



Linux namespace是虚拟化网络部分非常重要的基础知识 先做介绍

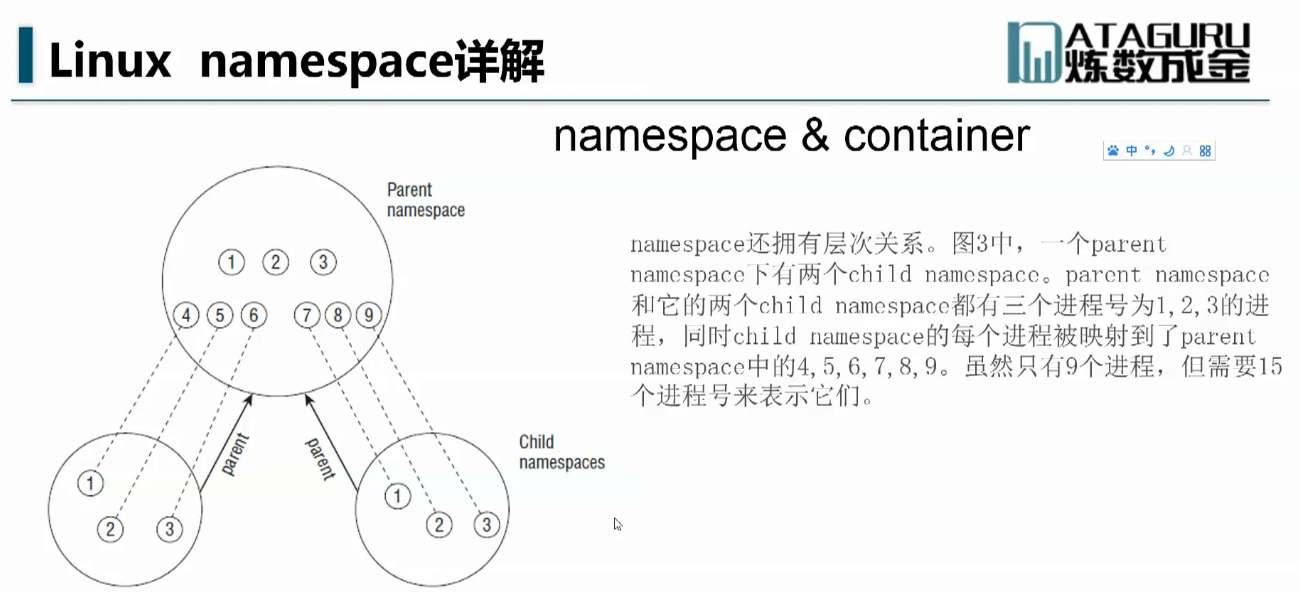
OVS技术是虚拟化的重要技术



Namespace相当于一个空间 可以放任意系统资源 比如 进程 网络 账号 文件系统 都可以拿到这个namespace中去 ---- 一旦某一个资源放到namespace之后 别的地方就看不到了 就隔离起来了 ---- 这样 进程ID会有相同的 编号一样 但是namespace不同 所以 不冲突 --- 每个用户ps只能列出来自己的namespace下面的进程 不能列出来别的namespace下面的进程

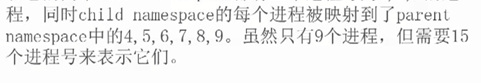
------ 容器就是把相关的进程 包括相关的网络资源 账号 包括文件系统 放到相关的namespace的中去 这就是一个容器

