Kubernetes: Flannel networking

Xiao Han
Oct 17, 2017 · 8 min read

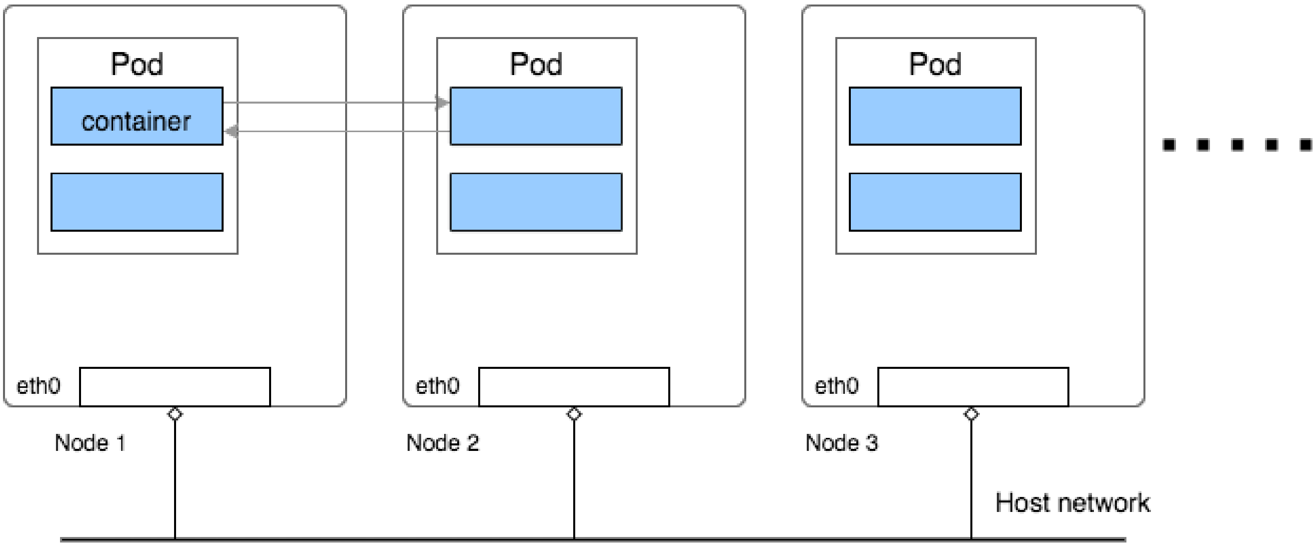
This article explains how flannel network works in Kubernetes

Kubernetes is an excellent tool for managing containerized applications at scale. But as you may know, working with kubernetes is not an easy road, especially the backend networking implementation. I have met many problems in networking and it cost me much time to figure out how it works.

In this article, I want to use the most simple implementation as an example, to explain kubernetes networking works. Hope this article can help those people like me who are studying kubernetes as a beginner.

Kubernetes networking model

The following graph shows a simple image of a kubernetes cluster:



Pods in Kubernetes

Kubernetes manages a cluster of Linux machines(might be cloud VM like AWS EC2 or physical servers), on each host machine, kubernetes runs any number of Pods, in each Pod there can be any number of containers. User’s application is running in one of those containers.

For kubernetes, Pod is the minimum management unit, and all containers inside one Pod shares the same network namespace, which means they have same network interface and can connect each other by using *localhost*

The official documentation says kubernetes networking model requires:

- all containers can communicate with all other containers without NAT
- all nodes can communicate with all containers (and vice-versa) without NAT
- the IP that a container sees itself as is the same IP that others see it as

We can replace all containers to Pods in above requirements, as containers share with Pod network.

Basically it means all Pods should be able to freely communicate with any other Pods in the cluster, even they are in different Hosts, and they recognized each other with their own IP address, just as the underlying Host does not exists. Also the Host should also be able to communicate with any Pod with it’s own IP address, without any address translation.

