# **linux 网络虚拟化： macvlan**

## **macvlan 简介**

macvlan 是 linux kernel 比较新的特性，可以通过以下方法判断当前系统是否支持：

$ modprobe macvlan

$ lsmod | grep macvlan

macvlan 19046 0

如果第一个命令报错，或者第二个命令没有返回，则说明当前系统不支持 macvlan，需要升级系统或者升级内核。

macvlan 允许你在主机的一个网络接口上配置多个虚拟的网络接口，这些网络 interface 有自己独立的 mac 地址，也可以配置上 ip 地址进行通信。macvlan 下的虚拟机或者容器网络和主机在同一个网段中，共享同一个广播域。macvlan 和 bridge 比较相似，但因为它省去了 bridge 的存在，所以配置和调试起来比较简单，而且效率也相对高。除此之外，macvlan 自身也完美支持 VLAN。

如果希望容器或者虚拟机放在主机相同的网络中，享受已经存在网络栈的各种优势，可以考虑 macvlan。

## **各个 linux 发行版对 macvlan 的支持**

macvlan 对kernel 版本依赖：Linux kernel v3.9–3.19 and 4.0+。几个重要发行版支持情况：

* ubuntu：>= saucy(13.10)
* RHEL(Red Hat Enterprise Linux): >= 7.0(3.10.0)
* Fedora: >=19(3.9)
* Debian: >=8(3.16)

各个发行版的 kernel 都可以自行手动升级，具体操作可以参考官方提供的文档。

以上版本信息参考了这些资料：

* [List of ubuntu versions with corresponding linux kernel version](https://askubuntu.com/questions/517136/list-of-ubuntu-versions-with-corresponding-linux-kernel-version" \t "https://cizixs.com/2017/02/14/network-virtualization-macvlan/_blank)
* [Red Hat Enterprise Linux Release Dates](https://access.redhat.com/articles/3078" \t "https://cizixs.com/2017/02/14/network-virtualization-macvlan/_blank)

## **四种模式**

* private mode：过滤掉所有来自其他 macvlan 接口的报文，因此不同 macvlan 接口之间无法互相通信
* vepa(Virtual Ethernet Port Aggregator) mode： 需要主接口连接的交换机支持 VEPA/802.1Qbg 特性。所有发送出去的报文都会经过交换机，交换机作为再发送到对应的目标地址（即使目标地址就是主机上的其他 macvlan 接口），也就是 hairpin mode 模式，这个模式用在交互机上需要做过滤、统计等功能的场景。
* bridge mode：通过虚拟的交换机讲主接口的所有 macvlan 接口连接在一起，这样的话，不同 macvlan 接口之间能够直接通信，不需要将报文发送到主机之外。这个模式下，主机外是看不到主机上 macvlan interface 之间通信的报文的。
* passthru mode：暂时没有搞清楚这个模式要解决的问题

VEPA 和 passthru 模式下，两个 macvlan 接口之间的通信会经过主接口两次：第一次是发出的时候，第二次是返回的时候。这样会影响物理接口的宽带，也限制了不同 macvlan 接口之间通信的速度。如果多个 macvlan 接口之间通信比较频繁，对于性能的影响会比较明显。

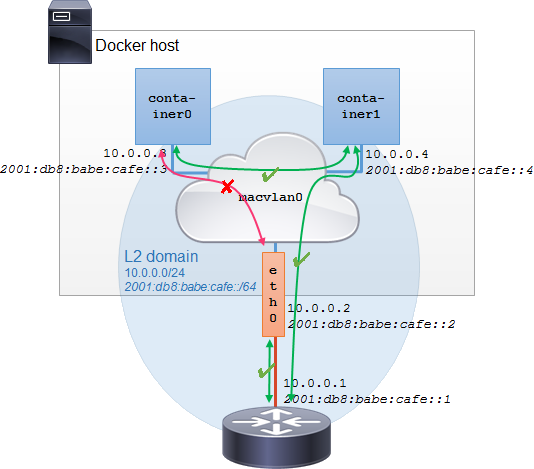
private 模式下，所有的 macvlan 接口都不能互相通信，对性能影响最小。

bridge 模式下，数据报文是通过内存直接转发的，因此效率会高一些，但是会造成 CPU 额外的计算量。

****NOTE****：如果要手动分配 mac 地址，请注意本地的 mac 地址最高位字节的低位第二个 bit 必须是 1。比如 02:xx:xx:xx:xx:xx。

## **实验**

为了方便演示，我们会创建出来两个 macvlan 接口，分别放到不同的 [network namespace](https://cizixs.com/2017/02/10/network-virtualization-network-namespace) 里。整个实验的网络拓扑结构如下：



[图片来源](http://hicu.be/docker-networking-macvlan-bridge-mode-configuration" \t "https://cizixs.com/2017/02/14/network-virtualization-macvlan/_blank)

首先还是创建两个 network namespace：

[root@localhost ~]*# ip netns add net1*[root@localhost ~]*# ip netns add net2*

然后创建 macvlan 接口:

[root@localhost ~]*# ip link add link enp0s8 mac1 type macvlan*[root@localhost ~]*# ip link*

