第31讲 | 容器网络之Calico：为高效说出善意的谎言

2018-07-27 刘超

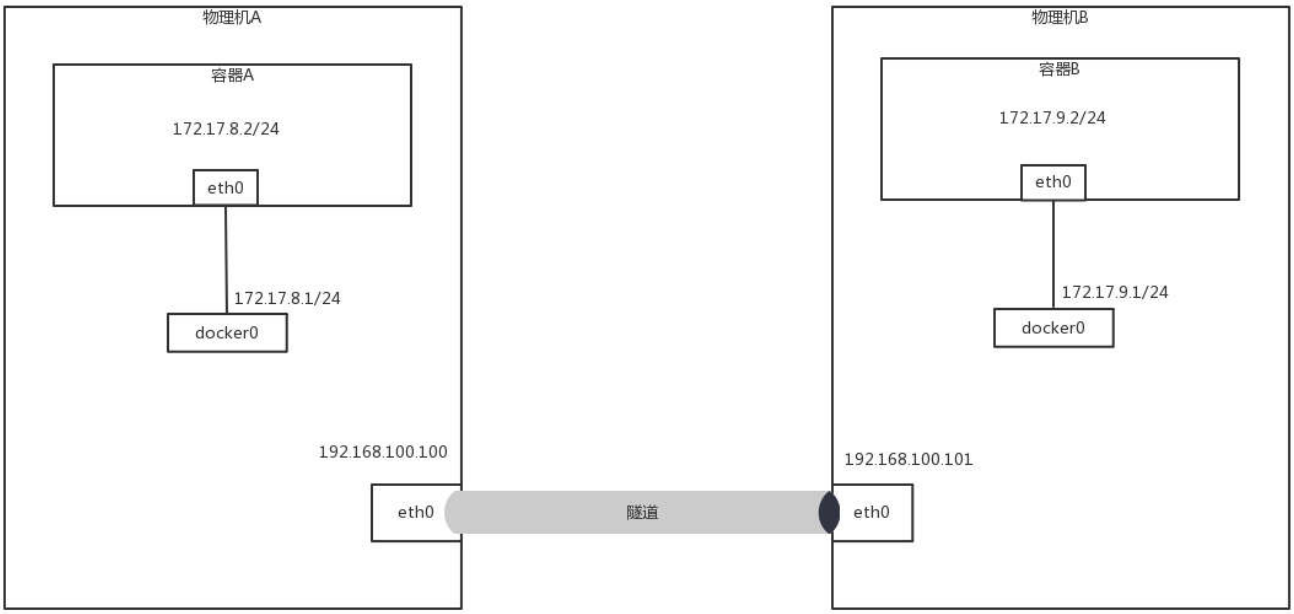
第31讲 | 容器网络之Calico：为高效说出善意的谎言
朗读人：刘超　17′42″ | 8.12M

上一节我们讲了 Flannel 如何解决容器跨主机互通的问题，这个解决方式其实和虚拟机的网络互通模式是差不多的，都是通过隧道。但是 Flannel 有一个非常好的模式，就是给不同的物理机设置不同网段，这一点和虚拟机的 Overlay 的模式完全不一样。

在虚拟机的场景下，整个网段在所有的物理机之间都是可以“飘来飘去”的。网段不同，就给了我们做路由策略的可能。

Calico 网络模型的设计思路

我们看图中的两台物理机。它们的物理网卡是同一个二层网络里面的。由于两台物理机的容器网段不同，我们完全可以将两台物理机配置成为路由器，并按照容器的网段配置路由表。



例如，在物理机 A 中，我们可以这样配置：要想访问网段 172.17.9.0/24，下一跳是 192.168.100.101，也即到物理机 B 上去。

这样在容器 A 中访问容器 B，当包到达物理机 A 的时候，就能够匹配到这条路由规则，并将包发给下一跳的路由器，也即发给物理机 B。在物理机 B 上也有路由规则，要访问 172.17.9.0/24，从 docker0 的网卡进去即可。

