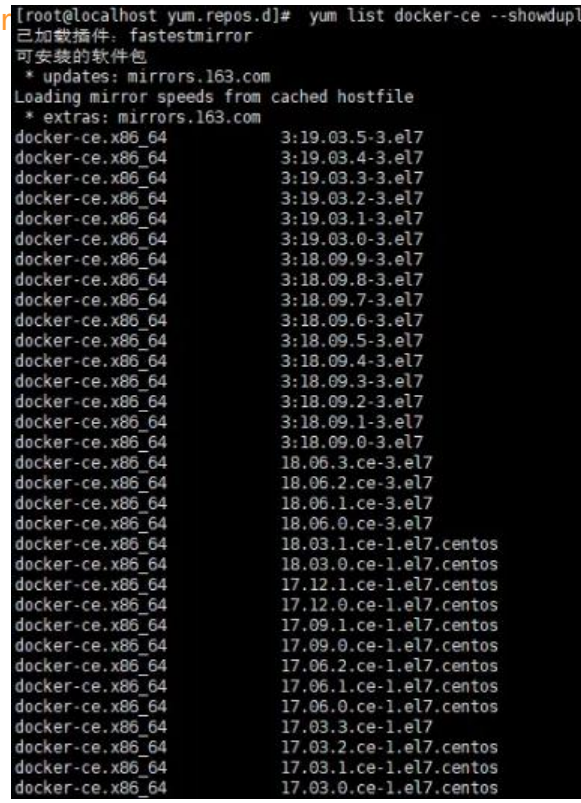
- 查看yum仓库，查看是否有docker　ll /etc/yum.repos.d/



如果用的厂商的服务器（阿里云、腾讯云）一般都会有docker仓库，如果用的是虚拟机或者公司的服器基本会没有。

- 安装软件包, yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的　ym install -y yum-utils device-mapper-persistent-data lvm2
- 安装仓库　yum-config-manager --add-repo https://download.docker.com/linux/centos/docer-ce.repo

- 查看docker版本　yum list docker-ce --showduplicates | sort -r



- 安装docker　yum install docker-ce

以上语句是是安装最新版本的Docker，你也可以通过yum install docker-ce-<VERSION> 安装指定本

- 启动docker　systemctl start docker
- 验证安装是否正确　dokcer run hello-world

```
[root@localhost yum.repos.d]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b[image]25: Pull complete
Dig[illegible]a256:4df8ca8a7e309c256d60d7971ea14c27672fc0d10c5f303856d7bc48f8cc17ff
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/
```

docker 重要命令

镜像相关

● 搜索镜像　docker search 如docker search nginx Docker就会在Docker Hub中搜索含有“ngin”这个关键词的镜像仓库

```
[root@VM_0_7_centos yum.repos.d]# docker search nginx
NAME                              DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
nginx                             Official build of Nginx.                        12245               [OK]
jwilder/nginx-proxy               Automated Nginx reverse proxy for docker con…   1690                                    [OK]
richarvey/nginx-php-fpm           Container running Nginx + PHP-FPM capable of…   746                                     [OK]
linuxserver/nginx                 An Nginx container, brought to you by LinuxS…   82
bitnami/nginx                     Bitnami nginx Docker Image                      73                                      [OK]
tiangolo/nginx-rtmp               Docker image with Nginx using the nginx-rtmp…   60                                      [OK]
nginxdemos/hello                  NGINX webserver that serves a simple page co…   32                                      [OK]
jc21/nginx-proxy-manager          Docker container for managing Nginx proxy ho…   30
jlesage/nginx-proxy-manager       Docker container for Nginx Proxy Manager        29                                      [OK]
nginx/nginx-ingress               NGINX Ingress Controller for Kubernetes         22
privatebin/nginx-fpm-alpine       PrivateBin running on an Nginx, php-fpm & Al…   19                                      [OK]
schmunk42/nginx-redirect          A very simple container to redirect HTTP tra…   17                                      [OK]
blacklabelops/nginx               Dockerized Nginx Reverse Proxy Server.          12                                      [OK]
centos/nginx-18-centos7           Platform for running nginx 1.8 or building n…   12
raulr/nginx-wordpress             Nginx front-end for the official wordpress:f…   12                                      [OK]
nginxinc/nginx-unprivileged       Unprivileged NGINX Dockerfiles                  11
centos/nginx-112-centos7          Platform for running nginx 1.12 or building …   10
nginx/nginx-prometheus-exporter   NGINX Prometheus Exporter                       9
sophos/nginx-vts-exporter         Simple server that scrapes Nginx vts stats a…   5                                       [OK]
1science/nginx                    Nginx Docker images that include Consul Temp…   5                                       [OK]
mailu/nginx                       Mailu nginx frontend                            5                                       [OK]
pebbletech/nginx-proxy            nginx-proxy sets up a container running ngin…   2                                       [OK]
ansibleplaybookbundle/nginx-apb   An APB to deploy NGINX                          1                                       [OK]
wodby/nginx                       Generic nginx                                   0                                       [OK]
centos/nginx-110-centos7          Platform for running nginx 1.10 or building …   0
```

