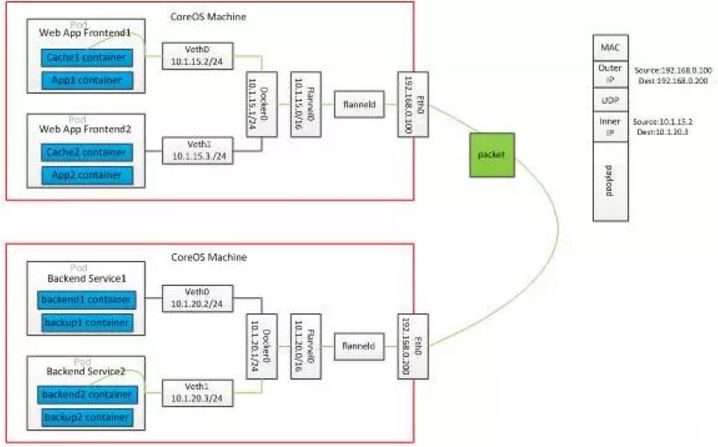
# 现有的跨主机容器网络解决方案

## Flannel容器网络

Flannel 是由 CoreOS 主导的解决方案。Flannel 为每一个主机的 Docker daemon 分配一个IP段，通过 etcd 维护一个跨主机的路由表，容器之间 IP 是可以互相连通的，当两个跨主机的容器要通信的时候，会在主机上修改数据包的 header，修改目的地址和源地址,经过路由表发送到目标主机后解包。封包的方式，可以支持udp、vxlan、host-gw等，但是如果一个容器要暴露服务，还是需要映射IP到主机侧的。



## ****Calico 网络方案****

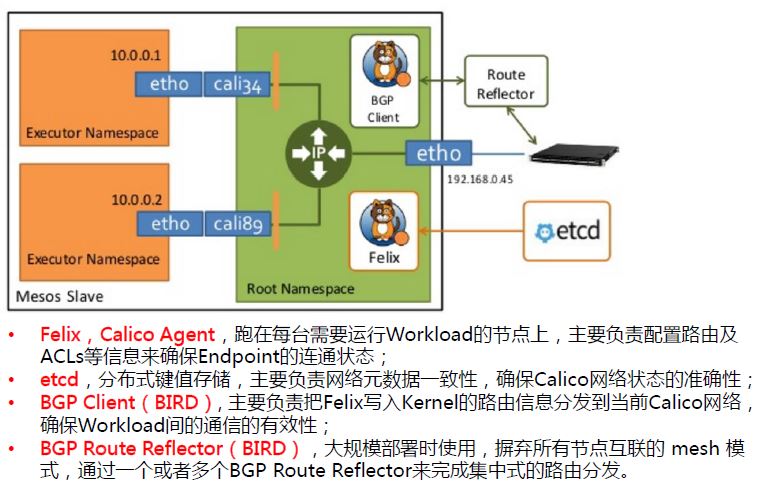


图 7  calico网络

Calico是个年轻的项目，基于BGP协议.完全通过三层路由实现，对网络不熟悉的同学可能都没有听说过。Calico的目标很大，可以应用在虚机，物理机，容器环境中。在Calico运行的主机上可以看到大量由linux路由组成的路由表，这是calico通过自有组件动态生成和管理的。这种实现并没有使用隧道，没有NAT，导致没有性能的损耗，性能很好，从技术上来看是一种很优越的方案。这样做的好处在于，容器的IP可以直接对外部访问，可以直接分配到业务IP，而且如果网络设备支持BGP的话，可以用它实现大规模的容器网络。但BGP带给它的好处的同时也带给他的劣势，BGP协议在企业内部还很少被接受，企业网管不太愿意在跨网络的路由器上开启BGP协议。

## **跨主机的容器网络解决方案不外乎三大类：**

隧道方案：比如Flannel的 VxLan。特点是对底层的网络没有过高的要求，一般来说只要是三层可达就可以，只要是在一个三层可达网络里，就能构建出一个基于隧道的容器网络。问题也很明显，一个大家共识是随着节点规模的增长复杂度会提升，而且出了网络问题跟踪起来比较麻烦，大规模集群情况下这是需要考虑的一个点。

路由方案：路由技术从三层实现跨主机容器互通，没有NAT，效率比较高，和目前的网络能够融合在一起，每一个容器都可以像虚拟机一样分配一个业务的IP。但路由网络也有问题，路由网络对现有网络设备影响比较大，路由器的路由表应该有空间限制一般是两三万条。而容器的大部分应用场景是运行微服务，数量集很大。如果几万新的容器IP冲击到路由表里，导致下层的物理设备没办法承受；而且每一个容器都分配一个业务IP，业务IP消耗会很快。

