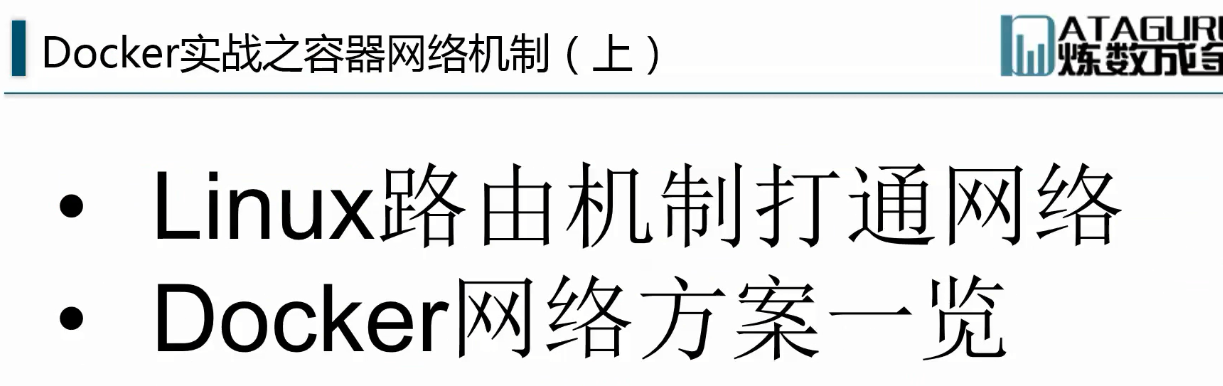
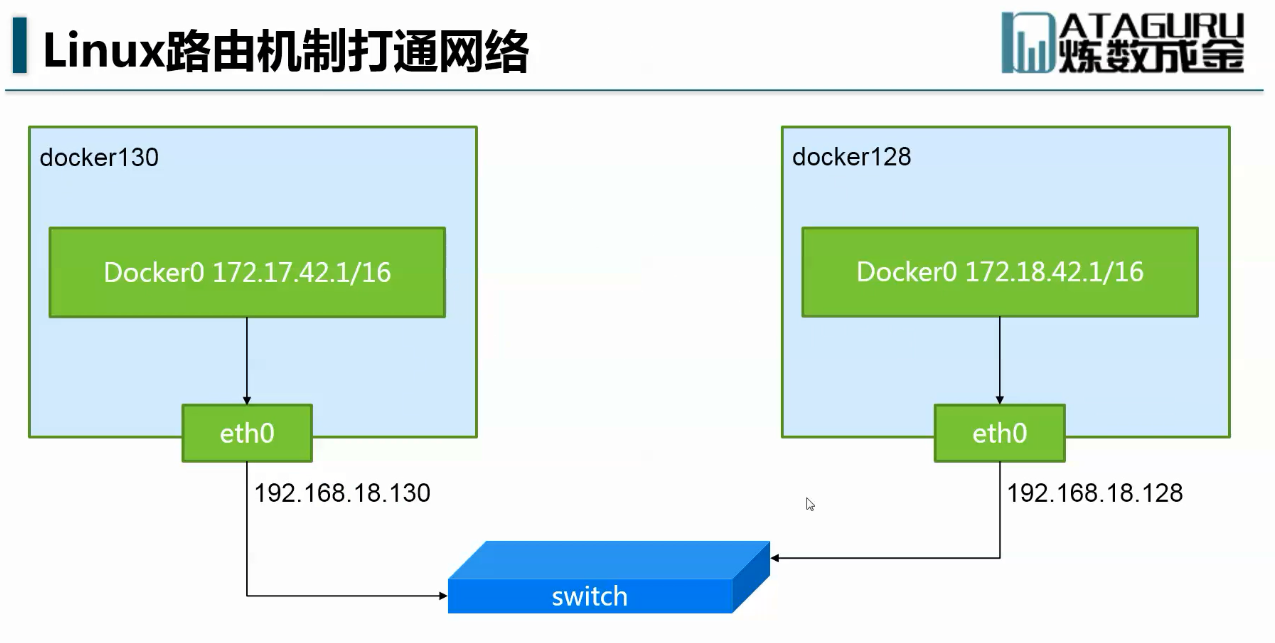


