docker四种网络模式

 李杰 发布于 2021-07-12

一.为什么要了解docker网络

当你开始大规模使用Docker时，你会发现需要了解很多关于网络的知识。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，然后介绍一些自定义网络模式。

二.docker 网络理论

Docker使用Linux桥接（参考《Linux虚拟网络技术》），在宿主机虚拟一个Docker容器网桥(docker0)，Docker启动一个容器时会根据Docker网桥的网段分配给容器一个IP地址，称为Container-IP，同时Docker网桥是每个容器的默认网关。因为在同一宿主机内的容器都接入同一个网桥，这样容器之间就能够通过容器的Container-IP直接通信。

Docker网桥是宿主机虚拟出来的，并不是真实存在的网络设备，外部网络是无法寻址到的，这也意味着外部网络无法通过直接Container-IP访问到容器。如果容器希望外部访问能够访问到，可以通过映射容器端口到宿主主机（端口映射），即docker run创建容器时候通过 -p 或 -P 参数来启用，访问容器的时候就通过 [宿主机IP]:[容器端口] 访问容器。

当你安装Docker时，它会自动创建三个网络。你可以使用以下 docker network ls 命令列出这些网络：

```
# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
857db65319fa        bridge              bridge              local
c16cf8722909        host                host                local
d39a88b56801        none                null                local
```

三.docker的四类网络模式

网络模式	配置	说明
bridge模式	--net=bridge	（默认为该模式）此模式会为每一个容器分配、设置IP等，并将容器连接到一个docker0虚拟网桥，通过docker0网桥以及Iptables nat表配置与宿主机通信。
host模式	--net=host	容器和宿主机共享Network namespace。
container模式	--net=container:NAME_or_ID	容器和另外一个容器共享Network namespace。kubernetes中的pod就是多个容器共享一个Network namespace。
none模式	--net=none	该模式关闭了容器的网络功能。

