MSInteract 2019【DD03】KubernetesとFlannelでWindows上にPod間VXLAN Overlayネットワークを構成

2019年6月29日

System Center User Group Japan
金井　崇

セッションアンケートにご協力ください

5セッション目アンケート⇒

Twitterハッシュタグ

#msinteract19 #DD03

自己紹介

- 名前：金井　崇
- 所属：某国内ISP
- 仕事：IaaSのインフラ設計・構築・運用・提案を担当
- 興味：最近はKubernetes、コンテナなどコンピューティングの
  最適な利用方法に興味があります。
- MVP：Cloud and Data Center Management (2017/03～)
- SNSなど：
  - Facebook　https://www.facebook.com/anikundesu
  - Blog　http://www.takanyan.net/
  - Twitter　@anikundesu
  - LinkedIn　https://jp.linkedin.com/in/takashikanai/ja
  - SlideShare　https://www.slideshare.net/anikundesu/presentations

一瞬だけ宣伝

2019/7/20(土)

SCUGJ 第20回勉強会

https://scugj.connpass.com/event/135520/

一瞬だけ宣伝

- 「異能Vation」は総務省と角川アスキー総合研究所の共同プログラムです。
- 奇想天外なチャレンジ・アイデアを募集中！
- 募集期間：2019年6月3日11:00～7月31日18:00
- https://www.inno.go.jp/

注意事項

本セッションは、**Windows Server 2019 英語版**と**Kubernetes 1.14**での検証結果をもとに記述をしています。バージョンによっては挙動が異なる場合がありますので、ご留意ください。

本セッション資料は、個人で準備した環境において、個人的に実施した検証・結果を基に記載しています。あくまで個人の意見・見解であり、所属する会社・組織及びマイクロソフト社とは関係がございません。所属する会社・組織・マイクロソフト社の正式な回答・見解ではない事に留意してください。

本資料を閲覧した事により問題が生じた場合、または問題が発生しかけた場合、または生じた一切の不利益について、発表者は一切の責任を負う事はできませんのでご了承ください。

本日のアジェンダ

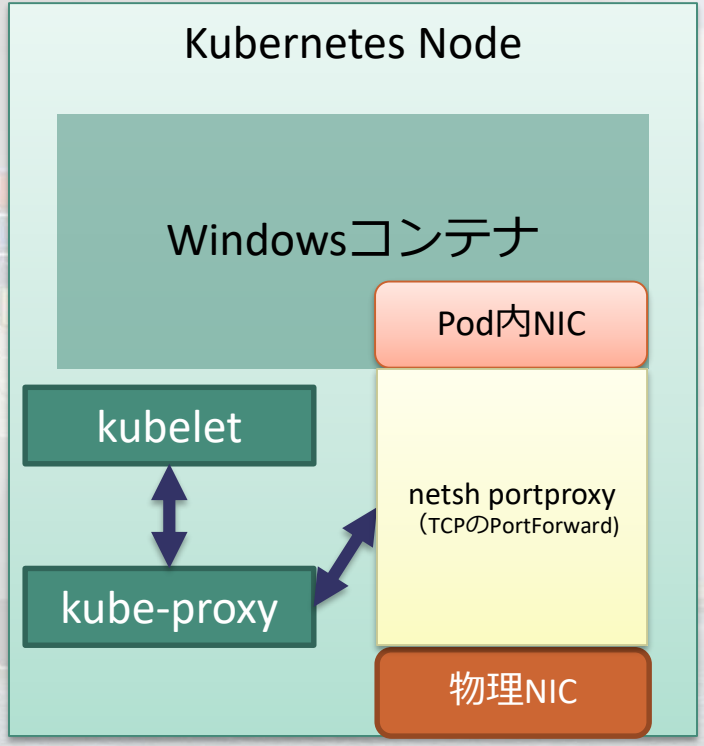
1. KubernetesのWindows対応GAまでの道のりとこれから
2. VXLAN Overlay構成でのコンテナ間通信（Linux）
3. VXLAN Overlay構成でのコンテナ間通信（Windows）
4. FlannelによるVXLAN Overlay構成の構築手順紹介

