Kubernetes网络

前言

- 自Docker技术诞生以来，使用容器技术用于开发、测试甚至是生产环境的企业或组织与日俱增。然而，在生产环境中使用合适的网络方案部署基于容器的服务依然是一个巨大的挑战。这不但涉及网络中各组件的互连互通方式，也涉及到容器间通过隔离以确保安全性。本章主要讲述容器网络模型的演进和Kubernetes的网络模型及插件。

目标

- 学完本课程后，您将能够：
  - 描述Kubernetes网络模型及实现方式
  - 区别主流CNI插件及相关技术

目录

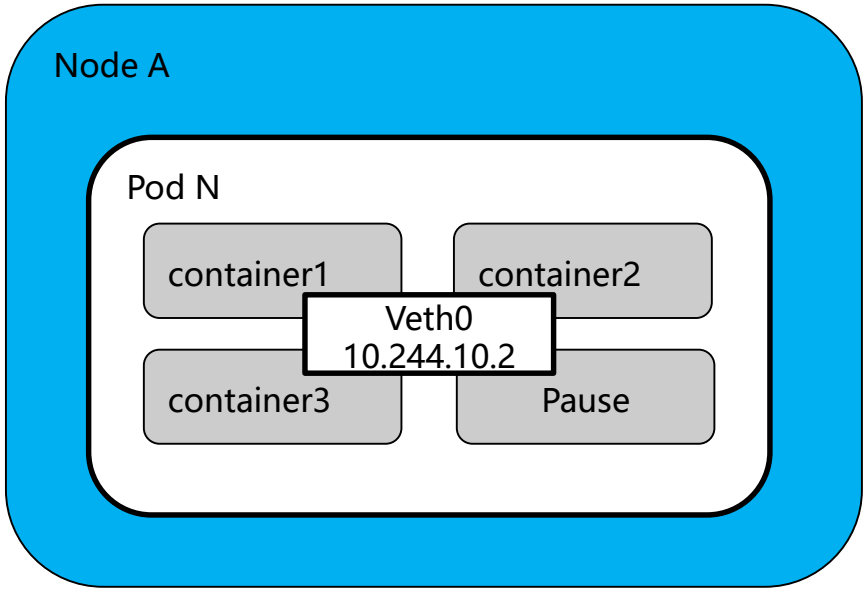
Kubernetes的网络模型设计目标

- Kubernetes网络模型用于满足以下四类通信需求：

1. 同一Pod内容器间的通信（Container to Container）
2. Pod间的通信（Pod to Pod）
3. Service到Pod间的通信（Service to pod）
4. 集群外部与Service之间的通信（external to Service）

容器间通信

- Pod内各容器共享同一网络名称空间，该名称空间通常由基础架构容器所提供。
  - 例如，由pause镜像启动的容器。所有运行于同一Pod内的容器与同一主机上的多个进程类似，彼此之间可通过环回接口完成交互，如图所示，Pod N内的Container1、Container 2、Container 3之间的通信即为容器间通信。