不同的容器 属于不同的namespace 是相互隔离的



Namespace有一个层次关系 parent ns和两个子ns

Pns会看到自己和cns的进程 这样 pns自己有三个进程 又看到了cns各自三个进程 所以 移动看到了九个进程 -----

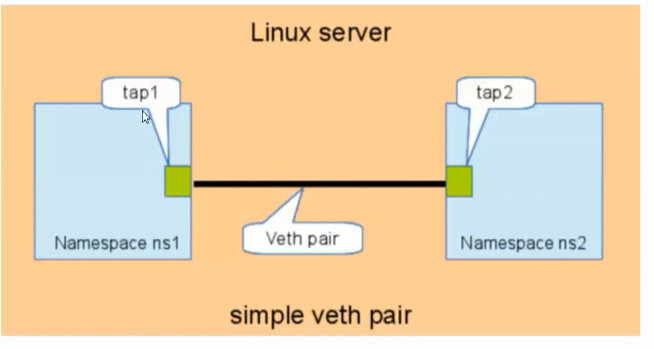


Pns一般不会遇到

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

 无数个地球 你在另外一个地球仍然存在 不同的地球就是不同的ns

\*\*\*\*不同的ns中如何交互

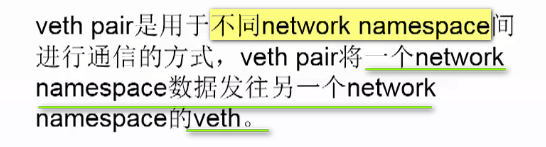
 veth pair是软件实现的网络接口 两边是不同的ns

Veth pair把两个不同的空间打通 和普通的网线不一样

但是openstack中没有将veth pair说清楚

这个是点对点通信

------------ veth pair最最关键的一点 **就是把两个不同ns打通**



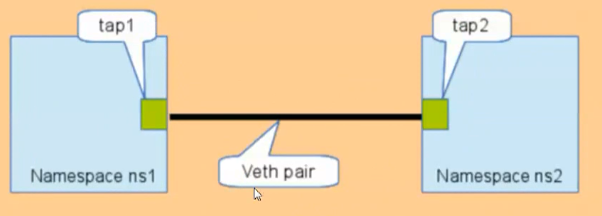


上面这个图是点对点的方式 

那么多个之间怎么通信？  这就是Linux另外一个技术 Linux Bridge

**Linux Bridge相当于一个软件交换机 和交换机的模式相似**

**---- 将不同的ns中的点打通 并且交互**

前面的这个是  点对点的 如果各自有四个 怎么交互

中间加一个bridge  为什么两边用了两个veth pair ? 因为这个Linux bridge是在主空间中 就是另外一个ns中 这样 前面说了 如果两个不同的ns要通信 就必须使用veth pair进行打通

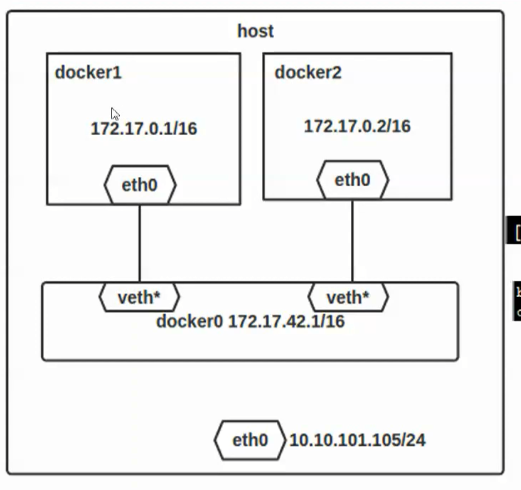
---- 三个空间 就可以完全交换 这个tap就相当于某一个ns的以太网卡 br-tap 就是bridge一端的以太网卡

Linux bridge仅仅是一个二层设备 所以 这里面 仅仅是mac地址的交换

Linux bridge 特殊在 有一个ip地址 可以做一些路由转发

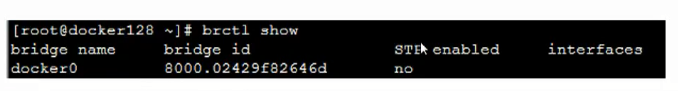
===========================

两个ns的连接是不知道的 一旦一个网卡插入到ns中之后 其实 另一端的ns 我们是不知道的 ----



Bridge

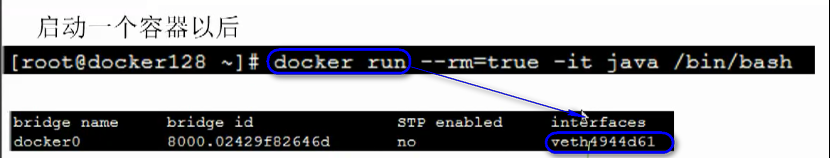
安装docker之后 默认建立docker0 bridge -----



正常情况下 里面的Bridge是空的 没有挂载 ---体现为interfaces那一栏什么都没有

【】

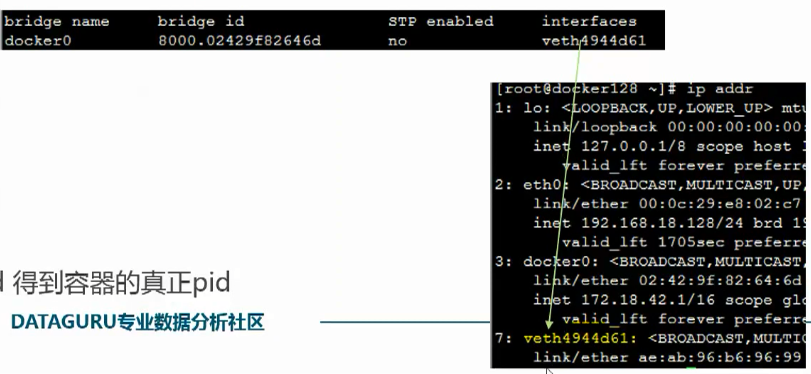
如果启动一个docker container之后



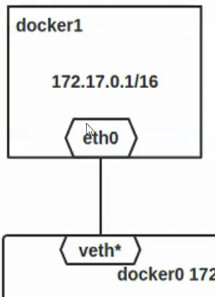
发现 docker0上面的interfaces下面 有一个veth4944d61 就是veth pair的一个端口