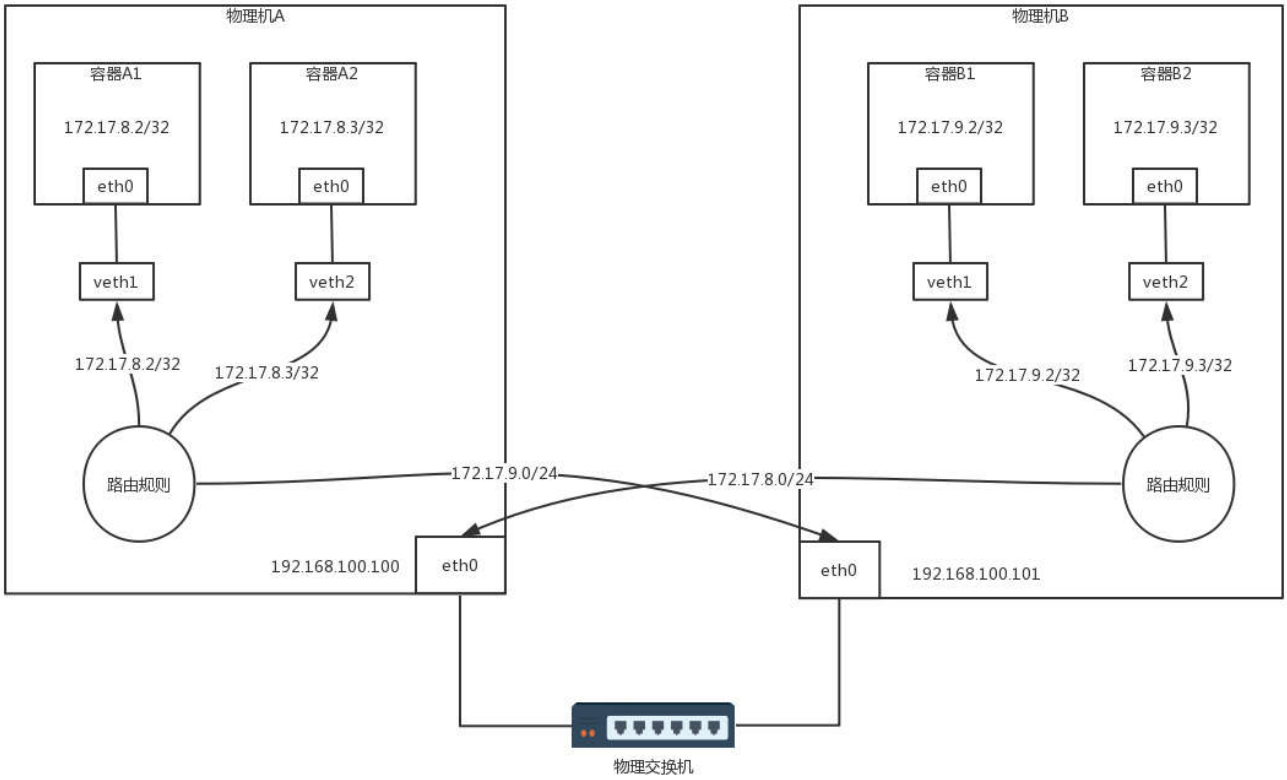
当容器 B 返回结果的时候，在物理机 B 上，可以做类似的配置：要想访问网段 172.17.8.0/24，下一跳是 192.168.100.100，也即到物理机 A 上去。

当包到达物理机 B 的时候，能够匹配到这条路由规则，将包发给下一跳的路由器，也即发给物理机 A。在物理机 A 上也有路由规则，要访问 172.17.8.0/24，从 docker0 的网卡进去即可。

这就是Calico 网络的大概思路，即不走 Overlay 网络，不引入另外的网络性能损耗，而是将转发全部用三层网络的路由转发来实现，只不过具体的实现和上面的过程稍有区别。

首先，如果全部走三层的路由规则，没必要每台机器都用一个 docker0，从而浪费了一个 IP 地址，而是可以直接用路由转发到 veth pair 在物理机这一端的网卡。同样，在容器内，路由规则也可以这样设定：把容器外面的 veth pair 网卡算作默认网关，下一跳就是外面的物理机。

于是，整个拓扑结构就变成了这个图中的样子。



Calico 网络的转发细节

我们来看其中的一些细节。

容器 A1 的 IP 地址为 172.17.8.2/32，这里注意，不是 /24，而是 /32，将容器 A1 作为一个单点的局域网了。

容器 A1 里面的默认路由，Calico 配置得比较有技巧。

```
default via 169.254.1.1 dev eth0
169.254.1.1 dev eth0 scope link
```

这个 IP 地址 169.254.1.1 是默认的网关，但是整个拓扑图中没有一张网卡是这个地址。那如何到达这个地址呢?

前面我们讲网关的原理的时候说过，当一台机器要访问网关的时候，首先会通过 ARP 获得网关的 MAC 地址，然后将目标 MAC 变为网关的 MAC，而网关的 IP 地址不会在任何网络包头里面出现，也就是说，没有人在乎这个地址具体是什么，只要能找到对应的 MAC，响应 ARP 就可以了。

ARP 本地有缓存，通过 ip neigh 命令可以查看。

```
169.254.1.1 dev eth0 lladdr ee:ee:ee:ee:ee:ee STALE
```

这个 MAC 地址是 Calico 硬塞进去的，但是没有关系，它能响应 ARP，于是发出的包的目标 MAC 就是这个 MAC 地址。

在物理机 A 上查看所有网卡的 MAC 地址的时候，我们会发现 veth1 就是这个 MAC 地址。所以容器 A1 里发出的网络包，第一跳就是这个 veth1 这个网卡，也就到达了物理机 A 这个路由器。

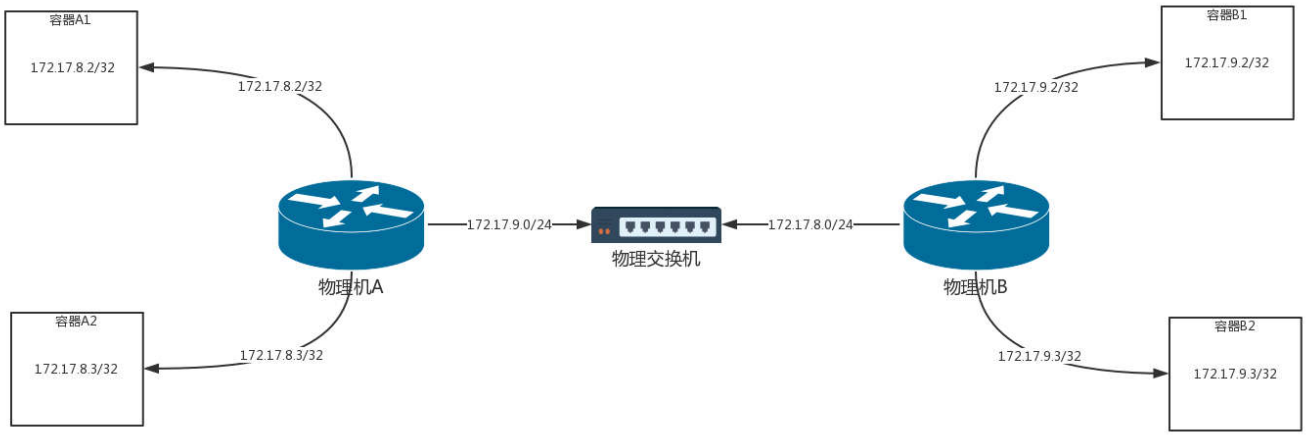
在物理机 A 上有三条路由规则，分别是去两个本机的容器的路由，以及去 172.17.9.0/24，下一跳为物理机 B。

```
172.17.8.2 dev veth1 scope link
172.17.8.3 dev veth2 scope link
172.17.9.0/24 via 192.168.100.101 dev eth0 proto bird onlink
```

