- 建模
- 性能和可伸缩性
- 领域驱动设计
- AOP
- 设计模式
- 安全
- 云计算
- SOA

特别专题 架构 & 设计

手机百度云端架构设计与实践 —— 手机百度“云和端技术实践”沙龙

手机百度是诞生于移动互联网时代的一个超级APP，是中国用户最流行的搜索客户端，拥有巨大的活跃用户群。手机百度云端架构支撑了其庞大的自建业务，在高并发、大数据的情况下保持了高可用性和可靠性，并支撑着快速的迭代和业务的快速扩展，解决了同时兼容了客户端双平台、若干个历史版本。手机百度云端在PHP架构对超级APP的支持方面具有领先技术。 本题目引导传统PHP工程师开拓眼界、提升能力。从传统Web系统工程...

浏览所有 架构 & 设计

- 过程 & 实践
  - Agile
  - 领导能力
  - 团队协作
  - 敏捷技术
  - 方法论
  - 持续集成
  - 精益
  - 客户及需求

特别专题 过程 & 实践

腾讯最赚钱的部门是怎么做运维的|大牛V课堂

Geekbang带领大家深度了解腾讯互娱（游戏）的运维文化、体系建设、人员培养，值得中高端技术人群收藏阅读。

浏览所有 过程 & 实践

- 运维 & 基础架构
  - 性能和可伸缩性
  - 大数据
  - DevOps
  - 云计算
  - 虚拟化
  - NoSQL
  - 应用服务器

- 运维

特别专题 运维 & 基础架构

使用Neo4j进行全栈Web开发

在开发一个全栈web应用时，你可以在多种数据库之间进行选择。在本文中，作者将展开讨论，当你的数据模型中包含大量关联数据与关系时，Neo4j图形数据库是作为数据存储系统的一个良好选择。

浏览所有 运维 & 基础架构

- 企业架构
  - 企业架构
  - 业务流程建模
  - 业务/IT整合
  - Integration (EAI)
  - 治理
  - Web 2.0
  - SOA

特别专题 企业架构

使用Neo4j进行全栈Web开发

在开发一个全栈web应用时，你可以在多种数据库之间进行选择。在本文中，作者将展开讨论，当你的数据模型中包含大量关联数据与关系时，Neo4j图形数据库是作为数据存储系统的一个良好选择。

浏览所有 企业架构

- 移动
- Docker
- 开源
- 云计算
- 大数据
- 架构师
- 安全
- AWS
- QCon
- ArchSummit
- UnitedStack

Docker网络详解及pipework源码解读与实践

作者 冯明振 发布于 2015年1月14日 |

Docker作为目前最火的轻量级容器技术，有很多令人称道的功能，如Docker的镜像管理。然而，Docker同样有着很多不完善的地方，网络方面就是Docker比较薄弱的部分。因此，我们有必要深入了解Docker的网络知识，以满足更高的网络需求。本文首先介绍了Docker自身的4种网络工作方式，然后通过3个样例 —— 将Docker容器配置到本地网络环境中、单主机Docker容器的VLAN划分、多主机Docker容器的VLAN划分，演示了如何使用pipework帮助我们进行复杂的网络设置，以及pipework是如何工作的。

1. Docker的4种网络模式

我们在使用docker run创建Docker容器时，可以用--net选项指定容器的网络模式，Docker有以下4种网络模式：

- host模式，使用--net=host指定。
- container模式，使用--net=container:NAME_or_ID指定。
- none模式，使用--net=none指定。
- bridge模式，使用--net=bridge指定，默认设置。

下面分别介绍一下Docker的各个网络模式。

1.1 host模式

众所周知，Docker使用了Linux的Namespaces技术来进行资源隔离，如PID Namespace隔离进程，Mount Namespace隔离文件系统，Network Namespace隔离网络等。一个Network Namespace提供了一份独立的网络环境，包括网卡、路由、Iptable规则等都与其他的Network Namespace隔离。一个Docker容器一般会分配一个独立的Network Namespace。但如果启动容器的时候使用host模式，那么这个容器将不会获得一个独立的Network Namespace，而是和宿主机共用一个Network Namespace。容器将不会虚拟出自己的网卡，配置自己的IP等，而是使用宿主机的IP和端口。