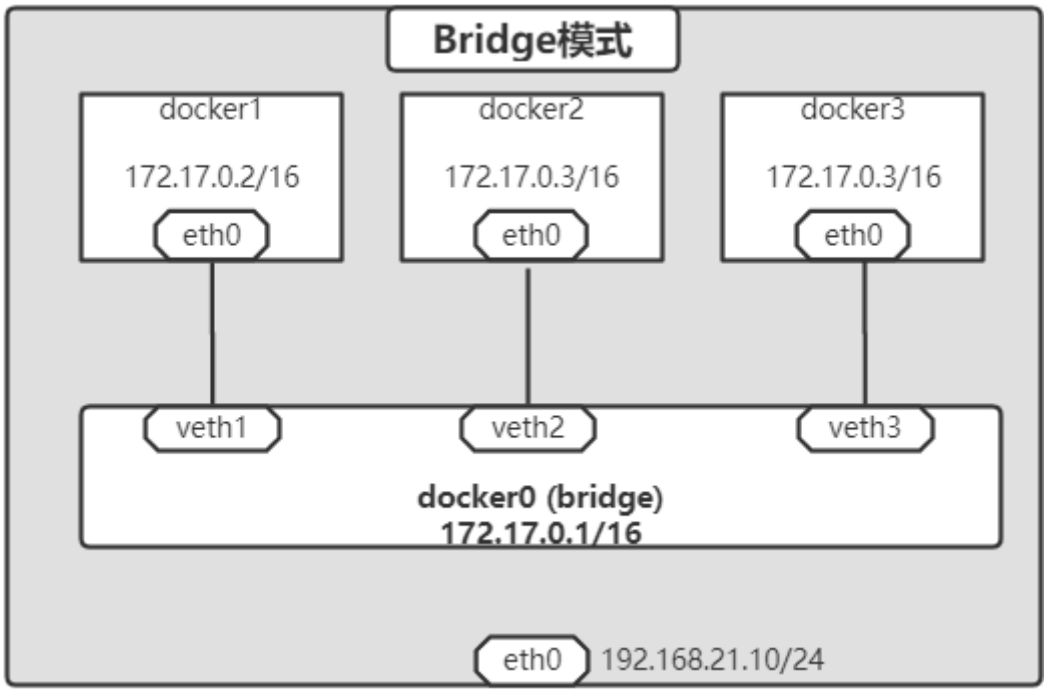
3.1 bridge模式

当Docker进程启动时，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。

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。在主机上创建一对虚拟网卡veth pair设备，Docker将veth pair设备的一端放在新创建的容器中，并命名为eth0（容器的网卡），另一端放在主机中，以vethxxx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。

bridge模式是docker的默认网络模式，不写--net参数，就是bridge模式。使用docker run -p时，docker实际是在iptables做了DNAT规则，实现端口转发功能。可以使用iptables -t nat -vnL查看。

bridge模式如下图所示：

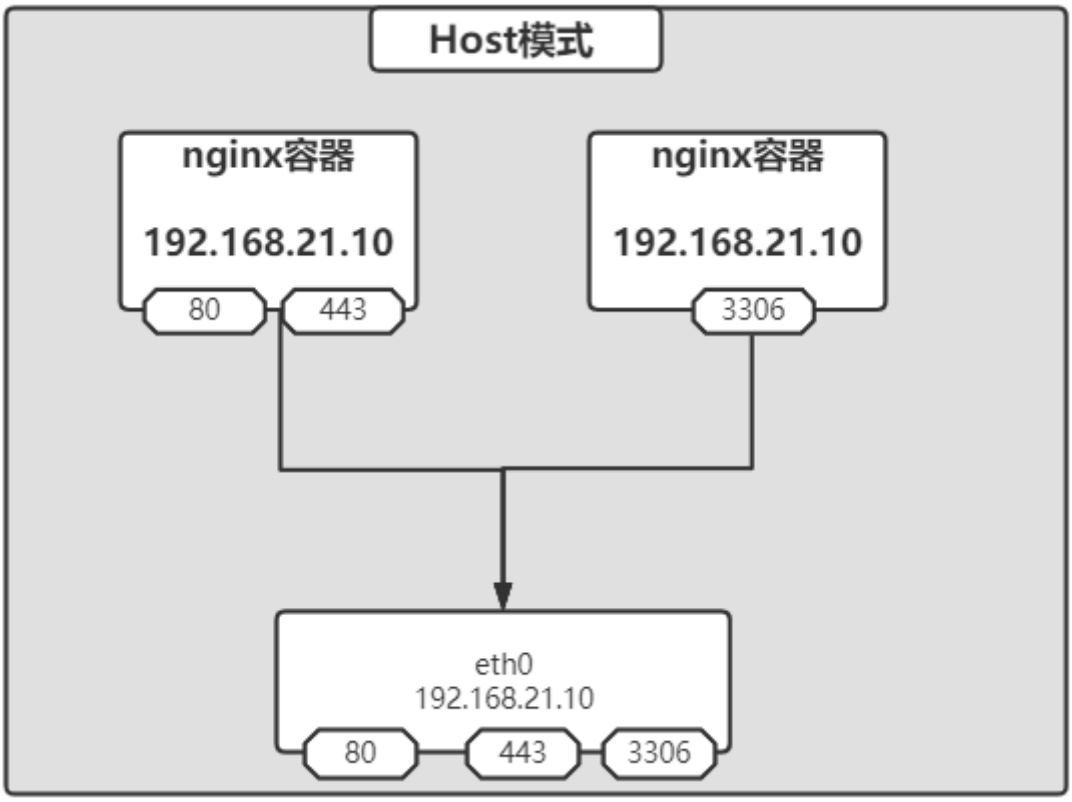


3.2 host模式

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。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。

使用host模式的容器可以直接使用宿主机的IP地址与外界通信，容器内部的服务端口也可以使用宿主机的端口，不需要进行NAT，host最大的优势就是网络性能比较好，但是docker host上已经使用的端口就不能再用了，网络的隔离性不好。

