# Container Network Interface. Обзор Сетевых подсистем.

# Не забудь включить запись!

# План

- Основные принципы работы сети в k8s
- Как работает сеть в k8s
- L3 vs. L2
- Pause containers
- Взаимодействие CNI, runtime, kubelet
- Обзор существующих плагинов
- Flanel
- Calico
- Canal
- Производительность плагинов
- Проблемы связанные с сетью

# Rules

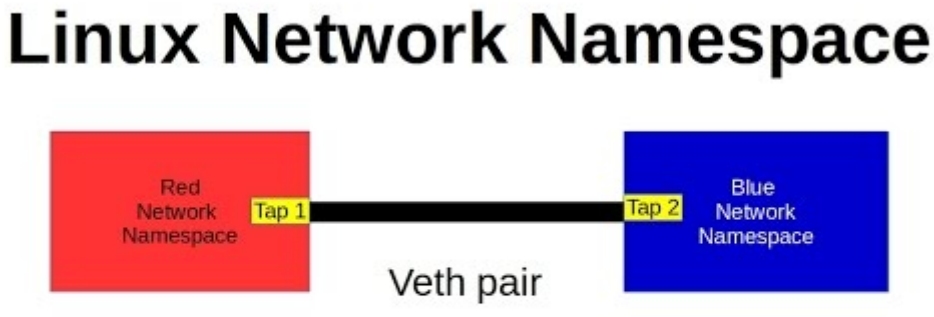
**В основе сетевой модели Kubernetes лежат правила:**

- Все контейнеры могут общаться со всеми контейнерами без использования NAT
- Все узлы могут общаться со всеми контейнерами без использования NAT
- IP, который видит контейнер, должен быть таким же, как его видят другие

# Rules

- Все служебные утилиты (kubelet, system daemons) могут общаться с всеми контейнерами на ноде
- Контейнеры которые используют хостовую сеть могут общаться со всеми контейнерами на всех нодах без использования NAT
- У каждого пода свой уникальный IP

# Network namespaces



# Network namespaces

- Технология ядра linux (такие как pid, mnt, uts, ipc, user)
- Linux kernel 2.6.24
- Использует системный вызов clone(), переданный флаг CLONE_NEWNET создаст новый ns для нового процесса
- Каждый ns имеет свой набор интерфейс и таблицу маршрутизации, не зависят от других ns

# Pause containers

- Резервирование и удержание пространства имен используемого всеми контейнерами пода.
- Управление процессами (Kill all zombies)
- IP-per-pod

# Kube proxy

- Запускается на каждой ноде
- Проксирует UDP, TCP и SCTP
- Не понимает HTTP ^_^
- Предаставляет балансировку нагрузки

