镜像生命周期管理

这是本专栏的第三部分：镜像篇，共 8 篇，帮助你由浅入深地认识和掌握 Docker 镜像的管理、构建、分发及原理。同时会深入源码了解 Docker 的构建系统，之后会结合实际经验，为你介绍如何在生产环境中对镜像镜像优化。下面我们一起进入第一篇的内容，主要涉及镜像的生命周期管理。

备注：为了避免术语的混淆，本课程中的"镜像"均指"Docker Image"，源码解析使用的是 Docker CE v19.03.4 版本。

在之前的内容中，我们多次使用 `docker run` 命令创建容器。比如，我们经常使用的一条命令 `docker run --rm -it alpine`，如果是首次运行此命令则会看到类似下面的输出：

复制

```
(MoeLove) ➜  ~ docker run --rm -it alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
89d9c30c1d48: Already exists
Digest: sha256:c19173c5ada610a5989151111163d28a67368362762534d8a8121ce95cf2bd5a
Status: Downloaded newer image for alpine:latest
/ #
```

通过这条命令，我们将名为 `alpine:latest` 的镜像下载到了本地，并使用此镜像启动了一个容器。我们来分解下此命令的输出，由此正式进入本篇。

docker run 下载镜像的行为解析

查找镜像

复制

```
Unable to find image 'alpine:latest' locally
```

首先 Docker 进行了本地镜像的查找，但是未在本地找到 `alpine:latest` 的镜像。这里有两个需要注意的点。

在给 `docker run` 传递镜像参数时候，它的基本格式为：

复制

```
name [ ":" tag ] [ "@" digest ]
```

但如果将其 name 稍微进行展开，则可扩展为以下形式：

复制

```
[registry "/"] [username "/"] [reponame "/"] pathname
```

此处，虽然我们在执行命令的时候给它传递的只是简单的 alpine 而已，但实际 Docker 在处理的过程中，会将其补全为 index.docker.io/library/alpine:latest。

默认配置可直接查看源码。

复制

```
//cli/vendor/github.com/docker/distribution/reference/normalize.go#L12-L17
var (
    legacyDefaultDomain = "index.docker.io"
    defaultDomain       = "docker.io"
    officialRepoName    = "library"
    defaultTag          = "latest"
)
```

index.docker.io/library 处存放的镜像为 Docker 官方镜像，所以在我们使用时，可以省略掉其中的一些内容，直接使用 alpine 作为替代。

复制

```
latest: Pulling from library/alpine
```

从这一行输出中也能印证一些我们前面的描述。

另一点是：如果未指定 Tag，则默认会使用 latest 作为镜像的 Tag。

下载镜像

接下来，Docker 去下载了此镜像，并使用此镜像启动了容器。

实际上，Docker 在首次执行 `docker run --rm -it alpine` 这条命令时，一共做了两次创建容器的动作，具体源码 如下：

复制

```
// cli/cli/command/container/create.go#L216-L241
    response, err := dockerCli.Client().ContainerCreate(ctx, config, hostConfig, netw

    if err != nil {
        if apiclient.IsErrNotFound(err) && namedRef != nil {
            fmt.Fprintf(stderr, "Unable to find image '%s' locally\n", reference.Fami


            // we don't want to write to stdout anything apart from container.ID
            if err := pullImage(ctx, dockerCli, config.Image, opts.platform, stderr);
                return nil, err
            }
            if taggedRef, ok := namedRef.(reference.NamedTagged); ok && trustedRef !=
                if err := image.TagTrusted(ctx, dockerCli, trustedRef, taggedRef); er
                    return nil, err
                }
            }
            // Retry
            var retryErr error
            response, retryErr = dockerCli.Client().ContainerCreate(ctx, config, host
            if retryErr != nil {
                return nil, retryErr
            }
        } else {
            return nil, err
        }
    }
```

即：直接从镜像创建容器，如果是因为镜像不存在导致失败，则下载镜像，之后重试创建容器。

镜像操作：增

注意：Docker 镜像可以存储在远端仓库中，比如 Docker Hub，当然也可以存储在运行着 Docker 服务的机器中。**本篇所指的镜像相关的操作，均是对运行着 Docker 服务的机器而言的。**