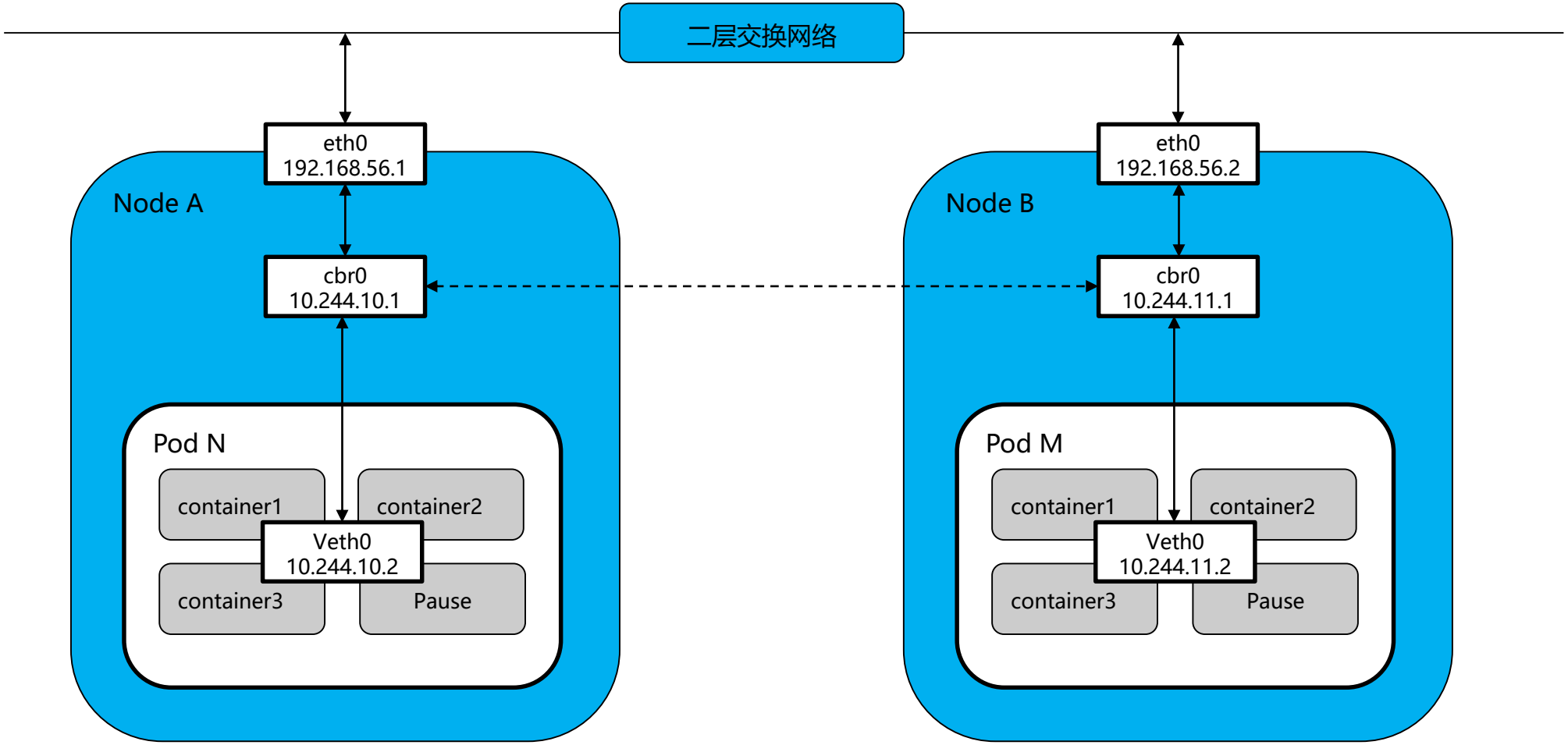


- 不同Pod之间不存在端口冲突的问题，因为每个Pod都有自己的IP地址。

Pod间通信

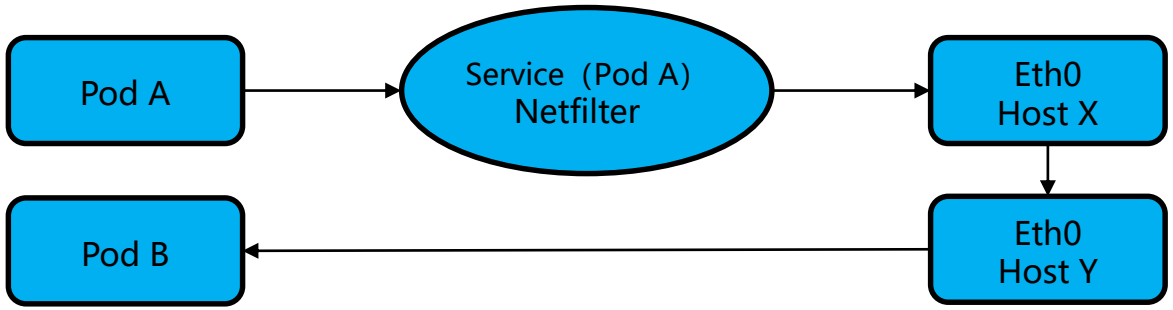
- Pod的IP是集群可见的，即集群中的任何其他Pod和节点都可以通过IP直接与Pod通信，这种通信不需要借助任何的网络地址转换、隧道或代理技术。Pod内部和外部使用的是同一个IP，这也意味着标准的命名服务和发现机制，比如DNS可以直接使用。
- 此类通信模型中的通信需求也是Kubernetes的各网络插件需要着力解决的问题，它们的实现方式有叠加网络模型和路由网络模型等，流行的解决方案有十数种之多，例如前面使用到的flannel。