# Kube proxy

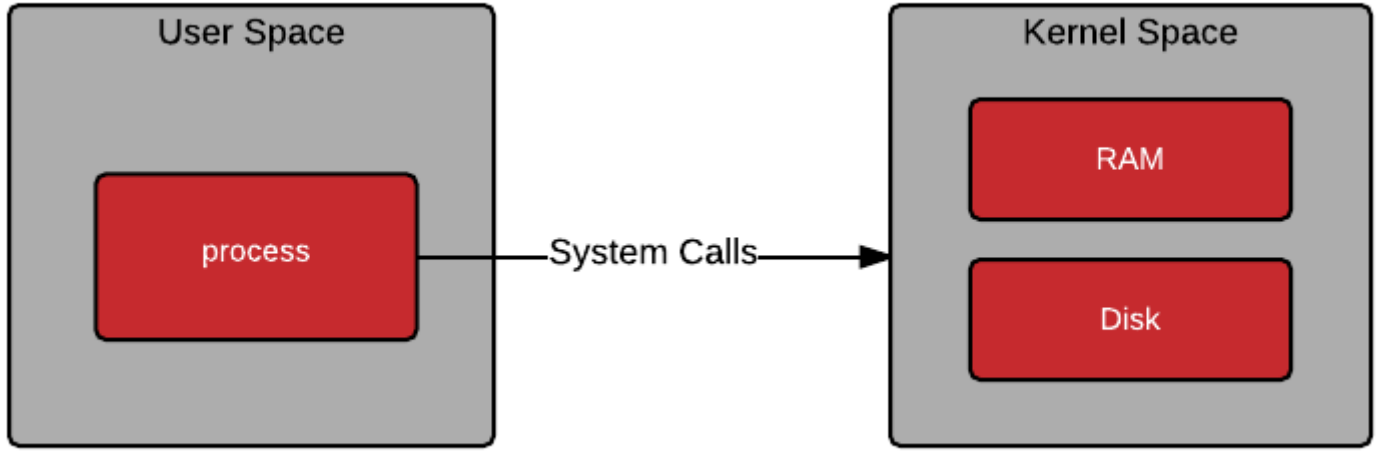
- Netfilter
- iptables

# Kube proxy

- userspace
- iptables
- IPVS

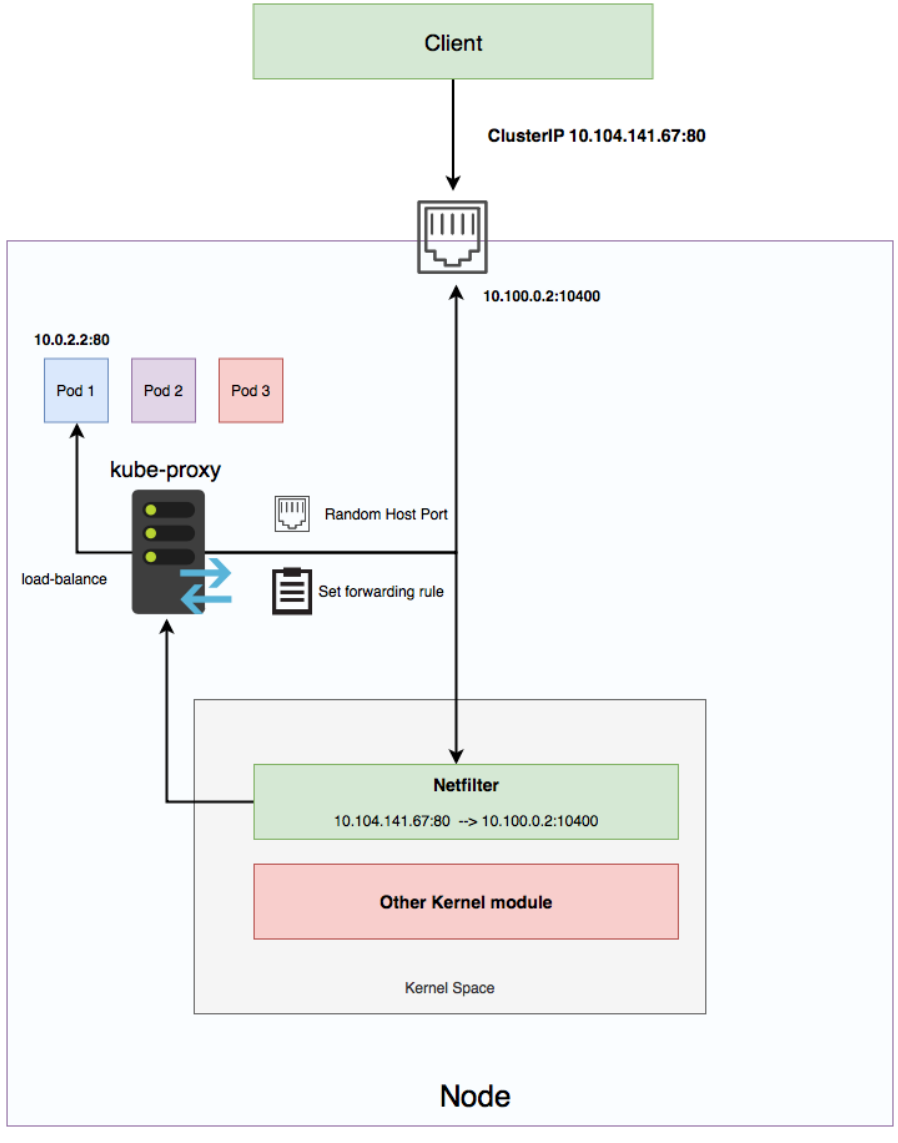
# Kube proxy

## Userspace vs. Kernel namespace



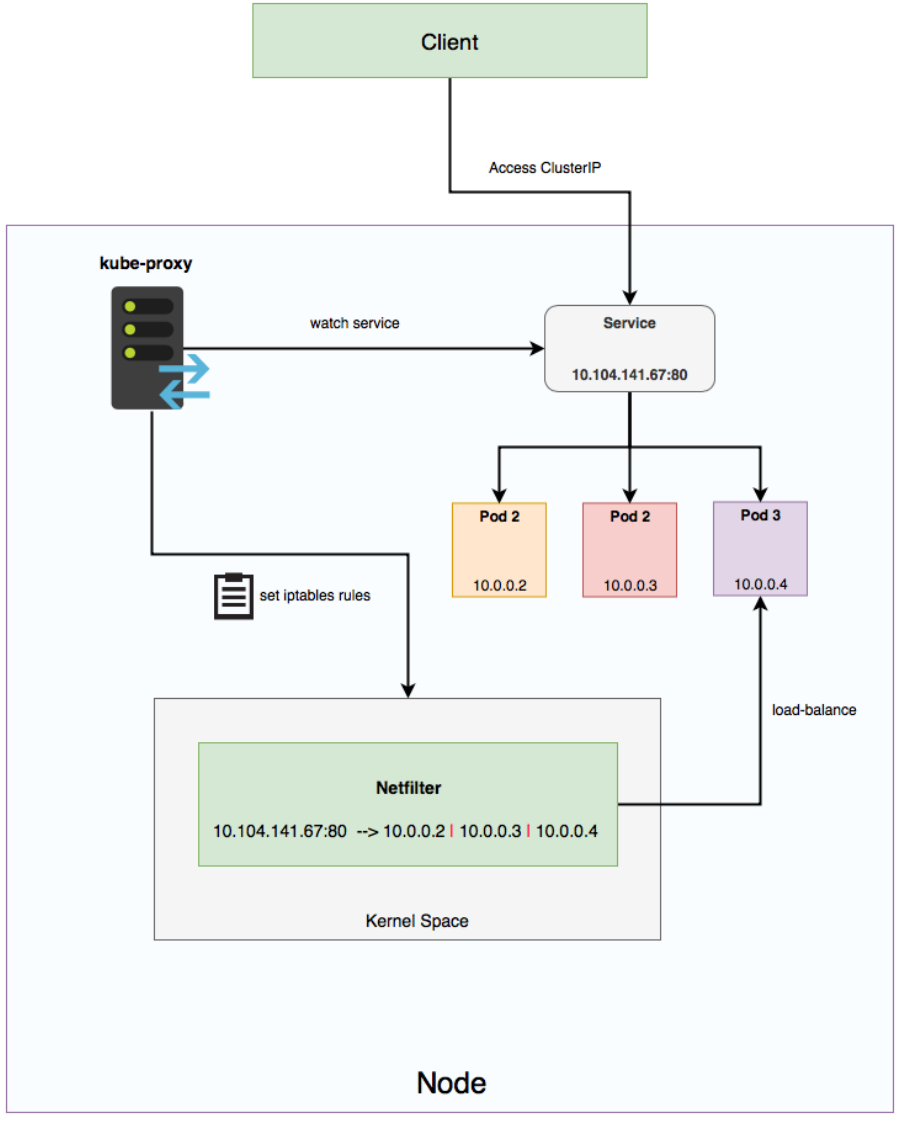
# Kube proxy

## Userspace mode


Client
ClusterIP 10.104.141.67:80
10.100.0.2:10400
10.0.2.2:80
Pod 1
Pod 2
Pod 3
kube-proxy
load-balance
Random Host Port
Set forwarding rule
Netfilter
10.104.141.67:80 --> 10.100.0.2:10400
Other Kernel module
Kernel Space
Node

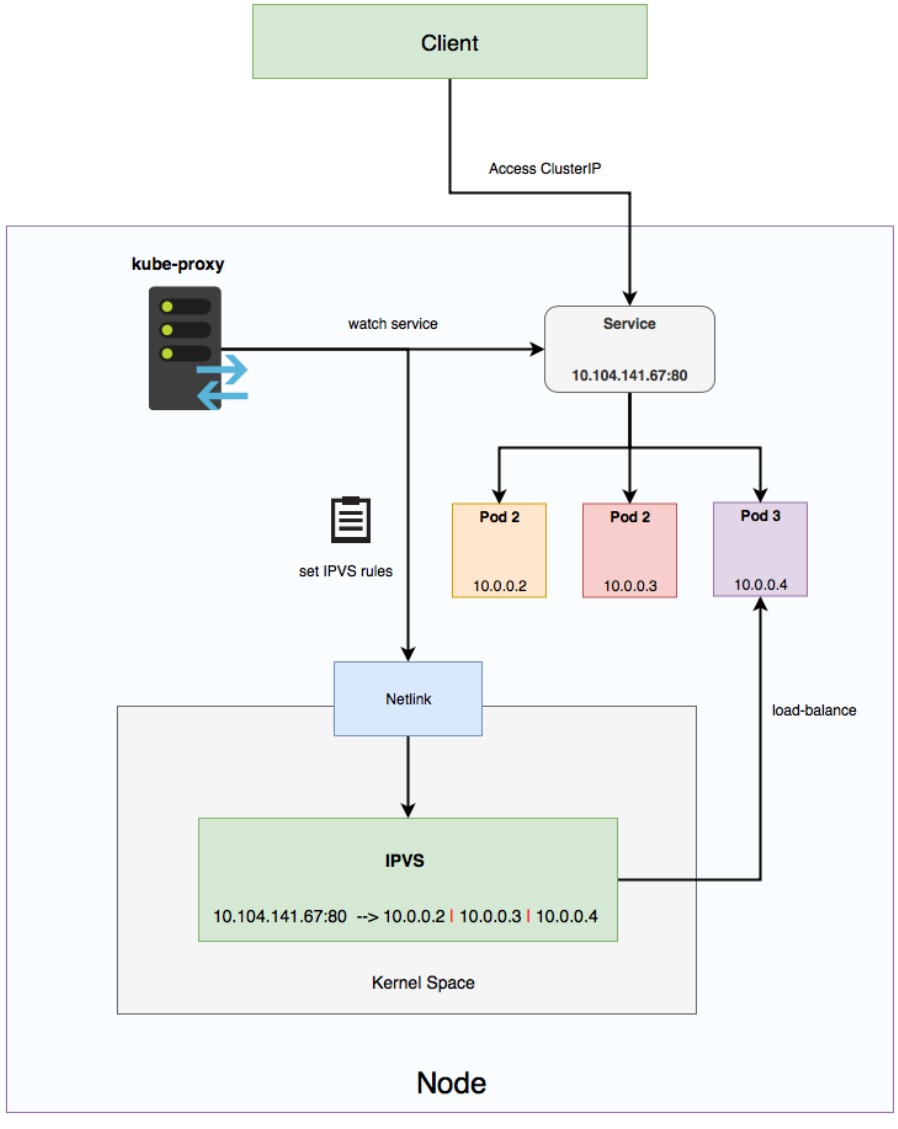
# Kube proxy

## Iptables Mode

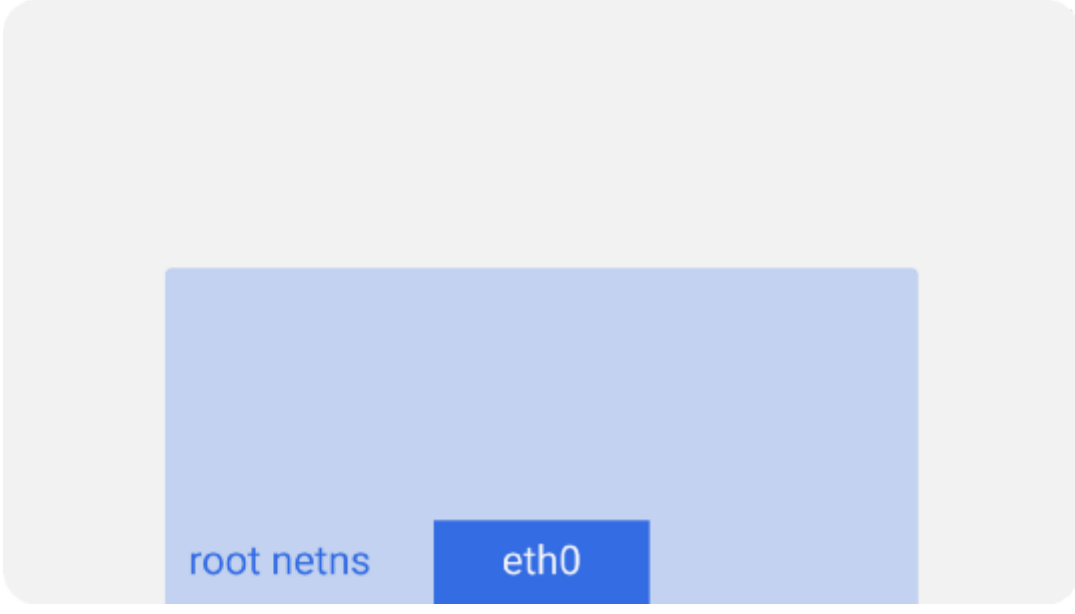

Client
Access ClusterIP
kube-proxy
watch service
Service
10.104.141.67:80
Pod 2
10.0.0.2
Pod 2
10.0.0.3
Pod 3
10.0.0.4
set iptables rules
load-balance
Netfilter
10.104.141.67:80 --> 10.0.0.2 | 10.0.0.3 | 10.0.0.4
Kernel Space
Node