Docker的网络部分





Docker130 和Docker128是虚机

需要让两个Docker Container的eth0互通

为什么叫路由机制？因为是效率最高 性能最好的

实际项目中也采用了这种机制代替了Kubernetes的FlyNails机制

具有生产实际的意义

物理网卡 网段

【ip/数字 这个数字表示子网掩码的位数 这样就可以知道是不是一个网段的ip地址了】

这是两个不重叠的子网 网段 ----- 两个独立的子网 交互只能通过路由

Docker上的ip和物理网卡的ip也不重叠

三个网段

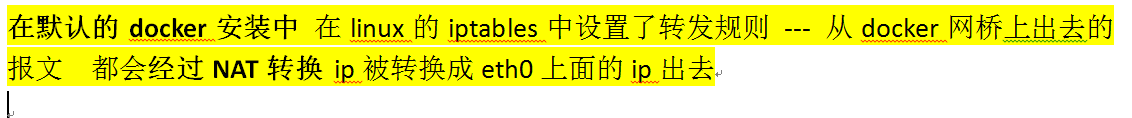
做法：

 重启是因为docker0重新创建 之后 地址就是修改过的了

【修改docker的配置文件的原因 就是 默认是网段】

【根据上一节课的交接 docker0上的Docker Container可以通过NAT和宿主机的网卡进行地址转发 这个转发规则是在安装Docker的时候 Docker在iptables中自动添加的

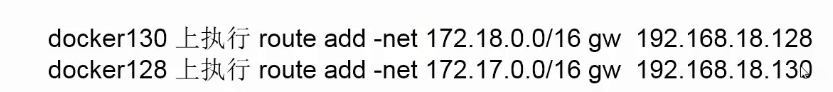
上一节课的笔记中



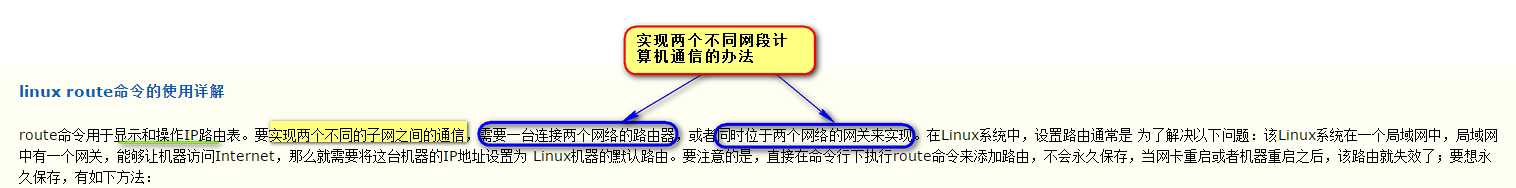
**所以 这个时候 docker container和自身网卡的通信是没有问题的**  现在需要做的是 手动配置虚机的路由表 直接去访问另一个网段的虚机

】

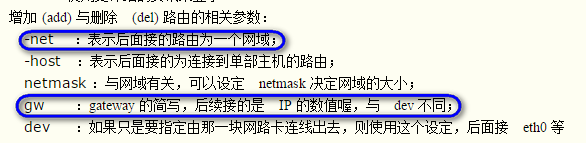
**因为路由上打通** 所以 分别在两个虚机上添加两个路由规则



【







我们这里面没有路由 那么就在各自的虚机的路由表上进行操作

Docker130的ip是 所做的操作是 route add –net 172.18.0.0 gw

网关 gateway一般是连接局域网的路由器 这里面分别使用对方的宿主机作为路由器 来访问另一个网段（因为另一个网段是docker的 docker在安装的时候 就在iptables做好了） 所以 把docker130的路由就当做docker128 docker128的另一端已经和另一个docker网段互联 ----- 所以 就设置 docker128作为docker130的网关

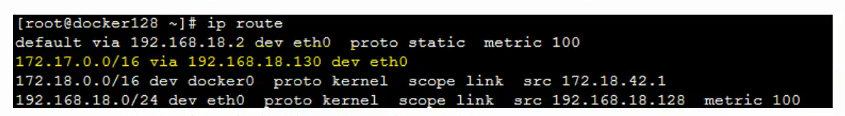
所以 命令就是

Route add –net 172.18.0.0/16(目标的docker网段) gw 192.168.18.128(当前主机的网关)

Route add –net 172.17.0.0/16(目标的docker网段) gw 192.168.18.130(当前主机的网关)