这个veth4944d61就是前面那个图中标记的  veth\*

现在查看一下 ip addr

 这样 可以看出来 一个端口 就是veth pair的另一端 【docker的这个图 和 linux bridge那个图很类似 一个容器 就是一个不同的ns 一个容器内的进程 和另一个容器内部的进程是不相互看到的】

【 】

每建立一个容器 一端就是容器的网卡 eth0 另一端 就挂接在docker0网桥上面 

现在做一个发现

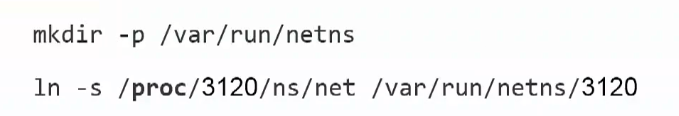
Docker中的ns和pid是一样的 ----- 进程的编号 就是ns的名称

---- 容器本身的pid不是指容器内部进程的pid ----- 如何看到容器本身的pid?

使用命令  这样可以获取这个容器本身的pid

这样查出来这个docker的pid就是3120

做下面两个操作

 这个3120就是docker的pid

Docker的命令空间是建立在/var/run/netns/${docker container pid} 这个目录下面

【】

但是 后面用到的 ip netns exec 3120这个是按照进程走的



需要在 这个进程下面  找ns

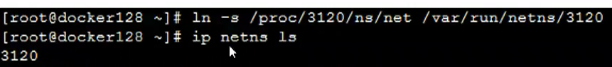
所以 把两者关联到一起 --- 也就是把docker container的pid相关的docker ns和ip netns的目录进行关联

Docker觉得ns是底层的 不需要让大家知道 ---- 所以 就没有按照标准的 在/proc/${pid}/ns/net下面 而是 自己在/var/run/netns/${pid}下面建立

放到这样的一个目录下

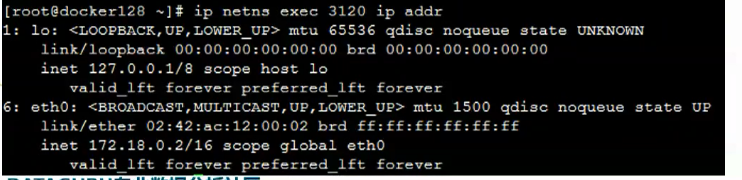
如果不知道ns的真相 觉得  这个从命名看 很形象 运行的网络命名空间的xxxx 但是 这个目录 是不符合 后面的ip netns这个工具的要求的 ---- 所以在这里建立一个软连接 一方面符合docker自身的ns的要求 另一方面 也符合标准的ip netns的要求

连过来以后



这样就查出来 有一个3120的ns

现在查看某一个ns的ip address



这个就是 在某一个ns空间执行一个命令 ----- ip nets exec 3120 ip addr ---- 就是在3120这个ns空间执行命令 ip addr

查看3120的ns的ip信息 ---- 可以看到  是分配docker容器的一个地址

这个eth0 就是和宿主机中的veth\*进行配对

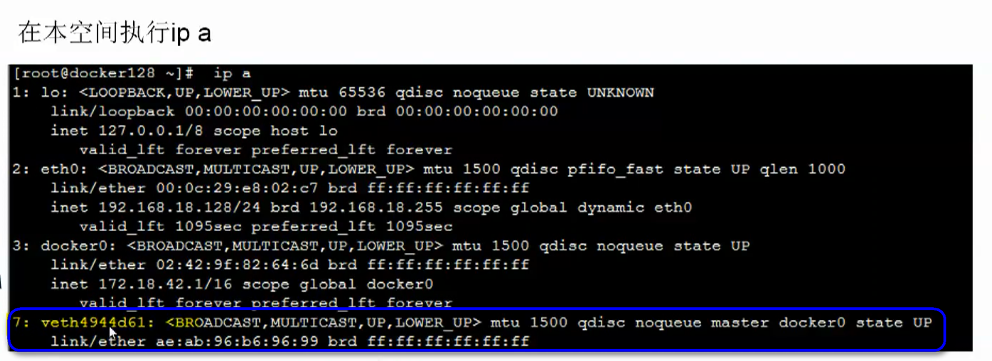
【其实这个ip也可以 使用docker xxxxx /bin/ls中 然后 执行ip addr查询到 但是 现在 是用来证明 docker container实际上就是一个linux中的namespace ----- 所以 用Linux中的ns命令对这个直接进行验证 】

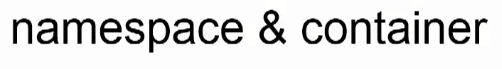
**【这样首先证明了 docker container实际上就是linux namespace的一种】**

**【然后再验证 这个docker container中的eth0 和 docker0的veth\*是否真的配对】**

 ---- 这个看到 在3120这个ns中 执行 ethtool –S eth0 ----发现 这个eth0的对端是peer ifindex是7

在宿主机中



这样验证了ns 3120中的eth0 和 docker0上的veth4944d61是配对的 ----- 就验证了 linux namespace和docker container的相似性 