# Kube proxy

## IPVS Mode

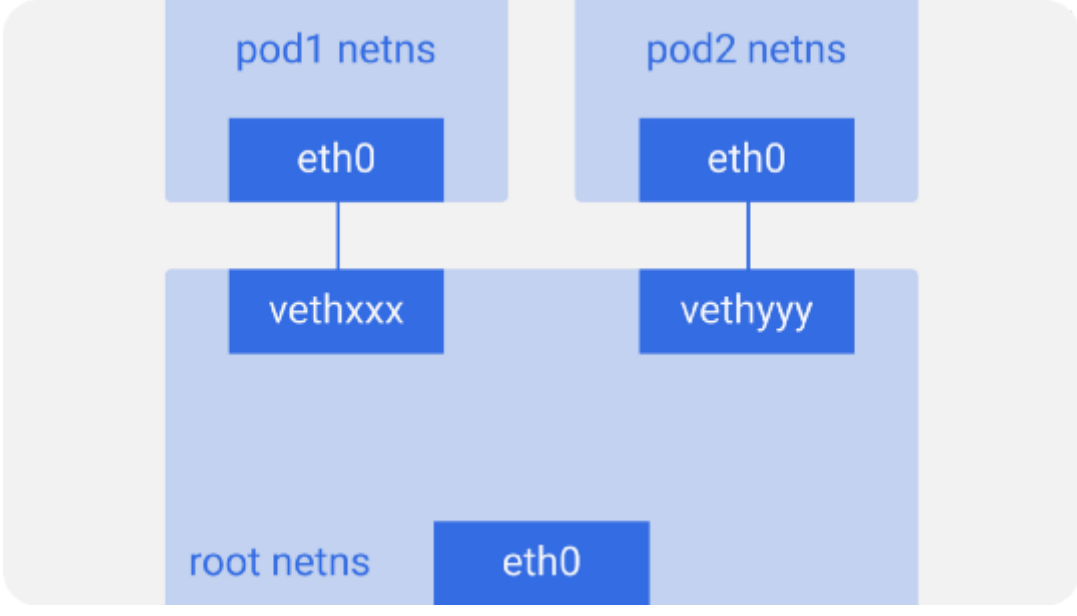

Client
Access ClusterIP
kube-proxy
watch service
Service
10.104.141.67:80
set IPVS rules
Pod 2
10.0.0.2
Pod 2
10.0.0.3
Pod 3
10.0.0.4
Netlink
load-balance
IPVS
10.104.141.67:80 --> 10.0.0.2 | 10.0.0.3 | 10.0.0.4
Kernel Space
Node

# Взаимодействие подов внутри хоста




Корневое пространство имен (root netns).

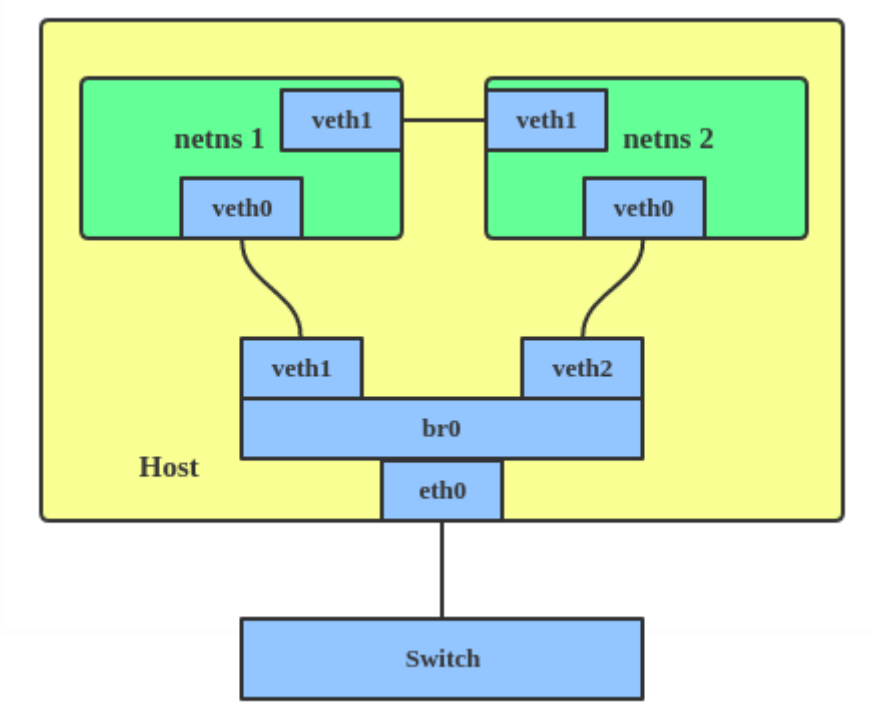
# Взаимодействие подов внутри хоста




У каждого пода есть свой netns

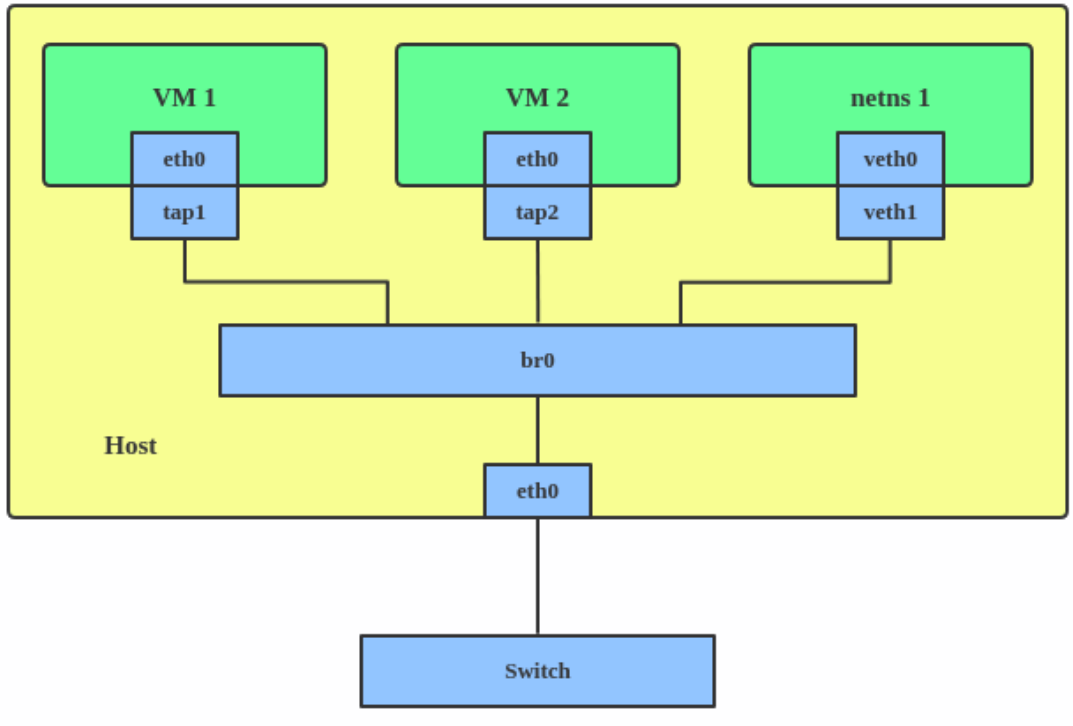
# VETH

The VETH (virtual Ethernet) представляет из себя локальный Ethernet. Устройства подключены попарно.