前面的例子中，已经展示了一种新增 Docker 镜像的方法了。即：通过 `docker run` 命令启动容器时，自动下载镜像。

另一种，则是通过 `docker pull` 或者 `docker image pull` 命令来主动下载镜像。例如：

复制

```
(MoeLove) ➜  ~ docker image pull --disable-content-trust=false alpine
Using default tag: latest
Pull (1 of 1): alpine:latest@sha256:c19173c5ada610a5989151111163d28a67368362762534
sha256:c19173c5ada610a5989151111163d28a67368362762534d8a8121ce95cf2bd5a: Pulling f
89d9c30c1d48: Already exists
Digest: sha256:c19173c5ada610a5989151111163d28a67368362762534d8a8121ce95cf2bd5a
Status: Downloaded newer image for alpine@sha256:c19173c5ada610a5989151111163d28a6
Tagging alpine@sha256:c19173c5ada610a5989151111163d28a67368362762534d8a8121ce95cf2
docker.io/library/alpine:latest
```

当然，`docker pull` 还支持一个 `--all-tags` 参数和 `--platform` 可以用于下载全部的 tag 和下载指定的平台的镜像。如果没有指定 `--platform` 参数，则默认使用 Docker 服务所运行机器的架构。（此功能当前仍然处于实验性质）

第三种方式：加载（load）

复制

```
# 查询镜像
(MoeLove) ➜  ~ docker image ls alpine:latest
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              latest              965ea09ff2eb        3 weeks ago         5.

# 使用 docker save 将镜像保存为一个 tar 包
(MoeLove) ➜  ~ docker image save -o alpine.tar alpine:latest

# 本地删除此镜像
(MoeLove) ➜  ~ docker rmi alpine:latest
Untagged: alpine:latest
Deleted: sha256:965ea09ff2ebd2b9eeec88cd822ce156f6674c7e99be082c7efac3c62f3ff652
# 查询镜像
(MoeLove) ➜  ~ docker image ls alpine:latest
REPOSITORY          TAG                 IMAGE ID            CREATED             SI

# 使用 docker image load 将镜像导入
(MoeLove) ➜  ~ docker image load -i alpine.tar
Loaded image: alpine:latest
Loaded image ID: sha256:965ea09ff2ebd2b9eeec88cd822ce156f6674c7e99be082c7efac3c62f
Loaded image ID: sha256:cdf98d1859c1beb33ec70507249d34bacf888d59c24df3204057f9a6c7
```

第四种方式：导入（import）

复制

```
# 启动一个容器
(MoeLove) ➜  ~ docker run -d  alpine sleep 999
06bdbb92222bf2aed9ff55d9af5a023928fae98533b116f8405f18b0f6860c0e

# 使用 docker export 将容器导出为一个 tar 包
(MoeLove) ➜  ~ docker export -o alpine-container.tar $(docker ps -ql)

# 使用 docker image import 将 tar 包导入，并传递新的 名称和 Tag
(MoeLove) ➜  ~ docker image import alpine-container.tar alpine:container
sha256:fc557c18e90e3374e604b9709786bc8d5aed7c58b14cb183786d7c0ab3d5f1ab

# 查询镜像
(MoeLove) ➜  ~ docker image ls alpine:container
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              container           fc557c18e90e        4 seconds ago       5.

# 使用镜像启动一个容器
(MoeLove) ➜  ~ docker run --rm -it  alpine:container sh
/ # cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.10.3
PRETTY_NAME="Alpine Linux v3.10"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://bugs.alpinelinux.org/"
```

第五种方式：commit

复制

```
# 查看我们刚才已经运行的容器
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
06bdbb92222b        alpine              "sleep 999"         6 minutes ago       Up

# 使用 docker commit 从正在运行的容器创建一个新镜像
(MoeLove) ➜  ~ docker commit  -a "Jintao Zhang" $(docker ps -ql) alpine:commit
sha256:f8a95cb5a1bed08f8b7ea80a4c8e61bdfc1ec69830b856c5bf97b4c9d7727b1b

# 查询镜像
(MoeLove) ➜  ~ docker image ls alpine:commit
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              commit              f8a95cb5a1be        5 seconds ago       5.
```

第六种方式：build

复制

```
(MoeLove) ➜  ~ mkdir alpine-dockerfile
(MoeLove) ➜  ~ cd alpine-dockerfile

# 写一个 Dockerfile
(MoeLove) ➜  alpine-dockerfile cat <<EOF > Dockerfile
FROM alpine
RUN apk add --no-cache curl
EOF

# 使用 docker build 构建镜像，通过 -t 指定其名称和 Tag
(MoeLove) ➜  alpine-dockerfile docker build -t alpine:from-dockerfile .
[+] Building 5.2s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 142B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:latest
 => CACHED [1/2] FROM docker.io/library/alpine
 => [2/2] RUN apk add --no-cache curl
 => exporting to image
 => => exporting layers
 => => writing image sha256:c1dce6f3d0b9f75e0f6e1e06298801d6b0b19b466ef2cf4bea015f
 => => naming to docker.io/library/alpine:from-dockerfile

# 查询镜像
(MoeLove) ➜  alpine-dockerfile docker image ls alpine:from-dockerfile
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              from-dockerfile     c1dce6f3d0b9        11 seconds ago      6.
```

