容器网络 Calico 基本原理和模拟

朱 娟 赵 兆

（南京理工大学 电子工程与光电技术学院，江苏 南京 210094）

摘 要：Docker 是一个开源的引擎，诞生于 2013 年，一直受到广泛的关注和讨论，目前已经发展得如火如荼。随着云计算、大数据的发展，Docker 的应用规模也越来越大，容器的使用越来越多，如何解决容器的通信问题变得愈加重要。Docker 的网络方式包括主要内置的 5 种用于通信的驱动方式，但是它们都具有一定的局限性。通过实验搭建集群环境，笔者采用 Calico 的网络模式进行测试同一台宿主机上的 Docker 容器之间以及跨主机的 Docker 容器之间的通信问题，有效地解决了容器间的通信问题。

关键词：Docker；网络；通信；容器；Calico

中图分类号：TP393.09 **文献标识码：**A **文章编号：**1003-9767（2019）24-169-04

The Basic Principle and Simulation of Calico Network in Container

Zhu Juan, Zhao Zhao

(School of Electronic and Optical Engineering, Nanjing University of Science & Technology, Nangjing Jiangsu 210094, China)

Abstract: Docker is an open source engine, which was born in 2013 and has been widely concerned and discussed. Currently, Docker has usered in used in many industries that involves informatization. With the development of cloud computing and big data, the application scale of Docker is getting larger and larger, and more and more containers are used now. Therefore, how to solve the communication problems of containers becomes more and more important. Docker's network mode includes five main built-in driver modes for communication, but they all have certain limitations. Now the cluster environment is built through experiments, and the network mode of Calico is adopted to test the communication problems between Docker containers on the same host and between Docker containers across hosts. This communication method can solve the communication problem between containers effectively .

Key words: Docker; network communication; container; Calico

0 引言

Docker 是一个开源项目，目标是实现轻量级的操作系统虚拟化方案。Docker 对进程进行封装，隔离于宿主机系统和其它容器的进程，这样每个程序都能有一个独立的运行环境，每个容器都有独立的网络、文件系统等，避免依赖等问题的相互干扰[1]。Docker 支持将程序批量移植到其他平台，即使在本地构建的程序也能快速地移植到其他平台，所以 Docker 极大地简化了开发人员和系统维护人员的工作，方便在多台服务器上同时批量上线部署维护软件程序。但是随着 Docker 的数量不断扩展，Docker 容器之间的通信也变得越来越麻烦，特别是集群内跨主机的容器之间要实现相互通信，使用 Docker 传统的的网络驱动模式，就会有一定的障碍，实施起来较为复杂。基于此，本实验在创建容器的时候使用 Calico 的网络方案，能够快速地帮助容器间建立通信连接，且方法简单。

1 Docker 简介

Docker 是一个开源的容器项目，基于 go 语言开发。Docker 的工作目标是通过对应用组件在容器内进行封装、部署、运行，对容器进行二次封装，生成一个镜像，这个镜像可以打包到任何一台装有 Docker 的主机上，在新的主机上就可以快速使用这个镜像生成新的容器。Docker 程序在主机上运行时，对进程进行封装，使其隔离于宿主机系统和其他进程，类似于一个装东西的容器，而且容器内装有该程序所需使用的一系列文件系统、网络、依赖包等环境[2]。对于后期程序移植、集群扩容、增加服务器节点、迁移都比较方便。

作者简介：朱娟（1991—），女，江苏南京人，硕士研究生。研究方向：通信工程。
赵兆（1979—），男，江苏南京人，博士研究生，副教授。研究方向：麦克风阵列信号处理、智能信号处理、大气被动声探测系统设计与实现。

Docker 的部署时间也都是以秒为单位的，十分迅速。除此之外，Docker 占用的硬件资源少，在使用过程中，Docker 容器共享宿主机内核，对操作系统进行虚拟化，性能损耗少，系统利用率高。