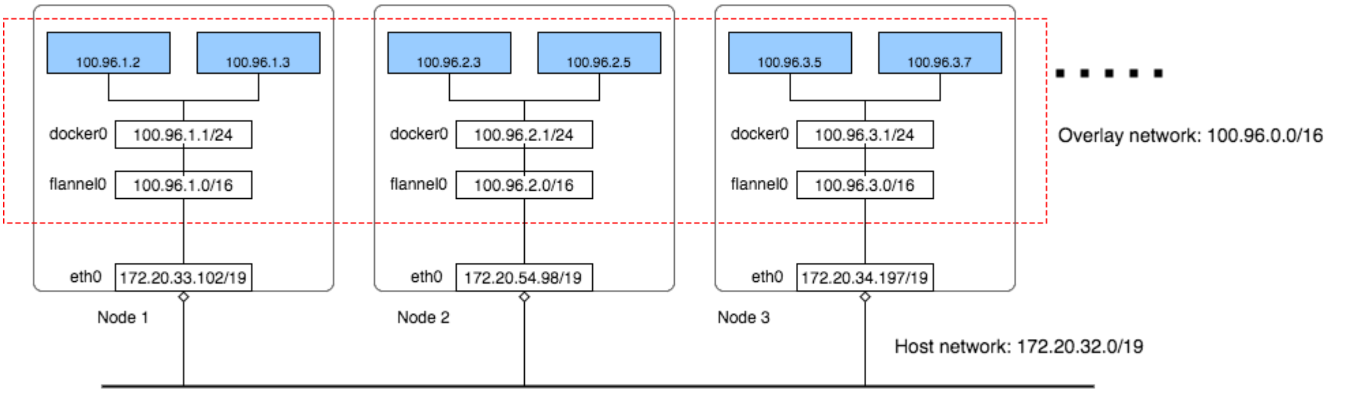
Kubernetes does not provide any default network implementation, rather it only defines the model and leaves to other tools to implement it. There are many implementations nowadays, flannel is one of them and one of the simplest. In the following sections, I will explain the Flannel’s UDP mode impelmentations.

The Overlay Network

Flannel is created by CoreOS for Kubernetes networking, it also can be used as a general software defined network solution for other purpose.

To achieve kubernetes' network requirements, flannel’s idea is simple: create another flat network which runs above the host network, this is the so-called *overlay* network. All containers(Pod) will be assigned one ip address in this overlay network, they communicate with each other by calling each other’s ip address directly.

To help explain, I use a small testing kubernetes cluster on AWS, there are 3 Kubernetes nodes in the cluster. Here is how the network looks like:



Flannel overlay network

There are three networks in this cluster:

AWS VPC network: all instances are in one VPC subnet `172.20.32.0/19`. They have been assigned ip addresses in this range, all hosts can connect to each other because they are in same LAN.

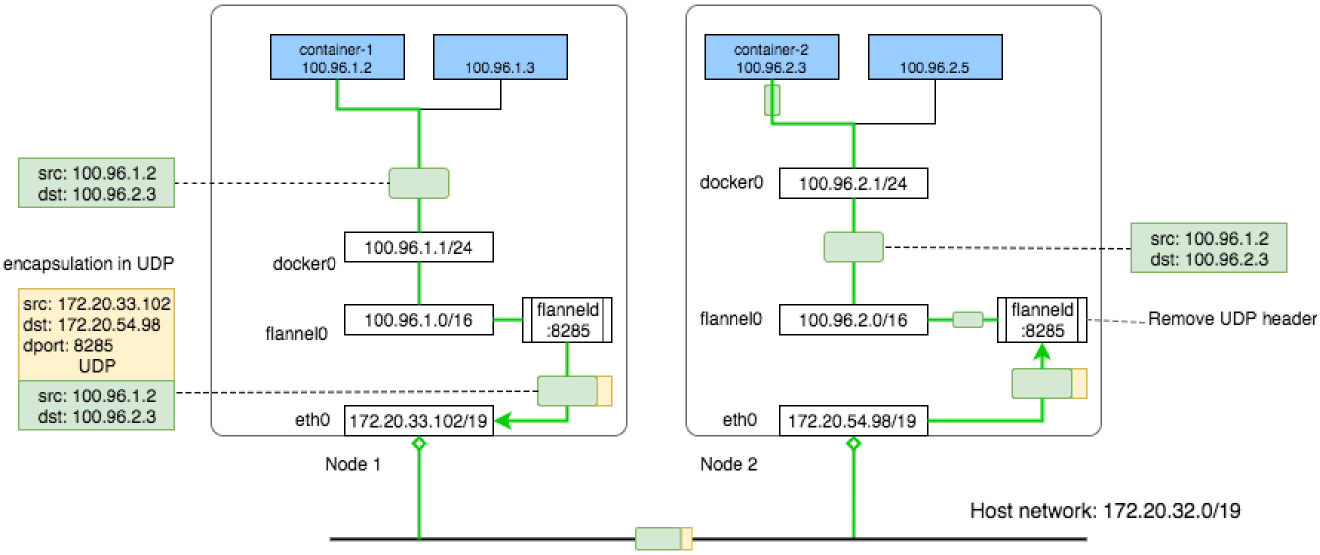
Flannel overlay network: flannel has created another network `100.96.0.0/16`, it's a bigger network which can hold 2^{16}(65536) addresses, and it's across all kubernetes nodes, each pod will be assigned one address in this range, later we will see how flannel achieves this.