● 下载镜像　docker pull 如docker pull nginx Docker就会在Docker Hub中下载含有“nginx”最版本的镜像

当然也可以使用docker pull reg.jianzh5.com/nginx:1.7.9 下载指定仓库地址标签的nginx镜像

● 列出镜像　docker images

```
[root@localhost yum.repos.d]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              4152a9608752        2 days ago          126MB
hello-world         latest              fce289e99eb9        10 months ago       1.84kB
```

● 删除镜像　docker rmi

如docker rmi hello-world删除我们刚刚下载的hello-world镜像

● 构建镜像　docker build

通过Dockerfile构建镜像，这个我们等下再拿出来详细说明。

容器相关

● 新建启动镜像　docker run

这个命令是我们最常用的命令，主要使用以下几个选项

① -d选项：表示后台运行

② -P选项（大写）：随机端口映射

③ -p选项（小写）：指定端口映射,**前面是宿主机端口后面是容器端口**，如docker run nginx -p 8080 80，将容器的80端口映射到宿主机的8080端口，然后使用localhost:8080就可以查看容器中nginx的迎页了④ -v选项：挂载宿主机目录，**前面是宿主机目录，后面是容器目录**,如docker run -d -p 80:80 v /dockerData/nginx/conf/nginx.conf:/etc/nginx/nginx.conf nginx 挂载宿主机的/dockerData/ginx/conf/nginx.conf的文件，这样就可以在宿主机对nginx进行参数配置了,注意目录需要用绝对路，不要使用相对路径，如果宿主机目录不存在则会自动创建。

⑤--rm : 停止容器后会直接删除容器，这个参数在测试是很有用，如docker run -d -p 80:80 --rm ng nx⑥--name : 给容器起个名字，否则会出现一长串的自定义名称如docker run -name niginx -d -p 0:80 - nginx

● 列出容器　docker ps

这个命令可以列出当前运行的容器，使用-a参数后列出所有的容器（包括已停止的）

● 停止容器　docker stop docker stop 5d034c6ea010 后面跟的是容器ID，也可以使用容器名称

● 启动停止的容器　docker start docker run是新建容器并启动，docker start是启动停止的容器,如ocker start 5d034c6ea010

● 重启容器　docker restart

此命令执行的过程实际是先执行docker stop,然后再执行docker start，如docker restart 5d034c6ea10

● 进入容器　docker exec -it 容器id /bin/bash

如docker exec -it 5d034c6ea010 /bin/bash，就相当于进入了容器本身的操作系统

● 删除容器　docker rm

如docker rm 5d034c6ea010 后面跟的是容器ID,删除容器之前需要先停止容器运行

● 数据拷贝　docker cp

此命令用于容器与宿主机之间进行数据拷贝,如docker cp 5d034c6ea010: /etc/nginx/nginx.conf /dckerData/nginx/conf/nginx.conf 将容器的目录文件拷贝到宿主机指定位置,容器ID可以替换成容器。

命令实战

如果我们需要一个nginx容器，并且需要在宿主机上直接修改nginx的配置文件、默认主页，在宿主机以实时看到容器nginx的日志。我们可以按照如下的方式一步一步完成。

● 使用--rm参数启动容器，方便删除　docker run -d -p 8081:80 --name nginx --rm nginx

● 进入容器，查看容器中配置文件、项目文件、日志文件的目录地址　docker exec -it 9123b67e42e /bin/bash

● 导出容器的配置文件　docker cp nginx:/etc/nginx/nginx.conf /dockerData/nginx/conf/nginx.onf导出配置文件 nginx.conf

docker cp nginx:/etc/nginx/conf.d /dockerData/nginx/conf/conf.d导出配置目录 conf.d

- 停止容器　docker stop 9123b67e428e，由于加了--rm参数，容器会自动删除
- 再以如下命令启动容器，完成目录挂载

```
docker run -d -p 8081:80 --name nginx \
-v /dockerData/nginx/conf/nginx.conf:/etc/nginx/nginx.conf \
-v /dockerData/nginx/conf/conf.d:/etc/nginx/conf.d \
-v /dockerData/nginx/www:/usr/share/nginx/html \
-v /dockerData/nginx/logs:/var/log/nginx nginx
```

