



Docker虚拟化容器应用与实战

第5章 Docker网络

本章目标

- 理解Docker 0
- --link
- 自定义网络
- 网络连通



理解Docker 0

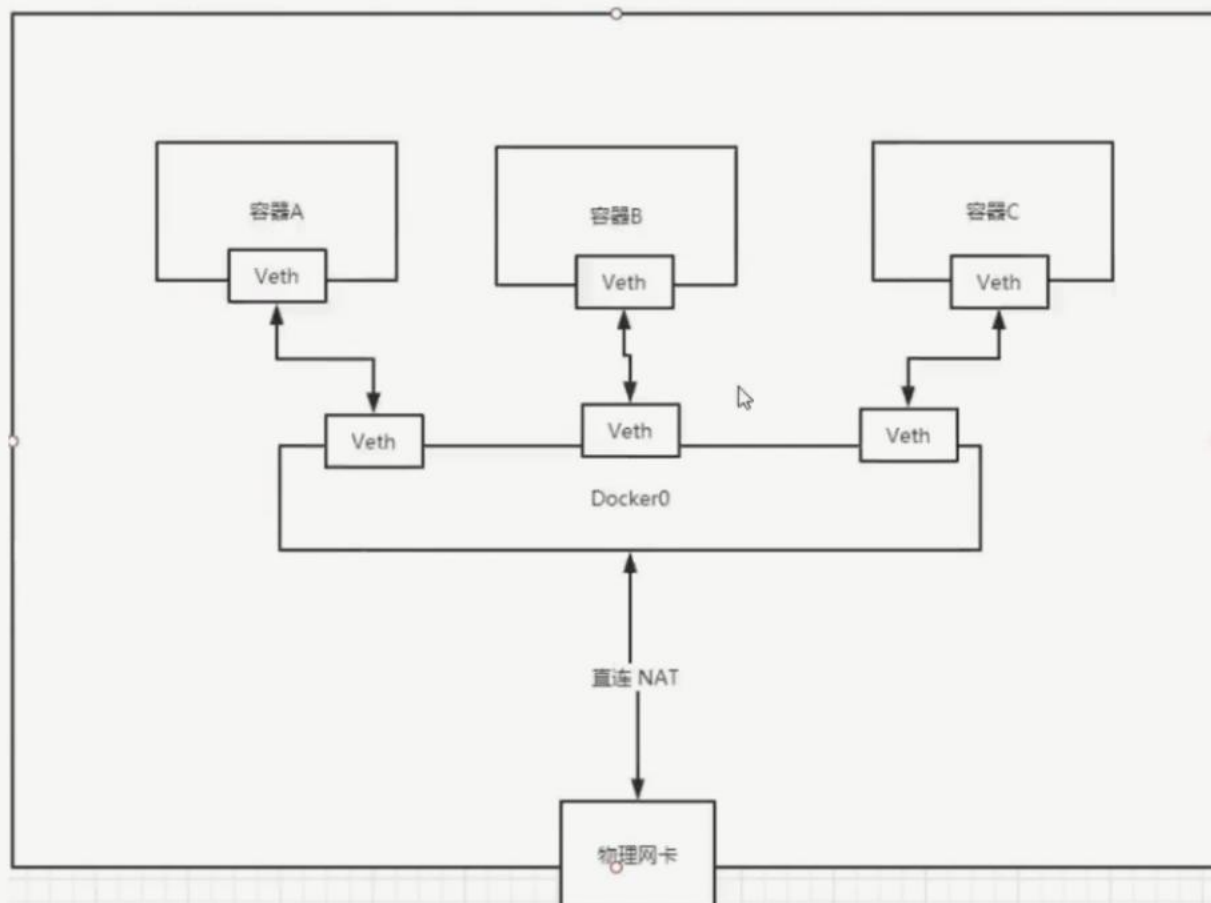
- 测试 运行一个tomcat
- `docker run -d --name tomcat01 tomcat`
- `ip addr`



- 1、我们每启动一个docker容器，docker就会给docker容器分配一个ip，我们只要安装了docker，就会有一个docker0桥接模式，使用的技术是veth-pair技术！
- 2、在启动一个容器测试，发现又多了一对网络，我们发现这个容器带来的网卡都是一对对的，veth-pair 就是一对虚拟设备接口，他们都是成对出现的，一端连着协议，一端彼此相连，正因为有veth-pair 充当一个桥梁，连接各种虚拟网络设备。 OpenStack,Docker容器之间的连接，OVS的连接，都是使用evth-pair技术
- 小结： Docker使用的是Linux的桥接，宿主机是一个Docker容器的网桥 docker0
- Docker中所有网络接口都是虚拟的，虚拟的转发效率高（内网传递文件）
- 只要容器删除，对应的网桥一对就没了！
- ****结论****： tomcat01和tomcat02公用一个路由器，docker0。所有的容器不指定网络的情况下，都是docker0路由的，docker会给我们的容器分配一个默认的可用ip