例如，我们在10.10.101.105/24的机器上用host模式启动一个含有web应用的Docker容器，监听

tcp80端口。当我们在容器中执行任何类似ifconfig命令查看网络环境时，看到的都是宿主机上的信息。而外界访问容器中的应用，则直接使用10.10.101.105:80即可，不用任何NAT转换，就如直接跑在宿主机中一样。但是，容器的其他方面，如文件系统、进程列表等还是和宿主机隔离的。

1.2 container模式

在理解了host模式后，这个模式也就好理解了。这个模式指定新创建的容器和已经存在的一个容器共享一个Network Namespace，而不是和宿主机共享。新创建的容器不会创建自己的网卡，配置自己的IP，而是和一个指定的容器共享IP、端口范围等。同样，两个容器除了网络方面，其他的如文件系统、进程列表等还是隔离的。两个容器的进程可以通过lo网卡设备通信。

1.3 none模式

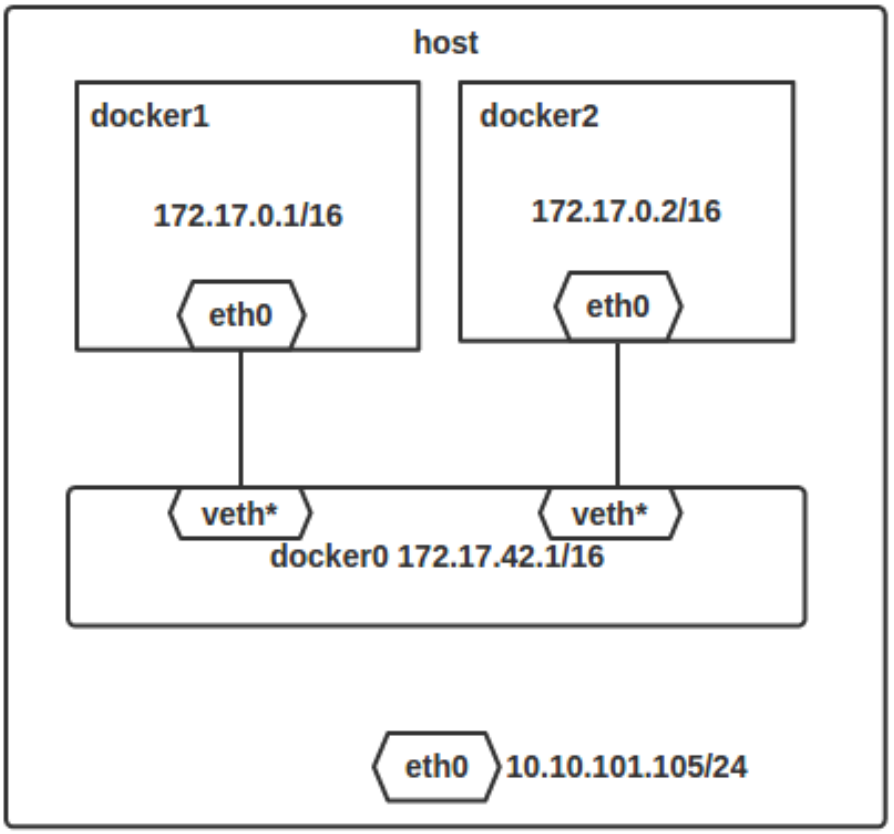
这个模式和前两个不同。在这种模式下，Docker容器拥有自己的Network Namespace，但是，并不为Docker容器进行任何网络配置。也就是说，这个Docker容器没有网卡、IP、路由等信息。需要我们自己为Docker容器添加网卡、配置IP等。

1.4 bridge模式

bridge模式是Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将一个主机上的Docker容器连接到一个虚拟网桥上。下面着重介绍一下此模式。