===== 这时候知道 这样就知道pid是3120的容器挂在到docker0的上了 ----- 这个实战很重要

【

 这个centos6的内核不支持 必须升级内核 或者使用其它的CentOS的版本】

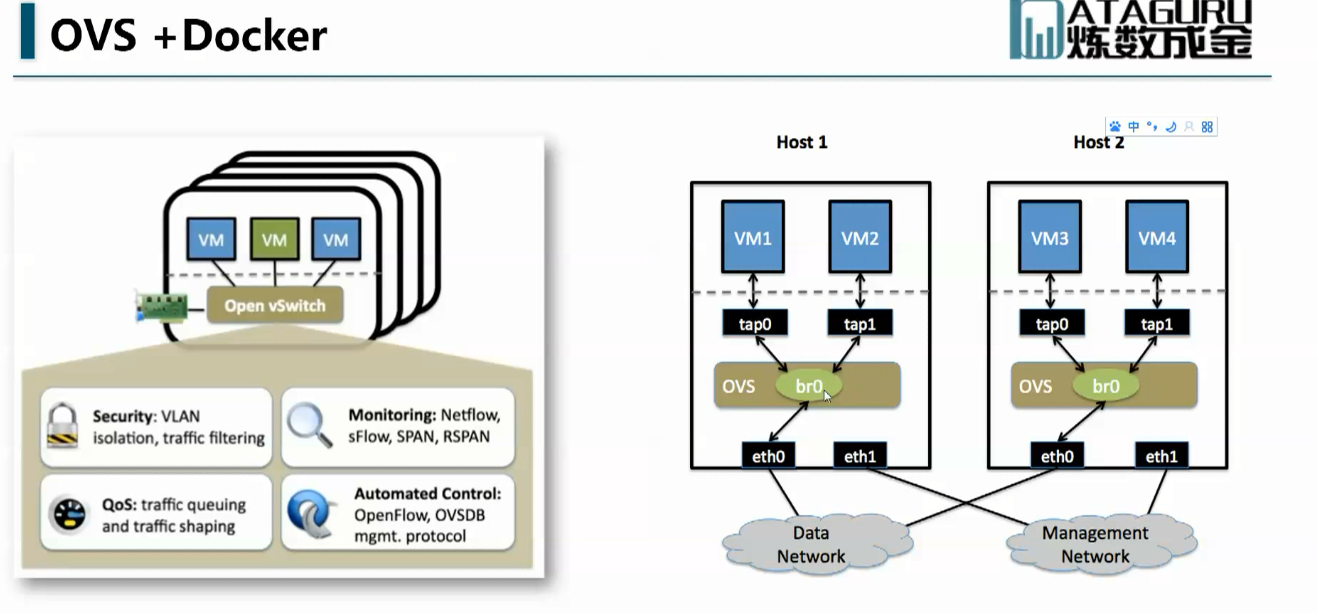
OVS ---- 就是把一个机器 虚拟化多个机器的时候 必须又网络方案的打通 --- 为了让多个虚拟机进行通信 ----- 那么这个交换机 就是 这个交换机必须是软件实现的 ---- 所以 开源的就是OVS ---- 是一种非常强大的交换机

支持很多 VLAN 流量整形 高级检控sflow 等等 高端的交换机上才有 --- 这个ovs是很高端 功能很强大的软件交换机 希望大家多多了解ovs 这个也是openstack里面的难点

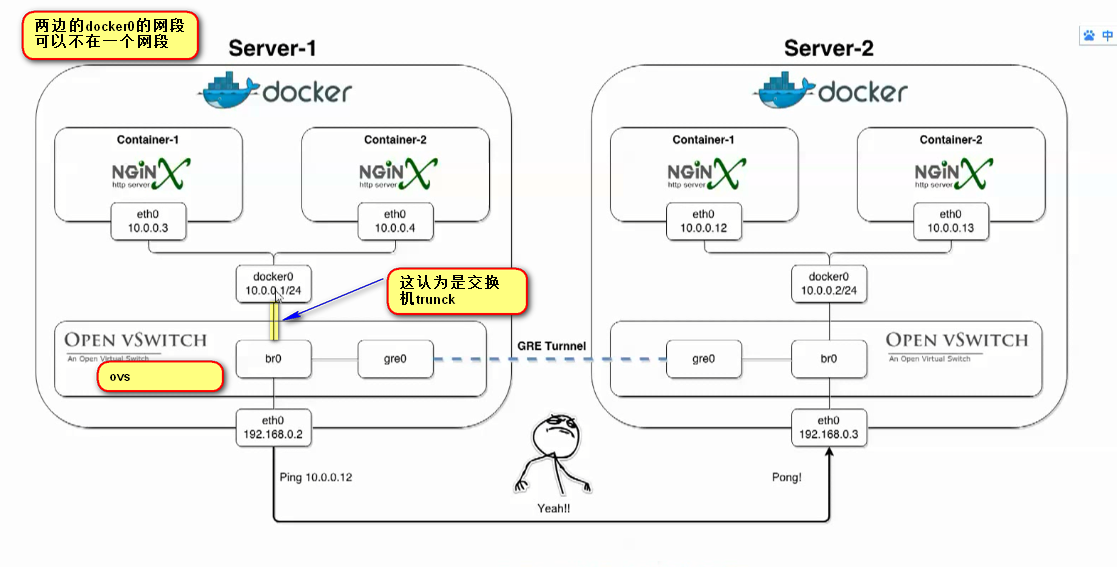
加了ovs之后 docker网络就变化了 ------ docker container 就看成VM 是没问题的

