Container and Kubernetes Networking 101
Before we begin

Poll

Arun Sriraman
(Software Engineer)
Platform9 Systems Inc
@arun_sriraman sarun87

Aditya Amar
Sr. Principal Software Engineer
@adityaCal
Agenda

Part I - Container Networking 101

• Need for container networking
• Linux networking constructs
  – Bridge drivers
  – Network Namespace
• Intro to docker networking - the CNM model
• Docker networking drivers and its comparison

Part II - Kubernetes Networking 101

• K8s networking fundamentals
• Kubernetes communication
  – Container-to-Container
  – Pod-to-Pod
  – Pod-to-Service
  – Service-to-external
• Container Network Interface
• CNI backend (Flannel, Calico)
Part I - Container Networking
Containers need to talk to:
- outside world and vice-versa
- the host machine (maybe)
- other containers running within and across hosts

We also need to be able to:
- load balance traffic between containers
- provide multi-tenancy
- automatically discover services provided by other containers

This sounds very similar to VMs and VM networking....
<table>
<thead>
<tr>
<th></th>
<th>Virtual Machines</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate networking stack</td>
<td>Network namespaces used to achieve network isolation</td>
<td>Service (typically) gets a separate IP; Service (typically) maps to multiple containers. With Kubernetes, services have their own IP</td>
</tr>
<tr>
<td>Multiple services run inside a single VM; the VM gets an IP - services may or may not be addressed explicitly.</td>
<td></td>
<td>Microservices implemented using Containers leads to more integrated Service Discovery</td>
</tr>
<tr>
<td>Service Discovery and Load balancing (typically) done outside the VM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scaling needs are not that high</td>
<td></td>
<td>Scaling needs at least an order of magnitude higher</td>
</tr>
</tbody>
</table>
Linux networking constructs

- The Linux Bridge device
- Network Namespaces
- Virtual Ethernet Devices
- iptables
Network namespaces

Process started with a new network namespace gets its own private network stack with
- network interfaces (including lo)
- routing tables
- iptables rules
- sockets (ss, netstat)

```c
flags = CLONE_NEWPID|
       CLONE_NEWNS|CLONE_NEWNET;
cpid = clone(child_function, childstack, flags, (void *)argv);
```
Add a network namespace

List of network namespaces

Namespace A

Host network namespace
Linux bridge and veth interface

http://blog.arunsriraman.com/2017/01/container-namespaces-deep-dive-into.html
Container Network Model (CNM)

- Project started by Docker
- Separate networking from container runtime as a library
- Components
  - Sandbox
  - Endpoint
  - Network
- Implemented using libnetwork
Mapping CNM to Libnetwork (Docker)
Libnetwork contd.

- Service Discovery
- Load Balancing
- Network Control Plane

Libnetwork (implementation of the CNM spec)

- Native Network driver
- Native IPAM driver
- Remote Network driver
- Remote IPAM driver

bridge, host, none, overlay

libnetwork built-in IPAM driver

weave
Contiv
Infoblox

Docker Engine
Default Bridge Driver

- Responsible for creating the docker0 bridge.
- Connects docker containers to the network using a veth pair
- Provides out-of-the-box support for bridge based container networking
- Allows creation of user-defined bridges

```
docker network create --driver bridge <name>
```
User Defined Bridge

- Container namespace
  - container 172.54.0.5
    - eth0
    - veth
  - Host
    - eth0 192.168.0.2
    - Host namespace
    - eg_bridge
    - veth
  - Container namespace
    - container 10.5.0.2/24
      - eth0
      - veth
  - Container namespace
    - container 10.5.0.3/24
      - eth0
      - veth
  - NAT
External Access for Containers

$ docker run -d --net eg_bridge -p 6000:80 nginx
- Multi-host networking
- First-class citizen in docker networking
- Uses swarm-distributed control plane for centralized mgmt, stability & security
- Uses VXLAN encap (decouples container n/w from physical n/w)
- Overlay datapath entirely in kernel space
Macvlan Driver

Diagram showing network namespaces and container addresses:

- **Host**: Container namespace with addresses 10.4.0.3/24 and 10.4.0.4/24, connected with eth0 interfaces.
- **macvlan** namespace with eth0 10.4.0.2.
- **Network Fabric** with Gateway 10.4.0.1/24.