1.4.1 bridge模式的拓扑

当Docker server启动时，会在主机上创建一个名为docker0的虚拟网桥，此主机上启动的Docker容器会连接到这个虚拟网桥上。虚拟网桥的工作方式和物理交换机类似，这样主机上的所有容器就通过交换机连在了一个二层网络中。接下来就要为容器分配IP了，Docker会从RFC1918所定义的私有IP网段中，选择一个和宿主机不同的IP地址和子网分配给docker0，连接到docker0的容器就从这个子网中选择一个未占用的IP使用。如一般Docker会使用172.17.0.0/16这个网段，并将172.17.42.1/16分配给docker0网桥（在主机上使用ifconfig命令是可以看到docker0的，可以认为它是网桥的管理接口，在宿主机上作为一块虚拟网卡使用）。单机环境下的网络拓扑如下，主机地址为10.10.101.105/24。



Docker完成以上网络配置的过程大致是这样的：

1. 在主机上创建一对虚拟网卡veth pair设备。veth设备总是成对出现的，它们组成了一个数据的通道，数据从一个设备进入，就会从另一个设备出来。因此，veth设备常用来连接两个网络设备。

2. Docker将veth pair设备的一端放在新创建的容器中，并命名为eth0。另一端放在主机中，以veth65f9这样类似的名字命名，并将这个网络设备加入到docker0网桥中，可以通过brctl show命令查看。

```
fmzhen@fmzhen-OptiPlex-3020:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.56847afe9799       no              veth4ddd3ee
                                                        vethcfd55de
```

3. 从docker0子网中分配一个IP给容器使用，并设置docker0的IP地址为容器的默认网关。

网络拓扑介绍完后，接着介绍一下bridge模式下容器是如何通信的。

1.4.2 bridge模式下容器的通信

在bridge模式下，连在同一网桥上的容器可以相互通信（若出于安全考虑，也可以禁止它们之间通信，方法是在DOCKER_OPTS变量中设置--icc=false，这样只有使用--link才能使两个容器通信）。

容器也可以与外部通信，我们看一下主机上的Iptable规则，可以看到这么一条

```
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
```

这条规则会将源地址为172.17.0.0/16的包（也就是从Docker容器产生的包），并且不是从docker0网卡发出的，进行源地址转换，转换成主机网卡的地址。这么说可能不太好理解，举一个例子说明一下。假设主机有一块网卡为eth0，IP地址为10.10.101.105/24，网关为10.10.101.254。从主机上一个IP为172.17.0.1/16的容器中ping百度（180.76.3.151）。IP包首先从容器发往自己的默认网关

docker0，包到达docker0后，也就到达了主机上。然后会查询主机的路由表，发现包应该从主机的eth0发往主机的网关10.10.105.254/24。接着包会转发给eth0，并从eth0发出去（主机的ip_forward转发应该已经打开）。这时候，上面的Iptable规则就会起作用，对包做SNAT转换，将源地址换为eth0的地址。这样，在外界看来，这个包就是从10.10.101.105上发出来的，Docker容器对外是不可见的。

那么，外面的机器是如何访问Docker容器的服务呢？我们首先用下面命令创建一个含有web应用的容器，将容器的80端口映射到主机的80端口。

```
docker run -d --name web -p 80:80 fmzhen/simpleweb
```

然后查看Iptable规则的变化，发现多了这样一条规则：

```
-A DOCKER ! -i docker0 -p tcp -m tcp --dport 80 -j DNAT --to-destination 172.17.0.5:80
```

此条规则就是对主机eth0收到的目的端口为80的tcp流量进行DNAT转换，将流量发往172.17.0.5:80，也就是我们上面创建的Docker容器。所以，外界只需访问10.10.101.105:80就可以访问到容器中得服务。