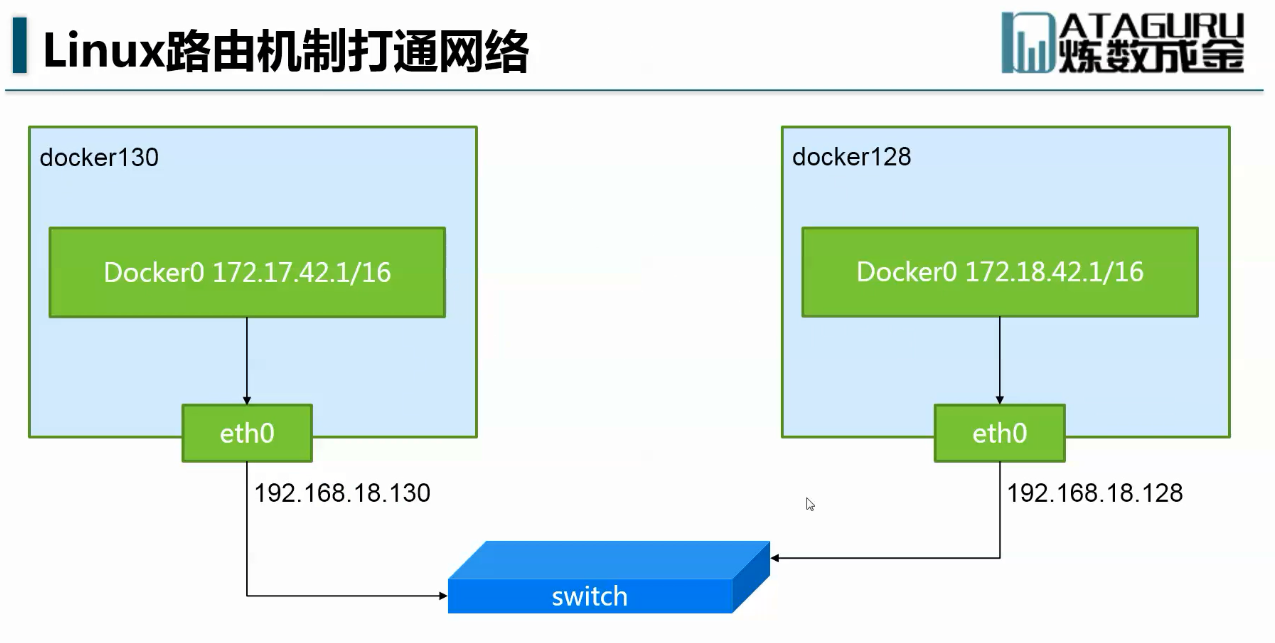
】

添加完之后：



【可以看到 要是访问 要通过192.168.18.2

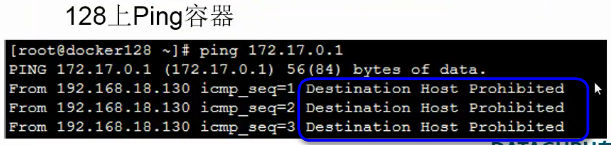
同一个网段的两格虚拟机 docker128和docker130 同一个网段的两个虚拟机通信 直接练到交换机上就可以了 所以 画的对