以上便是 6 种主要用于本地新增镜像的方式。但这里我没有对每种方式产生的影响做过多的介绍，在后续内容中，我们会继续深入学习。

镜像操作：查

上文中，已经多次介绍了，通过 `docker image ls` 可以查询镜像。它支持查询指定 Tag 的镜像，或在不指定具体 Tag 时查所有相同 Name 的镜像，或着不指定 Name 时，查询所有镜像。

复制

```
# 查询具体名称和 Tag 的镜像
(MoeLove) ➜  alpine-dockerfile docker image ls alpine:from-dockerfile
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              from-dockerfile     c1dce6f3d0b9        6 minutes ago       6.

# 查询具有相同名称的镜像
(MoeLove) ➜  alpine-dockerfile docker image ls alpine
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              from-dockerfile     c1dce6f3d0b9        6 minutes ago       6.
alpine              commit              f8a95cb5a1be        14 minutes ago      5.
alpine              container           fc557c18e90e        20 minutes ago      5.
alpine              latest              965ea09ff2eb        3 weeks ago         5.
alpine              <none>              cdf98d1859c1        7 months ago        5.
```

镜像操作：改

我们有四种方式做出修改。

只修改镜像名称或 Tag

通过 `docker image tag` 命令可实现镜像的“重命名”，例如：

复制

```
(MoeLove) ➜  ~ docker image ls alpine:latest
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              latest              965ea09ff2eb        3 weeks ago         5.

# 为镜像“重命名”
(MoeLove) ➜  ~ docker image tag alpine:latest alpine-new-name:new-tag

(MoeLove) ➜  ~ docker image ls alpine-new-name:new-tag
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine-new-name     new-tag             965ea09ff2eb        3 weeks ago         5
```

基于原镜像对内容做修改

复制

```
# 启动一个容器，为它新增一个文件
(MoeLove) ➜  ~ docker run --rm -it alpine:latest
/ # echo "在 /etc 下新增一个文件" > /etc/docker-course
```

打开另一个窗口，执行以下命令：

复制

```
# 查看刚才启动的容器
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             S
5504caff6d57        alpine:latest       "/bin/sh"           About a minute ago  U

# commit 成一个新镜像
(MoeLove) ➜  ~ docker commit  $(docker ps -ql) alpine:with-new-file
sha256:ef3592e1823b712b05911d8546f96098e47a1b06b53cfe685861d510c124a2dd

# 查询镜像
(MoeLove) ➜  ~ docker image ls alpine:with-new-file
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              with-new-file       ef3592e1823b        6 seconds ago       5.

# 启动容器
(MoeLove) ➜  ~ docker run --rm -it alpine:with-new-file
/ # cat /etc/docker-course
在 /etc 下新增一个文件
```

可以看到，我们通过 `docker commit` 创建的新镜像，保留了我们之前的修改。

不保留原镜像信息做修改

我们仍然使用前面例子中启动的容器，创建了 /etc/docker-course 文件。

打开另一个窗口，执行以下命令：

复制

```
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
5504caff6d57        alpine:latest       "/bin/sh"           5 minutes ago       Up

# 将容器导出为一个 tar 包
(MoeLove) ➜  ~ docker export -o new-file.tar $(docker ps -ql)

# import 该 tar 包
(MoeLove) ➜  ~ docker import  new-file.tar alpine:using-export
sha256:a5c6f761fba82341f739240eb7abf4191236adae3fd2a9b82096157b9c3d3dc1
(MoeLove) ➜  ~ docker image ls alpine:using-export
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              using-export        a5c6f761fba8        4 seconds ago       5

# 使用新镜像启动容器 - 这里有个报错，这是因为 export 的行为导致的，之后再介绍
(MoeLove) ➜  ~ docker run --rm -it alpine:using-export
docker: Error response from daemon: No command specified.
See 'docker run --help'.

# 为容器指定启动后执行的命令，可正常启动容器
(MoeLove) ➜  ~ docker run --rm -it alpine:using-export sh
/ # cat /etc/docker-course
在 /etc 下新增一个文件
```