图中的ovs的br0 相当于之前讲的docker0 ------- **只要连接到ovs br0上面的vm 就算打通了**

Ovs提供的协议支持 可以支持gre vxlan等虚拟化方式 所以 多台机器的的OVS就可以通信了 ---- 也不一定通过隧道 也可以通过物理网络



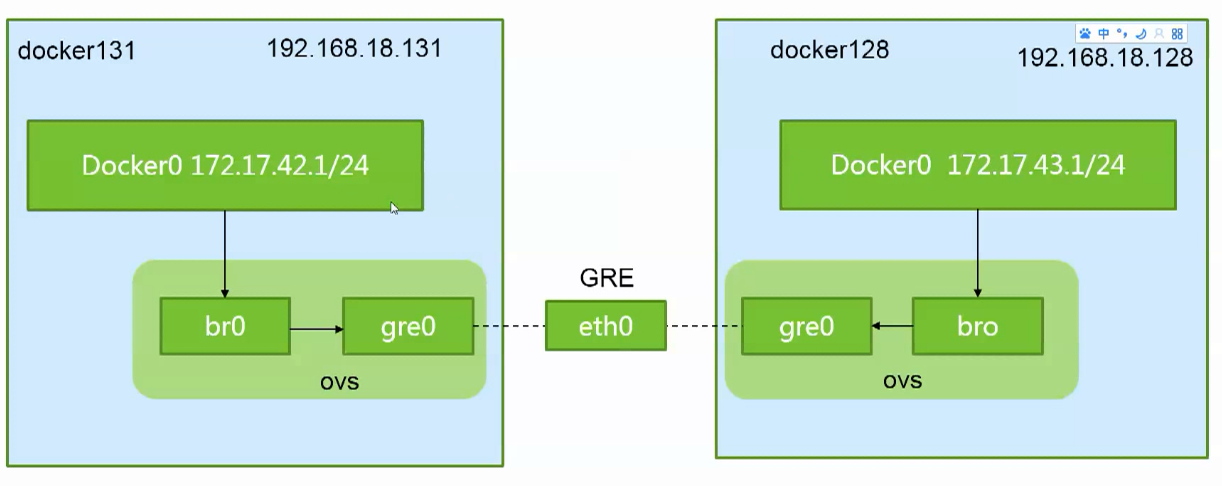
OVS下面的一个docker



Docker0和ovs都看做交换机 交换机之间通过trunck口连接 ------ ovs上会连接一个gre通到 ---- 虚线是因为这不是一个真实的路径 -------- 实际上还是通过eht0网卡出去 --- 原理上可以认为是通过gre0 tunnel出去的 --- gre是一个封装协议

可以和OpenStack的环境打通 因为ovs是标准的

这个是同一个子网的



这两个的里面不是同一组子网 和kubernetes的网络是一样的 ---kubernets网络 每一个端口是一个子网

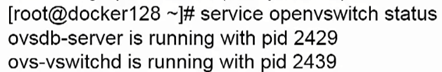
这样每个docker0上面各有254个空间

这个可以实现kubernetes的网络 两个不同的子网数据交换只能通过路由 三层 ----- 同一子网 走两层的 走mac ---------- linux支持路由转发 所以 主机可以配置