Diagram for another configuration:

- **Host**: Container namespace with addresses 10.4.0.3/24 and 10.4.0.4/24, connected with eth0 interfaces.
- **macvlan** namespace with eth0 10.4.0.2 and additional macvlan50 and macvlan70 namespaces.
- **Network Fabric** with Gateway 10.4.0.1/24.
Default Networks Created by Docker

‘bridge’ using **bridge** driver, ‘none’ using **null** driver, ‘host’ using **host** driver

```
arun-neotrekker:~ arunsriraman$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
544fd2b5b674        bridge              bridge             local
790b79d68240        host                host               local
6aaec591a006        none                null               local
```

Don’t want the bridge driver? Remove it by specifying OPTIONS

```
/etc/sysconfig/docker
OPTIONS="--bridge=none --log-driver=json-file"
```
### Compare Docker Network driver types

<table>
<thead>
<tr>
<th>Driver Features</th>
<th>Bridge / User defined bridge</th>
<th>Host</th>
<th>Overlay</th>
<th>Macvlan / ipvlan</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connectivity</strong></td>
<td>Same host</td>
<td>Same host</td>
<td>Multi-host</td>
<td>Multi-host</td>
</tr>
<tr>
<td><strong>Namespace</strong></td>
<td>Separate</td>
<td>Same as host</td>
<td>Separate</td>
<td>Separate</td>
</tr>
<tr>
<td><strong>External connectivity</strong></td>
<td>NAT</td>
<td>Use Host gateway</td>
<td>No external connectivity</td>
<td>Uses underlay gateway</td>
</tr>
<tr>
<td><strong>Encapsulation</strong></td>
<td>No double encap</td>
<td>No double encap</td>
<td>Double encap using Vxlan</td>
<td>No double encap</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>North, South external access</td>
<td>Need full networking control, isolation not needed</td>
<td>Container connectivity across hosts</td>
<td>Containers needing direct underlay networking</td>
</tr>
</tbody>
</table>
Part II - Kubernetes Networking
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

Quoted from K8s docs
Kubernetes networking

- Container-to-Container communication
- Pod-to-Pod communication
- Pod-to-Service (cluster internal) communication
- External-to-Service (cluster external) communication

Container CIDR

Service CIDR
Container-to-Container

Pod

Group of one or more containers with shared storage/network

https://github.com/sarun87/examples/blob/master/images/K8s_Pod.png
Container-to-Container takeaways

- Containers in a pod run on the same host.
- A pod generally represents a service unit of an application.
- Uses localhost (127.0.0.1) within the pod’s network namespace to communicate with each other.
- Containers in the same Pod cannot reuse ports.
- Pause container - Keeps the networking alive.
- New concepts: Pod, Pause container.
Pod-to-Pod takeaways

Currently supported networking models -

• Kubenet via kubelet (will be moved out to CNI)
• Multiple network backends via CNI (We’ll discuss this in depth later)

Network backend responsible for -

• Pod networking setup
• Pod-to-Pod networking setup (uses L3 BGP like Calico, network overlay like weave, flannel)
• New concepts: Kubelet, CNI, network backend
Service - an abstraction which defines a logical set of Pods and a policy by which to access them.

- A service is “generally” backed by pods (endpoints) using a “label selector”.
- Users can explicitly define an endpoint that isn’t backed by pods.
- K8s defines many types of services:
  - Internal: ClusterIP
  - External: NodePort, LoadBalancer, Ingress
Pod-to-Service (Cluster Internal)

1. main-site-pod.main-ns.abc -> 192.168.0.22/24 (ClusterIP)

2. GET 192.168.0.22/24

3. GET 10.0.2.3/24

https://github.com/sarun87/examples/blob/master/images/K8s_Pod_to_Service.png
Pod-to-Service takeaways

- Service is a logical definition/collection of pods.
- ClusterIP is allocated from the Services CIDR
- kube-proxy modes
  - userspace
  - iptables (our discussed example)
- **New concepts:** kube-proxy, kube-dns, Service, clusterIP, iptables
External-to-Service

Service type: **NodePort**
- Kubernetes master allocates a port from a flag-configured range (default: 30000-32767).
- Each Node will proxy that port (the same port number on every Node) into your Service

![Diagram](https://github.com/sarun87/examples/blob/master/images/K8s_NodePort.png)
Service type: **LoadBalancer**

- Fronts the K8s Service
- Traffic from load balancer is directed to backend Pods
- Exactly how that works depends on the cloud provider
- NodePort and ClusterIP to which LB will route are created automatically
External-to-Service III

**Ingress**

- An Ingress is a collection of rules that allow inbound connections to reach the cluster services.
- Ingress is useful since services typically have internal IPs/endpoints.
- All traffic that ends up at an edge router is either dropped or forwarded elsewhere.
- Gives services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting.

**External IPs**

- A public/external IP points to a node of the cluster.
- Service ingresses the requests from the external IP.
- Are not managed by K8s.

