Was ist der Unterschied zwischen Kubeproxy und einem CNI

In Kubernetes gibt es zwei Arten von IP: ClusterIP und Pod IP.

CNI

CNI kümmert sich um die Pod-IP.

Das CNI-Plugin konzentriert sich auf den Aufbau eines Overlay-Netzwerks, ohne das Pods nicht miteinander kommunizieren können. Die Aufgabe des CNI-Plugins besteht darin, dem Pod bei der Planung eine Pod-IP zuzuweisen, für diese IP ein virtuelles Gerät zu erstellen und diese IP von jedem Knoten des Clusters aus zugänglich zu machen.

In Calico wird dies durch N Host-Routen (N=the number of cali veth device) und M direkte Routen auf tun0 (M = die Anzahl der K8s-Cluster-Knoten) implementiert.

```
$ route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 10.130.29.1 0.0.0.0 UG 100 0 0 ens32
10.130.29.0 0.0.0.0 255.255.255.0 U 100 0 0 ens32
10.244.0.0 0.0.0.0 255.255.255.0 U 0 0 0 *
10.244.0.137 0.0.0.0 255.255.255.255 UH 0 0 0 calid3c6b0469a6
10.244.0.138 0.0.0.0 255.255.255.255 UH 0 0 0 calidbc2311f514
10.244.0.140 0.0.0.0 255.255.255.255 UH 0 0 0 califb4eac25ec6
10.244.1.0 10.130.29.81 255.255.255.0 UG 0 0 0 tunl0
10.244.2.0 10.130.29.82 255.255.255.0 UG 0 0 0 tunl0
```

In diesem Fall ist `10.244.0.0/16` das Pod-IP-CIDR und `10.130.29.81` ein Knoten im Cluster. Du kannst Dir sich vorstellen, dass eine TCP-Anfrage an 10.244.1.141 gemäß der 7. Regel an 10.130.29.81 gesendet wird. Und auf 10.130.29.81 wird es eine Routenregel wie diese geben:

```
10.244.1.141 0.0.0.0 255.255.255.255 UH 0 0 0 cali4eac25ec62b
```

Dadurch wird die Anfrage schließlich an den richtigen Pod gesendet.

kube-proxy

Die Aufgabe von kube-proxy ist recht einfach, er leitet lediglich Anfragen von der Cluster-IP zur Pod-IP um.

Kube-Proxy verfügt über zwei Modi: IPVS und Iptables. Wenn Dein Kube-Proxy im IPVS-Modus arbeitet, kannst Du die von Kube-Proxy erstellten Umleitungsregeln anzeigen, indem Du den folgenden Befehl auf einem beliebigen Knoten im Cluster ausführst:

```
ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 10.96.0.1:443 rr
-> 10.130.29.80:6443 Masq 1 6 0
-> 10.130.29.81:6443 Masq 1 1 0
-> 10.130.29.82:6443 Masq 1 0 0
TCP 10.96.0.10:53 rr
-> 10.244.0.137:53 Masq 1 0 0
-> 10.244.0.138:53 Masq 1 0 0
...
```

In diesem Fall siehst Du die Standard-Cluster-IP von CoreDNS 10.96.0.10 und dahinter zwei echte Server mit Pod-IP: 10.244.0.137 und 10.244.0.138.

Diese Regel ist, was kube-proxy erstellen soll, und sie ist es, was kube-proxy erstellt hat.

P.S. Der Iptables-Modus ist fast derselbe, aber die Iptables-Regeln sehen hässlich aus. Ich möchte es hier nicht einfügen. :P