除此之外，我们还可以自定义Docker使用的IP地址、DNS等信息，甚至使用自己定义的网桥，但是其工作方式还是一样的。

2. pipework的使用以及源码分析

Docker自身的网络功能比较简单，不能满足很多复杂的应用场景。因此，有很多开源项目用来改善Docker的网络功能，如pipework、weave、flannel等。这里，就先介绍一下pipework的使用和工作原理。

pipework是由Docker的工程师Jérôme Petazzoni开发的一个Docker网络配置工具，由200多行shell实现，方便易用。下面用三个场景来演示pipework的使用和工作原理。

2.1 将Docker容器配置到本地网络环境中

为了使本地网络中的机器和Docker容器更方便的通信，我们经常会有将Docker容器配置到和主机同一网段的需求。这个需求其实很容易实现，我们只要将Docker容器和主机的网卡桥接起来，再给Docker容器配上IP就可以了。

下面我们来操作一下，我主机A地址为10.10.101.105/24,网关为10.10.101.254,需要给Docker容器的地址配置为10.10.101.150/24。在主机A上做如下操作：

```
#安装pipework
git clone https://github.com/jpetazzo/pipework
cp ~/pipework/pipework /usr/local/bin/
#启动Docker容器。
docker run -itd --name test1 ubuntu /bin/bash
#配置容器网络，并连到网桥br0上。网关在IP地址后面加@指定。
#若主机环境中存在dhcp服务器，也可以通过dhcp的方式获取IP
#pipework br0 test1 dhcp
pipework br0 test1 10.10.101.150/24@10.10.101.254
#将主机eth0桥接到br0上，并把eth0的IP配置在br0上。这里由于是远程操作，中间网络会断掉，所以放在一条命令中执行。
ip addr add 10.10.101.105/24 dev br0; \
    ip addr del 10.10.101.105/24 dev eth0; \
    brctl addif br0 eth0; \
    ip route del default; \
    ip route add default gw 10.10.101.254 dev br0
```

完成上述步骤后，我们发现Docker容器已经可以使用新的IP和主机网络里的机器相互通信了。

pipework工作原理分析

那么容器到底发生了哪些变化呢？我们docker attach到test1上，发现容器中多了一块eth1的网卡，并且配置了10.10.101.150/24的IP，而且默认路由也改为了10.10.101.254。这些都是pipework帮我们配置的。通过查看源代码，可以发现pipework br0 test1 10.10.101.150/24@10.10.101.254是由以下命令完成的（这里只列出了具体执行操作的代码）。

```
#创建br0网桥
#若ovs开头，则创建OVS网桥 ovs-vsctl add-br ovs*
brctl addbr $IFNAME
#创建veth pair,用于连接容器和br0
ip link add name $LOCAL_IFNAME mtu $MTU type veth peer name $GUEST_IFNAME mtu $MTU
#找到Docker容器test1在主机上的PID,创建容器网络命名空间的软连接
DOCKERPID=$(docker inspect --format='{{ .State.Pid }}' $GUESTNAME)
ln -s /proc/$NSPID/ns/net /var/run/netns/$NSPID
#将veth pair一端放入Docker容器中，并设置正确的名字eth1
ip link set $GUEST_IFNAME netns $NSPID
ip netns exec $NSPID ip link set $GUEST_IFNAME name $CONTAINER_IFNAME
#将veth pair另一端加入网桥
#若为OVS网桥则为 ovs-vsctl add-port $IFNAME $LOCAL_IFNAME ${VLAN:+"tag=$VLAN"}
brctl addif $IFNAME $LOCAL_IFNAME
#为新增加的容器配置IP和路由
ip netns exec $NSPID ip addr add $IPADDR dev $CONTAINER_IFNAME
ip netns exec $NSPID ip link set $CONTAINER_IFNAME up
ip netns exec $NSPID ip route delete default
ip netns exec $NSPID ip route add $GATEWAY/32 dev $CONTAINER_IFNAME
```

