Calico: A Solution of Multi-host Network For Docker

Posted on 2015-09-06   |   In [Docker Multi-host Network](http://xelatex.github.io/categories/Docker-Multi-host-Network/)  |

* **UPDATE on Nov. 15** : Add chapter “Integrate Calico with Docker Network” to illustrate the new feature of Calico libnetwork plugin for Docker Engine v1.9.
* **UPDATE on Nov. 16** : Add chapter “Limitations”, describing limitations of Calico.
* **UPDATE on Nov. 16** : Add chapter “FAQ” for some frequently asked questions about Calico configuration.

**Introduction to Calico**

[Calico](https://github.com/projectcalico/calico) is a pure 3-layer protocol to support multi-host network communication for OpenStacks VMs and Docker containers. Calico does not use overlay network such as [falnnel](https://github.com/coreos/flannel" \t "_blank) and [libnetwork overlay driver](https://github.com/docker/libnetwork/blob/master/docs/overlay.md" \t "_blank), it is a pure Layer 3 approach with a vRouter implementation instead of a vSwitcher. Each vRouter propagates workload reachability information (routes) to the rest of the data center using BGP protocol.

This post focus on how to setup a multi-host networking for Docker containers with [calico-docker](https://github.com/projectcalico/calico-docker) and some advanced features.

**Environment**

**Environment Prerequisite**

* Two linux nodes (node1 and node2) with Ubuntu Linux distribution, either VM or physical machine is OK.
* Install docker on both nodes.
* Etcd cluster.

**Configuration & Download**

Setup two linux nodes with IP 192.168.236.130/131 and connect them physically or virtually, confirm that they can ping each other succesfully. Setup docker bridge (default is docker0) on two nodes. Let’s set two docker bridges with different network. Netowrk configuration details are as follows:

Node1

* IP: 192.168.236.130
* Docker bridge network: 192.168.1.0/24

Node2

* IP: 192.168.236.131
* Docker bridge network: 172.17.0.0/16

Install Docker, should be no error here.

Install Docker

|  |  |
| --- | --- |
| 1 2 | sudo apt-get install docker.io docker ps |

Download and run etcd, replace {node} with node0/1 seperately. We need at least two etcd node since the new version of etcd cannot run on single node.

Download and run etcd

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | curl -L https://github.com/coreos/etcd/releases/download/v2.2.1/etcd-v2.2.1-linux-amd64.tar.gz -o etcd-v2.2.1-linux-amd64.tar.gz tar xzvf etcd-v2.2.1-linux-amd64.tar.gz cd etcd-v2.2.1-linux-amd64 ./etcd -name {node} -initial-advertise-peer-urls http://{NODE\_IP}:2380 \  -listen-peer-urls http://0.0.0.0:2380 \  -listen-client-urls http://0.0.0.0:2379,http://127.0.0.1:4001 \  -advertise-client-urls http://0.0.0.0:2379 \  -initial-cluster-token etcd-cluster \  -initial-cluster node1=http://192.168.236.130:2380,node2=http://192.168.236.131:2380 \  -initial-cluster-state new |

Download calicoctl

Download calicoctl

|  |  |
| --- | --- |
| 1 | wget https://github.com/projectcalico/calico-docker/releases/download/v0.10.0/calicoctl |

**Start Calico Services**

Calico services in Docker environment are running as a Docker container using host network configuration. All containers configured with Calico services with use calico-node to communicate with each other and Internet.

Run the following commands on node1/2 to start calico-node

Run calico-node

|  |  |
| --- | --- |
| 1 | sudo calicoctl node --ip={host\_ip} |

You should see output like this on each node

calico@node1:~# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

40b177803c97 calico/node:v0.9.0 "/sbin/my\_init" 27 seconds ago Up 27 seconds calico-node

Before starting any containers, we need to configure an IP pool with the ipip and nat-outgoing options. Thus containers with an valid profile could have access to Internet. Run the following command on either node.

Configure IP pool

|  |  |
| --- | --- |
| 1 | calicoctl pool add 192.168.100.0/24 --ipip --nat-outgoing |

**Container Networking Configuration**

**Start Containers**

Firstly run a few containers on each host.