Section 1

KubernetesのWindows対応GAまでの道のりとこれから

Kubernetes v1.5：Win2016にα対応

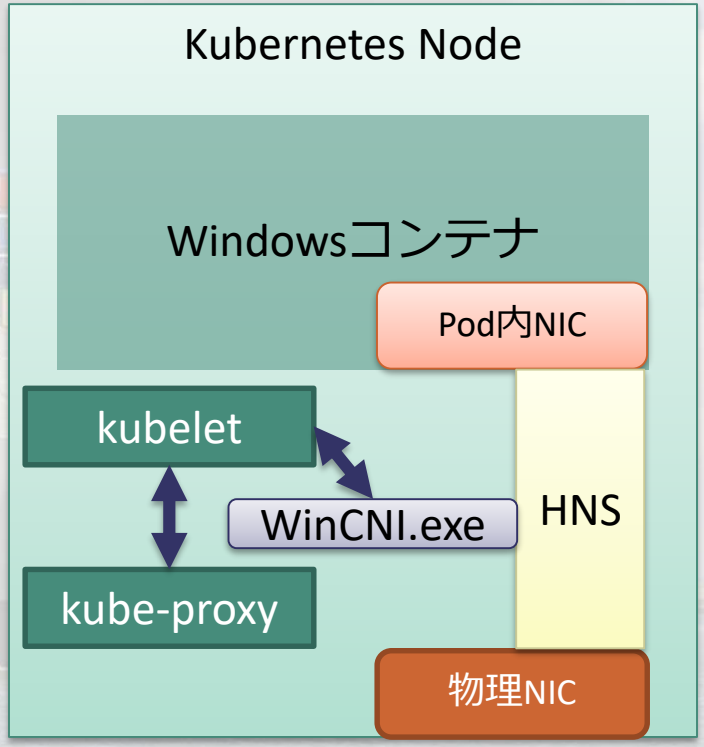
- 2016年12月にリリース
- Kubernetes Nodeに必須なkubeletとkube-proxyのみ実装。
- Kube-proxyの中身は「netsh portproxy」によるTCPのPortForwarding設定。
- だから、**UDP・ICMPはコンテナ内から外に出せなかった！**
- またconfigmapにも非対応だった。



Kubernetes v1.9：WinCNI登場とβ対応

- 2017年12月にリリース
- Windows Server version 1709対応
- WinCNI.exeを使ったCNI+HNS(Hyper-V仮想スイッチ)によるHost Gatewayモードへ対応。
- 上位L3SWルーティング、Open Switch対応。
- このころから私も触り始める。

参照：Kubernetes1.9でWindowsコンテナーをクラスタ化
https://www.slideshare.net/anikundesu/kubernetes19windows

