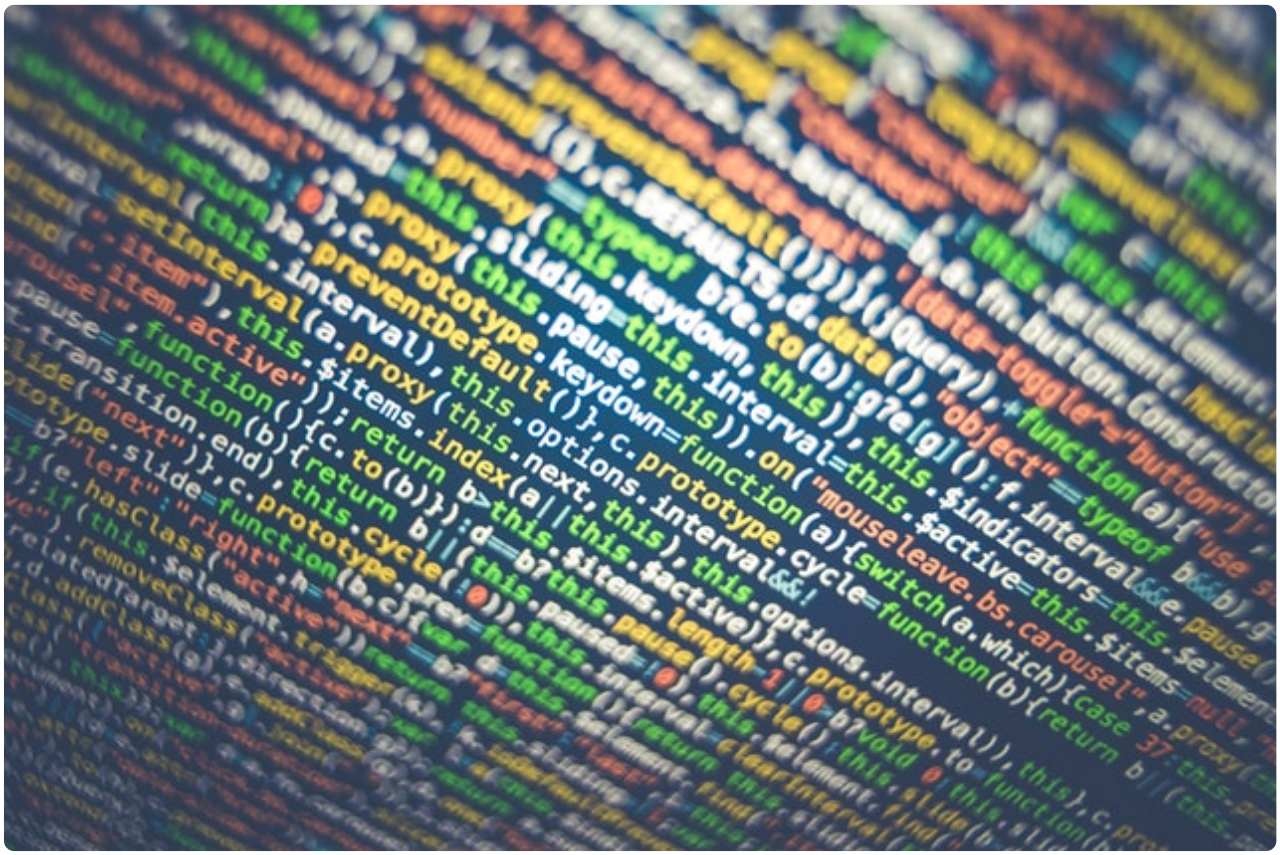
18 Docker 网络模式

更新时间：2020-08-26 09:57:10



> 机会不会上门来找人，只有人去找机会。——狄更斯

Docker 网络模式

在上一篇文章我们介绍了 Docker 网络的工作模式，包括 docker0 网桥和 iptables 等。其中 docker0 网桥是 Docker 默认网络模式，也就是 bridge 模式。