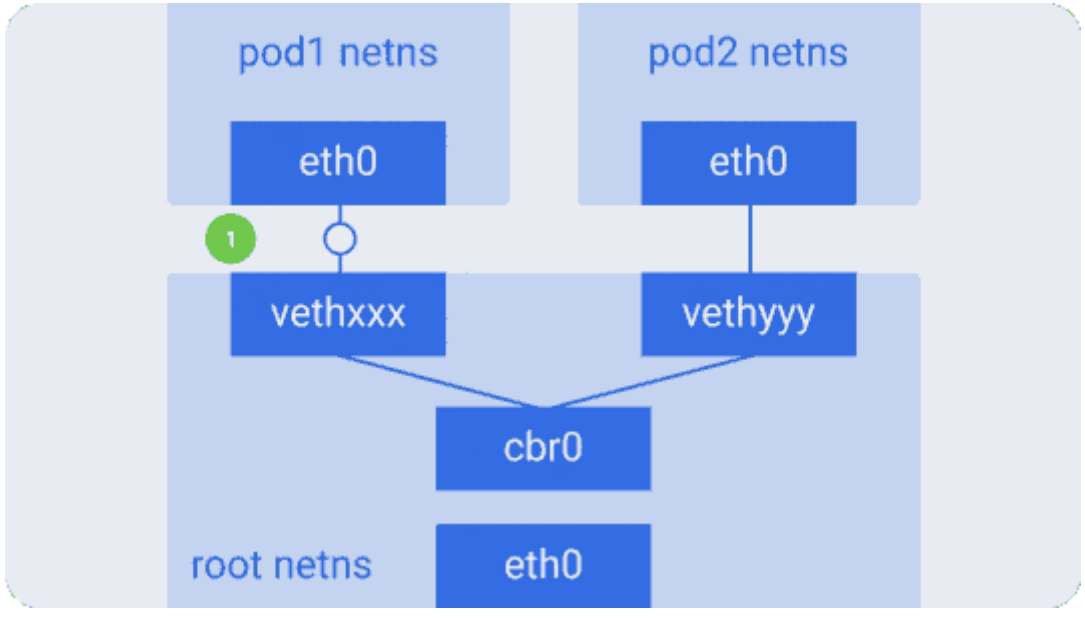


# Bridge

Linux bridge представляет из себя коммутатор. Он перенаправляет трафик между интерфейсами которые подключены к нему.



# Взаимодействие подов внутри хоста

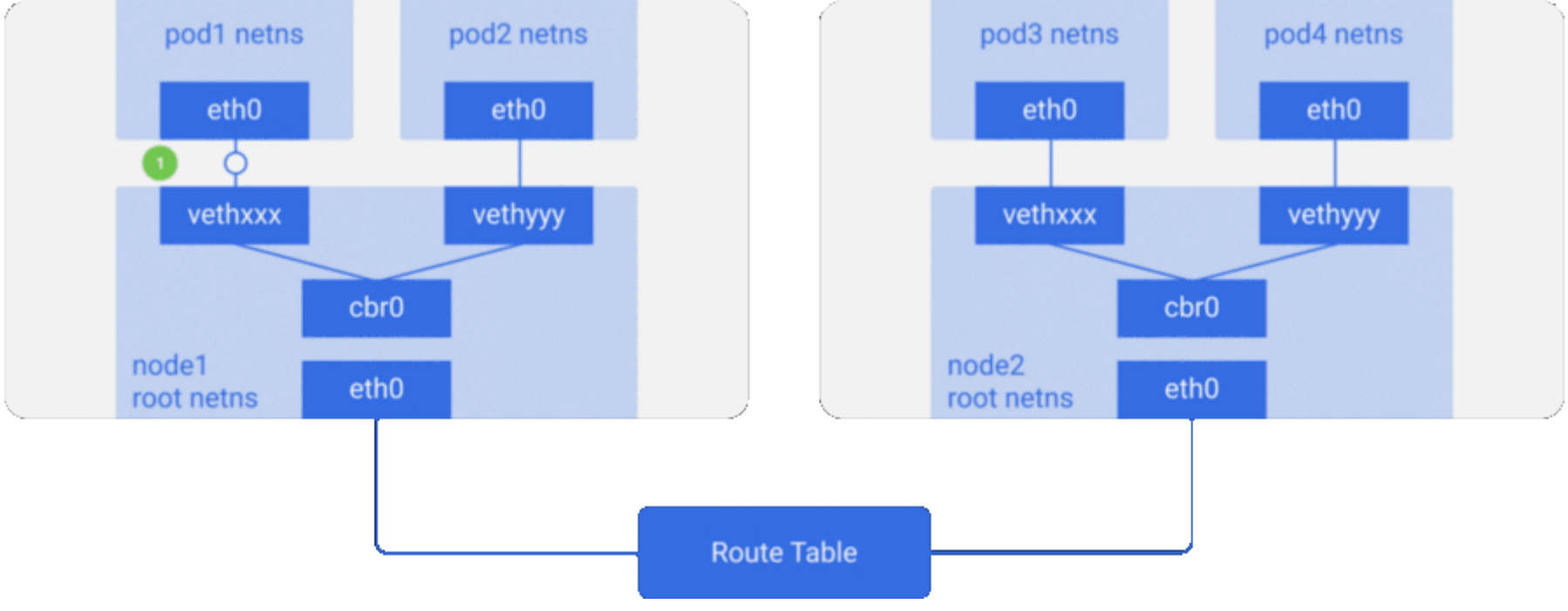



Взаимодействие pod1 и pod2

# Взаимодействие подов внутри хоста

- Пакет через eth0 покидает свой ns, попадает в root ns через vethxxx
- Попадает на bridge cbr0, посылает ARP запрос с целью поиска нужного адреса
- vethyyy посылает ответ что у него нужный IP адрес
- Пакет проходя через vethyyy попадает в netns принадлежащий pod2

# Взаимодействие подов расположенных на разных хостах



# Взаимодействие подов расположенных на разных хостах

- Через eth0 покидает netns и попадает в rootns через vethxxx
- Попадает в cbr0 через vethxxx и посылает ARP запрос с целью поиска нужного IP адреса
- Из cbr0 попадает в eth0, так как на узле нет нужного адреса
- Покидает ноду со значениями src=pod1 dst=pod4
- Согласно таблице маршрутизации отправляется на нужную ноду
- Попадает на основной интерфейс node2-eth0
- Пакет перенаправляется на bridge cbr0
- Через vethyyy попадает в netns пода