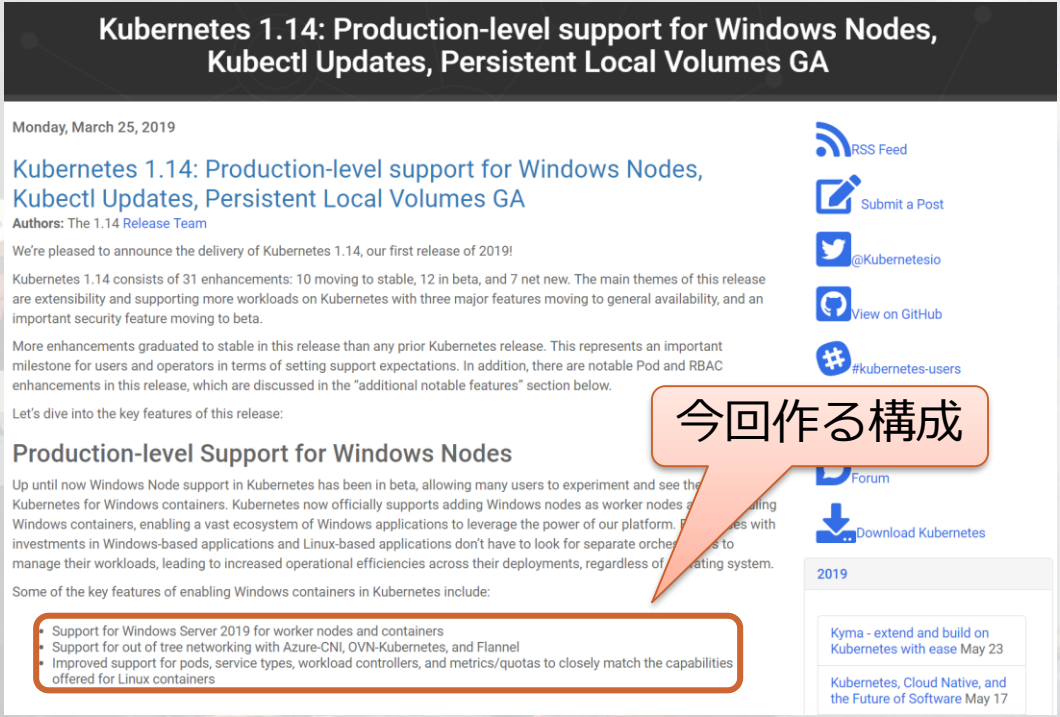


Kubernetes v1.14でWindows対応がGA

- Kubernetes v1.14でWindows対応がついにGA
- 【小ネタ】KubernetesのSIG-Windowsの議長はVMwareの「**Michael Michael（M2）**」さん。

彼は2013年まではMicrosoftにいてSCVMM 2008のProduct Mangerだった。書籍も出版。

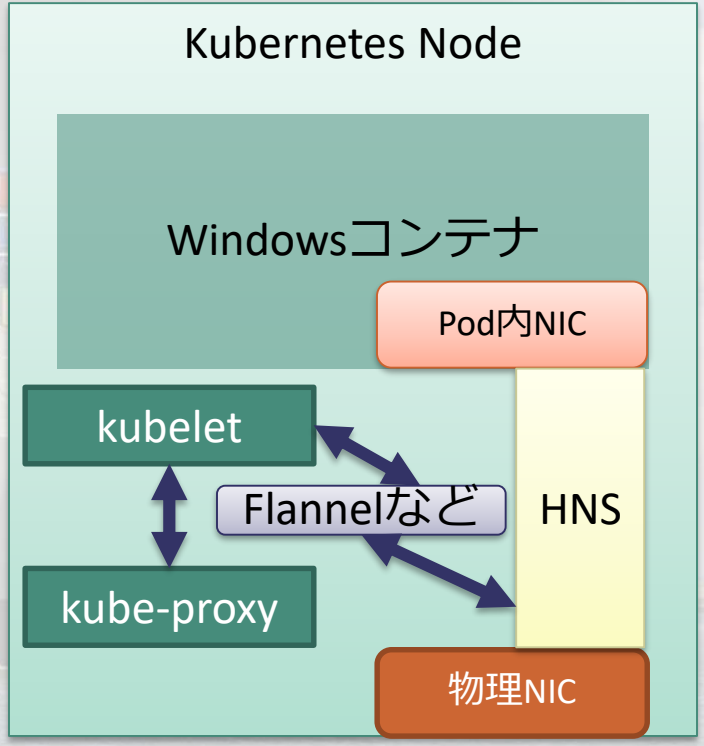
書籍：Mastering Virtual Machine Manager 2008 R2
https://www.amazon.com/dp/0470463325