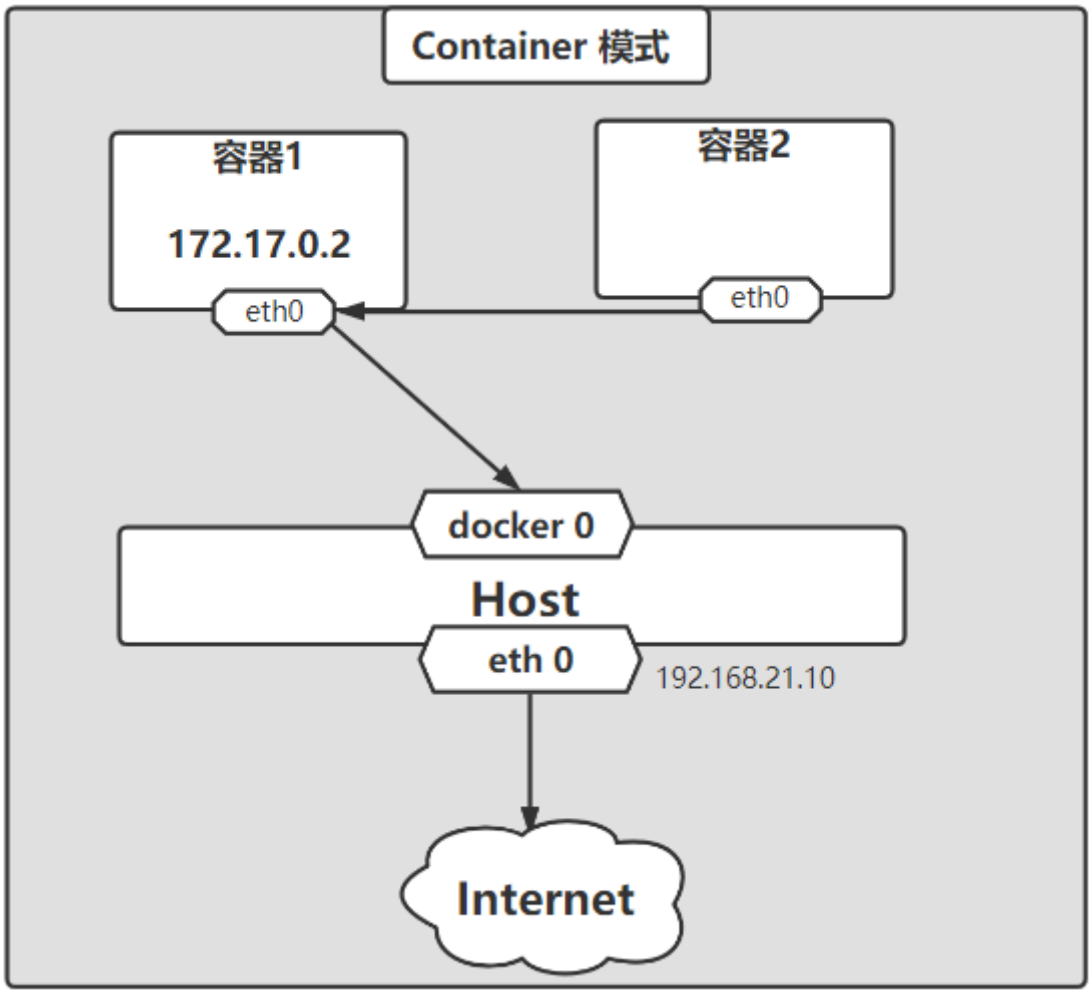
Host模式如下图所示：



3.3 container模式

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。

Container模式示意图：

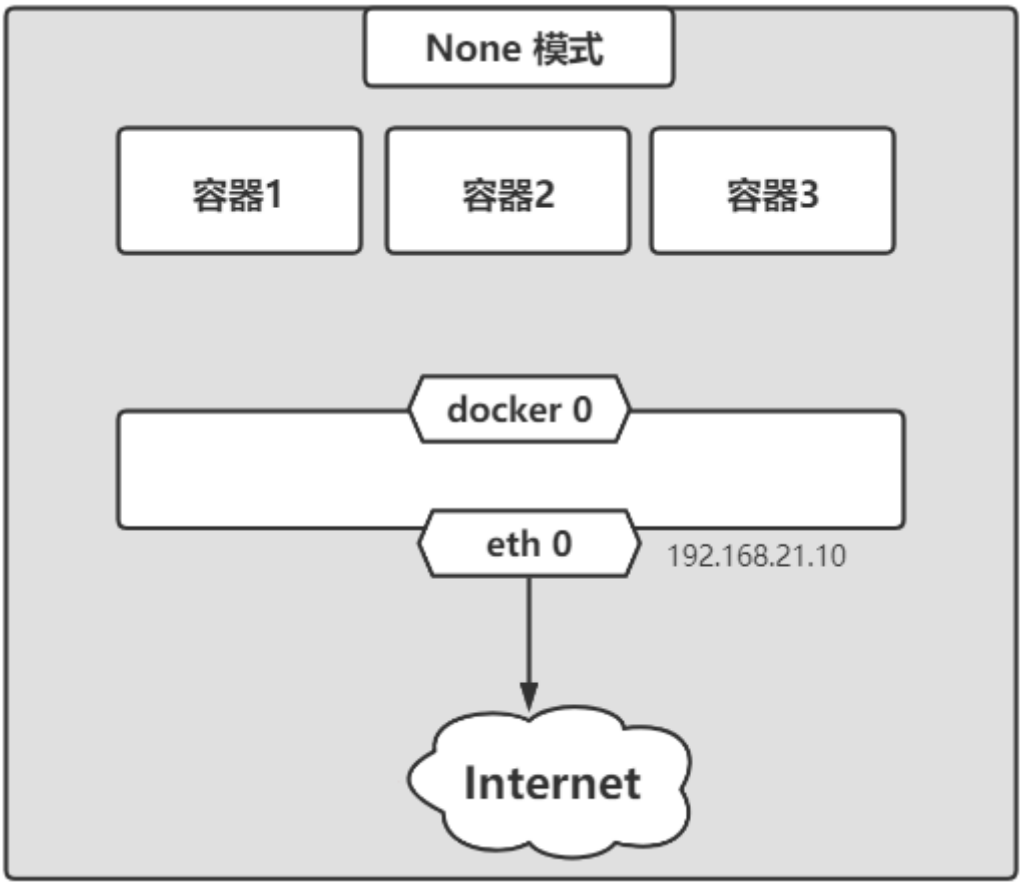


3.4 none模式

使用none模式，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。

这种网络模式下容器只有lo回环网络，没有其他网卡。none模式可以在容器创建时通过--network=none来指定。这种类型的网络没有办法联网，封闭的网络能很好的保证容器的安全性。

None模式示意图:



四.bridge模式下容器的通信

4.1 防火墙开启状态

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

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192.168.21.10/24，网关为192.168.21.255。从主机上一个IP为172.17.0.1/16的容器中ping百度（www.baidu.com）。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192.168.21.255/24。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192.168.21.10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 -itd --name=nginx_bridge --net=bridge -p 80:80 nginx
```

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 172.17.0.2:80
```

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172.17.0.2:80，也就是我们上面创建的Docker容器。所以，外界只需访问192.168.21.10:80就可以访问到容器中的服务。

4.2 防火墙关闭状态

如果docker网络使用了bridge模式，也不需要防火墙，那直接关掉FirewallD服务就可以了。可以解决诸多因为防火墙网络问题导致的docker容器端口不通的问题。

- docker网络官方文档：https://docs.docker.com/engin...

docker

阅读 6.2k • 发布于 2021-07-12