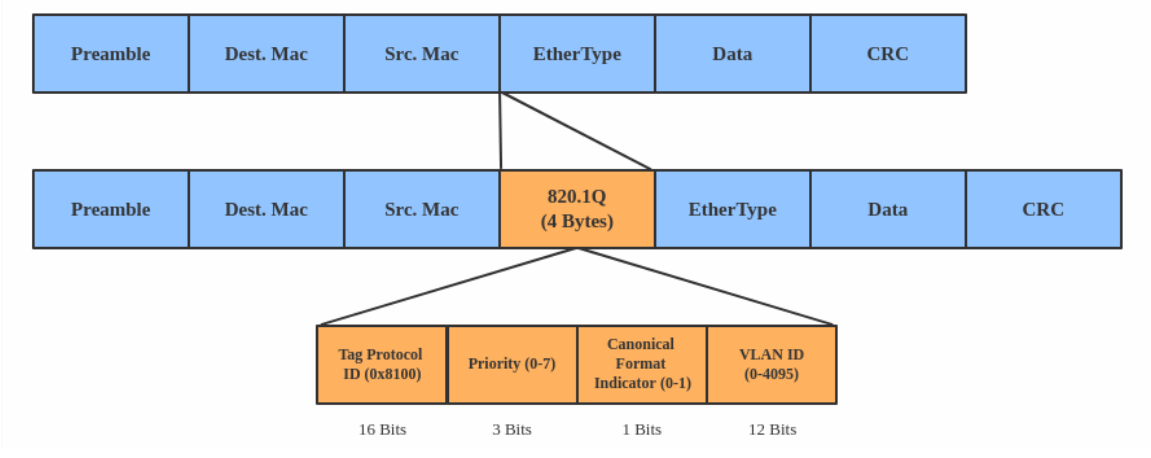
# Оверлейные сети

**Сеть создаваемая поверх основной сети.**

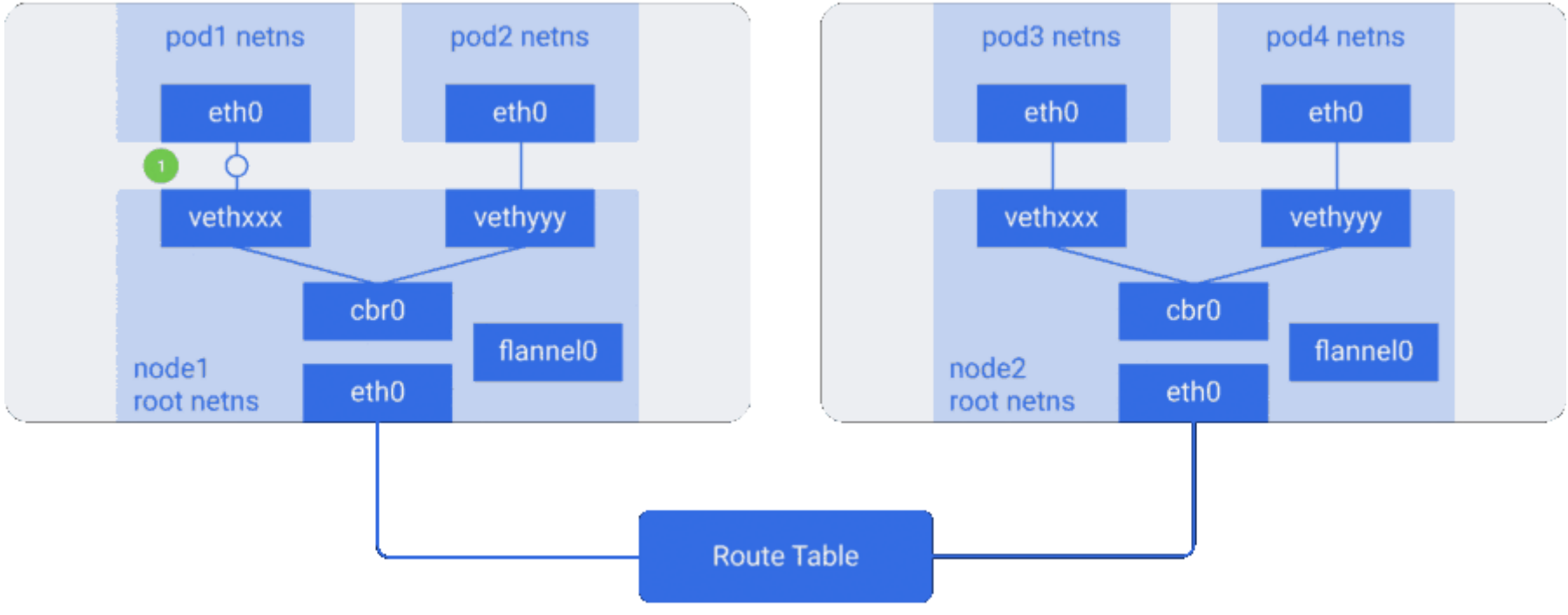
- Нехватка доступных адресов
- Невозможность управления маршрутизацией
- Ограничение по количеству маршрутов

# VXLAN

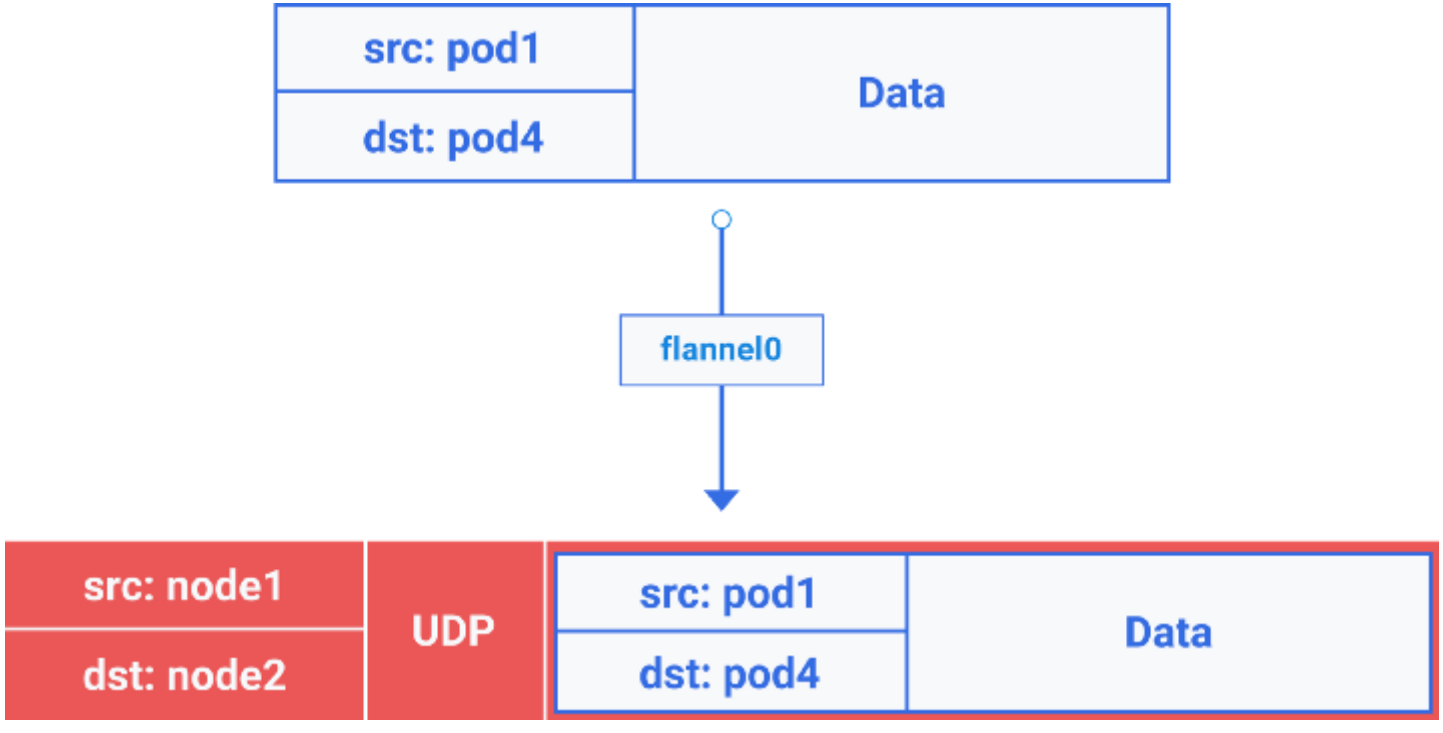
VXLAN (Virtual eXtensible Local Area Network) туннельный протокол созданный что бы решить проблему ограничения VLAN (4,096) в IEEE 802.1q. VXLAN инкапсулирует L2 фремы с VXLAN хидерами в UDP-IP пакет:



# Оверлейные сети



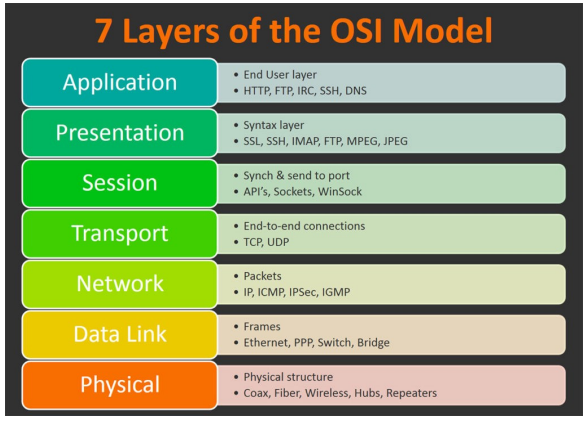
# Оверлейные сети



# Оверлейные сети

- Через eth0 покидает netns и оказывается в rootns на vethxxx
- На cbr0 делает ARP запрос с поиском IP адреса
  - Так как нет нужного IP, пакет отправляется в flannel0
  - flanneld получается IP нажначения (apiserver, etcd)
  - Пакет через flannel отправляется на нужную ноду
- Пакет попадает на eth0 узла node2
  - flannel переносит пакет в root ns.
  - пакет перенаправляется в cbr0
- На cbr0 делается ARP запрос с поиском нужного IP адреса
- Павет через vethyyy отправляется в нужный под