出典：https://kubernetes.io/blog/2019/03/25/kubernetes-1-14-release-announcement/

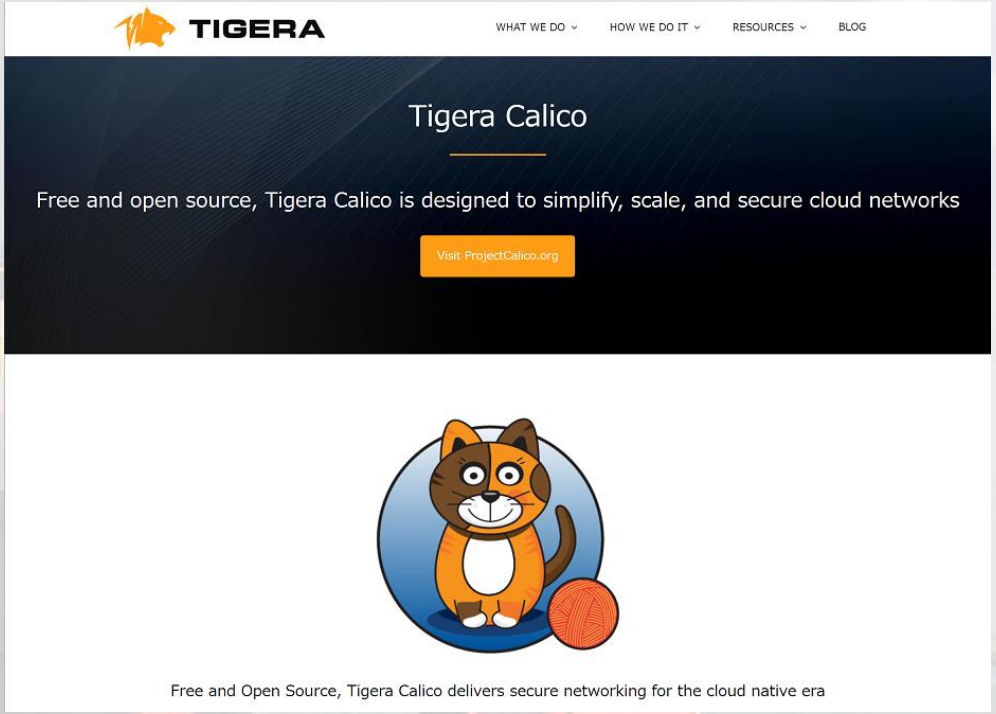
Kubernetes v1.14：CNI正式対応

- 2019年3月にリリース
- Windows Server 2019, 1903対応
- Kubernetes⇒CNI⇒Flannelなど⇒HNSの順でのネットワーク設定に正式対応。
- HNSでのVXLAN Overlayに正式対応。



Calico対応はどうなった？

- CalicoのWindows対応はTigera社にて実現。
- 有償版のみ。
- OSS版での対応はいつになるか未定。



https://www.tigera.io/tigera-calico/

Hyper-V Container対応はどうなる？

- Kubernetes v1.10で実験的に対応
  - Kubelet起動時の**HyperVContainer=true**指定
  - Podの**annotation**に以下の行を追記

```
experimental.windows.kubernetes.io/isolation-type=hyperv
```

- 今後は実装方式をCRI-ContainerD (v1.3でWindows対応予定）経由に変更するので上記方式は廃止予定。

参照：https://kubernetes.io/docs/setup/production-environment/windows/intro-windows-in-kubernetes/