1.1 Docker 框架介绍

Docker 的框架组成如图 1 所示。

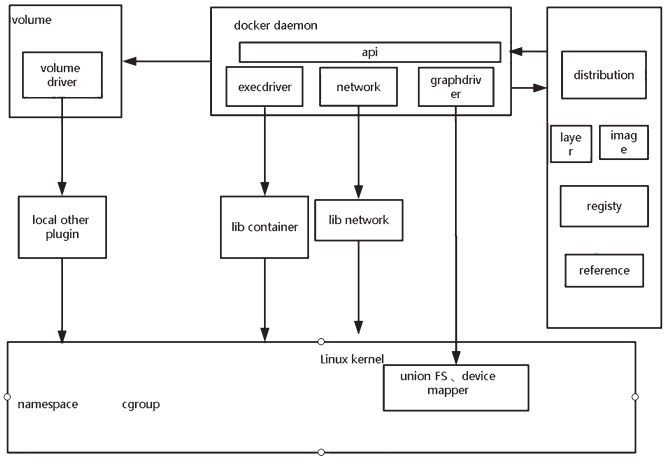

图 1 Docker 架构图

从图 1 可以看出，Docker 使用 C/S 的框架，即客户端 / 服务器的体系结构。Docker Daemon 是一个后台进程，运行在 Docker 系统里面，也是 Docker 框架的主要用户接口。该进程在后台启动一个 Api Server，用户通过 Docker Client 发送容器的管理操作命令，比如创建容器、保存镜像等，Docker Daemon 负责接收来自 Docker Client 发送的这些请求，然后将这些请求进行分发调度 [3]。通过 Driver 驱动，Docker 可以实现对 Docker 容器执行环境的创建和处理。包含管理容器镜像的 Graphdriver 驱动，配置容器内网络环境的 Network Driver 驱动，和用来创建和维护容器的 Execdriver 驱动。Docker Registry 负责保管 Docker 的镜像，Docker 从 Docker Registry 中下载镜像，这些镜像除了提供容器运行时所需的程序、库、资源、配置等文件，还包含容器运行时需要的一些配置参数、环境变量等。Docker 容器创建网络时，使用 Network 模块调用 Libnetwork, 然后创建并配置容器的网络，从而管理容器的 namespace、cgroups、apparmor、网络设备以及防火墙规则等。Docker 可以通过 Execdriver 模块限制容器的资源使用，运行容器的内部进程。Docker 容器创建数据卷时，可以通过 Volume 模块来调用具体的 Volume Driver，创建一个数据卷并对其进行挂载。

但是目前随着容器的应用率和应用场景越来越高，容器也不会简单地只运行在一台宿主机上，通常情况下，在一个集群内，各台主机上的多个容器是相互关联合作运行的，此时，就需要解决集群内跨主机的容器通信问题 [4]。

1.2 Docker 的通信模式介绍

Docker 可以为容器创建相互隔离的网络环境，在隔离的网络环境下，容器具有完全独立的网络栈，与宿主机隔离，也可以使容器共享主机或者其他容器的 namespace，基本可以满足开发者在各种场景下的需要。Docker 容器具有以下 5 种网络模式：

（1）Bridge：又称为桥接模式，Docker 创建容器之后，容器使用独立的 network namespace，并连接到 Docker0 的虚拟网卡。此时，通过 Docker0 的虚拟网桥与宿主机进行通信。

（2）Host：又称为主机模式，在创建容器时，开发者需要指定 --net=host 这个参数，才能实现这种模式。在这种模式下，容器可以直接使用宿主机的 IP 地址与外界通信，容器内部的服务端口也可以使用宿主机的端口，无需额外的 nat 转换。缺点是容器与宿主机共享 namespace，容器不再拥有独立的、隔离的网络栈。容器因此直接暴露在公网中，因此，容器的网络是不安全的。一般情况下，这种网络方式使用的比较少。

（3）None：在 None 模式下，Docker 容器拥有独立的 namespace，但是 Docker 容器在创建的时候没有进行任何网络配置。这是因为容器没有网卡、地址这些信息。在这种情况下，需要使用 –-net=none 的命令启动容器。

（4）Container：使用 Container 模式创建容器后，容器可以与指定的其他容器共享 network namespace，而不是和宿主机共享。但这两个容器只在网络、端口方面是共享的，其他的比如文件系统、进程列表方面还是隔离的，在这种情况下，需要指定 –-net=container:NAMEorID 的命令来启动容器。

用户自定义：Docker 的 1.9 版本开始引入了一整套的自定义网络命令和跨主机网络支持功能。Docker1.9 之上的版本自带了 Bridge 和 Overlay 两种类型的自定义网络 Driver，可以用于集成 Calico、weave、openvswitch 等第三方厂商的网络。和传统的网络连接模式相比，新的 Networking 可以跨越不同的物理和虚拟主机，连接不同的容器，并且用户可以轻松地停止、开启和重启容器，而不用担心破坏容器之间的相互连接。