On node1:

Run container on node1

|  |  |
| --- | --- |
| 1 2 | docker run --net=none --name worker-1 -tid ubuntu docker run --net=none --name worker-2 -tid ubuntu |

On node2:

Run container on node2

|  |  |
| --- | --- |
| 1 | docker run --net=none --name worker-3 -tid ubuntu |

**Configure Calico Networking**

Now that all the containers are running without any network devices. Use Calico to assign network devices to these containers. Notice that IPs assigned to containers should be in the range of IP pools.

On node1:

Configure network on node1

|  |  |
| --- | --- |
| 1 2 | sudo calicoctl container add worker-1 192.168.100.1 sudo calicoctl container add worker-2 192.168.100.2 |

On node2:

Configure network on node2

|  |  |
| --- | --- |
| 1 | sudo calicoctl container add worker-3 192.168.100.3 |

Once containers have Calico networking, they gain a network device with corresponding IP address. At this point them have access neither to each other nor to Internet since no profiles are created and assigned to them.

Create some profiles on either node:

Create profiles

|  |  |
| --- | --- |
| 1 2 | calicoctl profile add PROF\_1 calicoctl profile add PROF\_2 |

Then assign profiles to containers. Containers in same profile have access to each other. And containers in the IP poll created before won’t have access to Internet until added to a profile.

On node1:

Assign profiles to containers on node1

|  |  |
| --- | --- |
| 1 2 | calicoctl container worker-1 profile append PROF\_1 calicoctl container worker-2 profile append PROF\_2 |

On node2:

Assign profiles to containers on node2

|  |  |
| --- | --- |
| 1 | calicoctl container worker-3 profile append PROF\_1 |

Until now all configurations are done and we will test network connections of these containers afterwards.

**Testing**

Now check the connectivities of each containers. At this point every containers should have access to Internet, try and ping google.com:

Check Internet access

|  |  |
| --- | --- |
| 1 2 | docker exec worker-1 ping -c 4 www.google.com docker exec worker-2 ping -c 4 www.google.com |

Then check connections of containers in same profile:

Check inner profile access

|  |  |
| --- | --- |
| 1 | docker exec worker-1 ping -c 4 192.168.100.3 |

And containers not in same profile cannot ping each other:

Check access outer profile

|  |  |
| --- | --- |
| 1 | docker exec worker-1 ping -c 4 192.168.100.2 |

If we add worker-2 into profile PROF\_1, then worker-2 could ping worker-1 and worker-3.  
On node1:

Advanced check

|  |  |
| --- | --- |
| 1 2 3 | calicoctl container worker-2 profile append PROF\_1 docker exec worker-2 ping -c 4 192.168.100.1 docker exec worker-2 ping -c 4 192.168.100.3 |

**Performance Tests**

**Simple Test**

I perform a simple performance test using iperf to evaluate the network between two Calico containers. Run iperf -s on worker-1 and iperf -c 192.168.100.1 on worker-3. We can get the result:

root@39fdb1701da4:~# ./iperf -c 192.168.101.2

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

Client connecting to 192.168.101.2, TCP port 5001

TCP window size: 85.0 KByte (default)

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

[ 3] local 192.168.101.1 port 39187 connected with 192.168.101.2 port 5001

[ ID] Interval Transfer Bandwidth

[ 3] 0.0-10.0 sec 1.08 GBytes 927 Mbits/sec

Then run the same test on native host (node1 and node2):

calico@node2:~# iperf -c 192.168.236.130

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

Client connecting to 192.168.236.130, TCP port 5001

TCP window size: 85.0 KByte (default)

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

[ 3] local 192.168.236.131 port 54584 connected with 192.168.236.130 port 5001

[ ID] Interval Transfer Bandwidth

[ 3] 0.0-10.0 sec 2.57 GBytes 2.21 Gbits/sec

From the result we can see there’s a great gap between Calico network and native network. But according to the official documents and evaluations, calico network should be similar to the native network. **WHY???**

**Dive Deeper**

To find out the reason of slow network, firstly I test the network performance between workker-1 and worker-2, which are in the same host. The result is as follows:

root@51b78d9e6153:/# iperf -c 192.168.100.2

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

Client connecting to 192.168.100.3, TCP port 5001

TCP window size: 85.0 KByte (default)

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

[ 3] local 192.168.100.2 port 36476 connected with 192.168.100.3 port 5001

[ ID] Interval Transfer Bandwidth

[ 3] 0.0-10.0 sec 47.3 GBytes 40.6 Gbits/sec

Since speed of my net card is only 1Gbits/sec, it seems that containers on the same host connects each other directly without going through any network device. That really make all sense.