# L2 vs. L3

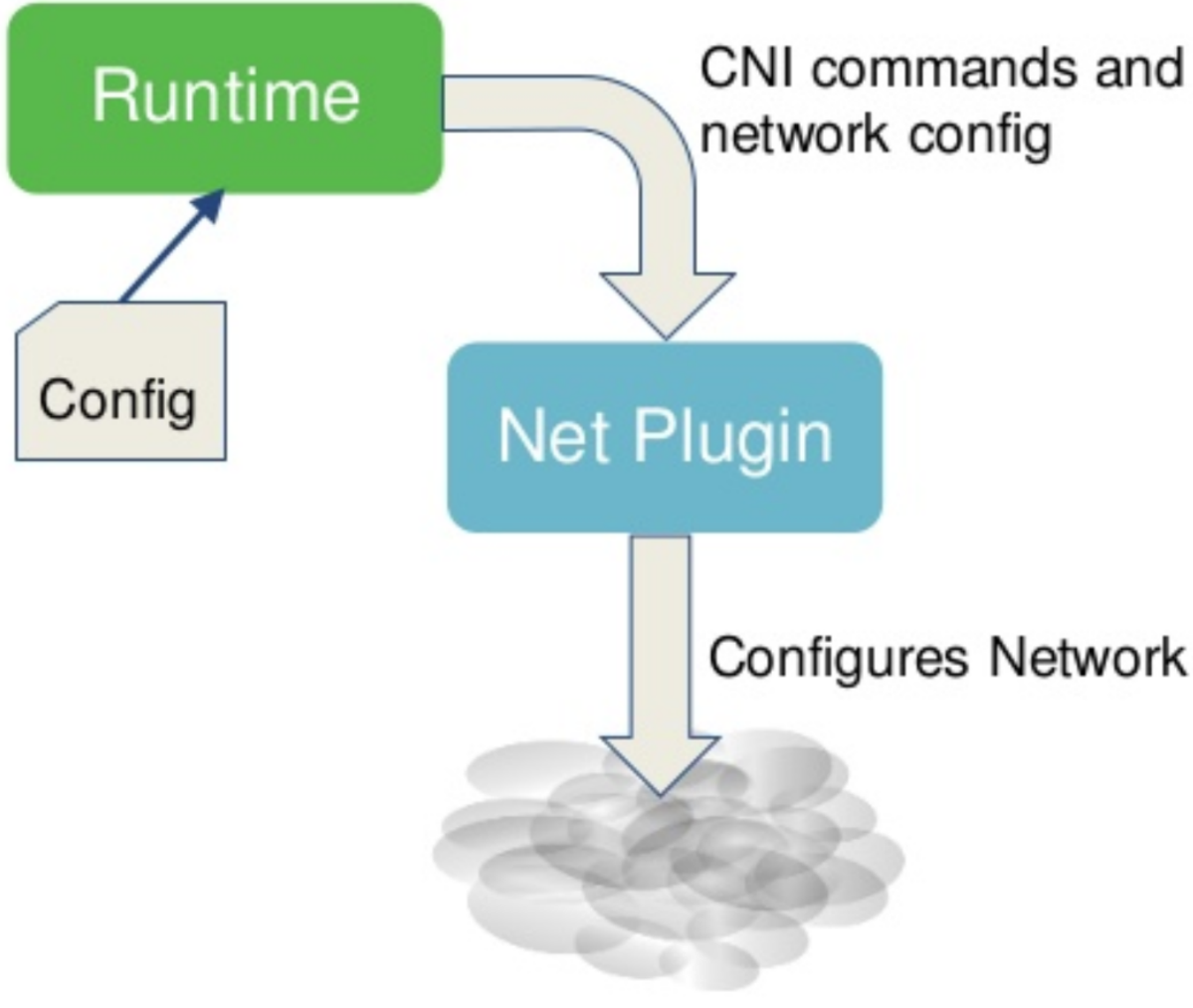



- L2: Канальный уровень. Взаимодействие сетей на физическом уровне, контроль ошибок. (frames, arp)
- L3 Сетевой уровень: Предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание ошибок и "заторов" (ipv4, ipv6)

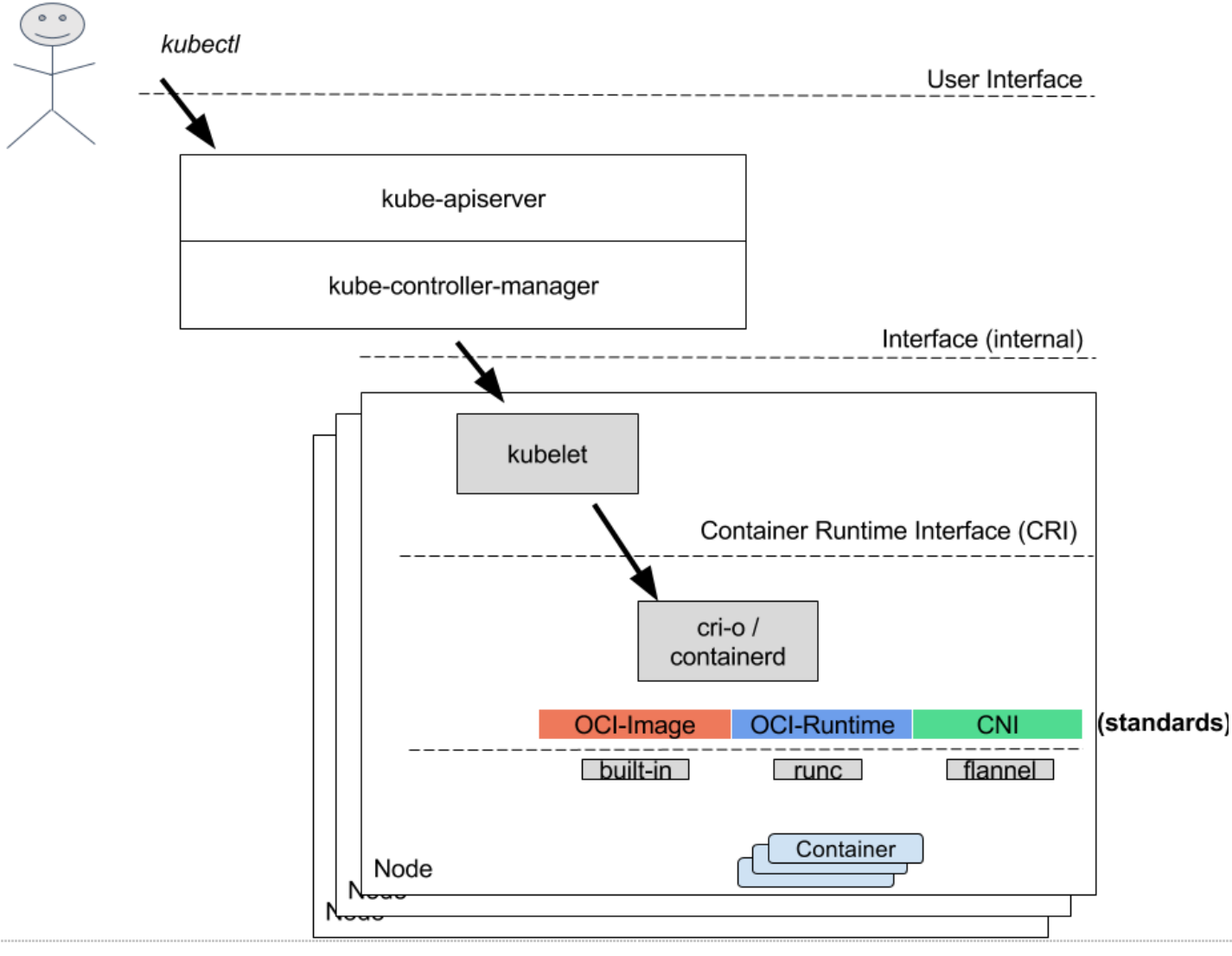
# CNI plugin

- Binary file
- CAP_NET_ADMIN Linux capabilities
- Store config in json file
- Информация должна передаваться в качестве переменных окружения
- Json to stdin through stdin
- Подключение контейнеров к сетям

# Взаимодействие CNI, runtime, kubelet



# Взаимодействие CNI, runtime, kubelet



# Взаимодействие CNI, runtime, kubelet

- Создание rootfs для контейнера
- Создать контейнер
- Присоеденить контейнер к сети
- Запустить процесс внутри контейнера

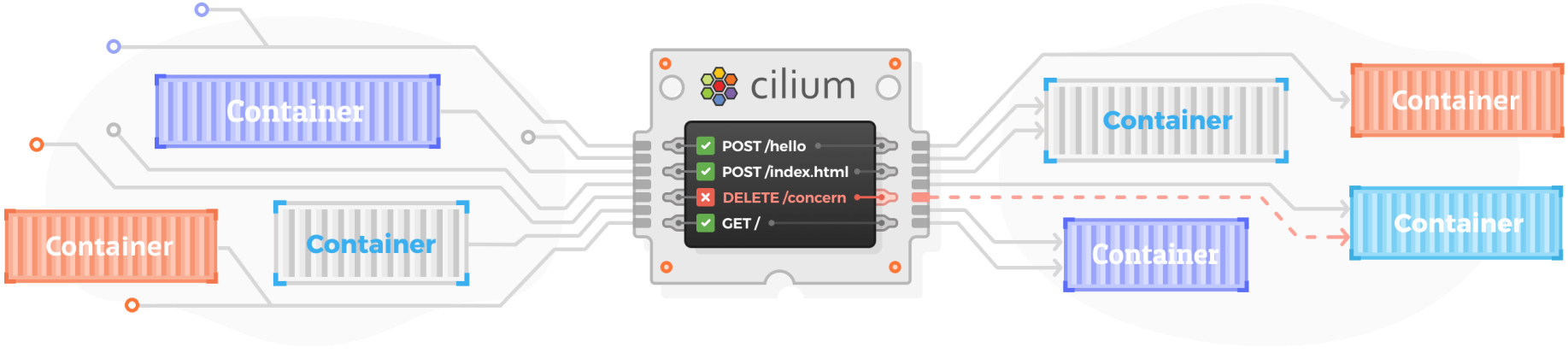
# Runtime CNI

**Provison phase**

- Проверка компонентов
- Создание bridge (не обязательно)
- **Runtime phase**

- Оркестратор сообщает рантайму что пора запускать контейнер
- Присоединение контейнера к сети
- Определение переменных окружения с информацией для CNI
- Запуск CNI