2 Calico 通信方式简介

Docker 中的 Libnetwork 提供了以上 5 种驱动方式，虽然它们都有各自的优点，但在使用过程中都有一定的缺陷。随着应用场景复杂度的不断提高，这些驱动模式不能满足新的需求。传统的二层网络通信方式需要依赖广播消息机制，随着集群内主机的数量不断增多，广播消息的开销也日益增多，Calico 的网络方案可以较好地解决这一问题。Calico 是一个纯三层的数据中心网络方案，Calico 的路由方案可以减少一定程度的资源开销，资源利用率更高。Calico 可以为容器和虚拟机等提供一个安全的网络互联方案。Calico 基于 BGP 协议和 Linux 的路由转发机制，不依赖特殊硬件，也不使用隧道或 NAT 技术，意味着 Workloads 之间路径更短更简单，配置更少，操作维护起来也更加简单，同时能提供更好的网络性能，并且更方便地部署在物理服务器、虚拟机或者容器环境下，提

高易维护和可交互性[5]。Calico 提供动态实施的网络安全策略，可使用简单的安全模型语言实现细粒度的安全控制。Calico 组网的核心原理就是 IP 路由，为每个容器或者 VM 等分配一个 Workload-endpoint，通过修改每个主机节点上的 Iptables 和路由表规则，实现容器间数据路由和访问控制，并通过 Etcd 协调节点配置信息的[6]。因此 Calico 需要搭配 Etcd 一起使用，并且需要运行在集群的每一个节点上。

2.1 Calico 架构介绍

Calico 的系统架构如图 2 所示。

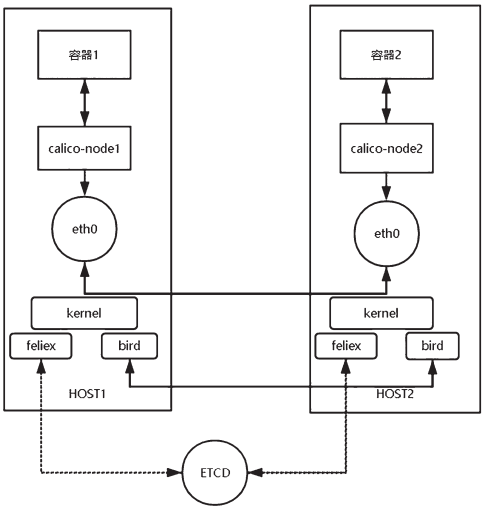

图 2 Calico 系统架构图

由图 1 可知，Calico 有以下 4 部分组成：

（1）Felix：即 Calico agent，其运行在每一台节点上，主要负责网络接口管理和路由配置以及 ACL 等。

（2）Etcd：Calico 模型的存储引擎，主要负责网络状态的准确性。

（3）BGP Client（BIRD）：Calico 为每一台 Host 部署一个 BGP Client，使用 BIRD 实现客户端管理。BIRD 是一个单独的持续发展的项目，实现了众多动态路由协议比如 BGP、OSPF、RIP 等。BGP Client 在 Calico 中的角色是监听主机上由 Felix 注入的路由信息，然后通过 BGP 协议广播告诉剩余 Host 节点，从而实现网络互通[7]。

（4）BGP Route Reflector(BIRD)：在大型网络规模中，可以通过一个或者多个 BGP Route Reflector 来完成集中式的路由分发，使所有 BGP Client 仅与特定 RR 节点互联并做路由同步，从而大大减少连接数。

2.2 基于 Calico 网络方案的容器通信方式

本次实验因为预算问题，集群内只有两台服务器，网络模型如图 3 所示。

如图 3 所示，本实验使用了两台 Centos7.5 的主机，Host1 设置为 Master,IP 地址为 172.17.0.3，Host2 设置为 Slave,IP 地址为 172.17.0.17，分别在两台主机上安装了 Docker、Etcd 和 Calico，并进行了配置。