In-Host docker network: inside each host, flannel assigned a `100.96.x.0/24` network to all pods in this host, it can hold 2^{8}(256) addresses. The docker bridge interface `docker0` will use this network to create new containers.

By this design, each container has it's own IP address, all fall into the overlay subnet `100.96.0.0/16`. The containers inside the same host can communicate with each other by the docker bridge `docker0`, which is simple so I will skip in this article. To communicate across hosts with other containers in the overlay network, flannel uses kernel route table and UDP encapsulation to achieve it, the following sections explain this.

Cross host container communication

Assume the container(let's call it container-1) in Node 1 which has the IP address `100.96.1.2` wants to connect to the container(let's call it container-2) in Node 2 with the IP address `100.96.2.3`, let's see how the overlay network enables the packets passing.



Cross host communication

First container-1 creates an IP packet with `src: 100.96.1.2 -> dst: 100.96.2.3`, the packet will go to the docker0 bridge as it's the container's gateway.

In each host, flannel runs a daemon process called `flanneld`, it creates some route rules in kernel's route table, here is what Node 1's route table looks like:

```
admin@ip-172-20-33-102:~$ ip route
default via 172.20.32.1 dev eth0
100.96.0.0/16 dev flannel0  proto kernel  scope link  src 100.96.1.0
100.96.1.0/24 dev docker0  proto kernel  scope link  src 100.96.1.1
172.20.32.0/19 dev eth0  proto kernel  scope link  src 172.20.33.102
```

As we can see, the packet's destination address `100.96.2.3` falls in the bigger overlay network `100.96.0.0/16`, so it matches the second rule, now kernel knows it should send the packet to `flannel0`.

`flannel0` is a TUN device also created by our `flanneld` daemon process, TUN is a software interface implemented in linux kernel, it can pass raw ip packet between user program and the kernel. It works in two directions:

- when it write IP packet to the `flannel0` device, the packet will be send to kernel directly, and kernel will route the packet according to its route table

- when an IP packet arrives to the kernel, and the route tables says it should be routed to `flannel0` device, kernel will send the packet directly to the process which created this device, which is the `flanneld` daemon process

As kernel sends packet to the TUN device, it will directly goes to the `flanneld` process, it sees the destination address is `100.96.2.3`, though we can see from the diagram that this address belongs to a container who runs on Node 2, but how does `flanneld` know this?