# Обзор сетевых плагинов: Cilium




- API-aware network security
- Linux BPF
- Visibility and security policies
- API-Protocol Visibility + Security (HTTP, gRPC, Kafka)
- Scalable

# BPF (Berkeley Packet Filter)

The Berkeley Packet Filter (BPF) provides a raw interface to data link layers, permitting raw link-layer packets to be sent and received. It is available on most Unix-like operating systems. In addition, if the driver for the network interface supports promiscuous mode, it allows the interface to be put into that mode so that all packets on the network can be received, even those destined to other hosts.

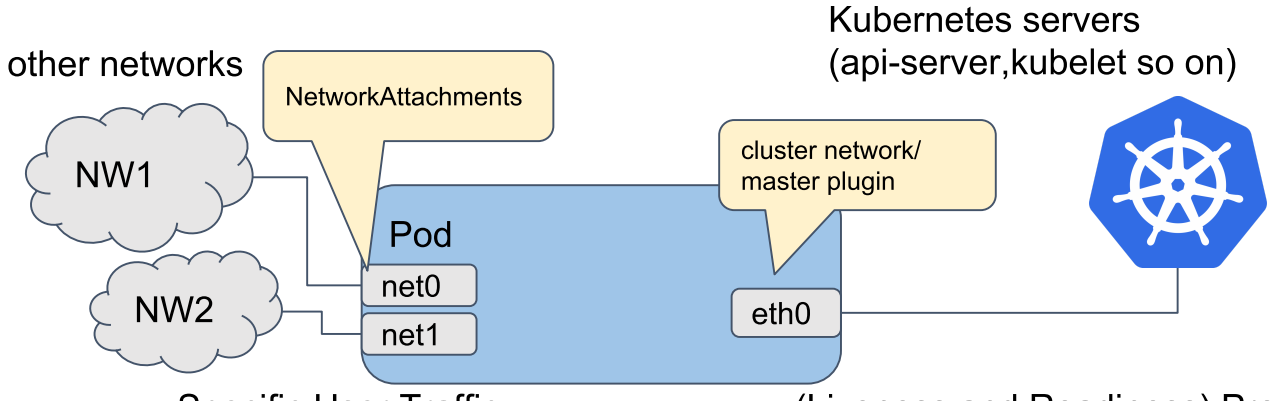
# Обзор сетевых плагинов: Weave

weaveworks

- Service Discovery
- Network security
- Resilience and Scaling
- Perfomance
- Multicast
- Load balancing

# Обзор сетевых плагинов: Multus


- Attach multiple network interfaces to pod




- Specific User Traffic
- (Liveness and Readiness) Probes
- Communication between API and Pod

# Обзор сетевых плагинов: Kube router


- IPVS/LVS based service proxy
- Pod Networking
- Network Policy Controller
- Advanced BGP Capabilities
- Standard Linux Networking
- Small Footprint
- High Performance

# Flannel


- CoreOS development
- Use etcd to store configuration
- L2 network
- Ecryption
- VXLAN
- ipv6

# Flannel

/coreos.com/network/config

flannel-network-config.json

```
etcdctl set /coreos.com/network/config < flannel-network-config.json
```

```
{
    "Network": "10.0.0.0/8",
    "SubnetLen": 20,
    "SubnetMin": "10.10.0.0",
    "SubnetMax": "10.99.0.0",
    "Backend": {
        "Type": "vxlan",
        "VNI": 100,
        "Port": 8472
    }
}
```

```
etcdctl get /coreos.com/network/config | jq .
```

# Calico

- Performance and flexibility
- L3 with BGP
- Network policy
- Istio integration
- Commercial support
- Load balancing
- ipv6