Pod间通信示意图



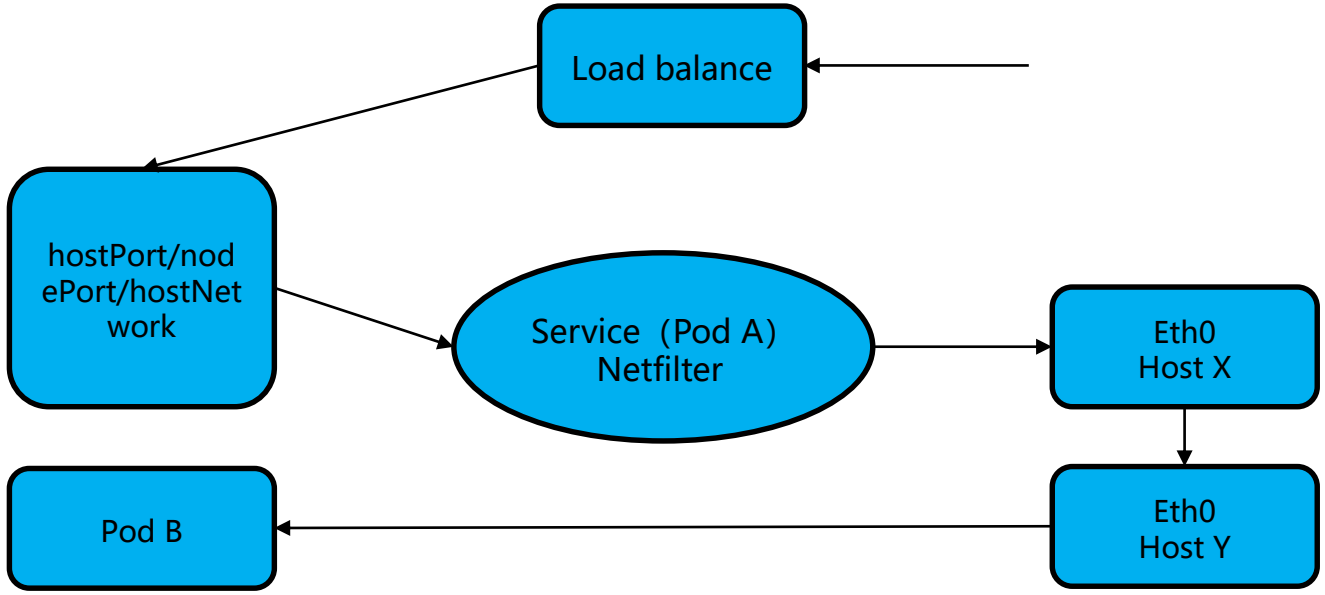
Service与Pod间的通信

- Pod间可以直接通过IP地址通信，但前提是Pod得知道对方的IP。在Kubernetes集群中，Pod可能会频繁的销毁和创建，也就是说Pod的IP不是固定的。
- 为了解决这个问题，Service提供了访问Pod的抽象层。无论后端的Pod如何变化，Service都作为稳定的前端对外提供服务。同时，Service还提供了高可用和负载均衡功能，Service负责将请求转发给正确的Pod。
- Service资源的专用网络也称为集群网络(Cluster Network)。



集群外部访问

- Kubernetes提供了两种方式让外界能够与Pod通信：
  - NodePort：Service通过Cluster节点的静态端口对外提供服务。外部可以通过<NodeIP>:<NodePort>访问Service。
  - LoadBalancer：Service利用load balancer（服务或设备）对外提供服务，LB负责将流量导向Service。