1. 首先pipework检查是否存在br0网桥，若不存在，就自己创建。若以"ovs"开头，就会创建OpenVswitch网桥，以"br"开头，创建Linux bridge。
2. 创建veth pair设备，用于为容器提供网卡并连接到br0网桥。
3. 使用docker inspect找到容器在主机中的PID，然后通过PID将容器的网络命名空间链接到/var/run/netns/目录下。这么做的目的是，方便在主机上使用ip netns命令配置容器的网络。因为，在Docker容器中，我们没有权限配置网络环境。
4. 将之前创建的veth pair设备分别加入容器和网桥中。在容器中的名称默认为eth1，可以通过pipework的-i参数修改该名称。
5. 然后就是配置新网卡的IP。若在IP地址的后面加上网关地址，那么pipework会重新配置默认路由。这样容器通往外网的流量会经由新配置的eth1出去，而不是通过eth0和docker0。(若想完全抛弃自带的网络设置，在启动容器的时候可以指定--net=none)

以上就是pipework配置Docker网络的过程，这和Docker的bridge模式有着相似的步骤。事实上，Docker在实现上也采用了相同的底层机制。

通过源代码，可以看出，pipework通过封装Linux上的ip、brctl等命令，简化了在复杂场景下对容器连接的操作命令，为我们配置复杂的网络拓扑提供了一个强有力的工具。当然，如果想了解底层的操作，我们也可以直接使用这些Linux命令来完成工作，甚至可以根据自己的需求，添加额外的功能。

2.2 单主机Docker容器VLAN划分

pipework不仅可以使用Linux bridge连接Docker容器，还可以与OpenVswitch结合，实现Docker容器的VLAN划分。下面，就来简单演示一下，在单机环境下，如何实现Docker容器间的二层隔离。

为了演示隔离效果，我们将4个容器放在了同一个IP网段中。但实际他们是二层隔离的两个网络，有不同的广播域。

```
#在主机A上创建4个Docker容器，test1、test2、test3、test4
docker run -itd --name test1 ubuntu /bin/bash
docker run -itd --name test2 ubuntu /bin/bash
docker run -itd --name test3 ubuntu /bin/bash
docker run -itd --name test4 ubuntu /bin/bash
#将test1，test2划分到一个vlan中，vlan在mac地址后加@指定，此处mac地址省略。
pipework ovs0 test1 192.168.0.1/24 @100
pipework ovs0 test2 192.168.0.2/24 @100
#将test3，test4划分到另一个vlan中
pipework ovs0 test3 192.168.0.3/24 @200
```

```
pipework ovs0 test4 192.168.0.4/24 @200
```

完成上述操作后，使用docker attach连到容器中，然后用ping命令测试连通性，发现test1和test2可以相互通信，但与test3和test4隔离。这样，一个简单的VLAN隔离容器网络就已经完成。

由于OpenVswitch本身支持VLAN功能，所以这里pipework所做的工作和之前介绍的基本一样，只不过将Linux bridge替换成了OpenVswitch，在将veth pair的一端加入ovs0网桥时，指定了tag。底层操作如下：

```
ovs-vsctl add-port ovs0 veth* tag=100
```