VLAN：所有容器和物理机在同一个 VLAN 中。

# ****PPTV Docker网络解决方案****

对比了几种解决方案之后，结合PPTV的实际情况：

1.网络组人力不足以维护一个Overlay网络，Overlay网络出问题排查复杂，会出现失控的状态。

2.隧道技术影响性能，不能满足生产环境对网络性能的要求。

3.开启bgp对现有网络改动太大，无法接受。

4.运维组同学希望能通过网络桥接的方案解决容器网络。

## 

## ****容器网络桥接****

## 最终，我们的解决方案，基于docker的bridge模式，将默认的docker bridge网桥替换为linuxbridge，把linuxbridge网段的ip加入到容器里，实现容器与传统环境应用的互通。

实现思路很简洁清晰，现在有一个Mesos主机

1.首先会在该主机上添加一个linux bridge，把主机网卡，可以是物理机的，也可以是虚拟机的，把这个网卡加入bridge里面，bridge配上网卡原本的管理IP。

2.创建一个新的docker bridge网络，指定bridge子网，并将该网络的网桥绑定到上一步创建的网桥上。

3.容器启动时候，指定容器网络为第二步中创建的bridge网络，同时为容器指定一个该网络子网内的IP。容器启动后网络IP默认即可与外界互通。

这里要注意的是第二步，我们的同学在研究这个方案的时候绕了一个很大的圈子，因为docker容器使用dockerbridge网络模式的时候，在容器启动时会默认把容器的网关指向到宿主机上的网桥IP，即linux bridge的IP，而这个IP并不是该网段的网关。所以我们需要在容器启动的时候将容器网关指向到实际的网关地址，而解决这个问题的方法在docker官方文档中并没有提到，我们最终是在一个issue里找到了解决办法。

链接在此  https://github.com/docker/docker/issues/20758

给个简单的例子

docker network create --gateway10.199.45.200 --subnet 10.199.45.0/24 -o com.docker.network.bridge.name=br-oak--aux-address "DefaultGatewayIPv4=10.199.45.1"  oak-net

关键参数:  --aux-address"DefaultGatewayIPv4=10.199.45.1"

以上边的命令为例，该命令中创建了一个dockerbridge网络，并与docker所在主机的br-oak网桥做桥接，该网络的使用了10.199.45.0/24这个子网，同时通过  --aux-address"DefaultGatewayIPv4=10.199.45.1" 这个参数将容器启动时的网关指向到10.199.45.1

通过网桥的方式解决容器网络有两个问题:

1.linux bridge 只能添加跟slavehost 同一个vlan的IP，也就是说容器IP必须要和宿主机在同一vlan下，这在一定程度上就限制了容器跨宿主机漂移的范围。

不过这个问题在PPTV的生产环境中天然不存在，因为我们的生产环境中，每个数据中心的主机都在一个很大的子网内，基本能满足容器在整个数据中心的任意节点下漂移。

2.要让容器IP在不同的宿主机上漂移，宿主机的docker网络需要使用同一个CIDR，也就是各宿主机的容器使用同一个网段。而不同宿主机的使用同一个容器网段就会涉及到IPAM的问题，因为宿主机的docker daemon只知道他本机上的容器使用了哪些IP，而这些IP在其他宿主机上有没有被使用，是不知道的。

在默认的docker bridge中，因为这些ip不会直接与外部通信，所以容器使用相同IP也不会有问题，但是当容器网络通过linux bridge打通以后，所有容器都是2层互通的，也就是会出现IP冲突的问题。

为了解决上边提到的问题，实现全局的IP管控，我们开发了IP池管理平台，实现对容器IP的分配管理。由这个平台管理的IP有三种状态:

1.未分配给应用

2.已非配给应用并且在使用中

3.已分配给应用但是当前未使用

管理平台以marathon上的app信息作为数据源，定期去调用marathon的API更新IP列表。当我们要在marathon上创建一个使用固定IP的容器时，首先会请求IP池管理平台的IP分配接口，请求的时候把app id发给分配接口，管理平台根据appid 判断这个应用是否是新应用，如果是新应用则从IP池中返回一个未使用的IP，并将此IP与应用关联。如果是已经存在的应用则分配已关联的IP。IP与应用关联之后，此IP就不会再分配给其他应用，除非IP池已经没有可用IP，这样做是为了防止应用如果重启或者重新构建的时候，IP有可能会被其他在同一时间启动的实例使用掉的风险。

IP池管理模块分配IP的流程图大致如下:

图 8  ip分配流程图

## ****后续工作****

网络方案搞定，固定IP搞定。我们要做的还有很多。

通过网桥的方式、解决了容器网络的问题，我们接下来还要面临其他的问题。

首当其冲的就是原先的服务自动注册、自动发现，不再适应了。因为原先的方案是基于NAT的模式做的，而现在实现了独立IP的功能。我们需要将现有的平台与PPTV内部的DNS做自动化对接，每当有容器创建和生成时，都会自动对容器的IP做DNS解析。

另外一个问题是负载均衡，PPTV的负载均衡基本都是通过LVS + nginx实现的，但对于后台的容器应用来说，每次扩容和缩容、或者创建新的应用，负载均衡的后端配置也是需要自动更新的。