23: mac1@enp0s8: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT

link/ether e2:80:1c:ba:59:9c brd ff:ff:ff:ff:ff:ff

创建的格式为 ip link add link <PARENT> <NAME> type macvlan，其中 <PARENT> 是 macvlan 接口的父 interface 名称，<NAME> 是新建的 macvlan 接口的名称，这个名字可以任意取。使用 ip link 可以看到我们刚创建的 macvlan 接口，除了它自己的名字之外，后面还跟着父接口的名字。

下面就是把创建的 macvlan 放到 network namespace 中，配置好 ip 地址，然后启用它：

[root@localhost ~]*# ip link set mac1@enp0s8 netns net1*

Cannot find device "mac1@enp0s8"[root@localhost ~]*# ip link set mac1 netns net1*[root@localhost ~]*# ip netns exec net1 ip link*

23: mac1@if3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT

link/ether e2:80:1c:ba:59:9c brd ff:ff:ff:ff:ff:ff[root@localhost ~]*# ip netns exec net1 ip link set mac1 name eth0*[root@localhost ~]*# ip netns exec net1 ip link*

23: eth0@if3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT

link/ether e2:80:1c:ba:59:9c brd ff:ff:ff:ff:ff:ff[root@localhost ~]*# ip netns exec net1 ip addr add 192.168.8.120/24 dev eth0*[root@localhost ~]*# ip netns exec net1 ip link set eth0 up*

同理可以配置另外一个 macvlan 接口，可以测试两个 ip 的连通性：

[root@localhost ~]*# docker exec 1444 ping -c 3 192.168.8.120*

PING 192.168.8.120 (192.168.8.120): 56 data bytes

64 bytes from 192.168.8.120: seq=0 ttl=64 time=0.130 ms

64 bytes from 192.168.8.120: seq=1 ttl=64 time=0.099 ms

64 bytes from 192.168.8.120: seq=2 ttl=64 time=0.083 ms

--- 192.168.8.120 ping statistics ---

3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max = 0.083/0.104/0.130 ms

## **在 docker 中的使用**

docker1.12 版本也正式支持了 macvlan 和 ipvlan 网络模式。

### **创建 macvlan 网络**

管理 macvlan 和其他类型的网络类似，通过 docker network 子命令就能实现。创建 macvlan 网络的时候，需要知道主机的网段和网关地址，虚拟网络要依附的物理网卡。

[root@localhost ~]*# docker network create -d macvlan --subnet=192.168.8.0/24 --gateway=192.168.8.1 -o parent=enp0s8 mcv*

9fad35e54a2f53c9314626f89cf8a705799ed382ddac01c865be1f4d04fdcb8f[root@localhost ~]*# docker network ls*

NETWORK ID NAME DRIVER SCOPE

e06b6e00dd3b bridge bridge local

823b7bb07c41 host host local

9fad35e54a2f mcv macvlan local

dc7c667aca19 none null local

选项说明：

* subnet：网络 CIDR 地址
* gateway：网关地址
* aux-address：不要分配给容器的 ip 地址。字典，以 key=value 对出现
* ip-range：指定具体的 ip 分配区间，也是 CIDR 格式，必须是 subnet 指定范围的子集
* opt(o)：和 macvlan driver 相关的选项，以 key=value 的格式出现
  + parent=eth0: 指定 parent interface
  + macvlan\_mode：macvlan 模式，默认是 bridge

### **运行容器**

创建好网络之后，我们就可以使用刚创建的网络运行两个容器，测试网络的连通性。

[root@localhost ~]*# docker run --net=mac192 -d --rm alpine top*

5e950cf86cda7b4e6fc4bd869834943edacaaf969051293021c75330bbc18b53[root@localhost ~]*# docker run --net=mac192 -d --rm alpine top*

9067c54aac79e65b3193a137e95a180a7e5cc4a2845cc664f53c17a244be3853[root@localhost ~]*# docker ps*

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

9067c54aac79 alpine "top" 7 seconds ago Up 6 seconds sharp\_hodgkin

5e950cf86cda alpine "top" 8 seconds ago Up 7 seconds peaceful\_chandrasekhar

[root@localhost ~]*# docker exec 906 ip addr*

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

16: eth0@if3: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue state UNKNOWN

link/ether 02:42:c0:a8:08:03 brd ff:ff:ff:ff:ff:ff

inet 192.168.8.3/24 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::42:c0ff:fea8:803/64 scope link

valid\_lft forever preferred\_lft forever[root@localhost ~]*# docker exec 5e9 ping -c 3 192.168.8.3*

PING 192.168.8.3 (192.168.8.3): 56 data bytes

64 bytes from 192.168.8.3: seq=0 ttl=64 time=0.226 ms

64 bytes from 192.168.8.3: seq=1 ttl=64 time=0.076 ms

64 bytes from 192.168.8.3: seq=2 ttl=64 time=0.095 ms

--- 192.168.8.3 ping statistics ---

3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max = 0.076/0.132/0.226 ms

需要注意的是，从容器中是无法访问所在主机地址的：

[root@localhost ~]*# docker exec 5e9 ping -c 3 192.168.8.110*

PING 192.168.8.110 (192.168.8.110): 56 data bytes

--- 192.168.8.110 ping statistics ---

3 packets transmitted, 0 packets received, 100% packet loss

这是 macvlan 的特性，目的是为了更好地实现网络的隔离，和 docker 无关。