DPDK Based Networking Products Enhance and Expand Container Networking

zhouzijiang@jd.com
Jingdong Digital Technology
Kubernetes Overview

- Pod to Pod communication
- Pod to Service communication
Flannel Overview

VXLAN encapsulation

<table>
<thead>
<tr>
<th>Outer Ethernet header</th>
<th>Outer IP header src: 192.168.0.1 dst: 192.168.0.2</th>
<th>Outer UDP header</th>
<th>Vxlan header</th>
<th>Inner Ethernet header</th>
<th>Inner IP header src: 10.10.10.2 dst: 10.10.20.3</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>eth0 10.10.10.2/24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eth0 10.10.10.3/24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bridge: cni0 10.10.10.1/24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vxlan: flannel.1 10.10.10.0/32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eth0 192.168.0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pod1</td>
<td>pod2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pod3</td>
<td>pod4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node1</td>
<td>node2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1、pods communicate with endpoints in k8s cluster, packets must be **encapsulated**
2、pods communicate with endpoints out of k8s cluster, packets must be **masqueraded**
   
   It will lead to extra overhead. Besides, it can’t meet some demands, e.g. pod wants to access white-list enabled application outside of k8s cluster

**Our goals:**
- no encapsulation
- no network address translation
- pods can be reached from everywhere directly

**Our Choice:**
- contiv with layer3 routing mode
Contiv Overview

- OVS to forward pod packets
- BGP to publish pod ip
1. User creates a new pod in k8s cluster
2. Netplugin requests a free ip 10.10.0.1 from netmaster
3. Netplugin creates a veth pair, such as vport1 and vvport1
4. Netplugin moves interface vport1 to pod network namespace and rename it to eth0
5. Netplugin sets ip and route in the pod network namespace
6. Netplugin adds vvport1 to ovs
7. Netplugin publishes 10.10.0.1/32 to bgp neighbor switch

- nw_dst=10.10.0.1  output:vvport1
- nw_dst=10.10.0.2  output:vvport2
Pod IP is Reachable in IDC Scope

10.10.0.2 (in cluster) ping 172.16.0.1 (outside cluster)

1、pod2 sends out packet through its eth0

```
Ethernet header
  src: 10.10.0.2
dst: 172.16.0.1
Payload
```

2、ovs receives packet from vvport2 and forwards it to host eth0

```
Ethernet header
  src: 10.10.0.2
dst: 172.16.0.1
Payload
```

3、switch receives packet and forwards it to host 172.16.0.1

```
Ethernet header
  src: 10.10.0.2
dst: 172.16.0.1
Payload
```

in the pod, in the host, in the underlying infrastructure, packet ip header is always the same
Contiv Optimization

1、multiple bgp neighbors support
2、reduce number of node’s ovs rules from magnitude of cluster to node
3、remove dns and load balance module from netplugin
4、add non-docker container runtime support, e.g. containerd
5、add ipv6 support
Load Balance: Native KubeProxy

```
<table>
<thead>
<tr>
<th></th>
<th>kube-proxy</th>
<th>iptables</th>
<th>eth0.100</th>
<th>eth0</th>
<th>eth0.200</th>
<th>inb01</th>
<th>vvport1</th>
<th>eth0</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod1</td>
<td>kube-proxy</td>
<td>iptables</td>
<td>eth0.100</td>
<td>eth0</td>
<td>eth0.200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pod2</td>
<td></td>
<td></td>
<td>eth0.100</td>
<td>eth0</td>
<td>eth0.200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pod3</td>
<td></td>
<td></td>
<td>eth0.100</td>
<td>eth0</td>
<td>eth0.200</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

.services endpoints
.data flow
.control flow

```
<table>
<thead>
<tr>
<th></th>
<th>kube-proxy</th>
<th>iptables</th>
<th>eth0.100</th>
<th>eth0</th>
<th>eth0.200</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pod1</td>
<td>kube-proxy</td>
<td>iptables</td>
<td>eth0.100</td>
<td>eth0</td>
<td>eth0.200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pod2</td>
<td></td>
<td></td>
<td>eth0.100</td>
<td>eth0</td>
<td>eth0.200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pod3</td>
<td></td>
<td></td>
<td>eth0.100</td>
<td>eth0</td>
<td>eth0.200</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Load Balance: DPDK-SLB

- Kube-Proxy on all nodes not needed
- SLB-Controller watches services and endpoints in K8S, dynamically sends VS and RS info to DPDK-SLB
DPDK-SLB: Control Plane

- SLB-Daemon: core process which does load balance and full NAT
- SLB-Agent monitors and configures SLB-Daemon
- OSPFD publishes service subnets to layer3 switch

- Admin core configures VS and RS info to worker cores
- KNI core forwards OSPF packets to kernel, the kernel then sends them to OSPFD
- Worker cores do the load balance

All data (config data, session data, local addrs) is per CPU, fully parallelizing packets processing
**DPDK-SLB: OSPF Neighbor**

- OSPF uses multicast address 224.0.0.5
- Flow Director: destination ip 224.0.0.5 bound to queue_x
- Dedicated KNI core to process OSPF packets
- OSPFD publishes service subnets to layer3 switch
DPDK-SLB: Data Plane

1. \{client\_ip, client\_port, vip, vport\}
2. rss selects a queue according to 5 tuple
3. worker\_1 does fullnat \{local\_ip1, local\_port, server\_ip, server\_port\}
4. worker\_1 saves session \{cip, cport, vip, vport, lip1, lport, sip, sport\}

The key point is that server-to-client packet must be placed on queue1, because only worker\_1 has the session.

1. \{server\_ip, server\_port, local\_ip1, local\_port\}
2. fdir selects a queue according to destination ip addr(local\_ip1 bound to queue\_1)
3. worker\_1 lookups session \{cip, cport, vip, vport, lip1, lport, sip, sport\}
4. worker\_1 does fullnat \{vip, vport, client\_ip, client\_port\}
Make Apps Run in the Container Cloud Seamlessly

- layer3 switch routes:
  10.10.0.1 nexthop node1
  10.10.0.4 nexthop node2
  service subnets nexthop dpdk-slb

- Pod IP can be reachable from vm1 outside k8s cluster
- Service IP can be reachable from vm2 outside k8s cluster
- Help apps to run in the container cloud and traditional environment at the same time
Thank You!

Q & A

zhouzijiang@jd.com

Jingdong Digital Technology