100-常用的容器网络实现原理

1. 容器网络解决的问题

容器网络需要解决的两大核心问题：

1. 容器ip地址的管理

   容器ip地址的管理包括容器IP地址的分配与回收

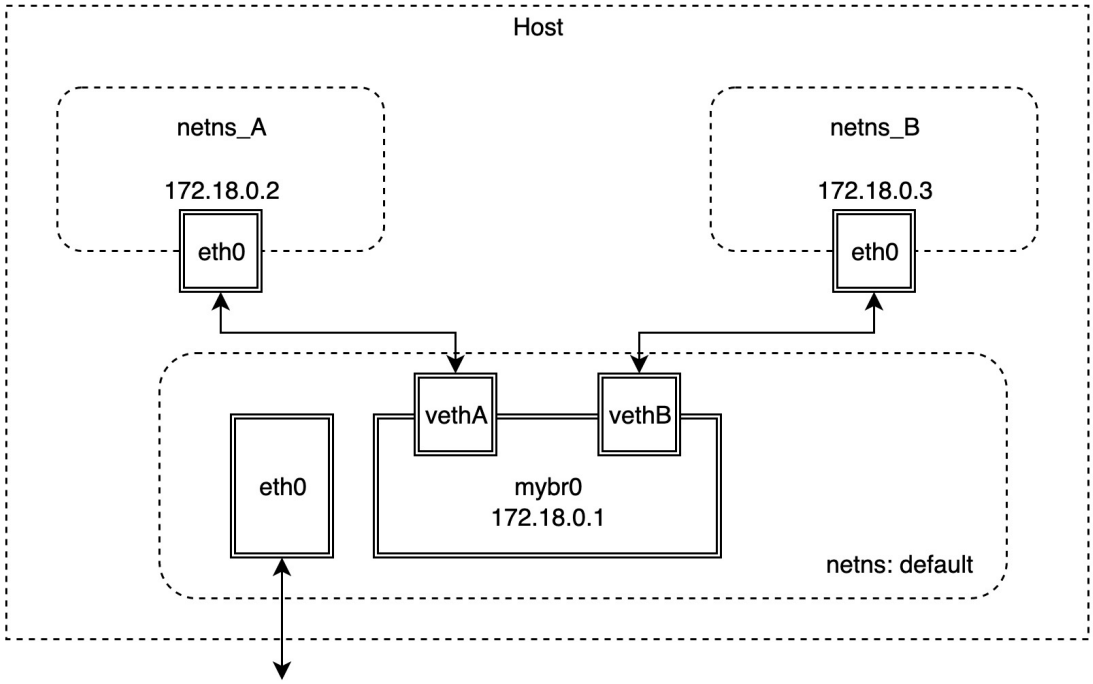
2. 容器之间的相互通信

   容器之间的通信包括：同一主机上的容器的通信和不同主机的容器的通信

2.容器经典网络模拟

2.1 同主机容器网络通信

Bridge桥接网络是docker默认的网络模型，如果我们在创建容器的时候，不指定网络模型，则默认使用bridge模型。bridge网络模型可以解决单宿主机上的容器之间的通信以及容器服务对外的暴露的问题。我们接下来就模拟一下 bridge 网络模型的实现，基本的网络拓扑图如下所示：



1. 首先创建两个netns网路命名空间

Shell | 复制代码

```
# ip netns add netns_A
# ip netns add netns_B
# ip netns
netns_B
netns_A
default
```

2. 在 default 网络命名空间中创建网桥设备 mybr0，并分配 IP 地址172.18.0.1/16使其成为对应子网的网关

Shell | 复制代码

```
# ip link add name mybr0 type bridge
# ip addr add 172.18.0.1/16 dev mybr0
# ip link show mybr0
12: mybr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state U
NKNOWN mode DEFAULT group default qlen 1000
    link/ether ae:93:35:ab:59:2a brd ff:ff:ff:ff:ff:ff
# ip route
...
172.18.0.0/16 dev mybr0 proto kernel scope link src 172.18.0.1
```

3. 创建 veth 设备

```shell
# ip link add vethA type veth peer name vethpA
# ip link show vethA
14: vethA@vethpA: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether da:f1:fd:19:6b:4a brd ff:ff:ff:ff:ff:ff
# ip link show vethpA
13: vethpA@vethA: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 86:d6:16:43:54:9e brd ff:ff:ff:ff:ff:ff
```