Section 2

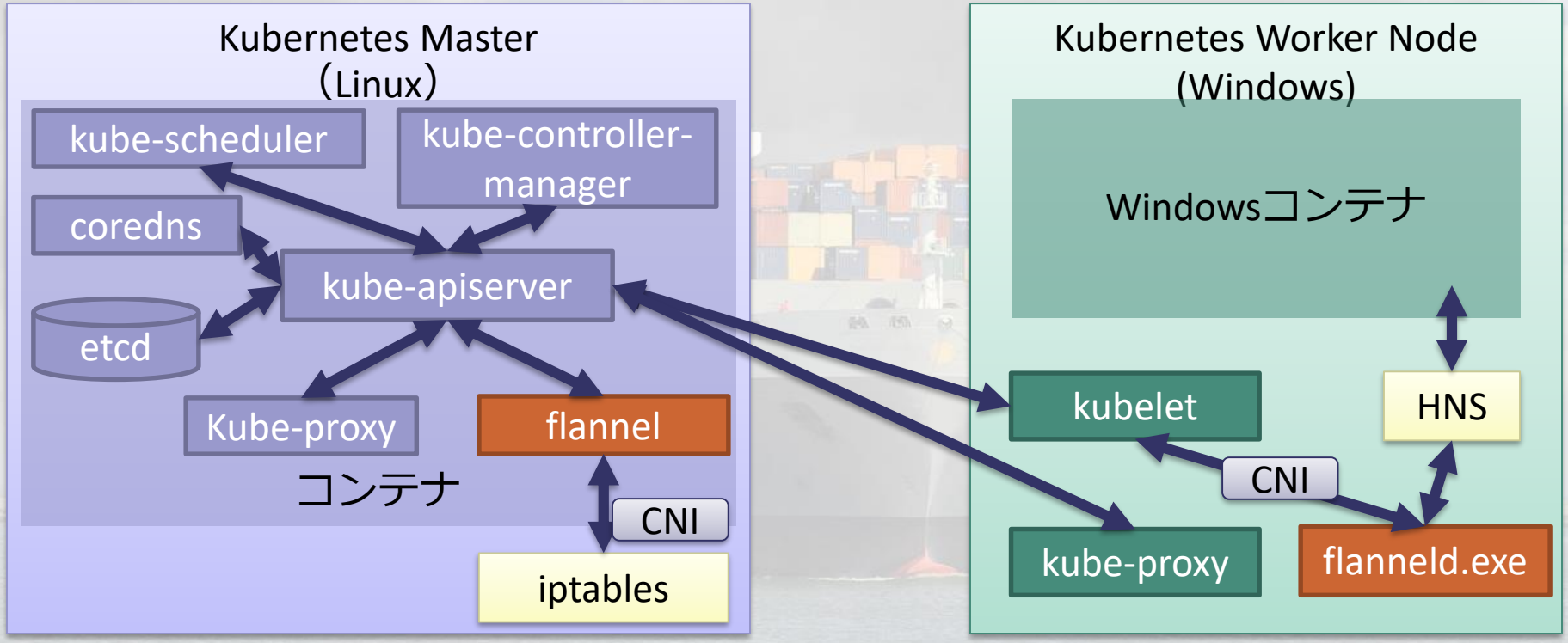
VXLAN Overlay構成でのコンテナ間通信（Linux）

コンテナ間ネットワークのトポロジー設計

今回はGAサポート対象となっているFlannel+VXLAN構成で検証

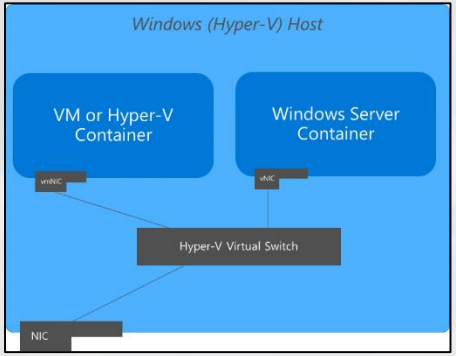
トポロジー	実装状況	特徴
上位L3ルーティング	L3スイッチでStatic Routing	手動でStatic Routingを設定
Host-Gateway	各コンテナホストでStatic Routing	各ホストで他ホスト向けStatic Routeを手動設定
Open vSwitch(OVS)& OVN with Overlay	STTなどのOverlayプロトコルをOVS上で実装	Hyper-V vSwitchがあるのにOVSを入れるとか・・・
FlannelとCNIプラグインによるHost Local or Overlay	各コンテナホストでStatic Routing or **VXLANカプセル化**	~~Host Gateway~~~~モードは大体~~ **VXLANも**動く
Calicoによるルーティング	BGPでL3ルーティング設定	**TIGERAにて有償サポート**（https://www.tigera.io/）

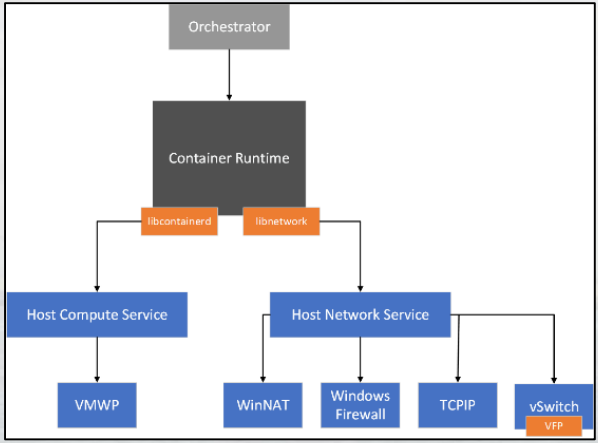
【復習】Kubernetes管理プロセス



【復習】Windowsコンテナーネットワーク

- コンテナNWはHyper-V vSwitchにつながる。
- Windows Serverコンテナの場合はホストOSのvNIC、Hyper-Vコンテナの場合は仮想マシンNICを利用
- HNS(Host Network Service)がvSwitchの設定・NAT、Filteringなどを設定する。