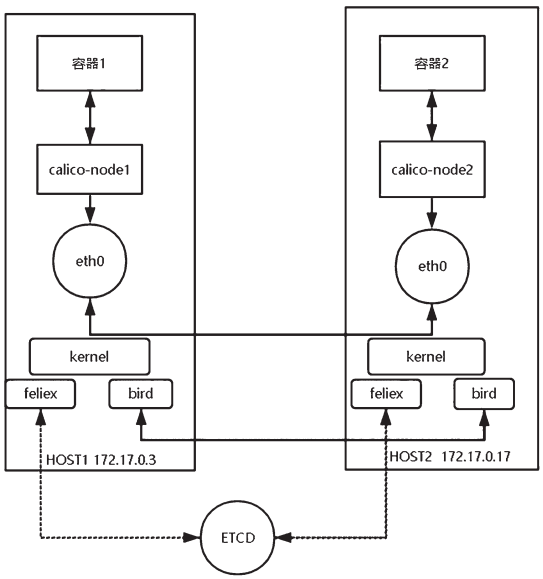

图 3 实验网络布局模型

因为容器不能跨越宿主机直接通信，本实验使用 Calico 的通信方式。当 Docker 容器创建之后，对各台主机上的 Etcd 进行配置，再下载并启动 Calico-node 容器，Calico 以容器的方式运行。所有节点安装部署完成之后，Calico 为容器生成一对 Veth pair，其中一端作为容器网卡加入容器的 namespace，帮助容器生成 IP 地址，另外一端停留在宿主机上，通过设置路由规则，将容器的 IP 暴露给宿主机的通信路由。同时，Calico 为每台 Host 预留了一段子网，作为容器可分配的 IP 范围，这样就可以根据子网的 CIDR 为每台主机生成路由规则。当集群中的容器需要与外界通信时，可以通过 BGP 协议将网关物理路由器加入集群中，使外界设备可以直接访问容器的 IP 地址。

测试结果如图 4 至图 7 所示。

```
root@eb17b79510ec:/# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.044 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 172.17.0.3: icmp_seq=5 ttl=64 time=0.046 ms
64 bytes from 172.17.0.3: icmp_seq=6 ttl=64 time=0.039 ms
64 bytes from 172.17.0.3: icmp_seq=7 ttl=64 time=0.041 ms
^Z
```

图 4 Slave 节点容器 Ping Master 主机

```
Processing triggers for libc-bin (2.28-10) ...
root@c0b0f0b8aa43:/# ping 172.17.10.64
PING 172.17.10.64 (172.17.10.64) 56(84) bytes of data.
64 bytes from 172.17.10.64: icmp_seq=1 ttl=62 time=0.633 ms
64 bytes from 172.17.10.64: icmp_seq=2 ttl=62 time=0.288 ms
64 bytes from 172.17.10.64: icmp_seq=3 ttl=62 time=0.314 ms
64 bytes from 172.17.10.64: icmp_seq=4 ttl=62 time=0.310 ms
64 bytes from 172.17.10.64: icmp_seq=5 ttl=62 time=0.305 ms
64 bytes from 172.17.10.64: icmp_seq=6 ttl=62 time=0.317 ms
```

图 5 Master 节点容器 Ping Slave 节点容器

```
[root@master ~]#  route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.17.0.1      0.0.0.0         UG    0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1002   0        0 eth0
172.17.0.0      0.0.0.0         255.255.240.0   U     0      0        0 eth0
172.17.10.0     0.0.0.0         255.255.255.255 UH    0      0        0 cali197e271f543
172.17.10.0     0.0.0.0         255.255.255.192 U     0      0        0 *
172.17.10.64    172.17.0.3      255.255.255.192 UG    0      0        0 tunl0
172.18.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.10.160.0    172.17.0.3      255.255.255.192 UG    0      0        0 tunl0
192.10.160.64   0.0.0.0         255.255.255.192 U     0      0        0 *
192.10.160.65   0.0.0.0         255.255.255.255 UH    0      0        0 cali574ab5b0d20
[root@master ~]#
```

图 6 Master 节点路由

```
[root@slave ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.17.0.1      0.0.0.0         UG    0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1002   0        0 eth0
172.17.0.0      0.0.0.0         255.255.240.0   U     0      0        0 eth0
172.17.10.0     172.17.0.17     255.255.255.192 UG    0      0        0 tunl0
172.17.10.64    0.0.0.0         255.255.255.255 UH    0      0        0 cali79b9451448c
172.17.10.64    0.0.0.0         255.255.255.192 U     0      0        0 *
172.18.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.10.160.0    0.0.0.0         255.255.255.192 U     0      0        0 *
192.10.160.1    0.0.0.0         255.255.255.255 UH    0      0        0 cali20601bb8db9
192.10.160.64   172.17.0.17     255.255.255.192 UG    0      0        0 tunl0
[root@slave ~]#
```