Docker原理



--link

- --link

探究: docker network inspect 网络id 网段相同

- docker inspect tomcat03



--link

- `docker exec -it tomcat02 ping tomca01` # ping不通ping: tomca01:
Name or service not known
- # 运行一个tomcat03 --link tomcat02
- `docker run -d -P --name tomcat03 --link tomcat02 tomcat` # 用tomcat03 ping
tomcat02 可以ping通
- `docker exec -it tomcat03 ping tomcat02`
- # 用tomcat02 ping tomcat03 ping不通



--link

```
→ docker-tomcat docker exec tomcat03 cat /etc/hosts  
127.0.0.1      localhost  
::1           localhost ip6-localhost ip6-loopback  
fe00::0       ip6-localnet  
ff00::0       ip6-mcastprefix  
ff02::1       ip6-allnodes  
ff02::2       ip6-allrouters  
172.17.0.3     tomcat02 9a9f30d83b57  
172.17.0.4     5f9331566980
```



赛博梦工厂

Cyber Works

--link

- --link 本质就是在hosts配置中添加映射
- 现在使用Docker已经不建议使用--link了!
- 自定义网络, 不适用docker0!
- docker0问题: 不支持容器名连接访问!





第5章 Docker网络

自定义网络

```
→ ~ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
ad5ada6a106f        bridge             bridge              local
19315b394ac7        host               host                local
8203922769a9        none               null                local
```

网络模式

- bridge：桥接 docker（默认，自己创建也是用bridge模式）
- none：不配置网络，一般不用
- host：和所主机共享网络
- container：容器网络连通（用得少！局限很大）



赛博梦工厂

Cyber Works

实战测试 (Docker网络)

我们直接启动的命令 --net bridge,而这个就是我们得docker0

bridge就是docker0

- docker run -d -P --name tomcat01 tomcat

等价于 => docker run -d -P --name tomcat01 --net bridge tomcat

docker0, 特点: 默认, 域名不能访问。 --link可以打通连接, 但是很麻烦!

我们可以 自定义一个网络

- docker network create --driver bridge --subnet 192.168.0.0/16 --gateway 192.168.0.1 mynet

```
→ ~ docker network create --driver bridge --subnet 192.168.0.0/16 --  
gateway 192.168.0.1 mynet  
aabfa0c0a0bdc266facb15e1e00e6e4d68c5ecec5c64919c0f8d04da9637ff0f  
→ ~ docker network ls  
NETWORK ID          NAME                DRIVER              SCOPE  
ad5ada6a106f        bridge             bridge              local  
19315b394ac7        host               host                local  
aabfa0c0a0bd        mynet              bridge              local  
8203922769a9        none              null                local  
→ ~
```



赛博梦工厂

Cyber Works

实战测试（部署Redis集群）

```
→ ~ docker network inspect mynet
[
  {
    "Name": "mynet",
    "Id": "aabfa0c0a0bdc266facb15e1e00e6e4d68c5ecec5c64919c0f8d04da9637ff0f",
    "Created": "2020-05-16T19:11:09.183843491+08:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "192.168.0.0/16",
          "Gateway": "192.168.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```



实战测试 (Docker网络)

```
→ ~ docker run -d -P --name tomcat-net-01 --net mynet tomcat  
1c3fbb2f22df905d80e8c447839822d78ddc98220ce66fd599c572ad235fa13a  
→ ~ docker run -d -P --name tomcat-net-02 --net mynet tomcat  
524f7b9526347398c63f058a68886f780728d9c09e87f83c92d6a0be73ae00ac
```