】

 看到130上面的docker容器的ip是

这时候 在128上



禁止的是因为  路由表中 这个地址不属于物理网卡 所以 默认被禁止掉 ---- 这个在防火墙中



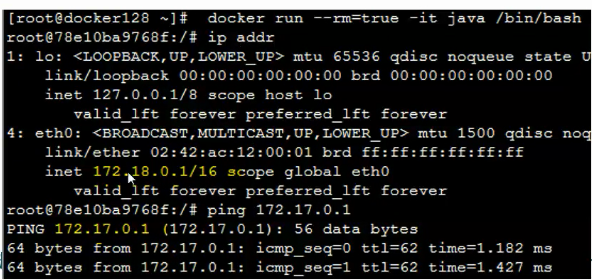
Iptables中 docker的nat规则清理掉 还有防火墙的默认规则清理掉

 宿主机上ping

这个时候 通了

路由通了

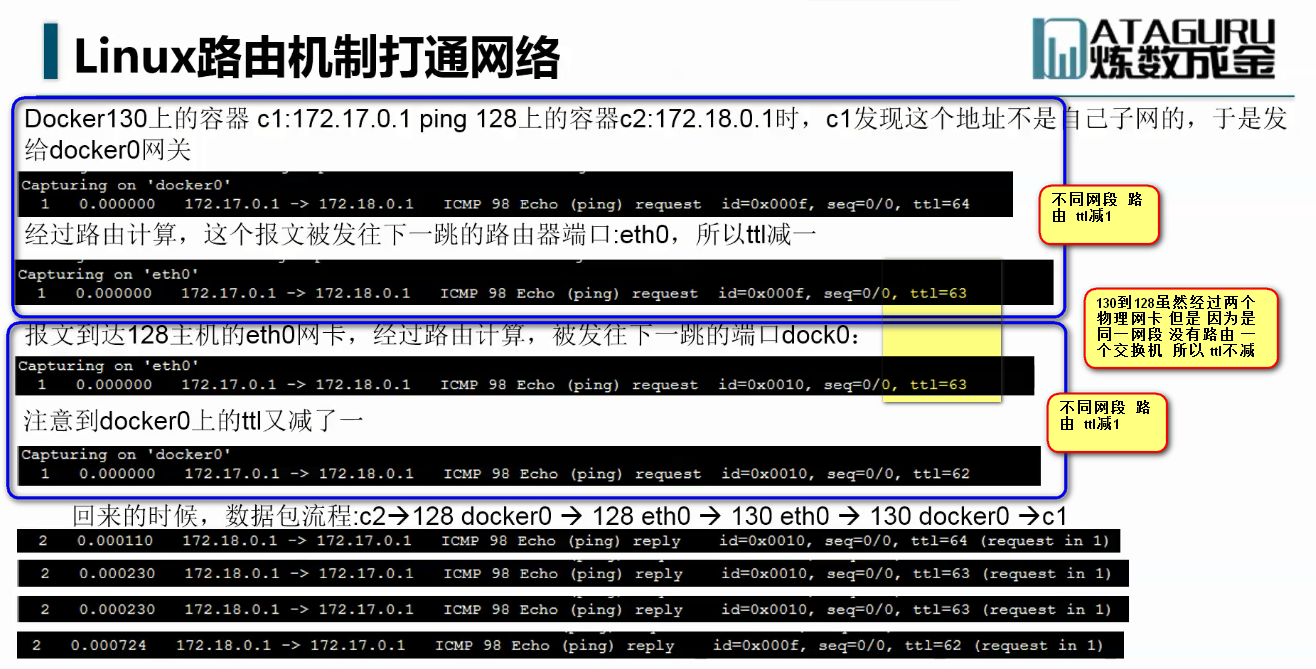
然后我们在容器中ping

 也可以ping通

---- 这样 两个不同网段的docker container就可以ping通 这种方式操作很简单 就是添加了宿主机的路由表

----------

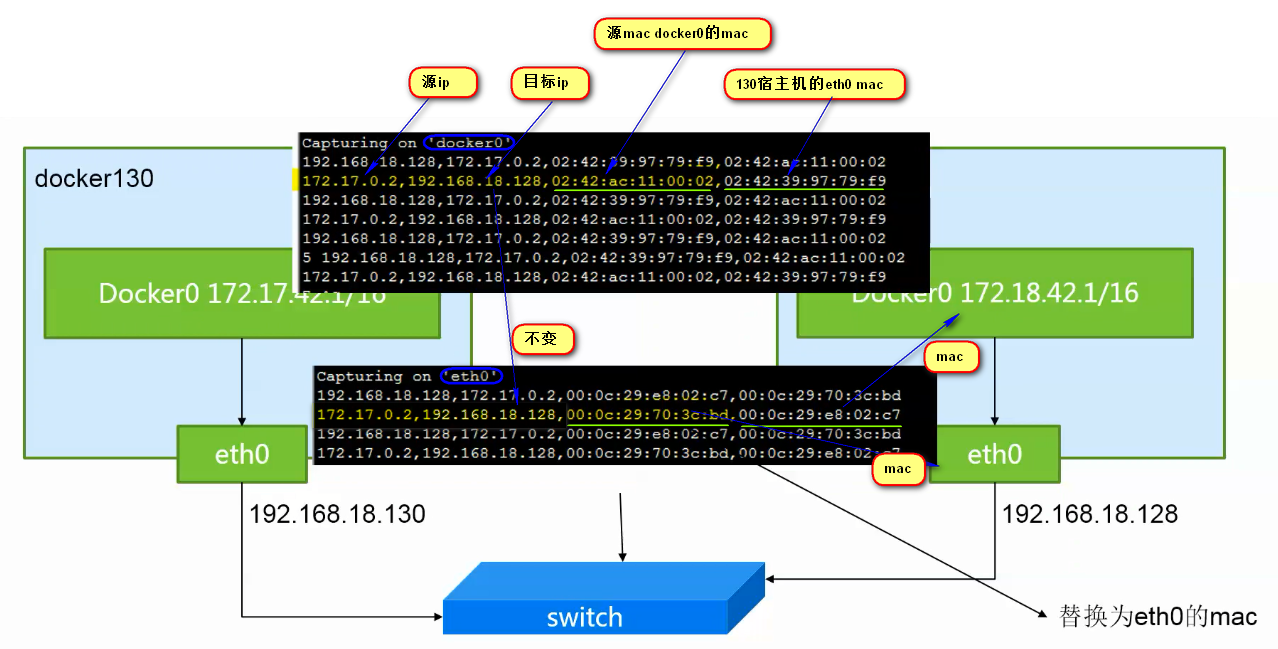




这里面变化就是因为路由的变化 导致包头的变化 效率非常高 对其他没有做任何变化