同理，物理机 B 上也有三条路由规则，分别是去两个本机的容器的路由，以及去 172.17.8.0/24，下一跳为物理机 A。

```
172.17.9.2 dev veth1 scope link
172.17.9.3 dev veth2 scope link
172.17.8.0/24 via 192.168.100.100 dev eth0 proto bird onlink
```

如果你觉得这些规则过于复杂，我将刚才的拓扑图转换为这个更加容易理解的图。

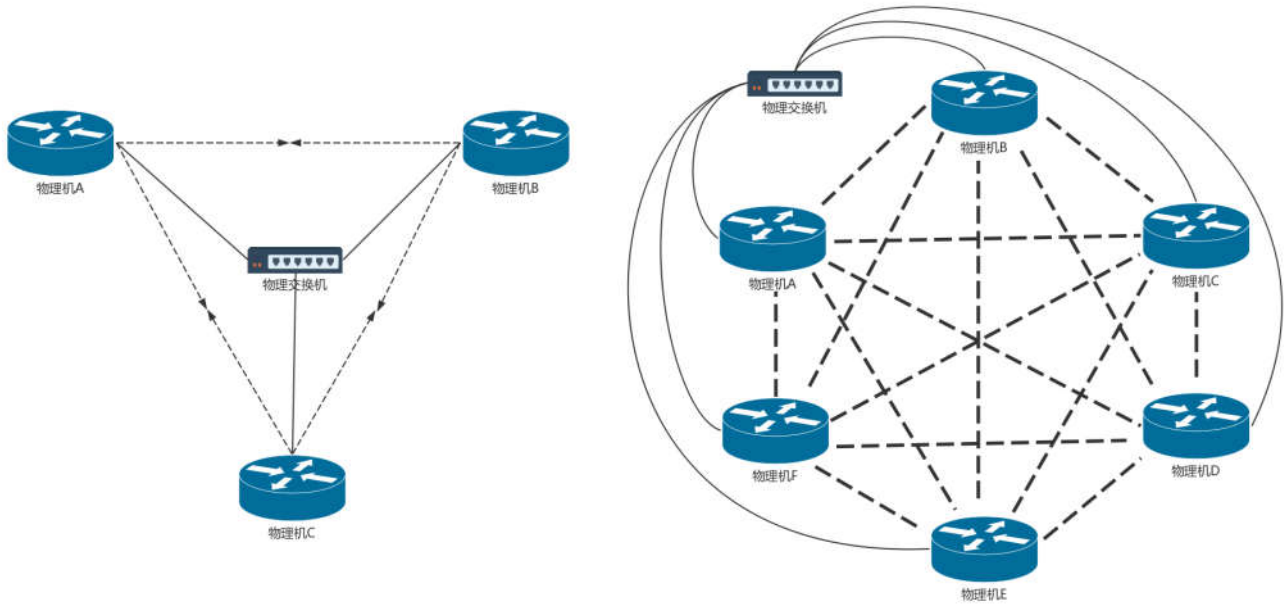


在这里，物理机化身为路由器，通过路由器上的路由规则，将包转发到目的地。在这个过程中，没有隧道封装解封装，仅仅是单纯的路由转发，性能会好很多。但是，这种模式也有很多问题。

Calico 的架构

路由配置组件 Felix

如果只有两台机器，每台机器只有两个容器，而且保持不变。我手动配置一下，倒也没啥问题。但是如果容器不断地创建、删除，节点不断地加入、退出，情况就会变得非常复杂。



就像图中，有三台物理机，两两之间都需要配置路由，每台物理机上对外的路由就有两条。如果有六台物理机，则每台物理机上对外的路由就有五条。新加入一个节点，需要通知每一台物理机添加一条路由。

这还是在物理机之间，一台物理机上，每创建一个容器，也需要多配置一条指向这个容器的路由。如此复杂，肯定不能手动配置，需要每台物理机上有一个 agent，当创建和删除容器的时候，自动做这件事情。这个 agent 在 Calico 中称为 Felix。