Note: If you came here to understand ingress specifically, let’s chat offline. I will cover this if time permits.
CNI - Container Network Interface

- Simple interface between container runtime & network
- **CNCF** project. Started by CoreOS for the *rkt* runtime

- Config passed to the NetPlugin by runtime then passed to IPAM
- CNI Interfaces - ADD, DEL
## CNI - plugins

### CNI Maintained

Plugins that create/delete interfaces
- bridge
- ipvlan
- lo
- macvlan
- vlan
- ptp

IPAM - IP address management
- dhcp
- host-local

### 3rd party/others

- flannel (now under CNI)
- calico
- canal
- weave
- Cilium
- Contrail
- Contiv
- Infoblox
- Romana
- Nuage
- .....

Github repo - [https://github.com/containernetworking/cni](https://github.com/containernetworking/cni)
Using CNI with individual containers

Eg: host-local IPAM. To ADD n/w to a container

```
$ CNI_COMMAND=ADD \
CNI_CONTAINERID=arun_container_01 \ 
CNI_NETNS=/var/run/netns/cni_ipam_eg \ 
CNI_IFNAME=eth0 \ 
CNI_PATH=/home/ubuntu/cni/bin \ 
./host-local < sample_ipam_config

{  
   "cniVersion": "0.3.1",  
   "ips": [{  
      "version": "4",  
      "address": "10.10.10.2/24",  
      "gateway": "10.10.10.1"  
   }],  
   "dns": {  
   
}  
}
```

```
$ cat sample_ipam_config
{
   "cniVersion": "0.3.1",
   "name": "example-network",
   "ipam": {
      "type": "host-local",
      "subnet": "10.10.10.0/24",
      "dataDir": "/home/ubuntu/sample_ipam_datadir"
   }
}
```
Flannel network backend

- Uses overlay network for host-host connectivity
- Backends - UDP, vxlan
- flanneld binary runs on every host
- Does **not** perform host-container networking.
- Via CNI, flannel delegates interface operations to bridge driver.

https://github.com/coreos/flannel/blob/master/packet-01.png
Calico network backend

- Pure L3 based network solution
- Router per node
- Uses BGP
- via CNI plugin - has its own IPAM driver as well
- Supports Kubernetes NetworkPolicy constructs
- BIRD protocol (BGP stack)
- ACL and L3 forwarding performed in the linux kernel
- Ease of debugging
- Scalable

https://github.com/sarun87/examples/blob/master/images/Calico_Architecture.png
## CNI backends summarized

<table>
<thead>
<tr>
<th>Plugin Features</th>
<th>flannel</th>
<th>calico</th>
<th>weave</th>
<th>canal</th>
<th>Contiv</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main / Networking Plugin</strong></td>
<td>Forwards to bridge driver</td>
<td>Yes</td>
<td>Yes (via bridge plugin)</td>
<td>Yes, bridge driver</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>IPAM</strong></td>
<td>host-local</td>
<td>calico-ipam</td>
<td>Weave-ipam / host-local</td>
<td>host-local</td>
<td>Contiv ipam</td>
</tr>
<tr>
<td><strong>Host-to-host networking</strong></td>
<td>Overlay - UDP and VXLAN</td>
<td>BGP L3 routing based</td>
<td>Fast data-path and weave router sleeve (VXLAN)</td>
<td>Calico + Flannel</td>
<td>Overlay - VXLAN and VLAN based networks using a vSwitch</td>
</tr>
<tr>
<td><strong>K8s NetworkPolicy support</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Limited</td>
<td>L3 IP, Scalable</td>
<td>Scalable. Fast data-path makes it more efficient</td>
<td>Scalable with advantage of easy setup that flannel brings</td>
<td>Integrates with ACI fabric. Highly scalable with ACI</td>
</tr>
<tr>
<td><strong>Debugability</strong></td>
<td>Easy with UDP</td>
<td>Easy since it uses IP</td>
<td>Weave CLI has multiple debugging commands</td>
<td>Mix of calico+flannel</td>
<td>Community and documentation</td>
</tr>
</tbody>
</table>

More to come..
Thank You

Help me to better help you next time.

Questions/Feedback:

@arun_sriraman
Macvlan - VEPA mode

- Virtual Ethernet Port Aggregator is the default macvlan mode
- Data sent directly via ethernet card
- External devices should support hairpin/reflective relay
- Container traffic can be seen at phy switch
Macvlan - Bridge mode

- Containers on the same macvlan device are bridged
- No need to send traffic outside if target is on another macvlan device
- Trivial bridge with no learning required
- Simple & fast
Macvlan - Private mode

- Containers on the same macvlan device cannot talk to each other
- Container isolation
- External access allowed for all containers