到底高效么？linux本身的路由转发机制 非常高效 接近于思科中对等的路由器和交换机 实际中 比flynail的机制高效 **需要掌握**

经过路由的时候 mac地址变成自身的路由自身的地址

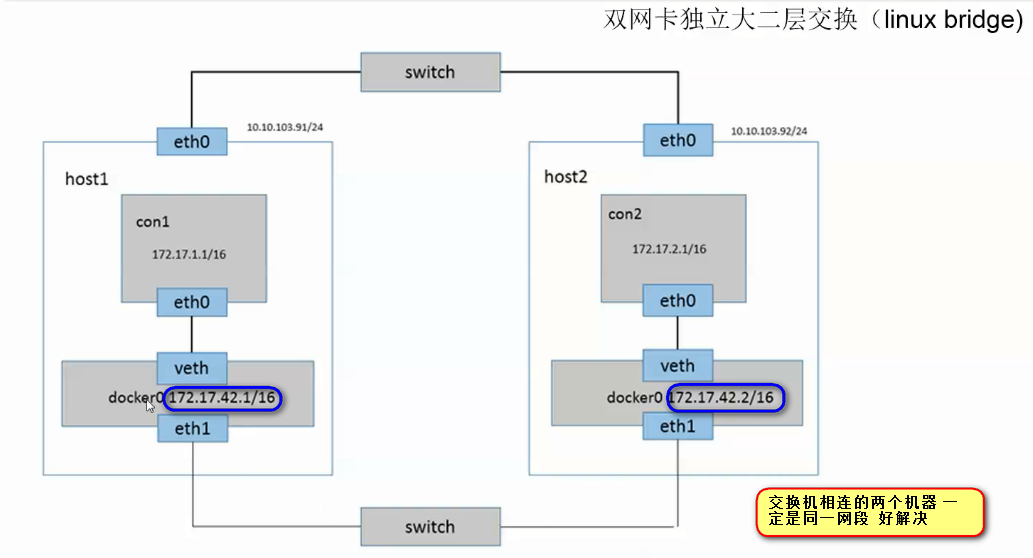


打印出上面的内容的命令是



=====================================

------ 另一个是 双网卡独立大二层交换



前面是三层的路由 这个直接是二层交换机 ----- 更高效

但是 缺点 多一个交换机 并且 同一个网段 广播风暴比较多 但是 三层的路由不会这样

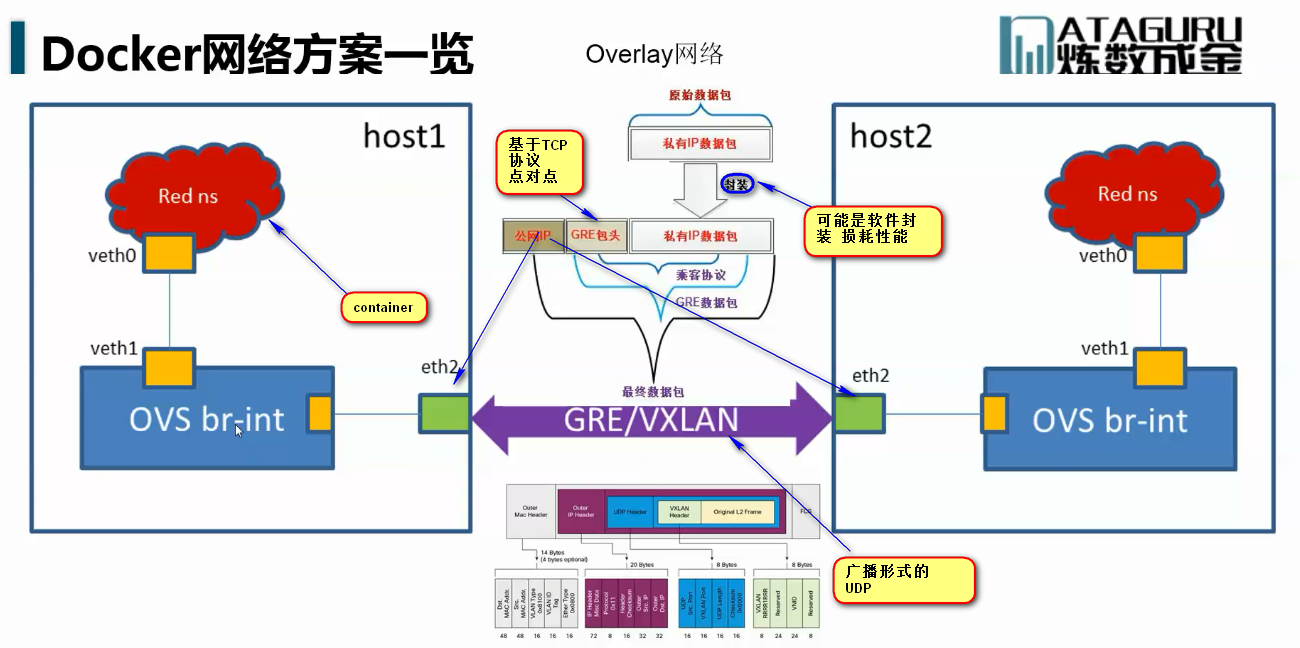
容器非常多 这个是一个非常多的局域网 广播mac地址就是泛红 但是 小网络可以

这个里面 con1表示container 1

---------------------

Overlay网络

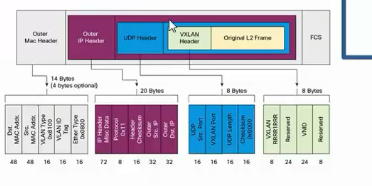