1. 概览

Docker 现在的网络模块是插件式的，只要按既定协议实现就可以使用。Docker 默认实现了五种网络模式如下（现在网络上面可以搜索到的文章都说 Docker 支持四种网络模式，其实是不准确的），我们可以在 `Docker run` 的时候通过参数 `--net` 指定。

bridge 模式

Docker 的默认网络模式。这种模式会将创建出来的所有 Docker 容器链接到 docker0 网桥或者自定义网桥上，所有的 Docker 容器处于同一个子网。

***host* 模式**

顾名思义，这种模式下，Docker 容器和宿主机使用同一个网络协议栈，也就是同一个 network namespace，和宿主机共享网卡、IP、端口等信息。好处是性能更好，缺点也很明显，没有做网络隔离。

***overlay* 模式**

这种模式在多个 Docker daemon 主机之间创建一个分布式网络，该网络位于 Docker 主机层次之上，允许容器之间加密通讯，需要处理容器之间和主机之间的网络包。

***macvlan* 模式**

macvlan 是 Linux 的一个内核模块，算是一个比较新的特性。本质上是一种网卡虚拟化技术，通过 macvlan 可以在同一个物理网卡上虚拟出多个网卡，通过不同的 Mac 地址在数据链路层进行网络数据的转发，一块网卡上配置多个 Mac 地址。Docker 的 macvlan 网络实际上就是使用 Linux 提供的 macvlan 驱动。

***none* 模式**

这种模式下 Docker 容器拥有自己的 network namespace，但是并不会做任何网络配置。换句话说，这个 Docker 容器除了 network namespace 自带的 lo 网卡（loopback，127.0.0.1）外没有其他任何网卡、IP 等信息。这种模式如果不做额外配置是无法使用的，要使用需要自己添加网卡等，也就是它给了用户最大的自由度。

network plugins

除了上面默认实现的五种网络模式，你还可以使用第三方的网络插件。这部分需要较多篇幅，本篇文章暂时不介绍了。感兴趣的同学可以参考这篇文章。

2. Bridge 模式

虽然上一篇文章已经基于 bridge 模式做了分析，这里还是简单介绍一下 bridge 模式下，Docker 初始化容器网络的步骤：

- 创建一对虚拟网卡（veth pair）。
- 赋予其中一块网卡类似 “vethxxx” 的名字，将其绑定到 docker0 或者自定义网桥，用来连接宿主机的 network namespace。
- 将 veth pair 的另一块网卡放入新创建的 Docker 容器的 network namespace 中，命名为 eth0。
- 从网桥的子网中选取一个未使用的 IP 分配给 eth0，并为 Docker 容器网络设置路由和网格。

Docker 会自动创建 docker0 网桥，使用 bridge 模式的 Docker 容器默认使用 docker0 网桥，除此之外，你也可以使用自定义网桥（User-defined bridge network）。自定义网桥和默认 docker0 网桥的区别在于：

- 自定义网桥提供容器间的自定义 DNS 解析。默认网桥网络下的 Docker 容器只能通过 IP 地址交互，除非使用 `--link` 参数将多个 Docker 容器连接起来。
- 自定义网桥具有更好的隔离性。默认创建的 Docker 容器如果没有指定 `--network` 参数，都会连接到默认的 docker0 网桥上，这样相当于将所有不不相干的容器都置于一个同一个网络环境中，可能存在风险。自定义网桥相当于将 docker0 网桥按我们需要分隔成多个自定义网桥，毫无疑问，这样隔离性更好。
- 容器可以在运行时和自定义网桥进行绑定或者解绑。这个默认 docker0 网桥是不行的，需要停止容器。
- 每个自定义网桥可以自定义自己的配置，比如 MTU 和 iptables 规则等。但是如果使用默认 docker0 网桥，相当于共享配置。
- 通过默认网桥 Link 的 Docker 容器可以共享环境变量。所谓 Link 是指 `docker run` 的时候指定 `--link` 参数。这个在自定义网桥中是不行的，但是可以通过其他方式来实现，比如：
    - 将需要共享的数据放到 volume 中，多个 Docker 容器自行 mount。
    - 使用 `docker-compose` 启动多个 Docker 容器，将共享变量定义到 compose 文件中。