```
→ ~ docker exec tomcat-net-02 ping tomcat-net-01  
PING tomcat-net-01 (192.168.0.2) 56(84) bytes of data.  
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=1 ttl=64 time=0.133 ms  
^C  
→ ~ docker exec tomcat-net-01 ping tomcat-net-02  
PING tomcat-net-02 (192.168.0.3) 56(84) bytes of data.  
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=1 ttl=64 time=0.114 ms  
^C  
→ ~
```

我们自定义的网络docker当我们维护好了对应的关系，推荐我们平时这样使用网络！

好处：

redis -不同的集群使用不同的网络，保证集群是安全和健康的

mysql-不同的集群使用不同的网络，保证集群是安全和健康的



赛博梦工厂

Cyber Works



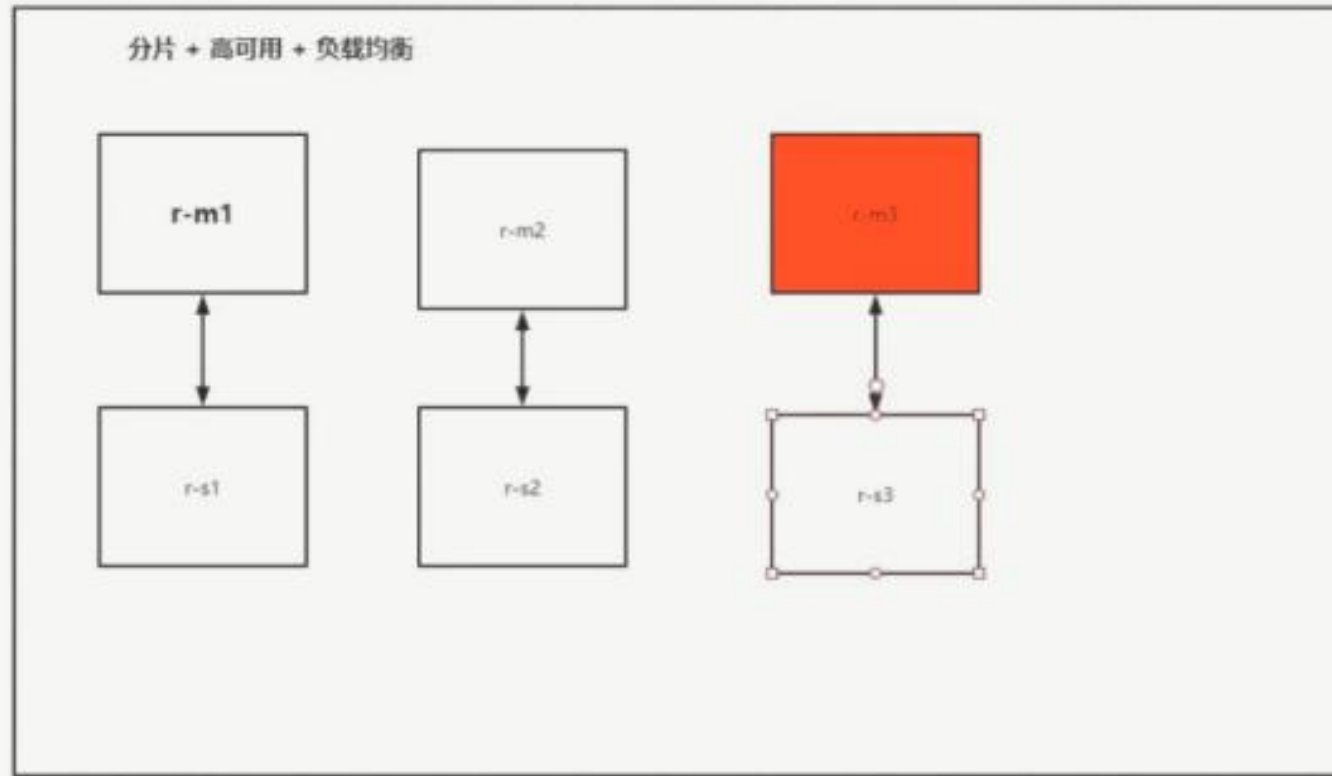
第5章 Docker网络

本章目标

- 理解Docker 0
- --link
- 自定义网络
- 网络连通



实战测试（部署Redis集群）



实战测试（部署Redis集群）

创建网卡

```
docker network create --subnet 192.168.1.0/24 --gateway 192.168.1.1  
redis
```



实战测试（部署Redis集群）

```
# 通过脚本创建六个redis配置文件
for port in $(seq 1 6);\
do \
mkdir -p /mydata/redis/node-{$port}/conf
touch /mydata/redis/node-{$port}/conf/redis.conf
cat << EOF >> /mydata/redis/node-{$port}/conf/redis.conf
port 6379
bind 0.0.0.0
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
cluster-announce-ip 192.168.1.1{$port}
cluster-announce-port 6379
cluster-announce-bus-port 16379
appendonly yes
EOF
done
```



赛博梦工厂

Cyber Works

实战测试（部署Redis集群）