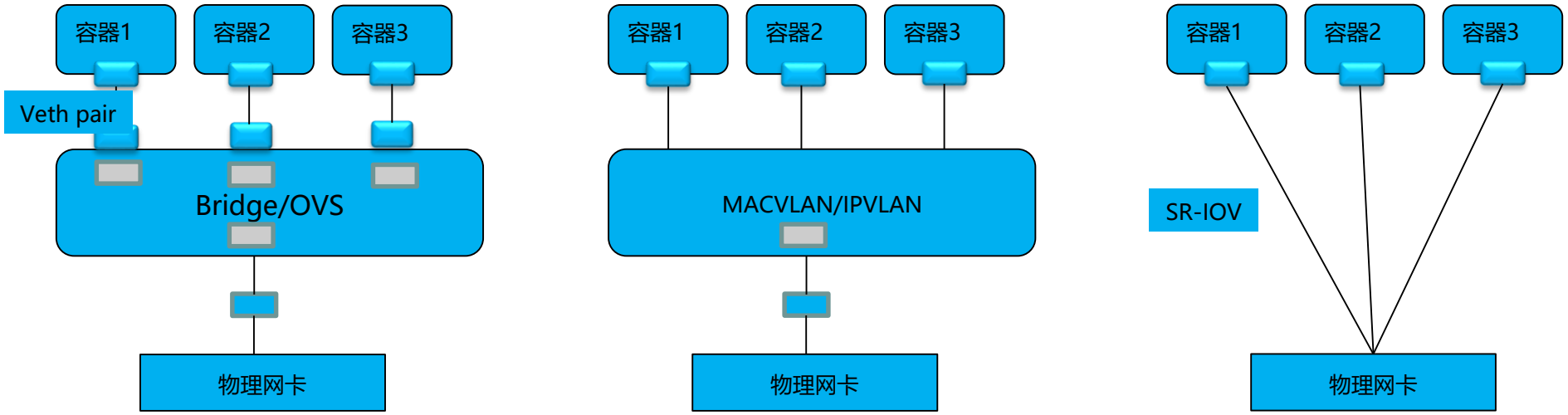
目录

Pod网络的实现方式

- 每个Pod对象内的基础架构容器均使用一个独立的Network namespace。
- 该Network namespace会共享给同一Pod内其他容器使用。
- 每个Network namespace均有其专用的独立网络协议栈及其相关的网络接口设备。
- 一个网络接口仅能属于一个Network namespace。
- 显然的，运行多个Pod必然要求使用多个Network namespace，也就需要用到多个网络接口设备。

虚拟网络接口的实现方式

- 虚拟网络接口的实现方案常见的有虚拟网桥、多路复用及硬件交换三种。



虚拟网络接口实现原理

ip link show

虚拟网桥:

- 创建一对虚拟以太网接口(veth)，一个接入容器内部，另一个置于根namespace内并借助于Linux内核桥接功能或Open VSwitch(OVS)关联至真实的物理接口。

多路复用:

- 多路复用可以由一个中间网络设备组成，它暴露了多个虚拟接口，可使用数据包转发规则来控制每个数据包转到的目标接口。

硬件交换:

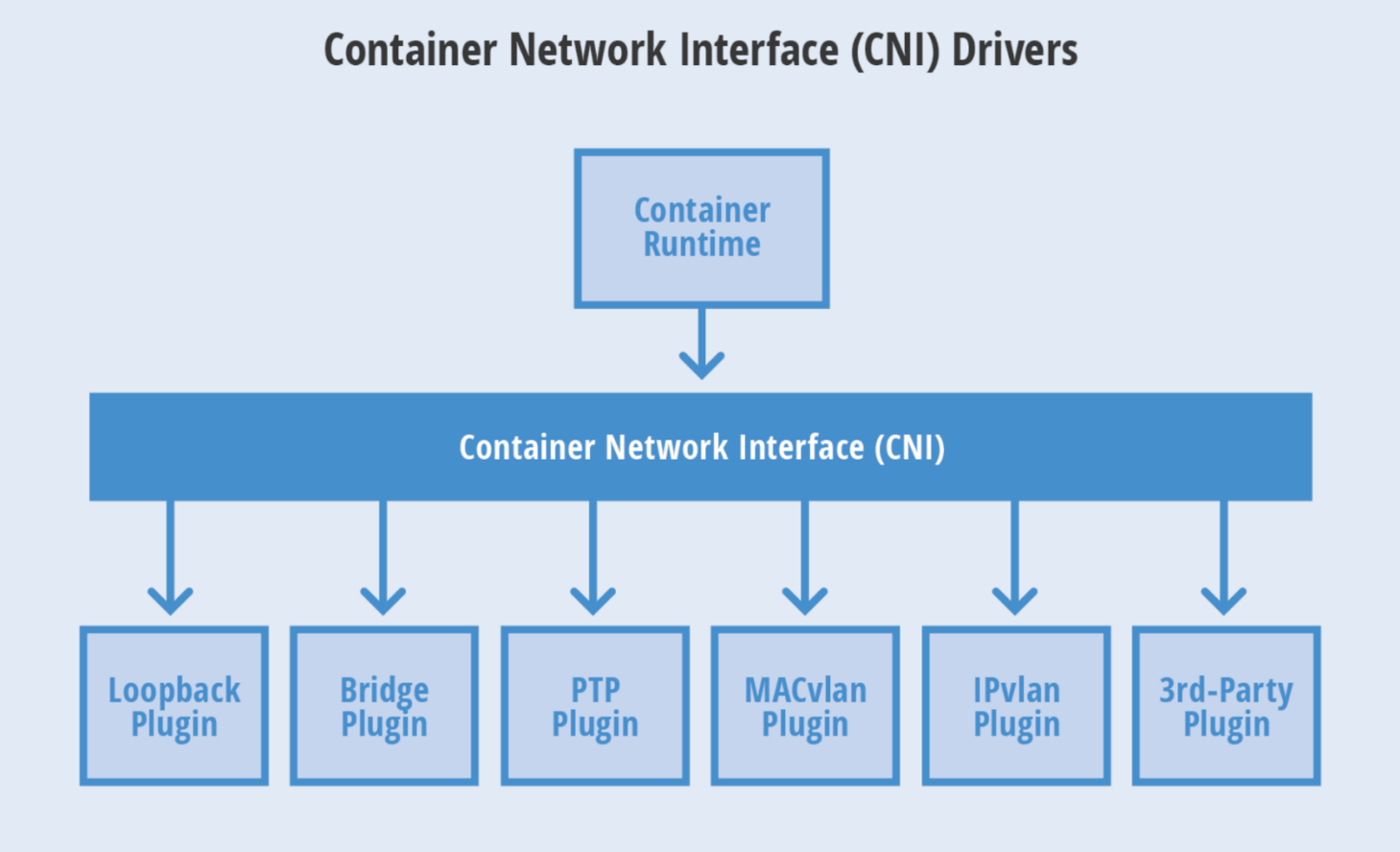
- 现今市面上的大多数NIC都支持单根I/O虚拟化(SR-IOV)，它是创建设备的一种实现方式。每个虚拟设备自身均表现为一个独立的PCI设备，并有自己的VLAN及与硬件强制关联的QoS。SR-OV提供了接近硬件级别的性能，但在公共云中通常是不可用的。

目录

容器网络模型规范CNI

- Kubernetes设计了网络模型，但将其实现交给了网络插件。于是，各种解决方案不断涌现。
- 为了规范及兼容各种解决方案，CoreOS和Google联合制定了CNI(Container Network Interface)标准，旨在定义容器网络模型规范。它连接了两个组件：容器管理系统和网络插件。
- CNI的设计思想是: 容器runtime在创建容器时，提前创建好Network namespace，然后调用CNI插件为这个Network namespace配置网络，而后再启动容器内的进程。

CNI Drivers



CNI的优点是支持多种容器runtime，不仅仅是Docker。CNI的插件模型支持不同组织和公司开发的第三方插件，这对运维人员来说很有吸引力，可以灵活选择适合的网络方案。

主流CNI插件项目

- Flannel：一个为Kubernetes提供叠加网络的网络插件，它基于Linux TUN/TAP，用UDP封装IP报文来创建叠加网络，并借助etcd维护网络的分配情况。
- Calico：一个基于BGP的三层网络插件，并且也支持网络策略来实现网络的访问控制；它在每台机器上运行一个vRouter，利用Linux内核来转发网络数据包，并借助iptables实现防火墙等功能。
- Canal：由Flannel和caco联合发布的一个统一网络插件，提供CNI网络插件，并支持网络策略。