- Ecapsulation (IPIP)
- 5000 nodes on cluster

# Calico

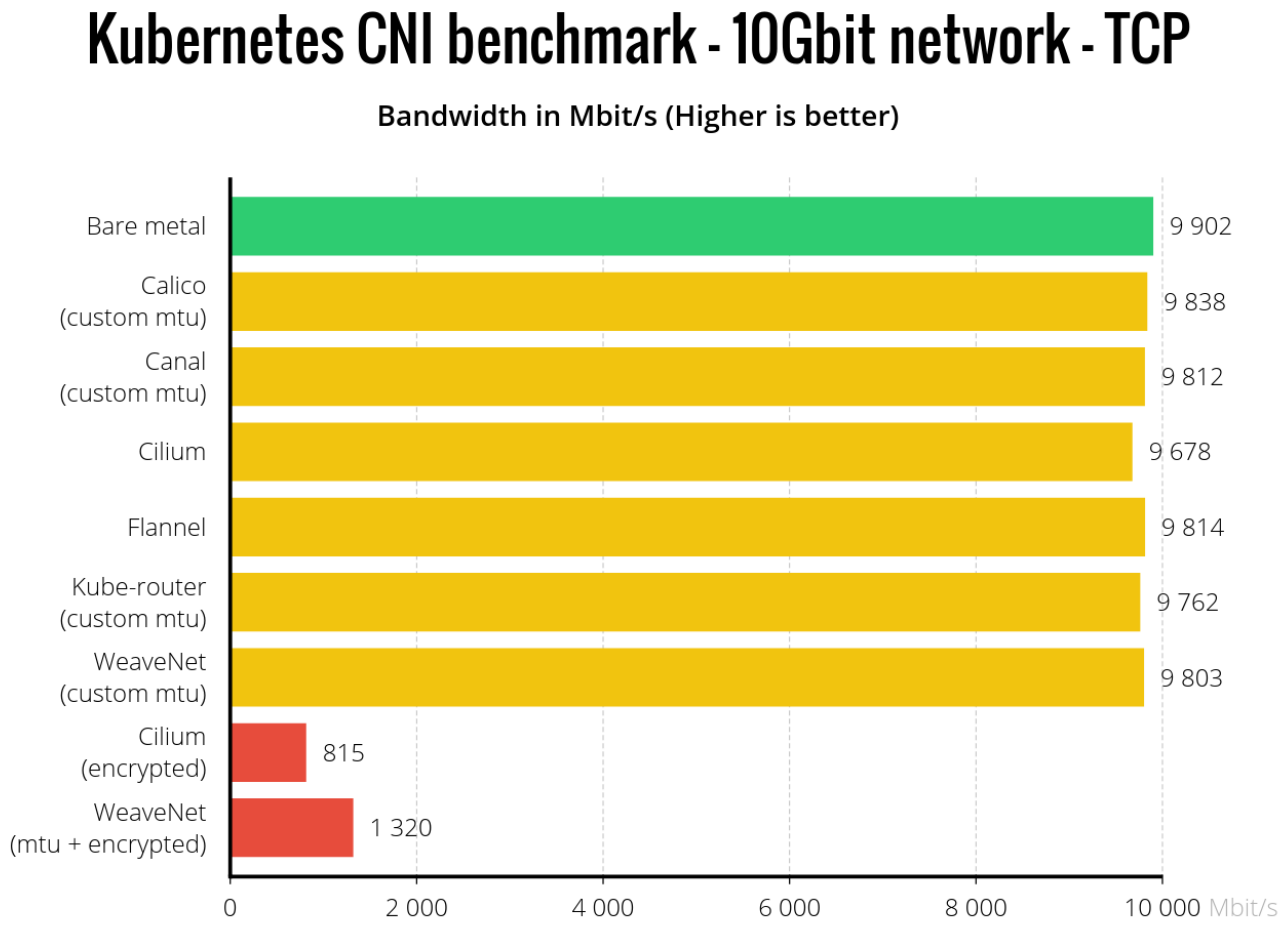
## Demo

- Calicoctl
- Simple network policy

# Canal

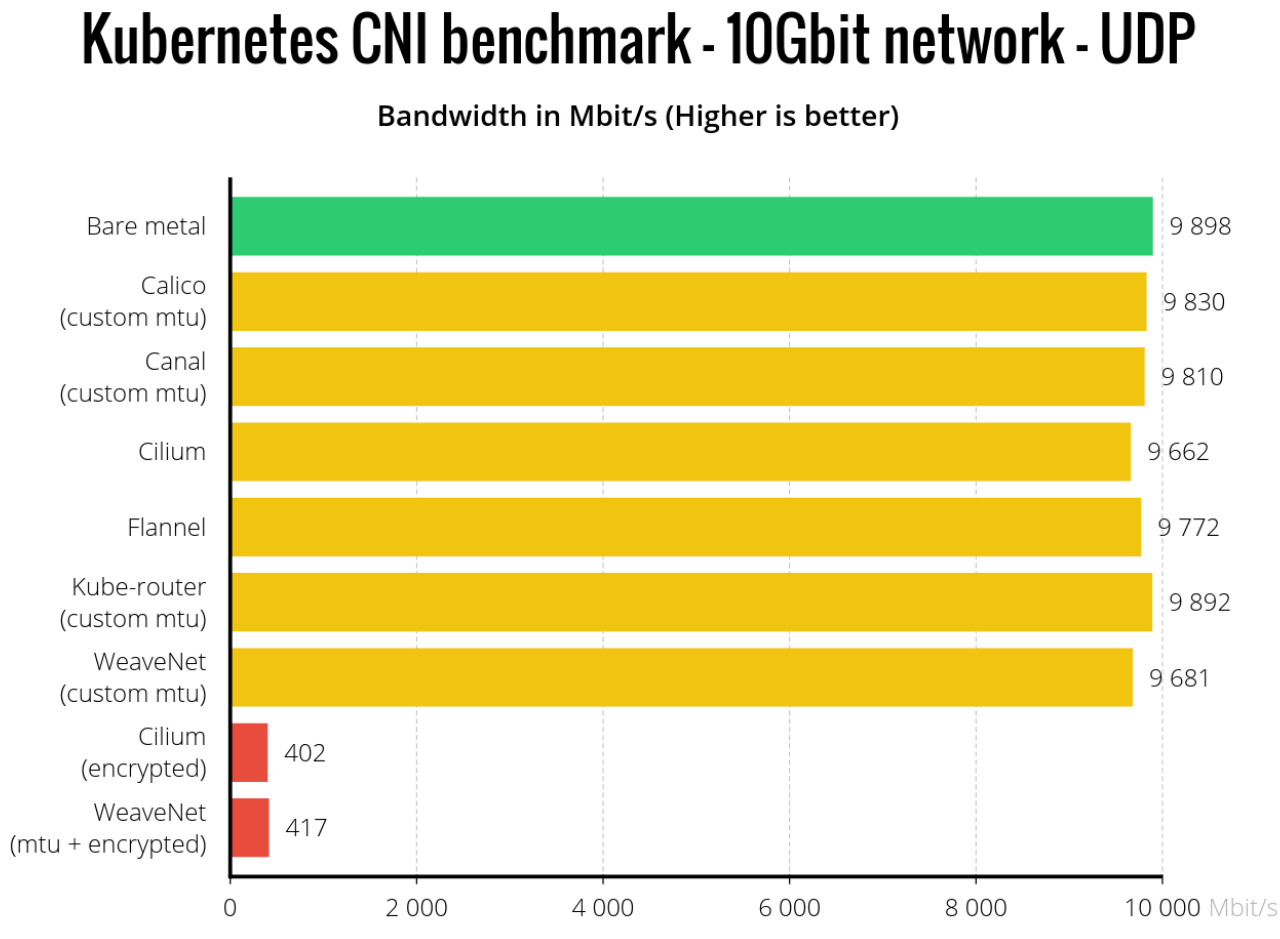
- Integrate calico and flannel
- Flannel network model
- Calico security model

# Network benchmark TCP



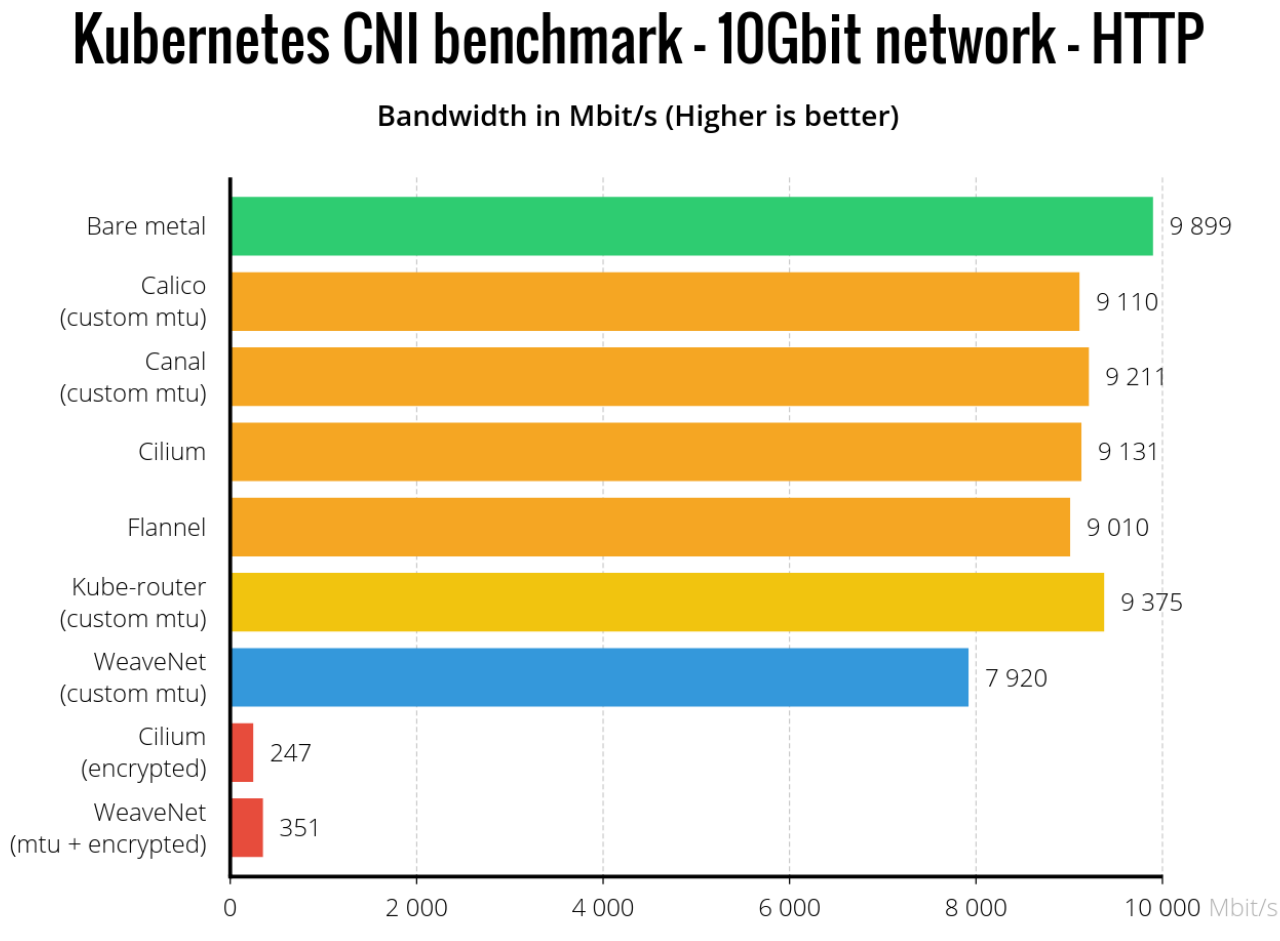

2019-04-05 - Alexis Ducastel - https://infrabuilder.com - Benchmark tool : iperf3

# Network benchmark UDP




2019-04-05 - Alexis Ducastel - https://infrabuilder.com - Benchmark tool : iperf3

# Network benchmark HTTP




2019-04-05 - Alexis Ducastel - https://infrabuilder.com - Benchmark tool : curl + nginx

# Проблемы с сетью

# Методы и инструменты дебага

- Finding pod cluster IP
- Finding Service IP
- Finding and Entering Pod Network Namespaces
- Finding a Pod's Virtual Ethernet Interface
- Inspecting Conntrack Connection Tracking
- Inspecting Iptables Rules
- Querying Cluster DNS
- IPVS Details