Then I dived deep into the documents and configurations of Calico and found such configuration of IP pool:

Configure IP pool

|  |  |
| --- | --- |
| 1 | calicoctl pool add 192.168.100.0/24 --ipip --nat-outgoing |

We use --ipip option when creating IP pool, which means Use IP-over-IP encapsulation across hosts. This option will enforce another layer of IP-over-IP encapsulation when packages traveling across hosts. Since our hosts node1 and node2 are in the same network (192.168.236.0/24), we could avoid this option and the speed should increase as supposed.

If your hosts located in different L2 network, which means can only connected to each other via IP network, you need to add --ipip options when starting Calico.

Run the following command on either node to override the previous IP pool configuration.

Configure IP pool

|  |  |
| --- | --- |
| 1 2 | calicoctl pool add 192.168.100.0/24 --nat-outgoing calicoctl pool show |

Then test networking between worker-1 and worker-3 again:

root@39fdb1701da4:~# ./iperf -c 192.168.101.2

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

Client connecting to 192.168.101.2, TCP port 5001

TCP window size: 85.0 KByte (default)

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

[ 3] local 192.168.101.1 port 39187 connected with 192.168.101.2 port 5001

[ ID] Interval Transfer Bandwidth

[ 3] 0.0-10.0 sec 2.74 GBytes 2.35 Gbits/sec

Hurray!!! That’s the native speed!

**Integrate Calico with Docker Network**

Calico can be integrated into Docker network after Docker released it’s v1.9 Docker Engine. Calico runs another container as Docker network plug-in, and integrates into Docker docker network commands.

Integrated Calico needs Docker Engine running on cluster mode. Stop original Docker daemon on node1/2 and run with cluster parameters:

Run Docker daemon with cluster params

|  |  |
| --- | --- |
| 1 2 | root@node1:~# sudo service docker stop root@node1:~# sudo /usr/bin/docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock --cluster-store=etcd://{ETCD\_IP}:4001 --cluster-advertise={NODE\_IP}:2375 |

Then run Calico with --libnetwork param:

Run calico node with libnetwork

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | root@node1:~# calicoctl node --libnetwork --ip={NODE\_IP} Calico node is running with id: 129d25cee92cc6d979ab3bed78482487c74fc136f0703991bc6572ceabb60cd1 Calico libnetwork driver is running with id: b29bb1f35c88096440afb740e23e433b52f2a4296747b915b6b212a98fc16a2c root@node1:~# docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES b29bb1f35c88 calico/node-libnetwork:v0.5.0 "./start.sh" 22 seconds ago Up 21 seconds calico-libnetwork 129d25cee92c calico/node:v0.9.0 "/sbin/my\_init" 22 seconds ago Up 21 seconds calico-node |

The new command docker network is introduced since Docker Engine v1.9 can be used to create a logical network. With the support of calico-libnetwork container, docker network can create a network with calico network driver as follows:

Create calico network

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | root@node1:~# docker network create --driver=calico --subnet=192.168.0.0/24 net1 root@node1:~# docker network ls NETWORK ID NAME DRIVER 42407f4bfbeb net1 calico 090c48443dc3 bridge bridge ff33bb080344 none null 62d6ae9141e5 host host |

You can see network net1 with driver type calico.

If you are running in a cloud environment (AWS, DigitalOcean, GCE), you will need to configure the network with --ipip and --nat-outgoing options. On either host, run:

Create calico network with options

|  |  |
| --- | --- |
| 1 | docker network create --driver=calico --opt nat-outgoing=true --opt ipip=true --subnet=192.168.0.0/24 net1 |

Note that we use the Calico driver calico. This driver is run within the calico-node container. We explictly choose an IP Pool for each network to avoid IP confliction. Then run docker directly with --net=net1 option without any other auxiliary configuration.