这个是主流的方案 是在openstack中的解决方案



VLAN是二层

VXLAN现在是国际标准

GRE是微软推的 以后可能是标准



Overlay网络的好处：不改变现在网络的侵入 不改变任何网络拓扑 && 可以在内部可以做非常复杂的虚拟网络 灵活性非常高

缺点： 性能损耗

--- 因为采用了封装方式 很多交换机 路由器都不支持

--- 每个报文都这么封装 消耗了内存 加长了处理实践 很多都是软件方面实现 ---- 也就是当前的硬件技术环境不支持封装和解封包的环境

**但是 Overylay是一种虚拟化的主流方式**

===============

Overylay用的最多的一个实现就是ovs open v switch 不讲了 现在无论openstack 还是ovs 都绕不过

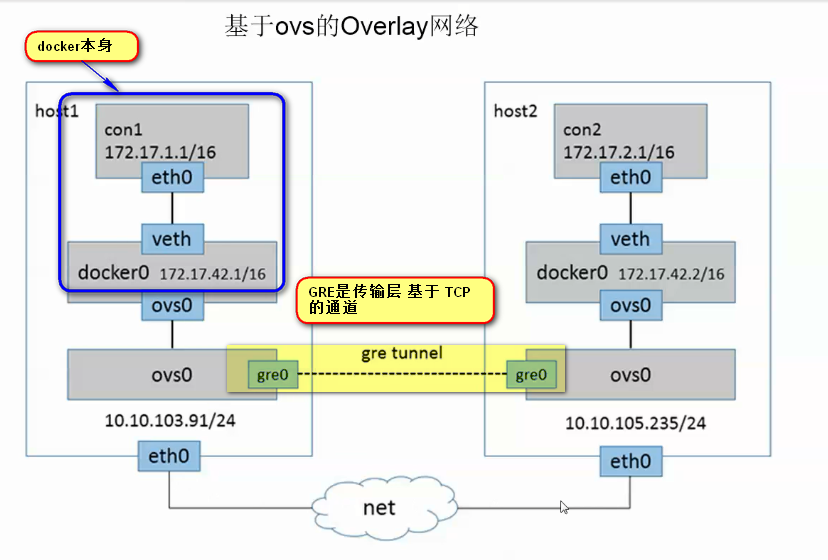
一台linux机器上 模拟出来一个交换机 机器上的虚机或者容器都可以接到这上面的