使用 Dockerfile

在前面已经介绍了使用 Dockerfile 来构建镜像的方式，这里再次进行演示：

复制

```
(MoeLove) ➜  ~ cd alpine-dockerfile
# 新建一个 Dockerfile
(MoeLove) ➜  alpine-dockerfile cat <<EOF > Dockerfile
FROM alpine
RUN echo "在 /etc 下新增一个文件" > /etc/docker-course
EOF

# 构建新镜像
(MoeLove) ➜  alpine-dockerfile docker build -t alpine:using-dockerfile-add-newfil
[+] Building 0.1s (4/5)
[+] Building 0.3s (4/5)
[+] Building 0.5s (4/5)
[+] Building 0.6s (5/5)
[+] Building 0.6s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 177B
 => [internal] load .dockerignore
 => [internal] load metadata for docker.io/library/alpine:latest
 => CACHED [1/2] FROM docker.io/library/alpine
 => [2/2] RUN echo "在 /etc 下新增一个文件" > /etc/docker-course
   0.5s
 => exporting to image
 => => exporting layers
 => => writing image sha256:91ee49603cd12793a1c8e2512103394caecd2a8781f85c1317096f
 => => naming to docker.io/library/alpine:using-dockerfile-add-newfile

# 查询镜像
(MoeLove) ➜  alpine-dockerfile docker image ls alpine:using-dockerfile-add-newfil
REPOSITORY          TAG                              IMAGE ID            CREATED
alpine              using-dockerfile-add-newfile     91ee49603cd1        11 seconds

# 启动容器
(MoeLove) ➜  alpine-dockerfile docker run --rm -it alpine:using-dockerfile-add-ne
/ # cat /etc/docker-course
在 /etc 下新增一个文件
```

可以看到，除了使用 `docker tag` 的方式外，其他方式都是新建了一个镜像。这也从侧面反映了 Docker 镜像内容的唯一性，内容如果发生改变，则镜像就不再是原来那个了。

镜像操作：删

前面我们创建了很多镜像，那如何对它进行删除呢？使用 `docker image rm` 即可。

复制

```
(MoeLove) ➜  ~ docker image ls alpine
REPOSITORY          TAG                              IMAGE ID            CREATED
alpine              using-dockerfile-add-newfile     91ee49603cd1        5 minutes a
alpine              using-export                     a5c6f761fba8        19 minutes
alpine              with-new-file                    ef3592e1823b        23 minutes
alpine              from-dockerfile                  c1dce6f3d0b9        51 minutes
alpine              commit                           f8a95cb5a1be        58 minutes
alpine              container                        fc557c18e90e        About an h

# 一个个删除
(MoeLove) ➜  ~ docker image rm alpine:using-dockerfile-add-newfile alpine:using-
Untagged: alpine:using-dockerfile-add-newfile
Deleted: sha256:91ee49603cd12793a1c8e2512103394caecd2a8781f85c1317096f3ebd908236
Untagged: alpine:using-export
Deleted: sha256:a5c6f761fba82341f739240eb7abf4191236adae3fd2a9b82096157b9c3d3dc1
Deleted: sha256:e6a6b86da14eeec38b50d1036d3e642aa1e2b86637476115e9f4146d886fc83e
Untagged: alpine:with-new-file
Deleted: sha256:ef3592e1823b712b05911d8546f96098e47a1b06b53cfe685861d510c124a2dd
Deleted: sha256:893539240a9abd401beb53b7a19b3b38a2725e87e86bada816aedd3ccbd1ea8c

# 查看结果
(MoeLove) ➜  ~ docker image ls alpine
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
alpine              from-dockerfile     c1dce6f3d0b9        51 minutes ago      6.
alpine              commit              f8a95cb5a1be        59 minutes ago      5
alpine              container           fc557c18e90e        About an hour ago   

# 批量删除
(MoeLove) ➜  ~ docker image rm $(docker image ls alpine -q)
Untagged: alpine:from-dockerfile
Deleted: sha256:c1dce6f3d0b9f75e0f6e1e06298801d6b0b19b466ef2cf4bea015fe469442da2
Untagged: alpine:commit
Deleted: sha256:f8a95cb5a1bed08f8b7ea80a4c8e61bdfc1ec69830b856c5bf97b4c9d7727b1b
Untagged: alpine:container
Deleted: sha256:fc557c18e90e3374e604b9709786bc8d5aed7c58b14cb183786d7c0ab3d5f1ab
Deleted: sha256:7a3bec6d41c7e0af550b27055d00af898e6ed17cff46e29c75bdc5e959d56d74
```

总结

本篇我为你介绍了容器镜像生命周期管理的基本操作，通过实际的操作来加深印象。

本篇的内容相对简单，但是在实际使用 Docker 的过程中，使用是非常频繁的。后续内容中，我们会对这些内容及其原理进行更多深入的介绍。