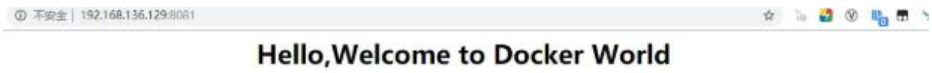
- 访问服务器地址　http://192.168.136.129:8081/

访问报错，这时候就进入宿主机的日志目录 /dockerData/nginx/logs查看日志2019/11/23 10:08:11 error] 6#6: *1 directory index of "/usr/share/nginx/html/" is forbidden, client: 192.168.136.1, s rver: localhost, request: "GET / HTTP/1.1", host: "192.168.136.129:8081"因为 /usr/share/nginx/ tml/被挂载到了服务器上面的 /dockerData/nginx/www目录下，原来的欢迎页面在 dockerData/ng nx/www是没有的，所有就报错了，这里我们随便建一个。

- 建立默认主页

```
#打开项目文件
cd /dockerData/nginx/www
#使用vim 创建并编辑文件
vi index.html
#此时我们会进入vim界面，按 i 插入，然后输入
<h1 align="center">Hello,Welcome to Docker World</h1>
#输入完后，按 esc，然后输入 :wq
```

- 再次访问浏览器地址



Dockerfile

我们可以使用Dockfile构建一个镜像，然后直接在docker中运行。Dockerfile文件为一个文本文件，面包含构建镜像所需的所有的命令，首先我们来认识一下Dockerfile文件中几个重要的指令。

指令详解

- FROM

选择一个基础镜像，然后在基础镜像上进行修改，比如构建一个SpringBoot项目的镜像，就需要选择java这个基础镜像，FROM需要作为Dockerfile中的第一条指令

如：FROM openjdk:8-jdk-alpine 基础镜像如果可以的话最好使用alpine版本的，采用alpline版本基础镜像构建出来的镜像会小很多。

- RUN

RUN指令用来执行命令行命令的。它有以下两种格式：

- shell 格式：RUN <命令>，就像直接在命令行中输入的命令一样。 RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
- exec 格式：RUN ["可执行文件", "参数1", "参数2"]，这更像是函数调用中的格式。

- CMD

此指令就是用于指定默认的容器主进程的启动命令的。

CMD指令格式和RUN相似，也是两种格式

- shell 格式：CMD <命令>
- exec 格式：CMD ["可执行文件", "参数1", "参数2"...]
- 参数列表格式：CMD ["参数1", "参数2"...]。在指定了 ENTRYPOINT 指令后，用 CMD 指定具的参数。

- ENTRYPOINT ENTRYPOINT 的格式和 RUN指令格式一样，分为 exec 格式和 shell 格式。 ENTYPOINT 的目的和 CMD 一样，都是在指定容器启动程序及参数。ENTRYPOINT 在运行时也可以替，不过比 CMD 要略显繁琐，需要通过 docker run 的参数 --entrypoint 来指定。

当指定了 ENTRYPOINT 后，CMD 的含义就发生了改变，不再是直接的运行其命令，而是将 CMD 内容作为参数传给 ENTRYPOINT 指令，换句话说实际执行时，将变为：

<ENTRYPOINT> "<CMD>"

- COPY & ADD

这2个指令都是复制文件，它将从构建上下文目录中 <源路径> 的文件/目录 复制到新的一层的镜像的 <目标路径> 位置。比如：COPY demo-test.jar app.jar 或 ADD demo-test.jar app.jar。

ADD指令比 COPY高级点，可以指定一个URL地址，这样Docker引擎会去下载这个URL的文件，如果DD后面是一个 tar文件的话，Dokcer引擎还会去解压缩。

我们在构建镜像时尽可能使用 COPY，因为 COPY 的语义很明确，就是复制文件而已，而 ADD 则含了更复杂的功能，其行为也不一定很清晰。

- EXPOSE

声明容器运行时的端口，这只是一个声明，在运行时并不会因为这个声明应用就会开启这个端口的服。在 Dockerfile 中写入这样的声明有两个好处，一个是帮助镜像使用者理解这个镜像服务的守护端，以方便配置映射；另一个用处则是在运行时使用随机端口映射时，也就是 docker run -P 时，会自随机映射 EXPOSE 的端口。

要将 EXPOSE 和在运行时使用 -p <宿主端口>:<容器端口> 区分开来。-p，是映射宿主端口和容器口，换句话说，就是将容器的对应端口服务公开给外界访问，而 EXPOSE 仅仅是声明容器打算使用什端口而已，并不会自动在宿主进行端口映射。

- ENV

这个指令很简单，就是设置环境变量，无论是后面的其它指令，如 RUN，还是运行时的应用，都可直接使用这里定义的环境变量。它有如下两种格式：

- ENV <key> <value>
- ENV <key1>=<value1> <key2>=<value2>...