这个实践 先把一台机器配置好 装上ovs 调试好 安装好 搞好之后 进行clone ----- 重新或者改掉ip地址 这时候可以做试验了

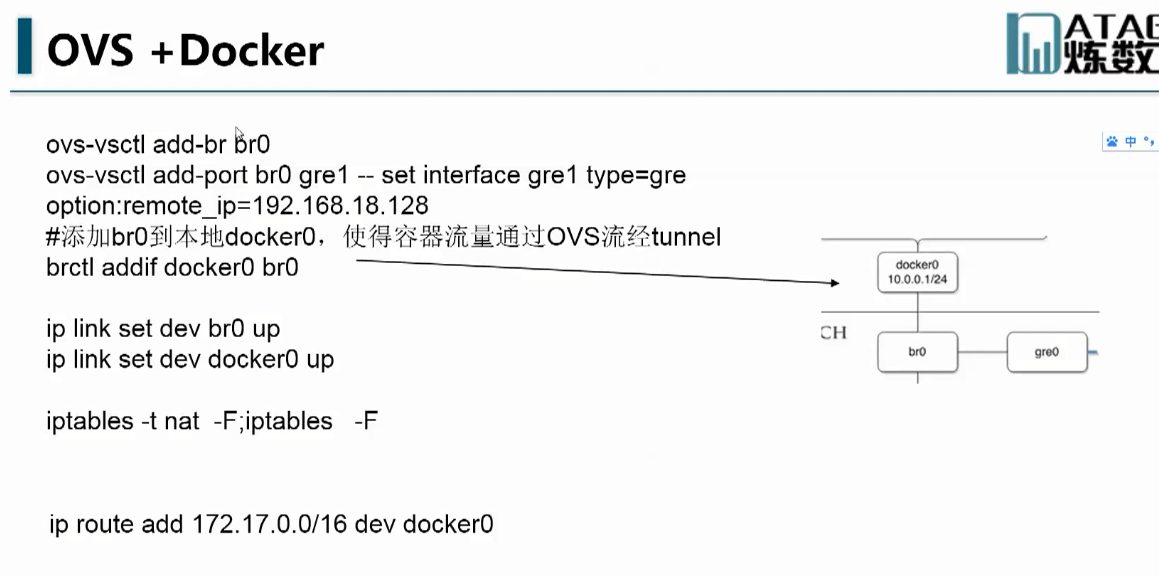
Docker默认不改

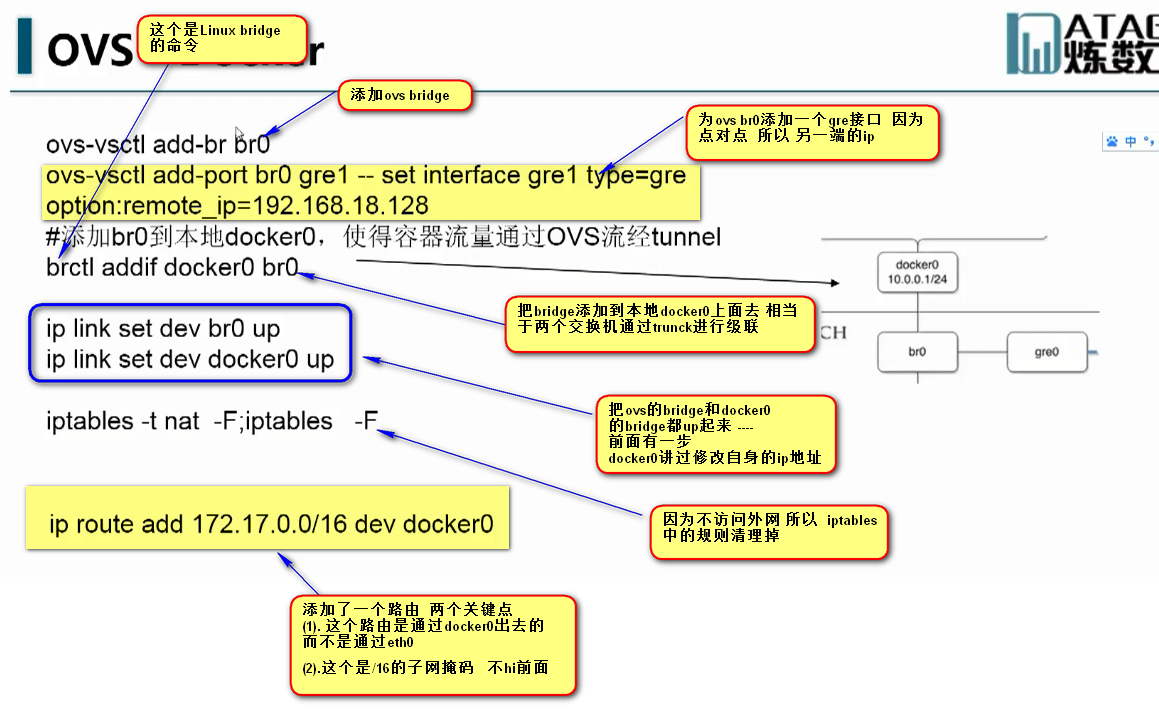


这话i兽可以看到两格ovs进程 

一个是server 另一个是vswitched

设置ovs





定义gre通道时候需要知道类型 和 对端 ----- 因为grep点对点 如果两个对端 就要有两格gre通道

所以 第二条命令写的是

**Ovs-vsctl add-port br0 gre1 – set interface gre1 type=gre option:remote\_ip=192.168.18.128**

对于 ip route add 172.17.0.0/16 dev docker0

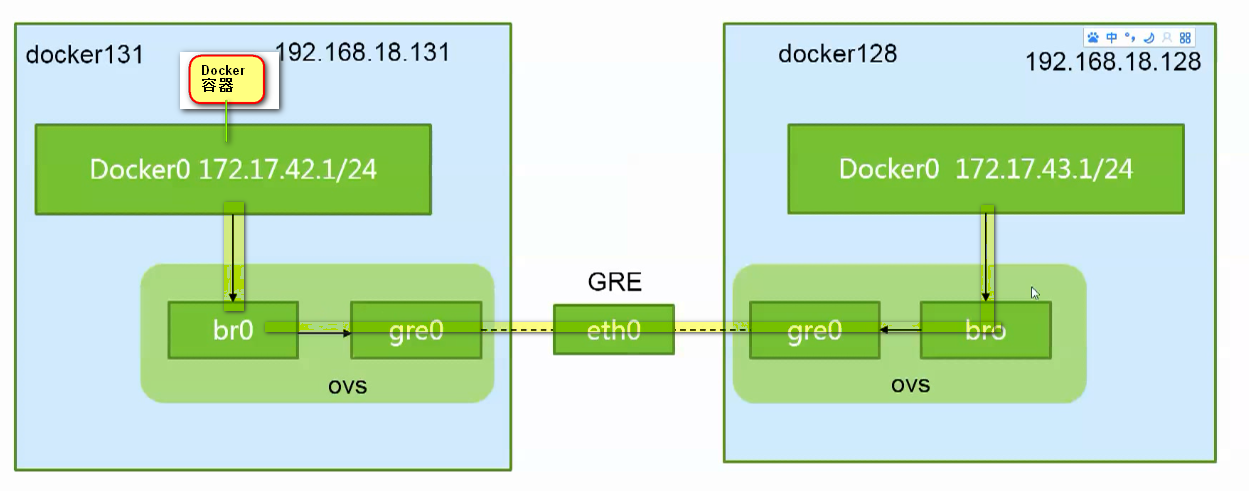
【





那么 上面 ip route add 172.17.0.0/16 dev docker0 含义就是 发往172.17.0.0/16的数据 都经过docker0做转发--- 添加了一条静态路由】