It so happens that flannel stores some information in a key-value storage service called *etcd*, if you know kubernetes you should not be suprised. In flannel case, we can just think it as a general key-value store.

flannel stores the subnet to host mapping information into the etcd service, we can see it by using `etcdctl` command:

```
admin@ip-172-20-33-102:~$ etcdctl ls /coreos.com/network/subnets
/coreos.com/network/subnets/100.96.1.0-24
/coreos.com/network/subnets/100.96.2.0-24
/coreos.com/network/subnets/100.96.3.0-24

admin@ip-172-20-33-102:~$ etcdctl get
/coreos.com/network/subnets/100.96.2.0-24
{"PublicIP":"172.20.54.98"}
```

so each `flanneld` process queries etcd to know each subnet belongs to which host, and compares the destination ip address with all subnets key stored in etcd. In our case, the address `100.96.2.3` will match the subnet `100.96.2.0-24`, and as we see the value stored in this keys says the Node ip is `172.20.54.98`.

Now `flanneld` knows the destination address, it then wraps the original IP packet into a UDP packet, with it’s own host’s ip as source address, and the target host’s IP as destination address. In each host, the `flanneld` process will listen on a default UDP port `:8285`. So it just need to set the UDP packet’s destination port to `8285`, and send it on the wire.

After the UDP packet arrives at the destination host, the kernel’s IP stack will send the packet to the `flanneld` process because that’s the user process who listens on UDP port `:8285`. Then `flanneld` will get the payload of the UDP packet, which is the original IP packet generated by the originate container, it simply write this packet to the TUN device `flannel0`, then the packet will directly pass to kernel as that’s how TUN works.

Same as in Node 1, the route table will decide where this packet should go, let’s see Node 2’s route table:

```
admin@ip-172-20-54-98:~$ ip route
default via 172.20.32.1 dev eth0
100.96.0.0/16 dev flannel0  proto kernel  scope link  src 100.96.2.0
100.96.2.0/24 dev docker0  proto kernel  scope link  src 100.96.2.1
172.20.32.0/19 dev eth0  proto kernel  scope link  src 172.20.54.98
```

the destination address of the IP packet is `100.96.2.3`, kernel will take the most precise match, which is the third rule. The packet will be send to `docker0` device. As `docker0` is a bridge device, and all containers in this host are connected to this bridge, the packet will finnaly be seen and received by the destination container-2.

Finally our packet finished the one way pass to the target, when contianer-2 sends packet back to container-1, the reverse route will work exactly the same way. That’s how cross host container communication works.

Configuring with docker network

In the above explanation, we have missed one point. That’s how we configure docker to use the smaller subnet like `100.96.x.0/24`?

It happens that `flanneld` will write it’s subnet information into a file in the host:

```
admin@ip-172-20-33-102:~$ cat /run/flannel/subnet.env
FLANNEL_NETWORK=100.96.0.0/16
FLANNEL_SUBNET=100.96.1.1/24
FLANNEL_MTU=8973
FLANNEL_IPMASQ=true
```

This information will be used to configure the docker daemon’s option, so docker can use the `FLANNEL_SUBNET` as its bridge network, then the in-host container network will work:

```
dockerd --bip=$FLANNEL_SUBNET --mtu=$FLANNEL_MTU
```

Packet copy and performance

The newer version of flannel does not recommend to use UDP encapsulation for production, says it should be only used for debugging and testing purpose. One reason is the performance.

Though the `flannel0` TUN device provides a simple way to get and send packet through the kernel, it has a performance penalty: the packet has to be copied back and forth from the user space to kernel space:

packet copy

as the above, from the original container process send packet, it has to be copied 3 times between user space and kernel space, this will increase network overhead in a significant way, so you should avoid using UDP in production if you can.

Conclusion

Flannel is one of the simplest implementation of kubernetes network model. It uses the existing docker bridge network and an extra Tun device with a daemon process to do UDP encapsulation. I explained the details of the core part: cross host container communication, and briefly mentioned the performance penalty. I hope this article helps people to understand the basics of kuberentes networking, with this knowledge you can start exploring the more interesting world of kubernetes networking, and figure out many advanced implementations like Calico, WeaveNet, Romana.

References

https://coreos.com/blog/introducing-rudder.html
http://backreference.org/2010/03/26/tuntap-interface-tutorial/

Kubernetes Flannel Networking Docker Container