```
# 通过脚本运行六个redis
for port in $(seq 1 6);\
do \
docker run -p 637${port}:6379 -p 1667${port}:16379 --name redis-${port} \
-v /mydata/redis/node-${port}/data:/data \
-v /mydata/redis/node-${port}/conf/redis.conf:/etc/redis/redis.conf \
-d --net redis --ip 192.168.1.1${port} redis:5.0.9 redis-server
/etc/redis/redis.conf #注意这段命令要写全，否则redis容器内无文件
done
docker exec -it redis-1 /bin/sh #redis默认没有bash
```



实战测试（部署Redis集群）

#创建集群

```
redis-cli --cluster create 192.168.1.11:6379 192.168.1.12:6379 192.168.1.13:6379  
192.168.1.14:6379 192.168.1.15:6379 192.168.1.16:6379 --cluster-replicas 1
```



赛博梦工厂
Cyber Works

实战测试（部署Redis集群）

```
Adding replica 192.168.1.14:6379 to 192.168.1.13:6379
M: f38861c044dfec6086d01861471a57ab43e8dd84 192.168.1.11:6379
slots:[0-5460] (5461 slots) master
M: ccc6a7f811838c4779ddd59c45dbd9982459f760 192.168.1.12:6379
slots:[5461-10922] (5462 slots) master
M: 1107a7761e73fb05be7eb0ea81e68886b4b69fdd 192.168.1.13:6379
slots:[10923-16383] (5461 slots) master
S: e51e0316af2a934ef85935414b8fd039126d72b3 192.168.1.14:6379
replicates 1107a7761e73fb05be7eb0ea81e68886b4b69fdd
S: 4561aebdelc21048536e07b9db3caa49b2098d7c 192.168.1.15:6379
replicates f38861c044dfec6086d01861471a57ab43e8dd84
S: d31142cbacblddc830b3ffb6b0ae6a34e37b0f98 192.168.1.16:6379
replicates ccc6a7f811838c4779ddd59c45dbd9982459f760
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
...
>>> Performing Cluster Check (using node 192.168.1.11:6379)
M: f38861c044dfec6086d01861471a57ab43e8dd84 192.168.1.11:6379
slots:[0-5460] (5461 slots) master
1 additional replica(s)
S: d31142cbacblddc830b3ffb6b0ae6a34e37b0f98 192.168.1.16:6379
slots: (0 slots) slave
replicates ccc6a7f811838c4779ddd59c45dbd9982459f760
M: 1107a7761e73fb05be7eb0ea81e68886b4b69fdd 192.168.1.13:6379
slots:[10923-16383] (5461 slots) master
1 additional replica(s)
S: e51e0316af2a934ef85935414b8fd039126d72b3 192.168.1.14:6379
slots: (0 slots) slave
replicates 1107a7761e73fb05be7eb0ea81e68886b4b69fdd
M: ccc6a7f811838c4779ddd59c45dbd9982459f760 192.168.1.12:6379
slots:[5461-10922] (5462 slots) master
1 additional replica(s)
S: 4561aebdelc21048536e07b9db3caa49b2098d7c 192.168.1.15:6379
slots: (0 slots) slave
replicates f38861c044dfec6086d01861471a57ab43e8dd84
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```



赛博梦工厂

Cyber Works

实战测试（部署Redis集群）

测试: redis-cli -c 回车

127.0.0.1:6379> cluster info #查看集群信息

127.0.0.1:6379> cluster nodes #查看节点信息

127.0.0.1:6379> set a b #redis中设定一个键值对, 假如处理的是
192.168.1.13

- docker stop redis-3 #停掉192.168.1.13

127.0.0.1:6379> redis-cli -c #重新连接一下集群

127.0.0.1:6379> get a #会显示出接替192.168.1.13的机器



赛博梦工厂

Cyber Works

第5章 Docker网络

实战部署个人博客 (WordPress)

- `docker pull mysql:5.7`
- `docker pull wordpress`
- `docker run -d --name db.wordpress -e MYSQL_ROOT_PASSWORD=admin mysql:5.7`
#sh -c后边的单引号内为在容器内执行的命令
- `docker exec db.wordpress sh -c 'exec mysql -uroot -p" $MYSQL_ROOT_PASSWORD" -e" create database wordpress" '`
- #启动一个 WordPress 容器, 将 db.wordpress 容器连接到 WordPress 容器即可:
- `docker run -d -p 8080:80 --name wordpress --link db.wordpress:mysql wordpress`



赛博梦工厂

Cyber Works



谢谢观赏