路由广播组件 BGP Speaker

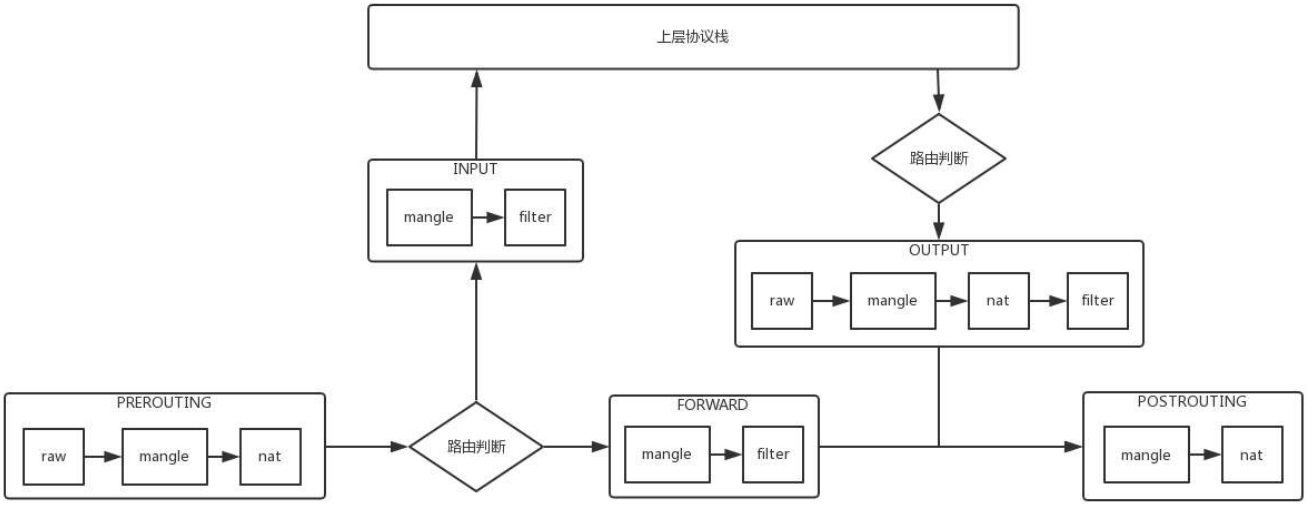
当 Felix 配置了路由之后，接下来的问题就是，如何将路由信息，也即将“如何到达我这个节点，访问我这个节点上的容器”这些信息，广播出去。

能想起来吗？这其实就是路由协议啊！路由协议就是将“我能到哪里，如何能到我”的信息广播给全网传出去，从而客户端可以一跳一跳地访问目标地址的。路由协议有很多种，Calico 使用的是 BGP 协议。

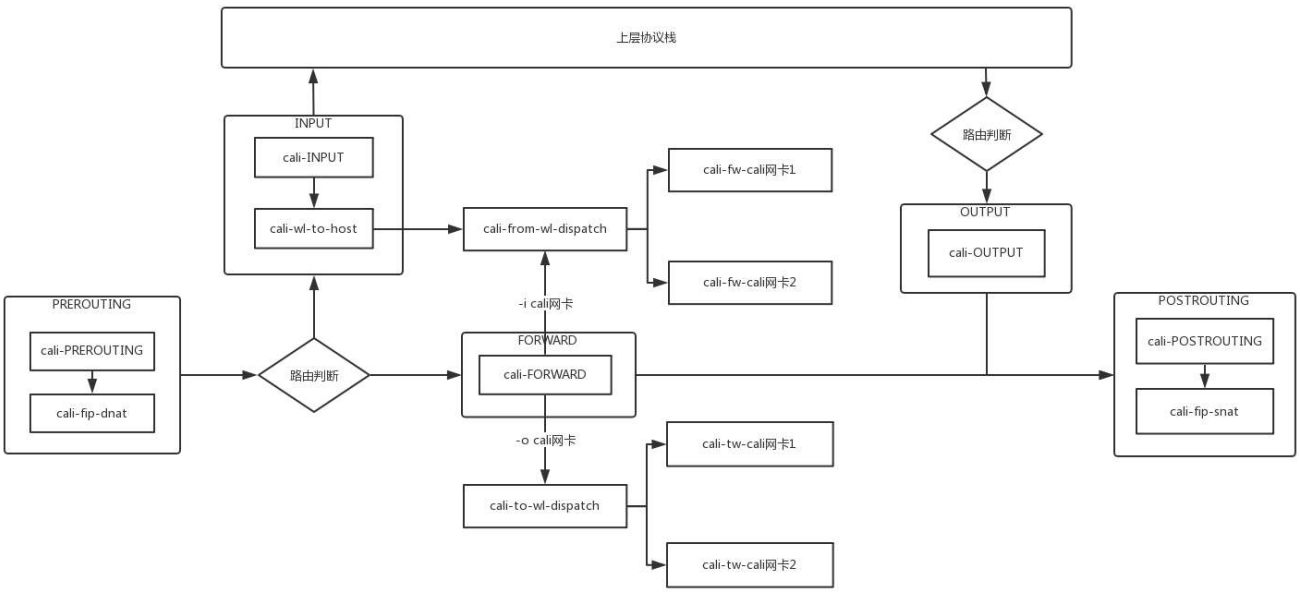
在 Calico 中，每个 Node 上运行一个软件 BIRD，作为 BGP 的客户端，或者叫作 BGP Speaker，将“如何到达我这个 Node，访问我这个 Node 上的容器”的路由信息广播出去。所有 Node 上的 BGP Speaker 都互相建立连接，就形成了全互连的情况，这样每当路由有所变化的时候，所有节点就都能够收到了。