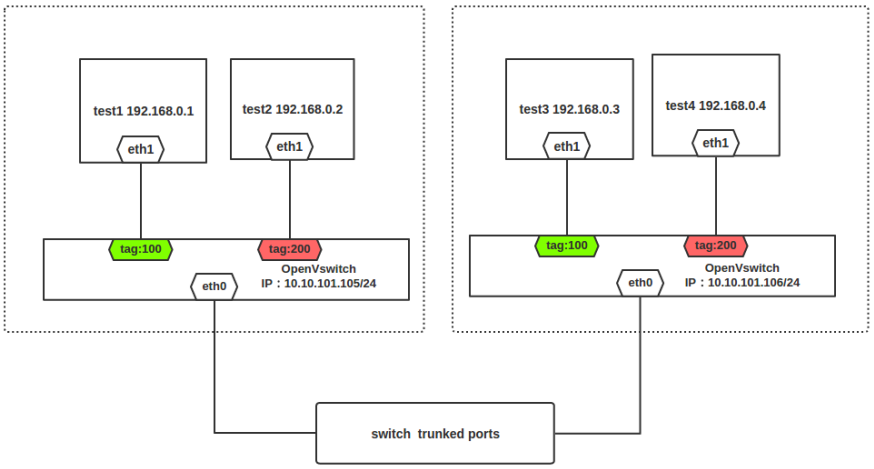
2.3 多主机Docker容器的VLAN划分

上面介绍完了单主机上VLAN的隔离，下面我们将情况延伸到多主机的情况。有了前面两个例子做铺垫，这个也就不难了。为了实现这个目的，我们把宿主机上的网卡桥接到各自的OVS网桥上，然后再为容器配置IP和VLAN就可以了。我们实验环境如下，主机A和B各有一块网卡eth0，IP地址分别为10.10.101.105/24、10.10.101.106/24。在主机A上创建两个容器test1、test2，分别在VLAN 100和VLAN 200上。在主机B上创建test3、test4，分别在VLAN 100和VLAN 200 上。最终，test1可以和test3通信，test2可以和test4通信。

```
#在主机A上
#创建Docker容器
docker run -itd --name test1 ubuntu /bin/bash
docker run -itd --name test2 ubuntu /bin/bash
#划分VLAN
pipework ovs0 test1 192.168.0.1/24 @100
pipework ovs0 test2 192.168.0.2/24 @200
#将eth0桥接到ovs0上
ip addr add 10.10.101.105/24 dev ovs0; \
    ip addr del 10.10.101.105/24 dev eth0; \
    ovs-vsctl add-port ovs0 eth0; \
    ip route del default; \
    ip route add default gw 10.10.101.254 dev ovs0

#在主机B上
#创建Docker容器
docker run -itd --name test3 ubuntu /bin/bash
docker run -itd --name test4 ubuntu /bin/bash
#划分VLAN
pipework ovs0 test1 192.168.0.3/24 @100
pipework ovs0 test2 192.168.0.4/24 @200
#将eth0桥接到ovs0上
ip addr add 10.10.101.106/24 dev ovs0; \
    ip addr del 10.10.101.106/24 dev eth0; \
    ovs-vsctl add-port ovs0 eth0; \
    ip route del default; \
    ip route add default gw 10.10.101.254 dev ovs0
```

完成上面的步骤后，主机A上的test1和主机B上的test3容器就划分到了一个VLAN中，并且与主机A上的test2和主机B上的test4隔离（主机eth0网卡需要设置为混杂模式，连接主机的交换机端口应设置为trunk模式，即允许VLAN 100和VLAN 200的包通过）。拓扑图如下所示（省去了Docker默认的eth0网卡和主机上的docker0网桥）：



除此之外，pipework还支持使用macvlan设备、设置网卡MAC地址等功能。不过，pipework有一个缺陷，就是配置的容器在关掉重启后，之前的设置会丢失。

3. 总结

通过上面的介绍，我相信大家对Docker的网络已经有了一定的了解。对于一个基本应用而言，Docker的网络模型已经很不错了。然而，随着云计算和微服务的兴起，我们不能永远停留在使用基本应用的级别上，我们需要性能更好且更灵活的网络功能。pipework正好满足了我们这样的需求，从上面的样例中，我们可以看到pipework的方便之处。但是，同时也应注意到，pipework并不是一套解决方案，它只是一个网络配置工具，我们可以利用它提供的强大功能，帮助我们构建自己的解决方案。

作者简介

冯明振，浙江大学SEL实验室硕士研究生，目前在云平台团队从事科研和开发工作。浙大团队对PaaS，Docker，大数据和主流开源云计算技术有深入的研究和二次开发经验，团队现将部分技术文章贡献出来，希望能对读者有所帮助。