Run docker

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 | root@node1:~# docker run --net net1 --name worker-1 -tid ubuntu root@node1:~$ docker exec worker-1 ifconfig cali0 Link encap:Ethernet HWaddr ee:ee:ee:ee:ee:ee  inet addr:192.168.0.3 Bcast:0.0.0.0 Mask:255.255.255.0  inet6 addr: fe80::ecee:eeff:feee:eeee/64 Scope:Link  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  RX packets:7 errors:0 dropped:0 overruns:0 frame:0  TX packets:7 errors:0 dropped:0 overruns:0 carrier:0  collisions:0 txqueuelen:1000  RX bytes:738 (738.0 B) TX bytes:578 (578.0 B)  eth1 Link encap:Ethernet HWaddr 02:42:ac:12:00:02  inet addr:172.18.0.2 Bcast:0.0.0.0 Mask:255.255.0.0  inet6 addr: fe80::42:acff:fe12:2/64 Scope:Link  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  RX packets:14 errors:0 dropped:0 overruns:0 frame:0  TX packets:7 errors:0 dropped:0 overruns:0 carrier:0  collisions:0 txqueuelen:0  RX bytes:1476 (1.4 KB) TX bytes:578 (578.0 B)  lo Link encap:Local Loopback  inet addr:127.0.0.1 Mask:255.0.0.0  inet6 addr: ::1/128 Scope:Host  UP LOOPBACK RUNNING MTU:65536 Metric:1  RX packets:0 errors:0 dropped:0 overruns:0 frame:0  TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  collisions:0 txqueuelen:0  RX bytes:0 (0.0 B) TX bytes:0 (0.0 B) |

A cali0 veth in container is created to communicate with other containers connected to the same net1. There’s a little difference compared with previous configuration, another eth1 veth is created to act as normal NIC.

**Limitations**

Calico implements a pure Layer-3 solution which encapsulate L3 package over IP or broadcast network. Though the pure Layer-3 solution brings greate performance, it also introduce a batch of limitations.

* Calico only supports **TCP, UDP, ICMP and ICMPv6** protocol. If you want to use other L4 protocols, you need to choose Flannel, Weave or Docker Overlay Network.
* Calico doesn’t have encryption data path. It’s not safe to build overlay network with Calico over untrusted network.
* The performance of Calico with IP-over-IP option is quite bad, which --ipip option is a must in a public data center connected with IP network.
* No IP overlap support. Though Calico community is developing a experimental feature that put overlap IPv4 packages into IPv6 package. But this is only an auxiliary solution and doesn’t fully support IP overlap technically.

**FAQ**

**Q: What is --ipip options used for when configuring Calico pool?**  
A: --ipip option means IP-over-IP mode. By default, calico broadcast the IP packages to all hosts through L2 switch and filter the packages by host’s routing table. For hosts connected with IP network, Calico need to encapsulate container’s IP packets in an outer IP packets and transfer to the remote host. So if you use Calico on a public data center, you’d better add --ipip option.  
  
  
**Q: How to assign Etcd address instead of using default value “localhost:4001”?**  
A: Run export ETCD\_AUTHORITY={ETCD\_HOST}:{ETCD\_PORT} on shell before running calico node.  
  
  
\*\*Q: What if I don’t want to use a distributed storage such as Etcd?  
A: Choose an alternative solution - [Weave](http://xelatex.github.io/2015/11/14/Weave-Network-Management-for-Docker/), which has an internal routing mechanism.

**References**

[1] Project Calico: <https://github.com/projectcalico/calico>  
[2] Calico Docker: <https://github.com/projectcalico/calico-docker>  
[3] Demenstration on calico-docker: <https://github.com/projectcalico/calico-docker>  
[4] Calico-docker in Yixin: [Paper URL](http://mp.weixin.qq.com/s?__biz=MzAwMDU1MTE1OQ==&mid=400983139&idx=1&sn=f033e3dca32ca9f0b7c9779528523e7e&scene=1&srcid=1101jklWCo9jNFjdnUum85PG&from=singlemessage&isappinstalled=0#wechat_redirect)