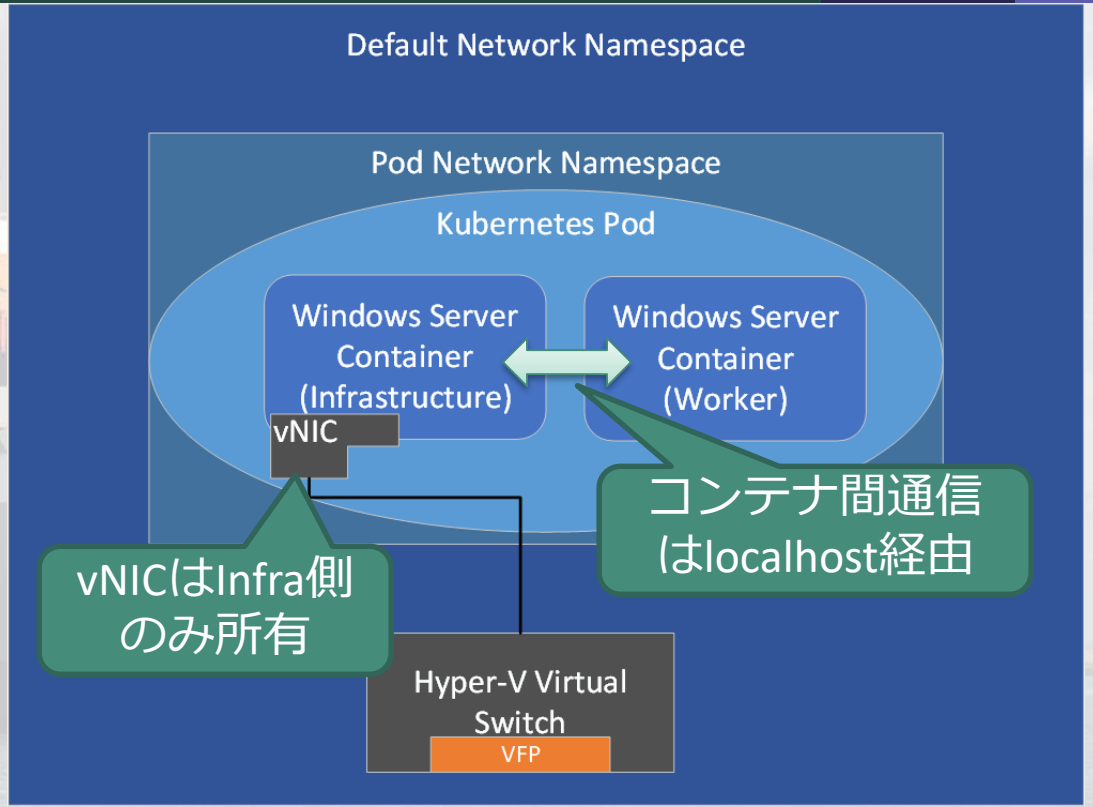




出典：Windowsのコンテナーネットワーク（コンテナーネットワークの概要）
https://docs.microsoft.com/ja-jp/virtualization/windowscontainers/container-networking/architecture

【復習】Windowsコンテナでの Pod実装

- Workerは利用者が作ったアプリ実行コンテナ
- Infrastructureは Kubernetesで自動的に作られるコンテナ（ただPingしてるだけ・・・）



出典：Windowsのコンテナーネットワーク（ネットワークとセキュリティの分離）
https://docs.microsoft.com/ja-jp/virtualization/windowscontainers/container-networking/network-isolation-security