42.1/24和43.1/24都是在docker上面的 --- 16就是这个子网扩大了



这样 因为 子网掩码扩大到16 ----- 这样 就直接让两边原本不是统一个子网的docker0 可以路由了 ------ 到对端也是 通过docker0通过gre出去的 所以 没有定义成eth0 --- 而是docker0出去 【注意 **不是一个网段 所以 通信 必须有路由**】

172.17.42.1/24和172.17.43.1/24 这两个小网段 需要合并成一个大网段 就是 172.17.0.0/16

这个就是 只要是访问172.17.0.0/16这个网络的 无论是本机的 还是对端的 都走docker0

【通过下面的例子 就可以理解

在docker128这台宿主机上ping对端docker130上面的docker0 的ip 是通的

\*\*就是 对于本机docker container 在没有上面那条路由命令的时候 实际上是可以直接按照 docker0 --- ovs bridge – eth0 –》eth0– ovs br – docker0 直接就过去了 gre tunnel

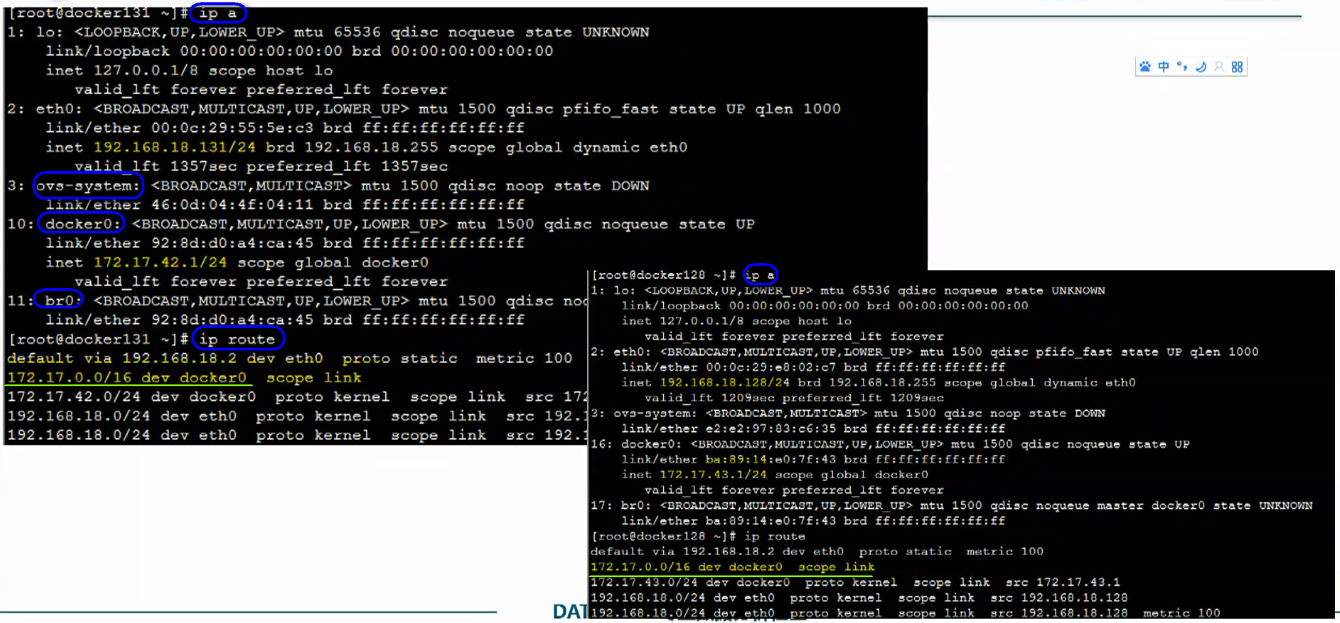
**实际上 前面也说过了 gre tunnel是逻辑的 实际上 也是封装成gre报文 发给eth0然后才发出去的**