目录

Flannel网络插件

- 各类CNI插件都至少要解决这两类问题：
  - 各运行容器的主机在网桥上默认使用同一个子网，跨节点会面临地址冲突的问题。
  - 即使跨节点时使用不同的子网，其报文也会因为缺乏路由信息而无法到达目的地址。

- 针对第一个问题，Flannel的解决方法是预留一个大网段，如10.244.0.0/16，而后自动为每个节点的Docker容器引擎分配一个子网，如10.244.1.0/24和10.244.2.0/24。子网和子网间能正常通信。
- 针对第二个问题，Flannel通过各种不同的后端（网络模型）来解决。

Flannel三种网络模型

- Flannel的后端也可称为网络模型。

VXLAN	• flannel的此种后端使用系统内核中的VXLAN模块封装报文，并通过隧道转发机制实现跨节点的Pod间通信，这也是Flannel的主流方式。
host-gw	• 即Host GateWay， 通过在节点上创建到达目标容器地址的路由直接完成报文转发。
UDP	• 使用普通UDP报文封装完成隧道转发，性能较低，仅在前两种方式无法支持时使用。

部署Flannel

- Flannel可以通过yml文件直接在kubernetes集群中部署。Flannel安装：

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

- 查看：

```
# kubectl get pod -o wide -n kube-system
```

```
NAME                                READY   STATUS    RESTARTS   AGE    IP              NODE              NOMINATED NODE
...
kube-apiserver-master1.vic.com      1/1     Running   0          3d5h   192.168.56.87   master1.vic.com   <none>
kube-flannel-ds-amd64-hxb7p         1/1     Running   0          3d4h   192.168.56.87   master1.vic.com   <none>
kube-flannel-ds-amd64-rjstm         1/1     Running   1          3d4h   192.168.56.89   node2.vic.com     <none>
kube-flannel-ds-amd64-zxgnn         1/1     Running   1          3d4h   192.168.56.90   node3.vic.com
…
```

测试Pod连通性

- 查询已有的Pod信息。

```
# kubectl get pod  -o wide
```

```
NAME                              READY   STATUS    RESTARTS   AGE     IP            NODE             NOMINATED NODE
nginx-deploy-8c5fc574c-67mp6      1/1     Running   0          3m43s   10.244.3.4    node2.vic.com    <none>
nginx-deploy-8c5fc574c-gv6rs      1/1     Running   0          3m43s   10.244.3.3    node2.vic.com    <none>
nginx-deploy-8c5fc574c-smtzb      1/1     Running   0          3m43s   10.244.2.5    node3.vic.com    <none>
…
```

- 测试Pod和主机之间连通性：

```
#ping 10.244.3.4
```

```
PING 10.244.3.4 (10.244.3.4) 56(84) bytes of data.
64 bytes from 10.244.3.4: icmp_seq=1 ttl=63 time=2.06 ms
64 bytes from 10.244.3.4: icmp_seq=2 ttl=63 time=0.432 ms
…
```

- 思考：如何进行Pod和Pod之间的连通性测试。

查看Flannel参数

```
# kubectl get cm kube-flannel-cfg -n kube-system
```

```
manifestsNAME             DATA    AGE
kube-flannel-cfg          2       3d1h
…
```

```
#kubectl get cm kube-flannel-cfg -n kube-system -o yaml
```

```
#kubectl get cm kube-flannel-cfg -n kube-
system -o yaml
apiVersion: v1
data:
  cni-conf.json: |
    {
      "name": "cbr0",
      "plugins": [
        {
          "type": "flannel",
          "delegate": {
            "hairpinMode": true,
            "isDefaultGateway": true
          }
        },
…
```

```
 "type": "portmap",
          "capabilities": {
            "portMappings": true
          }
        }
      ]
    }
  net-conf.json: |
    {
      "Network": "10.244.0.0/16",
      "Backend": {
        "Type": "vxlan"
      }
    }
```

flannel默认使用VXLAN作为后端

VXLAN

- VXLAN(Virtual extensible Local Area Network)：虚拟可扩展局域网，是VLAN展方案，采用MAC in UDP的封装方式，属于NVo3(Network Virtualization overayer3)中的一种网络虚拟化技术。VXLAN广泛应用于云数据中心的软硬件SDN网络解决方案中。