通信パケットの処理の流れがわからん！
これじゃ**Deep Diveセッション**と言えない！

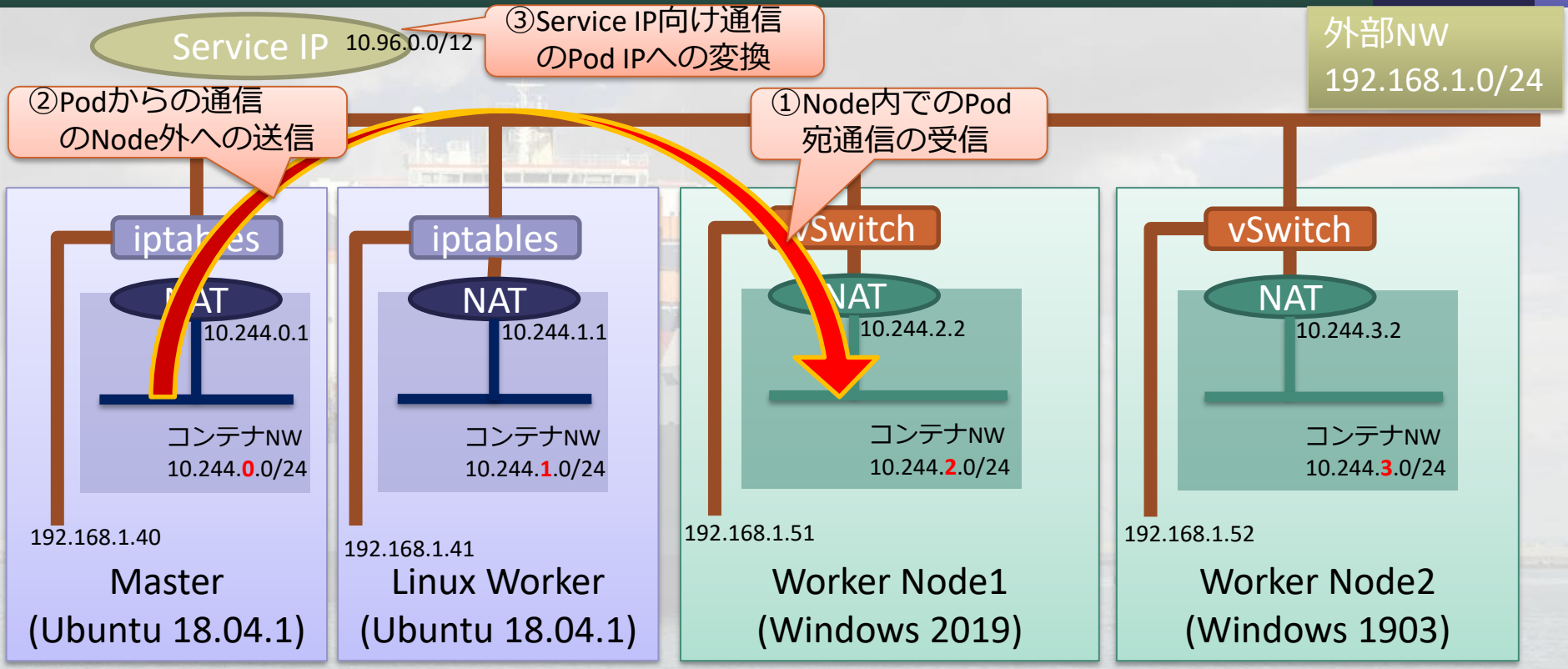
では、ここから**Deep Diveはじめます！**

理解したい通信の内容

1. Kubernetesクラスタ内のPod宛の通信の受信
2. 各Podから別Node上のPodやクラスタ外部への送信
3. KubernetesのService IP宛通信の宛先をPod宛に変換

   ⇒Pod宛に変換後は１，２の通信パターンになる。

想定するネットワーク構成



まずLinuxでは？

LinuxのコンテナNetworkingではしっかりパケットの処理の流れが解説されています。まずは下記の資料などをベースにLinuxでのパケット処理の流れを確認。