安全策略组件

Calico 中还实现了灵活配置网络策略 Network Policy，可以灵活配置两个容器通或者不通。这个怎么实现呢？



虚拟机中的安全组，是用 iptables 实现的。Calico 中也是用 iptables 实现的。这个图里的内容是 iptables 在内核处理网络包的过程中可以嵌入的处理点。Calico 也是在这些点上设置相应的规则。



当网络包进入物理机上的时候，进入 PREOUTING 规则，这里面有一个规则是 cali-fip-dnat，这是实现浮动 IP（Floating IP）的场景，主要将外网的 IP 地址 dnat 为容器内的 IP 地址。在虚拟机场景下，路由器的网络 namespace 里面有一个外网网卡上，也设置过这样一个 DNAT 规则。

接下来可以根据路由判断，是到本地的，还是要转发出去的。

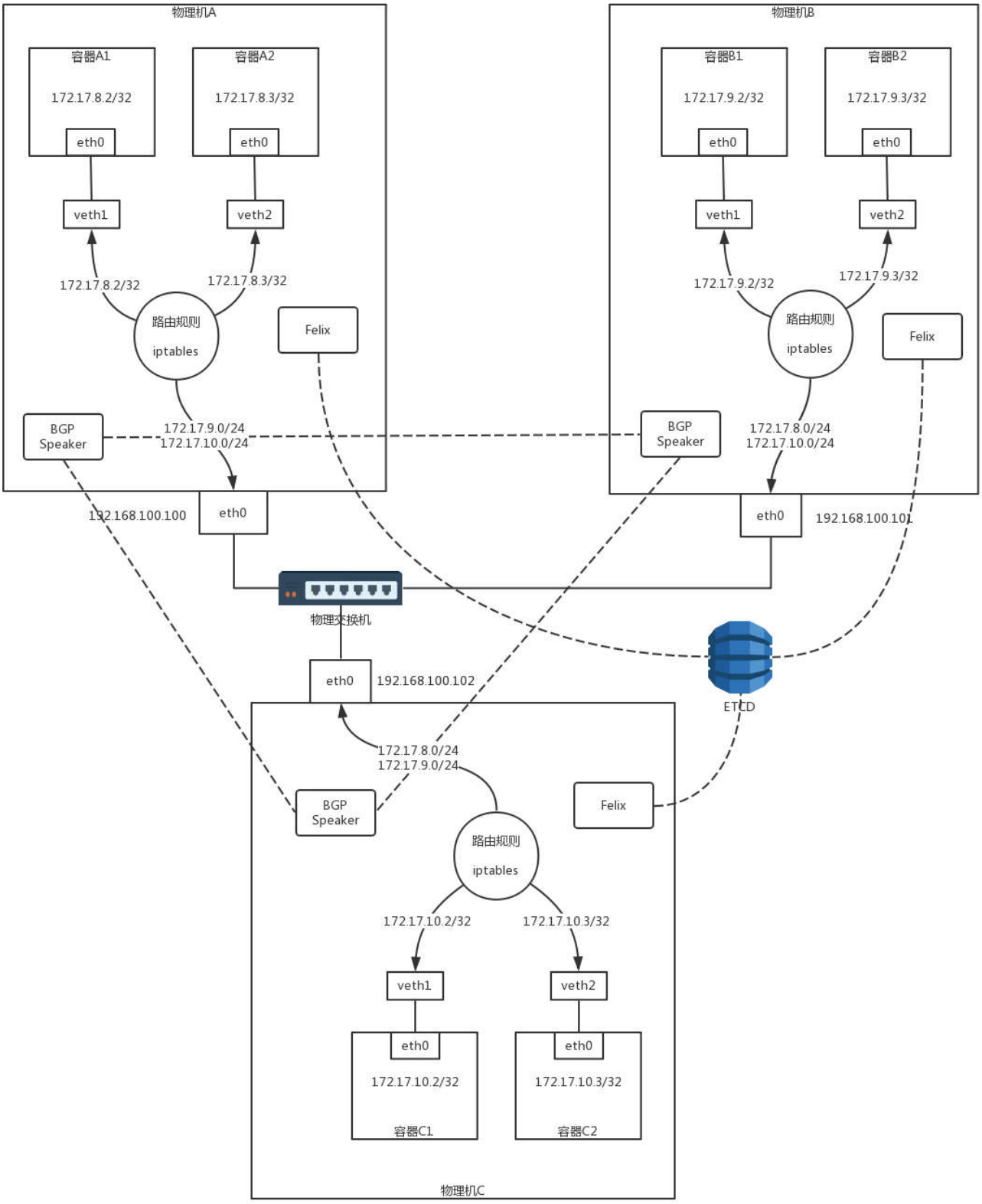
如果是本地的，走 INPUT 规则，里面有个规则是 cali-wl-to-host，wl 的意思是 workload，也即容器，也即这是用来判断从容器发到物理机的网络包是否符合规则的。这里面内嵌一个规则 cali-from-wl-dispatch，也是匹配从容器来的包。如果有两个容器，则会有两个容器网卡，这里面内嵌有详细的规则“cali-fw-cali 网卡 1”和“cali-fw-cali 网卡 2”，fw 就是 from workload，也就是匹配从容器 1 来的网络包和从容器 2 来的网络包。

如果是转发出去的，走 FORWARD 规则，里面有个规则 cali-FORWARD。这里面分两种情况，一种是从容器里面发出来，转发到外面的；另一种是从外面发进来，转发到容器里面的。