感谢郭蕾对本文的策划和审校。

- 领域
- 架构 & 设计
- 语言 & 开发

- 专栏
- 开放源代码
- pipework
- 源代码
- 开源
- Docker
- 网络管理

相关内容

您好，朋友！

您需要 注册一个InfoQ账号 或者 登录 才能进行评论。在您完成注册后还需要进行一些设置。

获得来自InfoQ的更多体验。

告诉我们您的想法

允许的HTML标签: a,b,br,blockquote,i,li,pre,u,ul,p

当有人回复此评论时请E-mail通知我

发送信息

社区评论

哈哈 by 胡 涛 Posted
很好很详细 by 张 鑫 Posted
nice by Wang ST Posted
关于将Docker容器配置到本地网络环境中的问题 by weihua liu Posted
Re: 关于将Docker容器配置到本地网络环境中的问题 by Jiang James Posted

哈哈 by "胡 涛"

哈哈

- 回复
- 回到顶部

很好很详细 by "张 鑫"

正好需要。

- 回复
- 回到顶部

nice by "Wang ST"

通俗易懂

- 回复
- 回到顶部

关于将Docker容器配置到本地网络环境中的问题 by "weihua liu"

我按照上述描述的启动一个容器并安装pipework，并配置相应的IP。
但是容器不能ping通主机的网络。

环境描述：
virtualbox中安装了CENTOS7 X86_64操作系统
网卡方式是桥接，选用了Intel Pro/1000 MT 桌面 （82540EM）控制芯片，混杂模式是全部允许

主机网络是10.247.58.*
主机IP是10.247.58.141，网关是10.247.58.1
1首先下载镜像docker pull centos
2启动镜像docker run -t -i --net=none --name test centos /bin/bash
3 安装上述文章描述下载pipework
4 查看我的网卡信息 ifconfig -a，其中对外连接的网卡是enp0s8
5 执行以下命令

```
pipework br0 test 10.247.58.143/24@10.247.58.1
ip addr add 10.247.58.141/24 dev br0; \
ip addr del 10.10.101.105/24 dev enp0s8; \
brctl addif br0 enp0s8; \
ip route del default; \
route add default gw 10.247.58.1 dev br0
```

执行完后，登录容器，发现已经有一个IP是10.247.58.143的网卡了，但是只能ping通10.247.58.141主机，不能ping通主机网络网关以及互联网。

求解！

- 回复
- 回到顶部

Re: 关于将Docker容器配置到本地网络环境中的问题 by "Jiang James"

我也是同样的问题，求解啊

- 回复
- 回到顶部

关闭

by

发布于

- 查看
- 回复
- 回到顶部

关闭

引用原消息

主题 您的回复

允许的HTML标签: a,b,br,blockquote,i,li,pre,u,ul,p

当有人回复此评论时请E-mail通知我

发送信息 取消

关闭

主题 您的回复

允许的HTML标签: a,b,br,blockquote,i,li,pre,u,ul,p

当有人回复此评论时请E-mail通知我

取消

关闭

OK

5 讨论

- 热点内容
- 10天
- 40天
- 近6个月

我们是否应该把后端构建为API 16

Android开发周报：Flyme OS开源、经典开源项目解析

架构师（2015年7月）

开发者眼中的Spring与Java EE

解析微服务架构（二）微服务架构综述 11

Google将Material Design带到CSS、HTML与JavaScript上

JVM源码分析之FinalReference完全解读

腾讯最赚钱的部门是怎么做运维的|大牛V课堂 1

程序员的生产力始于需求而非工具 1

Airbnb如何设计机器学习模型

深度内容

- 全部
- 文章
- 演讲
- 访谈
- 迷你书

手机百度云端架构设计与实践 —— 手机百度“云和端技术实践”沙龙

黎博 7月17日

王栋：美团的智能化推荐

王栋 7月17日

感觉设计

马力 7月17日

面向服务的质量保障

黄小微 7月17日

手机百度云端架构设计与实践 —— 手机百度“云和端技术实践”沙龙

韩超 7月16日

庖丁解牛迭代器，聊聊那些藏在幕后的秘密

陈嘉栋 7月16日

- 更早的 >