Ovs的一种就是Gre通道的 用gre来做 ---- 任何两个节点间打一个通道

Docker0网桥连接到ovs的一个端口上

Ovs的交换机上面开了一个gre的端口

Gre端口连接到另一端 形成一个tunnel



【形式上 也就是分层看 直接联通】

实际传送中 是要经过封包 然后 经过eth0网卡进行传送的

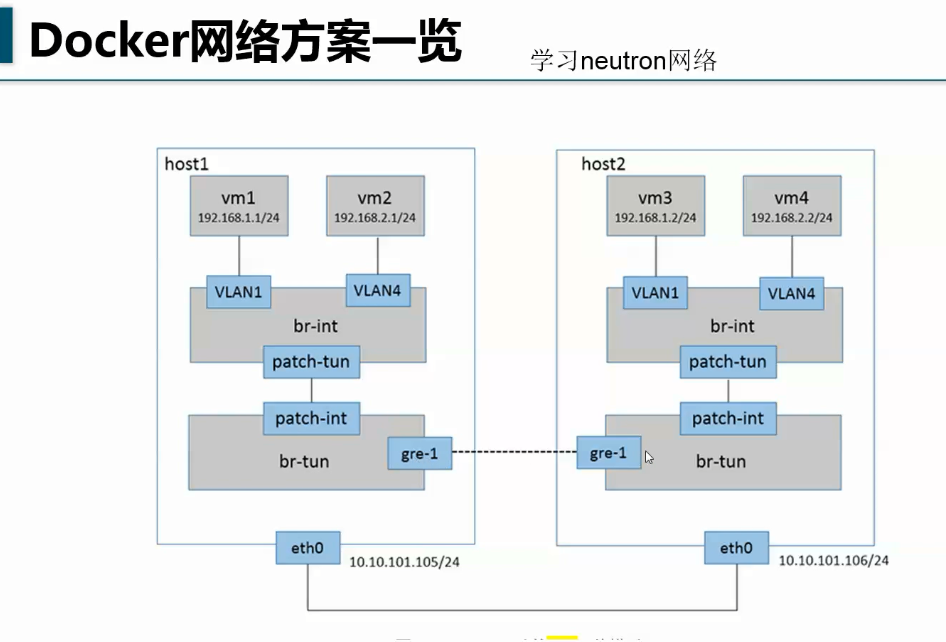
也就是

Tunnel不是直接过来的

也就是ovs0把上面的docker0传送过来的报文 封装成tcp包 然后 ovs再发给eth0

GRE比较容易配置

----- ovs的原理



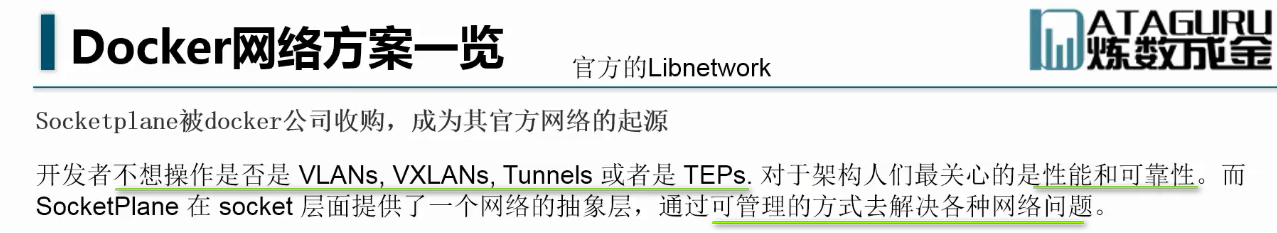
【这里面的vm1就相当于上面的docker0】

Ovs可以跟openstack进行融合 ----- 主流技术

Docker本身是没有解决网络问题的

原因是开发者 ---程序员对网络 硬件没有什么知识 不是很懂 ----- 其实docker开发者来说 不是网络专家 所以 绕过这个问题 因为此前 openstack又很成熟的网络方案

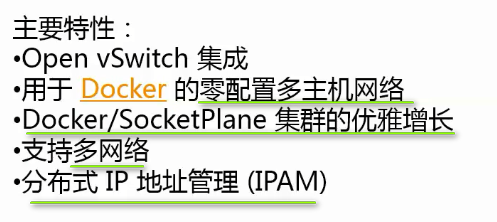
官方的解释



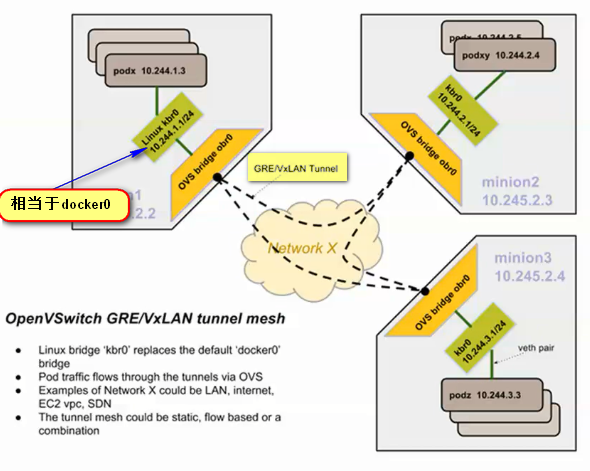
后来 这个问题不得不解决 收购了一个SocketPlane的公司 现在是官方网络的一个群

因为调研过程 用的多的是SocketPlane --- 所以 收购了

SocketPlane就是用的是ovs做的 ---做了一些工具 不用敲ovs的命令 变得方便



模型如下



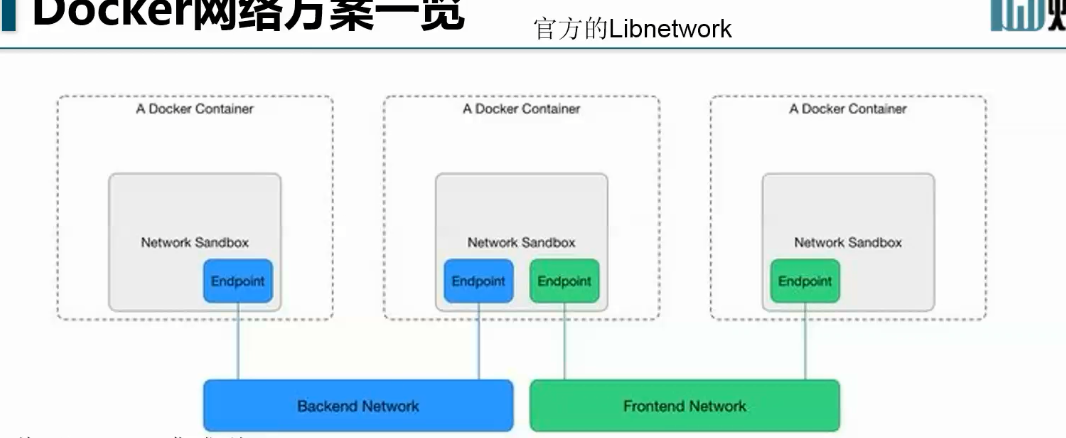
连接以后 可以通过GRE/VxLAN通道

还是使用了ovSwitch技术

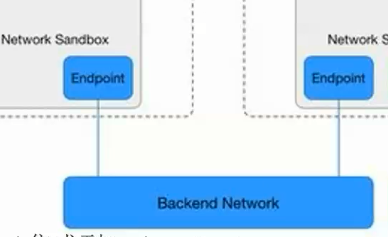
Docker收购了之后 做了一个很宏伟的蓝图 LibnetWork

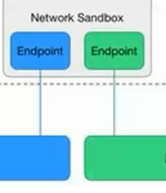
Sandbox沙箱机制 理解成交换机

Endpoint是交换机的一个端口



这样 就是 每一个Container内部 有一个沙箱机制 类似一个交换机 --- 有了交换机 就相当于 可以又很多主机插上来 ---- 这样 原本的一个Container就变成了一个Container网络 ----- 这样 因为沙箱机制（交换机）变成的网络 ==== Network SandBox