- 具体实现方式为：给普通数据报文添加到VXLAN报头后封装为外网可路由的UDP报文，然后以传统网络的通信方式传输该报文，待其到达目的主机后，去掉外网报头以及VXLAN报头，然后将报文转发给目的终端，整个过程中通信双方对物理网络无所感知。

- 上述过程相当于在传统的网络之上架起了由一条条VXLAN隧道构成的“高架”网络。

- VXLAN技术的引入使得逻辑网络拓扑和物理网络拓扑实现了一定程度的解耦，网络拓扑的配置对于物理设备的配置的依赖程度有所降低，配置更灵活更方便。

VXLAN后端和direct routing

- 传统的VXLAN后端使用隧道的通信方式会导致报文开销及流量增大。
- 于是Flannel的VXLAN后端还支持DirectRouting模式，该模式通过添加路由信息的方式，直接使用节点的二层网络进行Pod的通信报文的交互，仅在跨网段时才启用传统的隧道方式转发通信流量。
- 由于大部分场景中都省去了隧道首部开销，因此DirectRouting通信模式的性能基本接近于二层物理网络。

启用direct routing

- 仅使用VXLAN，未启用directrouting

```
# ip route show
```

```
default via 192.168.56.2 dev ens33
10.244.0.0/24 dev cni0 proto kernel scope link src 10.244.0.1
10.244.1.0/24 via 10.244.1.0 dev flannel.1 onlink
10.244.2.0/24 via 10.244.2.0 dev flannel.1 onlink
10.244.3.0/24 via 10.244.3.0 dev flannel.1 onlink
169.254.0.0/16 dev ens33 scope link metric 1002
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
192.168.56.0/24 dev ens33 proto kernel scope link src 192.168.56.87 …
```

- 启用directrouting后（启用步骤见下页胶片）

```
# ip route show
```

```
…
10.244.2.0/24 via 192.168.56.68 dev ens33
10.244.3.0/24 via 192.168.56.69 dev ens33
…
```

使配置生效

- 将flannel项目官方提供的配置清单下载至本地:
  - /etc/kubernetes/manifests/kube-flannel.yaml
- 卸载Flannel后将其ConfigMap资源kube-flannel-cfg的data字段的网络配置部分修改为如下

```
    net-conf.json: |
        {
          "Network": "10.244.0.0/16",
          "Backend": {
            "Type": "vxlan"
            "Directrouting": true
          }
        }
```

- 使用“ kubectl apply”命令重新应用于集群中

```
# kubectl apply -f kube-flannel.yaml
```

- 查看节点路由表

```
…
10.244.2.0/24 via 192.168.56.68 dev ens33
10.244.3.0/24 via 192.168.56.69 dev ens33
```

- 注意：该配置推荐在集群网络初始化时做

host-gw后端

- host-gw后端通过添加路由信息使节点直接发送Pod的通信报文，其工作方式类似于VXLAN后端的direct routing的功能，但不包括其 VXLAN的隧道转发能力。
- 配置方式类似上述direct routing配置方式

```
    net-conf.json: |
        {
          "Network": "10.244.0.0/16",
          "Backend": {
            "Type":  “host-gw"
          }
        }
```

网络策略

- 网络策略（Network Policy）是用于控制分组的Pod资源之间如何进行通信及分组的Pod资源如何与其他网络端点进行通信。
- 网络策略是Kubernetes的一种资源。Network Policy通过Label选择Pod，并指定其他Pod或外界如何与这些Pod通信。
- 默认情况下，所有Pod都是未隔离的，即任何来源的网络流量都能够访问Pod，没有任何限制。当为Pod定义了Network Policy，只有Policy允许的流量才能访问Pod。
- 不是所有的Kubernetes网络方案都支持Network Policy。比如Flannel就不支持，Calico是支持的。

知识小考

- 请阐述VXLAN direct routing 后端的优势。

本章总结

- Kubernetes的网络模型：容器间通信、Pod间通信、Service与Pod间的通信以及集群外部流量与Pod间的通信四种通信。
- Kubernetes网络模型通过具备CNI接口外部网络插件来实现，如Flannel、Calico和Canal等。
- Flannel支持host-gw、 VXLAN和UDP等后端，默认为VXLAN。


谢 谢
www.huawei.com