第一种情况匹配的规则仍然是 cali-from-wl-dispatch，也即 from workload。第二种情况匹配的规则是 cali-to-wl-dispatch，也即 to workload。如果有两个容器，则会有两个容器网卡，在这里面内嵌有详细的规则“cali-tw-cali 网卡 1”和“cali-tw-cali 网卡 2”，tw 就是 to workload，也就是匹配发往容器 1 的网络包和发送到容器 2 的网络包。

接下来是匹配 OUTPUT 规则，里面有 cali-OUTPUT。接下来是 POSTROUTING 规则，里面有一个规则是 cali-fip-snat，也即发出去的时候，将容器网络 IP 转换为浮动 IP 地址。在虚拟机场景下，路由器的网络 namespace 里面有一个外网网卡上，也设置过这样一个 SNAT 规则。

至此为止，Calico 的所有组件基本凑齐。来看看我汇总的图。



全连接复杂性与规模问题

这里面还存在问题，就是 BGP 全连接的复杂性问题。

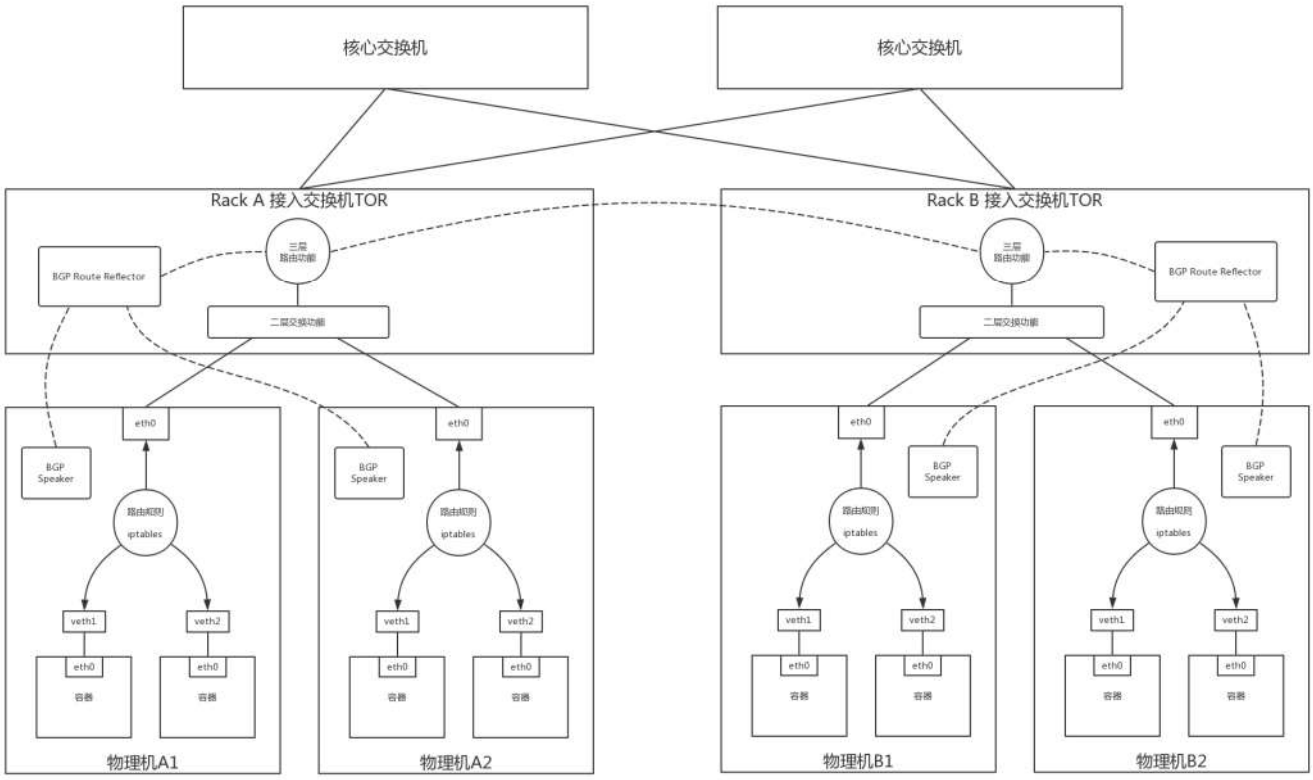
你看刚才的例子里只有六个节点，BGP 的互连已经如此复杂，如果节点数据再多，这种全互连的模式肯定不行，到时候都成蜘蛛网了。于是多出了一个组件 BGP Route Reflector，它也是用 BIRD 实现的。有了它，BGP Speaker 就不用全互连了，而是都直连它，它负责将全网的路由信息广播出去。

可是问题来了，规模大了，大家都连它，它受得了吗？这个 BGP Router Reflector 会不会成为瓶颈呢？

所以，肯定不能让一个 BGP Router Reflector 管理所有的路由分发，而是应该有多个 BGP Router Reflector，每个 BGP Router Reflector 管一部分。

多大算一部分呢？咱们讲述数据中心的时候，说服务器都是放在机架上的，每个机架上最顶端有个 TOR 交换机。那将机架上的机器连在一起，这样一个机架是不是可以作为一个单元，让一个 BGP Router Reflector 来管理呢？如果要跨机架，如何进行通信呢？这就需要 BGP Router Reflector 也直接进行路由交换。它们之间的交换和一个机架之间的交换有什么关系吗？