图 7　Slave 节点路由

由图可知，在这两台主机上创建的容器能够跨越本身的宿主机访问集群里的其他主机，而且各个节点同一网段内的容器是可以相互访问的。

3　结　语

自从问世之后，Docker 容器之间的网络通信问题一直是被关注的焦点，Docker 容器的通信方式有 Host、Bridge、None、Container，还有用户自定义的模式，跨主机的通信方式有基于 Overlay 的网络模式、基于三层 SDN 的网络模式等。本实验采用的 Calico 通信方式是基于三层 SDN 的网络模式。Calico 的网络方案为每个容器分配完全可路由的 IP 地址，工作负载可以在没有 IP 封装或网络地址转换的情况下进行通信，简化了故障排除工作，实现了裸机性能和更好的互操作性。此外，Calico 还可以与现在流行的 Kubernetes、Open stack 等结合使用，已经成为 Docker 容器通信方式的合理选择。

参考文献

[1] 肖小芳 , 宋建新 .Docker 网络通信研究与实现 [J]. 通讯世界 ,2017(22):1-2.

[2] 杨保华 , 戴王剑 , 曹亚仑 .Docker 技术入门与实战 [M]. 第 3 版 . 北京 : 机械工业出版社 ,2018:132-136.

[3](英) 奈吉尔波尔顿 . 深入浅出 Docker[M]. 李瑞丰 , 刘康 , 译 . 北京 : 人民邮电出版社 ,2017:36-42.

[4] 华为 Docker 实践小组 .Docker 进阶与实战 [M]. 北京 : 机械工业出版社 ,2016:120-130.

[5] 孙宏亮 .Docker 源码分析 [M]. 北京 : 机械工业出版社 ,2016:89-100.

[6] 李金榜 , 尹烨 , 刘天斯 , 等 . 循序渐进学 Docker[M]. 北京 : 机械工业出版社 ,2017:200-204.

[7] 余何 .PaaS 实现与运维管理 : 基于 Mesos ＋ Docker ＋ ELK 的实战指南 [M]. 北京 : 电子工业出版社 ,2015:78-82.

（上接第 168 页）

4　流量图对广西某医科大学校园网重点监测的作用

利用流量图对广西某医科大学校园网的某些网络或服务器进行监测，如对校内主页服务器点监测，生成的流量图如图 5 所示，这是一个访问流量图。在 9 ~ 11 时，可以看出这段时间访问主页服务器流量比较大，包括校外和校内的，还可以发现从 10 时 20 分到 10 时 50 分之间的流量为 0，在校园网正常的情况下，可以判断主页服务器网络不通或者服务器出现了问题。

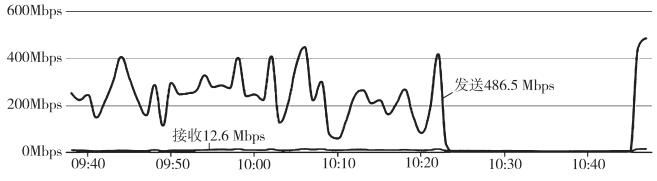

图 5　主页服务器流量异常图

5　结　语

用基于 SNMP 协议的监控系统创建流量图形，以形成连续的记录图形，可以直观地反映网络状态。通过对校园网形成的流量图分析，能够掌握校园网络的历史记录和实时状态，对于管理监控校园网络变化有着重要作用。在查找网络故障时，流量图与网络拓扑图相结合，就能判断网络故障点的时间和空间，改变了校园网络管理员繁锁的网络故障查找工作过程，提高了网络管理员的工作效率，节省了查找故障的时间，加快了解决故障过程，更好地发挥了网络管理员的校园网络管理角色，更清晰地把握了整个校园网络的运行状况，保障了校园网的安全正常运行 [5]。流量图为优化校园网络、改变网络应用策略提供了直观的参照依据。流量图在广西某医科大学校园网管理中提高了网络管理员的工作效率，为优化网络提供了依据，符合广西某医科大学校园网络管理的实际要求。

参考文献

[1] 黄志明 . 基于 SNMP 的路由器流量监控系统的设计分析 [J]. 电子世界 ,2018(10):205.

[2] 王静 . 局域网网络流量监控与管理方案 [J]. 福建电脑 ,2014,30(3):173-174.

[3] 蔡向阳 . 校园网流量管理问题及其控制措施 [J]. 宁波职业技术学院学报 ,2014,18(4):102-105.

[4] 赵韬 . 计算机网络流量监控的设计与实现 [J]. 电子技术与软件工程 ,2014(7):29.

[5] 彭晓云 , 郭正球 , 黎湖广 . 基于 Cacti 的图形化校园网络监控平台研究与实现 [J]. 信息与电脑 : 理论版 ,2015(1):14-15.