4. 将上一步创建的 veth 设备对的一端 vethA 连接到 mybr0 网桥并启动

```shell
# ip link set dev vethA master mybr0
# ip link set vethA up
# bridge link
14: vethA state LOWERLAYERDOWN @vethpA: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 master mybr0 state disabled priority 32 cost 2
```

5. 将 veth 设备对的另一端 vethpA 放到网络命名空间 netns_A 中并配置 IP 启动

```shell
# ip link set vethpA netns netns_A
# ip netns exec netns_A ip link set vethpA name eth0
# ip netns exec netns_A ip addr add 172.18.0.2/16 dev eth0
# ip netns exec netns_A ip link set eth0 up
# ip netns exec netns_A ip addr show type veth
13: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 86:d6:16:43:54:9e brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.2/16 scope global eth0
       valid_lft forever preferred_lft forever
```

6. 现在就可以验证从 netns_A 网络命名空间中访问 mybr0 网关

```shell
# ip netns exec netns_A ping -c 2 172.18.0.1
PING 172.18.0.1 (172.18.0.1) 56(84) bytes of data.
64 bytes from 172.18.0.1: icmp_seq=1 ttl=64 time=0.096 ms
64 bytes from 172.18.0.1: icmp_seq=2 ttl=64 time=0.069 ms

--- 172.18.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.069/0.082/0.096/0.016 m
```

7. 若想要从 netns_A 网络命名空间中访问非172.18.0.0/16的地址，就需要增加一条默认默认路由

```shell
# ip netns exec netns_A ip route add default via 172.18.0.1
# ip netns exec netns_A ip route
default via 172.18.0.1 dev eth0
172.18.0.0/16 dev eth0 proto kernel scope link src 172.18.0.2
```

> Note: 如果你此时尝试去 ping 其他的公网地址，例如 google.com，是 ping 不通的，原因是 ping 出去的数据包（ICMP 包）的源地址没有做源地址转换(snat)，导致 ICMP 包有去无回；docker是通过设置 iptables 实现源地址转换的。

8. 接下来，按照上述步骤创建连接 default 和 netns_B 网络命名空间 veth 设备对

```shell
# ip link add vethB type veth peer name vethpB
# ip link set dev vethB master mybr0
# ip link set vethB up
# ip link set vethpB netns netns_B
# ip netns exec netns_B ip link set vethpB name eth0
# ip netns exec netns_B ip addr add 172.18.0.3/16 dev eth0
# ip netns exec netns_B ip link set eth0 up
# ip netns exec netns_B ip route add default via 172.18.0.1
# ip netns exec netns_B ip add show eth0
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue st
ate UP group default qlen 1000
    link/ether 0e:2f:c6:de:fe:24 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.3/16 scope global eth0
       valid_lft forever preferred_lft forever
# ip netns exec netns_B ip route show
default via 172.18.0.1 dev eth0
172.18.0.0/16 dev eth0 proto kernel scope link src 172.18.0.3
```

9. 默认情况下 Linux 会把网桥设备 bridge 的转发功能禁用，所以在 netns_A 里面是 ping 不通 netns_B 的，需要额外增加一条 iptables 规则才能激活网桥设备 bridge 的转发功能

```shell
# iptables -A FORWARD -i mybr0 -j ACCEPT
```

10. 现在就可以验证两个网络命名空间之间可以互通

```shell
# ip netns exec netns_A ping -c 2 172.18.0.3
PING 172.18.0.3 (172.18.0.3) 56(84) bytes of data.
64 bytes from 172.18.0.3: icmp_seq=1 ttl=64 time=0.091 ms
64 bytes from 172.18.0.3: icmp_seq=2 ttl=64 time=0.093 ms

--- 172.18.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1027ms
rtt min/avg/max/mdev = 0.091/0.092/0.093/0.001 ms
# ip netns exec netns_B ping -c 2 172.18.0.2
PING 172.18.0.2 (172.18.0.2) 56(84) bytes of data.
64 bytes from 172.18.0.2: icmp_seq=1 ttl=64 time=0.259 ms
64 bytes from 172.18.0.2: icmp_seq=2 ttl=64 time=0.078 ms

--- 172.18.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1030ms
rtt min/avg/max/mdev = 0.078/0.168/0.259/0.091 ms
```