3. Host 模式

Host 模式可以通过参数 `--network host` 指定，比如我们使用 host 模式启动一个 nginx 容器。

```
[root@docker ~]# docker run --rm -d --network host --name host_nginx nginx
38a4b19971e5f503dc902ba070d4dec270f0737197e574f50eb9dff253c56129
```

Nginx 进程会使用 80 端口，那么我们看一下刚才启动 nginx 容器有没有占用宿主机的 80 端口。首先我们要获取到容器对应的宿主机上面的进程 pid，使用命令 `docker top` 命令。

```
[root@docker ~]# docker ps | grep host_nginx
38a4b19971e5        nginx               "nginx -g 'daemon of…"   3 minutes ago       Up 3 minutes                            host_nginx
[root@docker ~]# docker top 38a4b19971e5
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                28480               28460               0                   20:29               ?                   00:00:00            nginx: master process nginx -g daemon off;
101                 28506               28480               0                   20:29               ?                   00:00:00            nginx: worker process
```

上面的输出表示 nginx 的 Docker 容器启动了两个进程 nginx master 和 nginx worker，分别对应到宿主机的 28480 和 28506 号进程。然后我们通过 `netstat` 命令查看 nginx master 进程有没有占用宿主机的 80 端口。答案很明显是的。

```
[root@docker ~]# netstat -anlp | grep 28480
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      28480/nginx: master
```

Host 模式的优缺点都很明显。

- **缺点**：没有和宿主机的 network namespace 进行隔离。可能会存在端口冲突的情况，比如 nginx 镜像的 Docker 容器会使用 80 端口，那么我们就不能以 host 模式启动两个容器，不然会冲突。
- **优点**：共用同一个 network namespace 也就意味没有个多个 network namespace 之间的数据转发，性能更好。

4. none 模式

None 模式就是禁止 Docker 容器的网络，没啥可以多说的，我们还是以一个实际的例子来好了。

```
[root@docker ~]# docker run --rm -ti --network none --name none-net-busybox busybox:latest sh
/ # ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

我们首先通过 `--network none` 参数启动一个 none 模式的 busybox 容器，然后在容器中通过 ifconfig 查看发现只有一个 loopback 网卡，这也就意味这个 Docker 容器是不可访问的，也就是 none 模式的含义。

5. overlay && macvlan

Overlay 和 Macvlan 模式我们这里就不做过多介绍了，Overlay 模式网络我们后面在 Kubernetes 网络相关内容再介绍，毕竟现在的跨主机的 Docker 容器部署基本都是通过 Kubernetes 来部署的。

Macvlan 相当于是处理 VM 迁移到 Docker 容器的历史遗留问题使用的方式，大部分人应该都应用不到，这里不做过多介绍。感兴趣的同学可以参考这里。

6. 最佳实践

官方给了一个针对各个网络模式的选择使用建议：

- ***User-defined bridge network*** 适用于同一个宿主机上多个 Docker 容器进行通信。这里的 ***user-defined*** 可以理解为自定义网桥，不适用 docker0 网桥，这样可以更灵活地设置子网和 iptables。
- ***Host networks*** 适用于 Docker 容器的网络不需要和宿主机进行隔离的场景，比如对于网络性能比较敏感的场景。
- ***Overlay networks*** 适用于运行在多个宿主机上 Docker 容器之间的通信情况。
- ***Macvlan networks*** 适用于 VM 迁移的场景，这样每个 Docker 容器看起来和物理主机一样。
- ***Third-party network plugins*** 适用于将 Docker 和特定网络协议栈整合的场景。

7. 总结

本篇文章介绍了 Docker 支持的集中网络模式，并重点介绍了最常用的 bridge 和 host 模式。由于篇幅有限，macvlan 和 第三方的 network plugin 没有做介绍，感兴趣的同学可以自行查阅。

}

17 Docker 网络初探

19 容器间网络通信：link 的技术原理解析