Docker 核心架构...

上一篇，我们正式进入了本专栏第四部分 “架构篇”的学习。本篇是第一个主题 “Docker 核心架构及拆解” 的中篇。在上一篇中，我为你从较高的层次介绍了 Docker 基础的核心组件，包括 containerd 和 runc 等相关组件，知道了容器创建的一个基本的组件间的调用关系。本篇，我来为你介绍尚未介绍到的其他相关组件。

docker-proxy

我们来回忆下之前介绍过的，如何将容器的端口暴露出来。在 `docker run ...` 的时候，通过 `-p` 或者 `-P` 选项可以将容器内的端口暴露出来，映射到主机上。

比如，我运行一个 Nginx 的容器，并将其 80 端口映射到主机的 8765 端口上：

复制

```
(MoeLove) ➜  ~ docker run --rm -d -p 8765:80 nginx
6e2597332e8d6ba74c3b0f59122743148c2a5e83be5763dc11d676abe3518f07
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
6e2597332e8d        nginx               "nginx -g 'daemon of…"   51 seconds ago
(MoeLove) ➜  ~ curl -I localhost:8765
HTTP/1.1 200 OK
Server: nginx/1.17.6
Date: Sun, 22 Dec 2019 23:51:31 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 19 Nov 2019 12:50:08 GMT
Connection: keep-alive
ETag: "5dd3e500-264"
Accept-Ranges: bytes
```

可以看到，通过 curl 可以成功通过此 8765 端口访问到 Nginx。在主机上查看此端口的监听情况：

复制

```
(MoeLove) ➜  ~ sudo netstat -ntlp |grep 8765
tcp6       0      0 :::8765                 :::*                    LISTEN      27
```

可以看到改端口是被 27363 号进程占用，进程名为 `docker-proxy`，查看进程详细信息：

复制

```
(MoeLove) ➜  ~ ps -ef |grep  27363 |grep -v grep
root     27363 31274  0 07:49 ?        00:00:00 /usr/bin/docker-proxy -proto tcp -
```

我们也可以直接执行以下命令，查看 `docker-proxy` 所支持参数的含义：

复制

```
(MoeLove) ➜  ~ docker-proxy -h
Usage of docker-proxy:
  -container-ip string
        container ip
  -container-port int
        container port (default -1)
  -host-ip string
        host ip
  -host-port int
        host port (default -1)
  -proto string
        proxy protocol (default "tcp")
```

很明显 docker-proxy 的参数的就是容器和主机这两端。

根据这里的现象，我们可以暂时认为 docker-proxy 主要是在负责容器的端口映射到主机上，但是 dockerd 包含一个 `--userland-proxy` 的参数默认是 true，我们可以将其设置为 false 来看看会发生什么：

复制

```
(MoeLove) ➜  dockerd --userland-proxy=false
```

使用以上命令启动 Docker Daemon，接下来按前面的例子，再次启动一个容器看看会发生什么：

```
(MoeLove) ➜  ~ docker run -d -p 9765:80 nginx
8940aa6b4f8b4d56e84899d57d0ecbf46d045540390bddb312054010562fc91a
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
8940aa6b4f8b        nginx               "nginx -g 'daemon of…"   7 minutes ago

(MoeLove) ➜  ~ curl -I localhost:9765
HTTP/1.1 200 OK
Server: nginx/1.17.6
Date: Sun, 22 Dec 2019 23:59:31 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 19 Nov 2019 12:50:08 GMT
Connection: keep-alive
ETag: "5dd3e500-264"
Accept-Ranges: bytes
```

会发现基本没什么变化，再次查看此端口的占用情况：

```
(MoeLove) ➜  ~ netstat  -ntlp |grep 9765 |grep -v grep
tcp        0      0 :::9765                 :::*                    LISTEN      31
```

会发现，这次端口是被 dockerd 占用的而不再是 docker-proxy，同时在机器上查看也发现没有任何 docker-proxy 相关的进程。 说明通过 `--userland-proxy=false` 可以去除掉对 docker-proxy 的依赖。

这样虽然对于基础使用没什么太多影响，不过也会有些其他问题，在本篇中我们暂且跳过，在第 44 篇《docker-proxy 的原理》中，我会为你深入内部解析 docker-proxy 的原理，以及 userland-proxy 是否启用 docker-proxy 会有哪些区别和影响。我们继续进行本篇的后续内容。

docker-init

在 `dockerd` 和 `docker run` 时，都支持一个 `--init` 的参数，它的含义及对我们使用的区别是什么呢？我们来进行以下实践：

```
(MoeLove) ➜  dockerd --init=true
```

使用以上命令启动 Docker Daemon，需要注意的是 `--init` 默认是 false 的，这里我们将其设置为 true。

接下来启动一个容器，进行测试：

```
(MoeLove) ➜  docker run --rm -d redis:alpine
792d699b1e4a05c020ff87927904804bdc205c4a46ab1a066987aa1bbc5c8dcd
(MoeLove) ➜  docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
792d699b1e4a        redis:alpine        "docker-entrypoint.s…"   3 seconds ago
(MoeLove) ➜  docker exec $(docker ps -ql) ps -ef
PID   USER     TIME  COMMAND
    1 root      0:00 /sbin/docker-init -- docker-entrypoint.sh redis-server
    6 redis     0:00 redis-server
   13 root      0:00 ps -ef
```

会发现，容器中 PID 为 1 的进程不是 redis-server 了，而是 docker-init。

接下来，我们在 `docker run` 中传递 `--init=false` 的选项，启动一个容器：

```
(MoeLove) ➜  docker run --rm -d --init=false redis:alpine
1868c2c443b600d18e6c064cc8436c9bd45c8738a89bc8c5568cb92c9791946c
(MoeLove) ➜  docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
1868c2c443b6        redis:alpine        "docker-entrypoint.s…"   3 seconds ago
(MoeLove) ➜  docker exec $(docker ps -ql) ps -ef
PID   USER     TIME  COMMAND
    1 redis     0:00 redis-server
   12 root      0:00 ps -ef
```

可以看到容器中 PID 为 1 的进程成了 redis-server。

在我们日常的使用中，通常不会为 dockerd 配置 `--init=true` 的选项，如果配置了此参数的话，容器内所有进程都将成为 docker-init 的子进程，外部的的信号将由 docker-init 进行处理，比如说 `docker stop` 发送的停止容器的信号等。

但如果保持默认配置的话，docker-init 实际不会被使用的，所以它也是一个可选项。

总结

本篇，我为你介绍了在安装 Docker 时，安装至系统中的两个工具 docker-proxy 和 docker-init。如果保持默认配置，docker-proxy 将会绑定我们通过 `-p` 选项暴露的端口号；而 docker-init 在默认配置下是不启用的，除非你更改 dockerd 的配置，或者是在 `docker run` 的时候加上 `--init` 参数才会启用。

docker-proxy 在后续课程中会有专门的一篇来讲述其工作原理及对我们的影响。现在重点说一下 docker-init 对实际使用时的影响。

如果容器内的应用程序不处理或者忽略掉了全部的信号，那很可能会导致启动在前台的容器（`docker run -it xxx` 这种方式）无法直接通过 Ctrl + C 的方式停止容器。

但如果加上 `--init` 选项的话，容器内所有进程都是 docker-init 的子进程，由 docker-init 负责接收和处理中止信号也就很容易可以中止容器了。

下篇，是本主题的第三篇，我会为你将 Docker 的组件和核心架构给组织起来，并为后续课程做好铺垫和准备。