有没有觉得在这个场景下，一个机架就像一个数据中心，可以把它设置为一个 AS，而 BGP Router Reflector 有点儿像数据中心的边界路由器。在一个 AS 内部，也即服务器和 BGP Router Reflector 之间使用的是数据中心内部的路由协议 iBGP，BGP Router Reflector 之间使用的是数据中心之间的路由协议 eBGP。



这个图中，一个机架上有多台机器，每台机器上面启动多个容器，每台机器上都有可以到达这些容器的路由。每台机器上都启动一个 BGP Speaker，然后将这些路由规则上报到这个 Rack 上接入交换机的 BGP Route Reflector，将这些路由通过 iBGP 协议告知到接入交换机的三层路由功能。

在接入交换机之间也建立 BGP 连接，相互告知路由，因而一个 Rack 里面的路由可以告知另一个 Rack。有多个核心或者汇聚交换机将接入交换机连接起来，如果核心和汇聚起二层互通的作

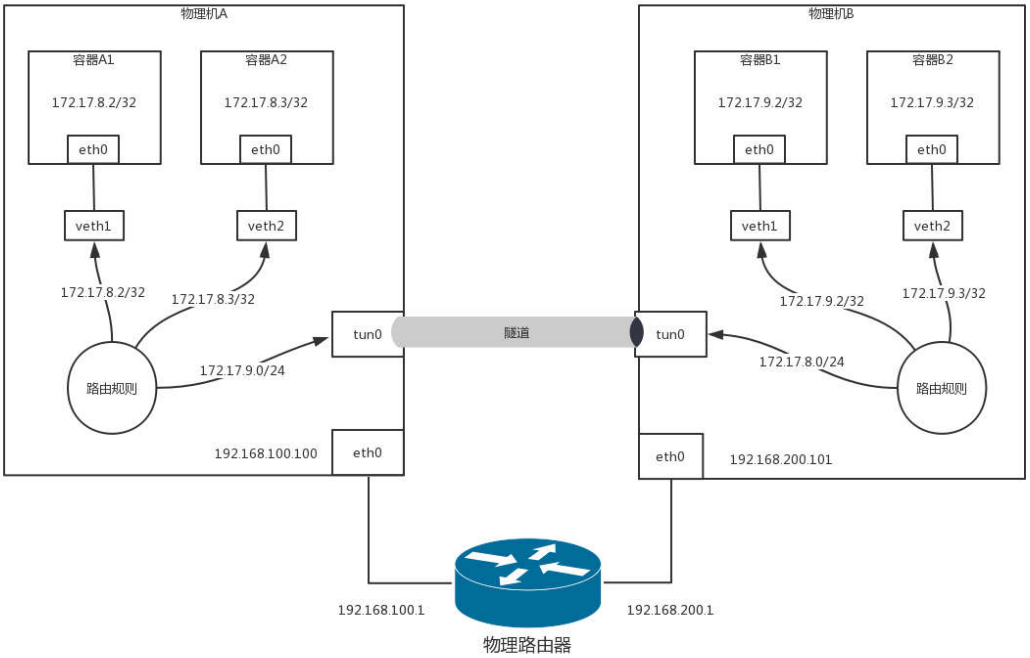
用，则接入和接入之间之间交换路由即可。如果核心和汇聚交换机起三层路由的作用，则路由需要通过核心或者汇聚交换机进行告知。

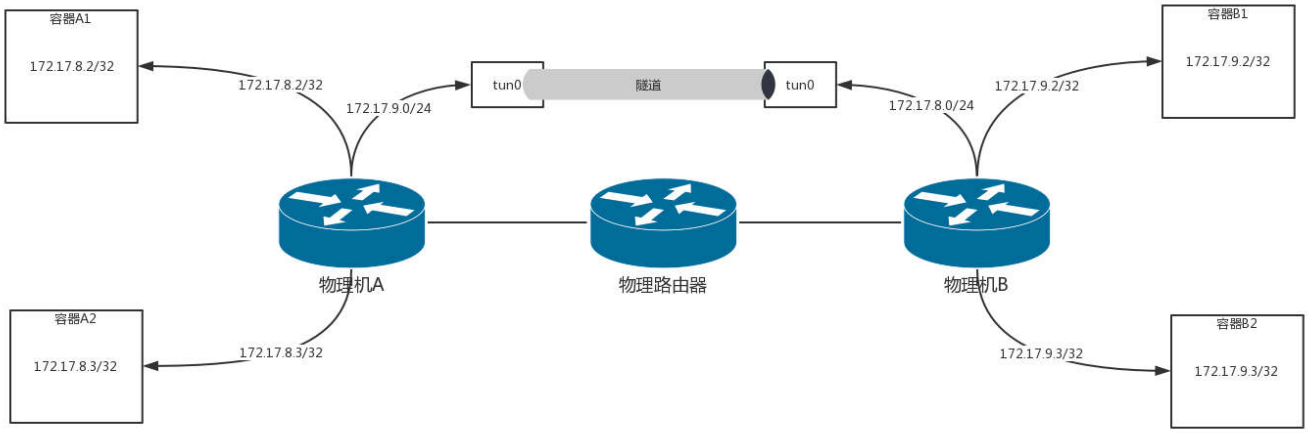
跨网段访问问题

上面的 Calico 模式还有一个问题，就是跨网段问题，这里的跨网段是指物理机跨网段。

前面我们说的那些逻辑成立的条件，是我们假设物理机可以作为路由器进行使用。例如物理机 A 要告诉物理机 B，你要访问 172.17.8.0/24，下一跳是我 192.168.100.100；同理，物理机 B 要告诉物理机 A，你要访问 172.17.9.0/24，下一跳是我 192.168.100.101。