- VOLUME

该指令使容器中的一个目录具有持久化存储的功能，该目录可被容器本身使用，也可共享给其他容器当容器中的应用有持久化数据的需求时可以在Dockerfile中使用该指令。如 VOLUME /tmp

这里的 /tmp 目录就会在运行时自动挂载为匿名卷，任何向 /tmp 中写入的信息都不会记录进容器存层，从而保证了容器存储层的无状态化。当然，运行时可以覆盖这个挂载设置。比如：

```
docker run -d -v mydata:/tmp xxxx
```

- LABEL

你可以为你的镜像添加labels，用来组织镜像，记录版本描述，或者其他原因，对应每个label，增加LABEL开头的行，和一个或者多个键值对。如下所示：

```
LABEL version="1.0"
LABEL description="test"
```

Dockerfile实战

我们以一个简单的SpringBoot项目为例构建基于SpringBoot应用的镜像。

功能很简单，只是对外提供了一个 say接口，在进入这个方法的时候打印出一行日志，并将日志写入志文件。

```
@SpringBootApplication
@RestController
@Log4j2
public class DockerApplication {

    public static void main(String[] args) {
        SpringApplication.run(DockerApplication.class, args);
    }

    @GetMapping("/say")
    public String say(){
        log.info("get say request...");
        return "Hello,Java日知录";
    }

}
```

我们使用maven将其打包成jar文件，放入一个单独的文件夹,然后按照下面步骤一步步构建镜像并执行

- 在当前文件夹建立Dockerfile文件，文件内容如下：

```
FROM openjdk:8-jdk-alpine
#将容器中的/tmp目录作为持久化目录
VOLUME /tmp
#暴露端口
EXPOSE 8080
#复制文件
COPY docker-demo.jar app.jar
#配置容器启动后执行的命令
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

- 使用如下命令构建镜像

docker built -t springboot:v1.0 .

```
[root@localhost docker]# docker build -t springboot:v1.0 .
Sending build context to Docker daemon  19.51MB
Step 1/5 : FROM openjdk:8-jdk-alpine
 ---> a3562aa0b991
Step 2/5 : VOLUME /tmp
 ---> Running in 01d753104dd0
Removing intermediate container 01d753104dd0
 ---> 6d9934a81c15
Step 3/5 : EXPOSE 9090
 ---> Running in f66c30b97e78
Removing intermediate container f66c30b97e78
 ---> f16781bdd6ba
Step 4/5 : COPY docker-demo.jar app.jar
 ---> 5288afe82080
Step 5/5 : ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
 ---> Running in 46b52b0ff959
Removing intermediate container 46b52b0ff959
 ---> 3666da7bec35
Successfully built 3666da7bec35
Successfully tagged springboot:v1.0
```

-t 指定镜像的名称及版本号，注意后面需要以 . 结尾。

- 查看镜像文件

```
[root@localhost docker]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
springboot          v1.0                3666da7bec35        11 minutes ago      124MB
nginx               latest              4152a9608752        3 days ago          126MB
openjdk             8-jdk-alpine        a3562aa0b991        6 months ago        105MB
```

- 运行构建的镜像　docker run -v /app/docker/logs:/logs -p 8080:8080 --rm --name springboot springboot:v1.0
- 浏览器访问　http://192.168.136.129:8080/say

- 在宿主机上实时查看日志

tail -100f /app/docker/logs/docker-demo-info.log

```
[root@localhost logs]# tail -100f docker-demo-info.log
2019-11-23 14:47:37,519 INFO  DockerApplication:50 - Starting DockerApplication v0.0.1-SNAPSHOT on 71f5c2a0094a with PID 1 (/app.jar started by root in /)
2019-11-23 14:47:37,546 INFO  DockerApplication:648 - No active profile set, falling back to default profiles: default
2019-11-23 14:47:39,355 INFO  Http11NioProtocol:173 - Initializing ProtocolHandler ["http-nio-8080"]
2019-11-23 14:47:39,402 INFO  StandardService:173 - Starting service [Tomcat]
2019-11-23 14:47:39,408 INFO  StandardEngine:173 - Starting Servlet engine: [Apache Tomcat/9.0.26]
2019-11-23 14:47:39,543 INFO  [/]:173 - Initializing Spring embedded WebApplicationContext
2019-11-23 14:47:40,341 INFO  Http11NioProtocol:173 - Starting ProtocolHandler ["http-nio-8080"]
2019-11-23 14:47:40,408 INFO  DockerApplication:59 - Started DockerApplication in 3.51 seconds (JVM running for 4.778)
2019-11-23 14:50:56,184 INFO  [/]:173 - Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-11-23 14:50:56,266 INFO  DockerApplication:20 - get say request...
```

小结

本文详细介绍了日常使用Docker过程中经常用到的相关指令，每小结都用一个综合实战将所有指令合在一起使用，领悟这些实际用例可以让你学习Docker事半功倍。