在 IPv6 集群上部署 Cilium

利用 Linux 原生路由在二层和三层网络上组建 Kubernetes 集群.

---

1. Cilium 配置

经过一个下午的调试, 我成功地部署了只有 IPv6 的 K8S 集群 (kubeadm bootstrap).

Cilium 版本是 1.19.0.

```
cilium status
    /¯¯\
 /¯¯\__/¯¯\    Cilium:             OK
 \__/¯¯\__/    Operator:           OK
 /¯¯\__/¯¯\    Envoy DaemonSet:    OK
 \__/¯¯\__/    Hubble Relay:       OK
    \__/       ClusterMesh:        disabled

DaemonSet              cilium                   Desired: 3, Ready: 3/3, Available:
3/3
DaemonSet              cilium-envoy             Desired: 3, Ready: 3/3, Available:
3/3
Deployment             cilium-operator          Desired: 1, Ready: 1/1, Available:
1/1
Deployment             hubble-relay             Desired: 1, Ready: 1/1, Available:
1/1
```

可以从上面看到, 我的 DaemonSet 有 2 个 Pod. 也就是有一个 3 节点的集群. 事实上, 其中两台是 Dell R630, 和网关放在同一个交换机下. 还有一台是家中的软路由, 通过 wireguard 连接到网关. 属于既有二层, 又有三层的场景.

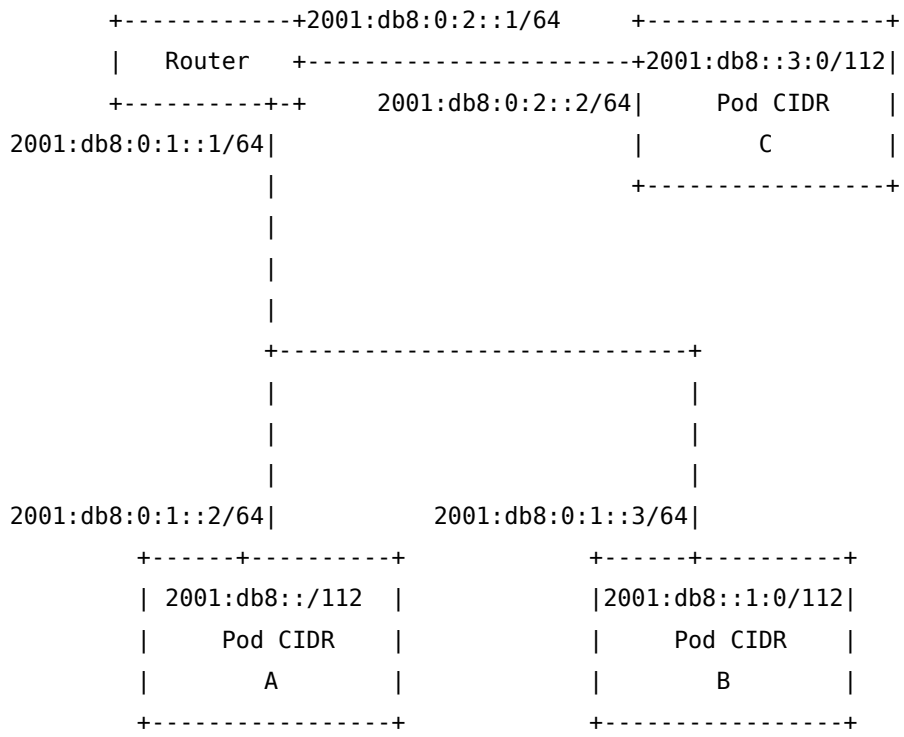

图 1　A, B 和 C 的拓扑

运行

```
cilium connectivity perf
```

发现集群成功跑到了 10Gps, 充分利用了天空工场集群的 10G 网络.

此外, Cilium 自动为所有 Pod 配置了与宿主机相同的 MTU, 现在的 Pod 互联也都是巨型帧了.

具体的配置如下:

```
routingMode: native
ipam:
  mode: kubernetes
ipv4:
  enabled: false
ipv6:
  enabled: true
prometheus:
  enabled: true
  serviceMonitor:
    enabled: true
operator:
  replicas: 1
  prometheus:
    enabled: true
    serviceMonitor:
      enabled: true
  dashboards:
```

```
    enabled: true
envoy:
  prometheus:
    enabled: true
    serviceMonitor:
      enabled: true
dashboards:
  enabled: true
k8s:
  requireIPv6PodCIDR: true
  requireIPv4PodCIDR: false
kubeProxyReplacement: true
autoDirectNodeRoutes: true
directRoutingSkipUnreachable: true
ipv6NativeRoutingCIDR: "2001:db8::/32"
enableIPv6Masquerade: false
bpf:
  lbExternalClusterIP: true
hubble:
  enabled: true
  metrics:
    enabled:
      - dns:query
      - drop
      - tcp
      - flow
      - icmp
      - http
    serviceMonitor:
      enabled: true
    dashboards:
      enabled: true
  relay:
    enabled: true
    prometheus:
      enabled: true
      serviceMonitor:
        enabled: true
```

2. Linux 原生路由

这段配置最重要的是什么呢? 其实只有这几段:

```
# 原生路由模式, 不使用 VXLAN/Geneve 隧道封装
routingMode: native
# 使用 Cilium eBPF datapath 替代 kube-proxy 处理 Kubernetes Service 转发
```

```yaml
kubeProxyReplacement: true
# 如果各节点位于同一个二层网络, 自动在节点之间添加到对端 PodCIDR 的直连路由
autoDirectNodeRoutes: true
# 该范围内的目的地址交给 Linux 路由栈处理, 不需要做 SNAT
ipv6NativeRoutingCIDR: "2001:db8::/32"
# 禁用 Cilium 对 Pod/endpoint 发出的 IPv6 流量做 masquerade
enableIPv6Masquerade: false
# 允许从集群外访问 ClusterIP 类型的 Service
bpf:
  lbExternalClusterIP: true
```

对于从来没有了解过 k8s 网络的读者. k8s 的网络需要达到下面的目标:

1. 节点之间相互通信
2. 同一节点的 Pod 相互通信
3. 不同节点的 Pod 相互通信
4. Pod 和宿主节点通信
5. Pod 和其他节点通信
6. 实现 Service CIDR 转发

因此, 再回顾 图 1, 在原生路由下, 网关配置路由表, 就已经可以实现除了 Service CIDR 转发的目标了.

```
2001:db8:0:1::/64 dev eth0
2001:db8::/112 via 2001:db8:0:1::1 dev eth0
2001:db8::1:0/112 via 2001:db8:0:1::2 dev eth0

2001:db8:0:2::/64 dev wg0
2001:db8::2:0/112 via 2001:db8:0:2::1 dev wg0
```

而 Cilium 自己会通过 BPF 处理送往 Service CIDR 的包, 这一步可以理解为, Cilium 把目的地址重写为 Pod 地址 (即 DNAT), 等包送回后, 再将地址改回对应的 Service IP 送出节点.

在以上的配置下, 任何一个节点都可以接受目的地址为 Service CIDR 的包.