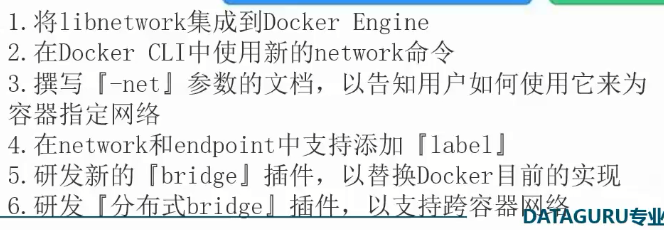
这样 两个不同的Docker Container就可以看成两个不同的NetWork 可以通过局域网 或者虚拟网 连接到一起

因为本身是一个沙箱 类似于交换机 所以 可以多个主机（虚拟） ----- 这样 就会有多个EndPoint 这样 一个Container内部 就可以又属于不同网络的两个交换机的EndPoint

这个就是交换机+端口的重新定义的概念

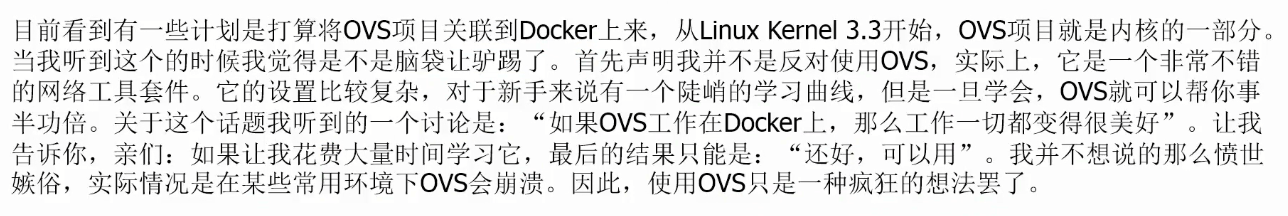
做了这个概念以后 提出了很多新名词 但是 懂网络的人 很难理解

目标就是 简单配置 就能让不同网络的container可以通信

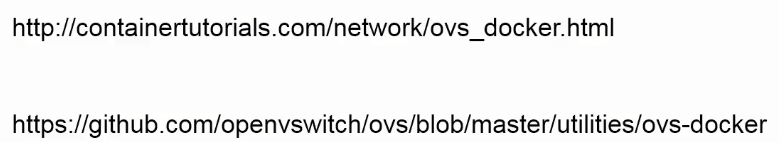
 计划和要做的

Label在K8中讲

Docker开发者说的OVS ---抵触OVS

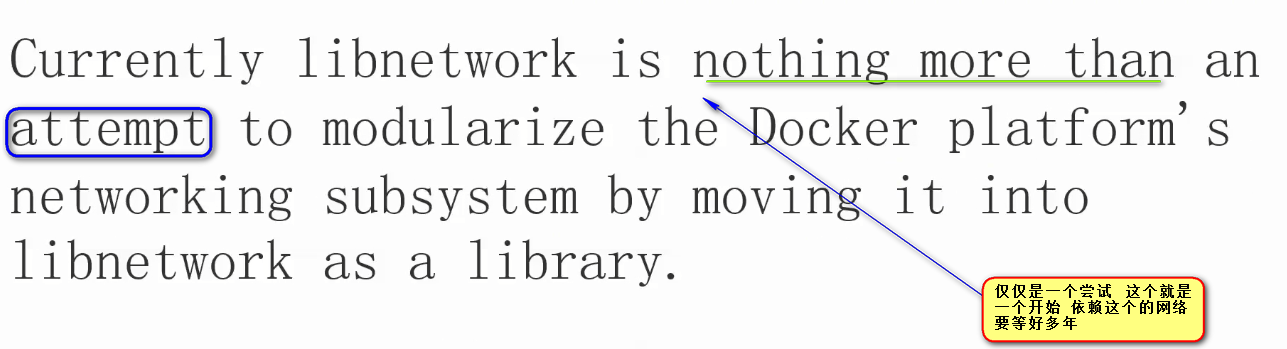


OVS影响非常广泛

 这个是OVS和Docker的东西

未来就是docker本身增加ovs的支持

现在的情况是什么？



Docker的网络 关系到K8 ------ 很重要