实际上，此时两个网络命名空间处于同一个子网中，所以网桥设备mybr0还是工作在二层（数据链路层），只需要对方的MAC地址就可以访问。

但是如果需要从两个网络命名空间访问其他网段的地址，这个时候网桥设备mybr0设置为默认网关地址就发挥作用了，来自两个网络命名空间的数据包发现目标ip地址并不是本子网地址，于是发送给网关mybr0，此时网桥设备mybr0其实工作在三层（IP网络层），它收到数据包之后，查看本地路由与目标IP地址，寻找吓一跳的地址。

2.2 不同主机容器网络通信

对于跨主机容器间的相互访问问题，我们能想到的最直观的解决方案就是直接使用宿主机网络，这时，容器完全复用复用宿主机的网络设备以及协议栈，容器 IP 就是主机 IP，这样，只要宿主机主机能通信，容器也就自然能通信。但是这样，为了暴露容器服务，每个容器需要占用宿主机上的一个端口，通过这个端口和外界通信。所以，就需要手动维护端口的分配，而且不能使不同的容器服务运行在一个端口上，正因为如此，这种容器网络模型很难被推广到生产环境。

因此解决跨主机通信的可行方案主要是让容器配置与宿主机不一样的 IP 地址，往往是在现有二层或三层网络之上再构建起来一个独立的 overlay 网络，这个网络通常会有自己独立的 IP 地址空间、交换或者路由的实现。由于容器有自己独立配置的 IP 地址，underlay 平面的底层网络设备如交换机、路由器等完全不感知这些 IP 的存在，也就导致容器的 IP 不能直接路由出去实现跨主机通信。

为了解决容器独立 IP 地址间的访问问题，主要有以下两个思路：

1. 修改底层网络设备配置，加入容器网络 IP 地址的管理，修改路由器网关等，该方式主要和 SDN(Software define networking) 结合。
2. 完全不修改底层网络设备配置，复用原有的 underlay 平面网络解决容器跨主机通信，主要有如下两种方式：

- **隧道传输(overlay)**：**将容器的数据包封装到宿主机网络**的三层或者四层数据包中，然后**使用宿主机的 IP 或者 TCP/UDP 传输到目标主机，目标主机拆包后再转发给目标容器**。overlay 隧道传输常见方案包括 VxLAN、ipip 等，目前使用 Overlay 隧道传输技术的主流容器网络有 Flannel 等。
- **修改主机路由**：把容器网络加到宿主机网络的路由表中，把宿主机网络设备当作容器网关，通过路由规则转发到指定的主机，实现容器的三层互通。目前通过路由技术实现容器跨主机通信的网络如 Flannel host-gw、Calico 等。

3.容器网络通信技术术语

- IPAM(IP Address Management): 即 IP 地址管理。IPAM 并不是容器时代特有的词汇，传统的标准网络协议比如 DHCP 其实也是一种 IPAM，负责从 MAC 地址分发 IP 地址；但是到了容器时代我们提到 IPAM，我们特指为每一个容器实例分配和回收 IP 地址，保证一个集群里面的所有容器都分配全局唯一的 IP 地址；主流的做法包括：基于 CIDR 的 IP 地址段分配地或精确为每一个容器分配 IP；
- overlay：在容器时代，就是在主机现有二层（数据链路层）或三层（IP网络层）基础之上在构建起来一个独立的网络平面，这个overlay网络通常会有自己独立的IP地址空间，交换或者路由的实现。
- IPIP：一种基于 Linux 网络设备 TUN 实现的隧道协议，允许将三层（IP）网络包封装在另外一个三层网络包之上发送和接收，详情请看揭秘 IPIP 隧道；
- IPSec：跟 IPIP 隧道协议类似，是一种点对点的加密通信协议，一般会用到 overlay 网络的数据隧道里；
- VxLAN：最主要是解决 VLAN 支持虚拟网络数量（4096）过少的问题而由 VMware、Cisco、RedHat 等联合提出的解决方案；VxLAN 可以支持在一个 VPC(Virtual Private Cloud) 划分多达 1600万个虚拟网络；
- BGP：主干网自治网络的路由协议，当代的互联网由很多小的 AS(Autonomous system)自治网络构成，自治网络之间的三层路由是由 BGP 协议实现的，简单来说，通过 BGP 协议 AS 告诉其他 AS

自己子网里都包括哪些 IP 地址段，自己的 AS 编号以及一些其他的信息；

- SDN(Software-Defined Networking)：一种广义的概念，通过软件方式快速配置网络，往往包括一个中央控制层来集中配置底层基础网络设施；