之所以能够这样，是因为物理机 A 和物理机 B 是同一个网段的，是连接在同一个交换机上的。那如果物理机 A 和物理机 B 不是在同一个网段呢？





例如，物理机 A 的网段是 192.168.100.100/24，物理机 B 的网段是 192.168.200.101/24，这样两台机器就不能通过二层交换机连接起来了，需要在中间放一台路由器，做一次路由转发，才

能跨网段访问。

本来物理机 A 要告诉物理机 B，你要访问 172.17.8.0/24，下一跳是我 192.168.100.100 的，但是中间多了一台路由器，下一跳不是我了，而是中间的这台路由器了，这台路由器的再下一跳，才是我。这样之前的逻辑就不成立了。

我们看刚才那张图的下半部分。物理机 B 上的容器要访问物理机 A 上的容器，第一跳就是物理机 B，IP 为 192.168.200.101，第二跳是中间的物理路由器右面的网口，IP 为 192.168.200.1，第三跳才是物理机 A，IP 为 192.168.100.100。

这是咱们通过拓扑图看到的，关键问题是，在系统中物理机 A 如何告诉物理机 B，怎么让它才能到我这里？物理机 A 根本不可能知道从物理机 B 出来之后的下一跳是谁，况且现在只是中间隔着一个路由器这种简单的情况，如果隔着多个路由器呢？谁能把这一串的路径告诉物理机 B 呢？

我们能想到的第一种方式是，让中间所有的路由器都来适配 Calico。本来它们互相告知路由，只互相告知物理机的，现在还要告知容器的网段。这在大部分情况下，是不可能的。

第二种方式，还是在物理机 A 和物理机 B 之间打一个隧道，这个隧道有两个端点，在端点上进行封装，将容器的 IP 作为乘客协议放在隧道里面，而物理主机的 IP 放在外面作为承载协议。这样不管外层的 IP 通过传统的物理网络，走多少跳到达目标物理机，从隧道两端看起来，物理机 A 的下一跳就是物理机 B，这样前面的逻辑才能成立。

这就是 Calico 的IPIP 模式。使用了 IPIP 模式之后，在物理机 A 上，我们能看到这样的路由表：

```
172.17.8.2 dev veth1 scope link
172.17.8.3 dev veth2 scope link
172.17.9.0/24 via 192.168.200.101 dev tun0 proto bird onlink
```

这和原来模式的区别在于，下一跳不再是同一个网段的物理机 B 了，IP 为 192.168.200.101，并且不是从 eth0 跳，而是建立一个隧道的端点 tun0，从这里才是下一跳。

如果我们在容器 A1 里面的 172.17.8.2，去 ping 容器 B1 里面的 172.17.9.2，首先会到物理机 A。在物理机 A 上根据上面的规则，会转发给 tun0，并在这里对包做封装：

- 内层源 IP 为 172.17.8.2；
- 内层目标 IP 为 172.17.9.2；
- 外层源 IP 为 192.168.100.100；

- 外层目标 IP 为 192.168.200.101。

将这个包从 eth0 发出去，在物理网络上会使用外层的 IP 进行路由，最终到达物理机 B。在物理机 B 上，tun0 会解封装，将内层的源 IP 和目标 IP 拿出来，转发给相应的容器。

小结

好了，这一节就到这里，我们来总结一下。

- Calico 推荐使用物理机作为路由器的模式，这种模式没有虚拟化开销，性能比较高。
- Calico 的主要组件包括路由、iptables 的配置组件 Felix、路由广播组件 BGP Speaker，以及大规模场景下的 BGP Route Reflector。
- 为解决跨网段的问题，Calico 还有一种 IPIP 模式，也即通过打隧道的方式，从隧道端点来看，将本来不是邻居的两台机器，变成相邻的机器。

最后，给你留两个思考题：

1. 将 Calico 部署在公有云上的时候，经常会选择使用 IPIP 模式，你知道这是为什么吗？
2. 容器是用来部署微服务的，微服务之间的通信，除了网络要互通，还需要高效的传输信息，例如下单的商品、价格、数量、支付的钱等等，这些要通过什么样的协议呢？

我们的专栏更新到第 31 讲，不知你掌握得如何？每节课后我留的思考题，你都有没有认真思考，并在留言区写下答案呢？我会从已发布的文章中选出一批认真留言的同学，赠送学习奖励礼券和我整理的独家网络协议知识图谱。

欢迎你留言和我讨论。趣谈网络协议，我们下期见！

精选留言

sunlight001 11

工作中接触不到，现在完全看不明白的举个手！

2018-07-27

_CountingStars 2

1.因为公有云中的虚拟机之间 不直接通过交换机互通 中间有路由 使用VPC有可能可以实现

2.微服务数据交换现在有两种主流方式 http 和 rpc

2018-07-27

balancer 0

老师，网络数据包到达网卡的时候，内核怎么定位这个数据包属于哪个进程的那个socketfd?

2018-07-29

张稀虹 0

提个建议 老师能不能在下一期文章出来的时候在前一期文章中更新问题的答案，感觉比较深的话题讨论区的讨论就比较少了

2018-07-28

固态U盘 0

感觉老师的这个课程标价有点低

2018-07-27