\*\*但是 对于宿主机上面 192.168.18.131这个 想ping通本机的docker container或者另一台机器的Docker-container 那就是完全不同的两个网段 不同的网段发送数据 必须有路由 所以 必须添加静态路由表 ----- 这里面 就借助了虚拟的docker0作为转发的路由器来做这件事

如果 对于docker128宿主机上面的信息 要发给另一台机器 借助docker0的通道 ---- 可以写成 ip route add 172.17.43.1/24 dev docker0 要是访问另一台机器的docker0 写法还是一样的 ip route add 172.17.42.1/24 dev docker0 ----- 两条指令合并起来 就是

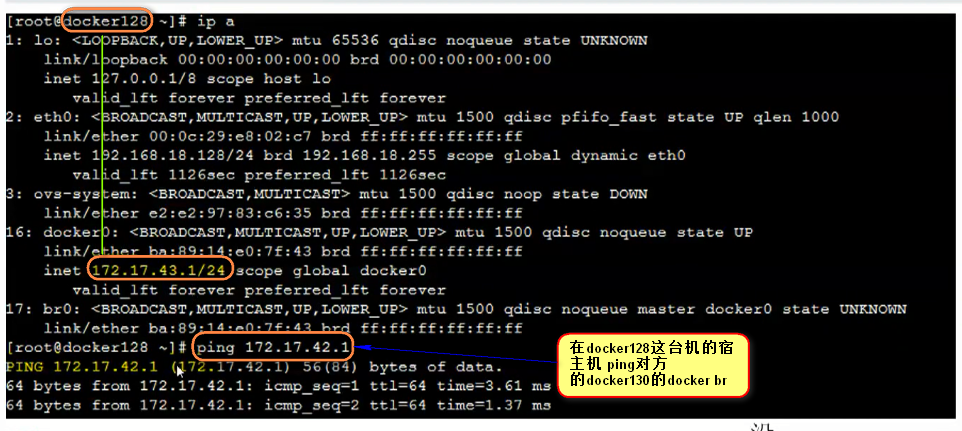
ip route add 172.17.0.0/16 dev docker0 ----------- 这样 两台虚拟机 各自执行同一条命令就好了 相当于添加了两条路由信息】

设计好之后 通了



----- 继续看

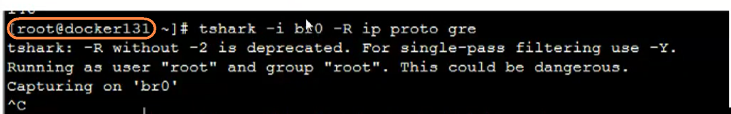
互相ping



Docker128主机上面的信息 ping对端的docker0 借助本端的docker0 走gre tunnel 发送到另一端的docker0

【但是 docker128 和docker130 是通一个网段 所以 通过本机的eth0就可以完成通信工作了】

抓包分析一下这个过程



在docker131上面 抓包分析 br0 ---因为br0 和docker0是联通的 ----

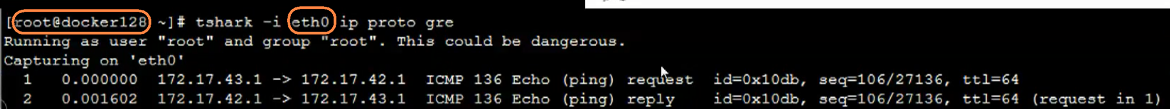
在br0上面 看看有没有gre的协议 但是 没有报文

就是gre这个报文并没有在ovs和docker0的网卡上出现【ovs上有一个Br0 然后 br0和docker0联通】【这块不理解】

【tshark option

-i 设置抓包的网络接口，不设置则默认为第一个非自环接口。

-R 设置读取过滤表达式（read filter expression）。不符合此表达式的流量同样不会被写入文件。注意，读取（显示）过滤表达式的语法和底层相关的抓包过滤表达式语法不相同，它的语法表达要丰富得多，请参考http://www.ethereal.com/docs/dfref/和http://www.ethereal.com/docs/man-pages/ethereal-filter.4.html。类似于抓包过滤表达式，在命令行使用时最好将它们quote起来。-Y （显示）过滤。】



【为什么只有eth0有 但是 对面的br0是没有的 。。。。？？？】