引用元

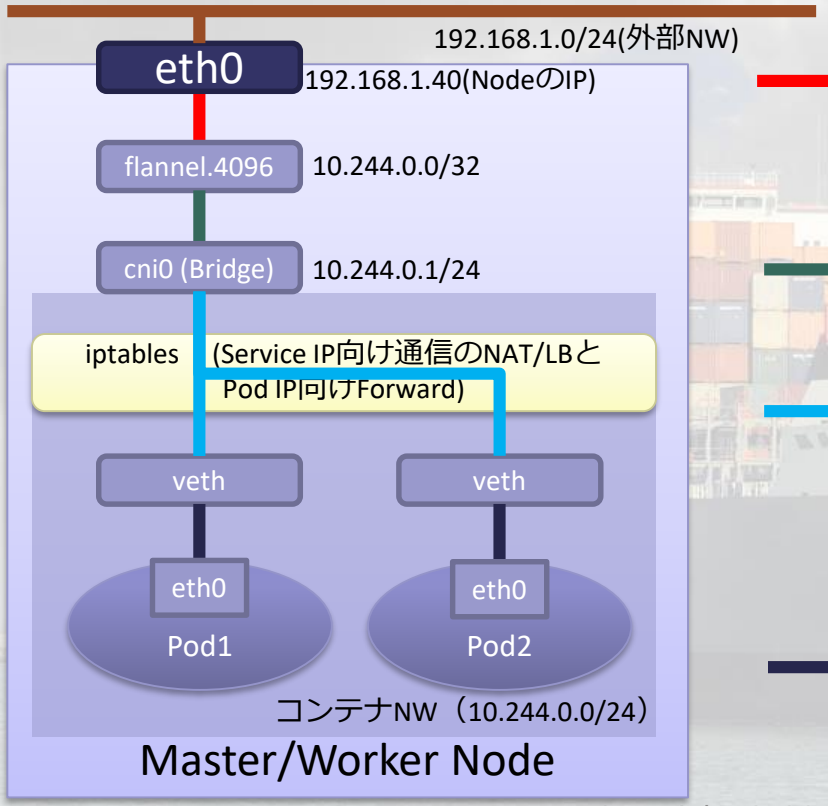
■市原さん：Container Networking Deep Dive
https://www.slideshare.net/hichihara/container-networking-deep-dive-94307233

■進藤さん：コンテナネットワーキング（CNI）最前線
https://www.slideshare.net/motonorishindo/cni-124981353

前提知識（Linux版）

- ip addrコマンド　（IPアドレス設定）
- ip routeコマンド　（Routing設定）
- Bridge fdbコマンド　（IPとMACの対応設定）
- ip linkコマンド　（Linkの設定）
- iptablesコマンド　（NAT, Filter）
- iptablesの動作仕様　（チェーン、Tableの順序）
- VXLANの仕様　（RFC7348）
  https://tools.ietf.org/html/rfc7348

Linux Node内ネットワーク概要



- VXLANカプセル化された後のNode外むけ通信（MTU 1,500）
- コンテナNWをまたぐL3通信（MTU 1,450）
- Pod向けの内部通信およびKubernetesのService IP向け通信の負荷分散およびNAT変換処理
- Pod内とNodeとのL2Bridge接続

物理NIC～VXLAN Overlay設定

① flannel.4096という名前のVTEP InterfaceがMTU 1450で作成される。
② VTEP経由の通信はeth0（物理NIC）に転送される
③ VXLAN通信はUDP 4789ポートでカプセル化する
④ 10.244.0.0/32(Pod Networkアドレス) がflannel.4096に設定される

```
# ip -d addr
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:01:0b:10 brd ff:ff:ff:ff:ff:ff promiscuity 0 numtxqueues 64 numrxqueues 64 gso_max_size 62780 gso_max_segs 65535
    inet 192.168.1.40/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
4: flannel.4096: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default  ①
    link/ether ce:7a:e5:af:6a:e9 brd ff:ff:ff:ff:ff:ff promiscuity 0
②vxlan id 4096 local 192.168.1.40 dev eth0 srcport 0 0 dstport 4789 ③nolearning ttl inherit ageing 300 udpcsum noudp6zerocsumtx
 noudp6zerocsumrx numtxqueues 1 numrxqueues 1 gso_max_size 62780 gso_max_segs 65535
④inet 10.244.0.0/32 scope global flannel.4096
       valid_lft forever preferred_lft forever
```

flannel0.4096とcni0の間の通信設定

① 自Node内のPod IP向け通信をcni0に集めるStatic Route設定。

② 別Node内のPod IP向け通信は".0"(VTEP)IP経由、flannel.4096デバイス経由で通信するStatic Routeを設定。

③ 別NodeのVTEP IPを解決する静的ARPテーブルを設定。

④ ③で指定したMACアドレスの転送先を別Nodeの外部IPに指定。

⇒"bridge fdb"はdeviceがVXLANの時のみVTEPがある外部IPを指定できる。

```
# ip route | grep 10.244
10.244.0.0/24 dev cni0 proto kernel scope link src 10.244.0.1 ①
10.244.2.0/24 via 10.244.2.0 dev flannel.4096 onlink ②
(以下略)
# ip neigh | grep flannel
10.244.2.0 dev flannel.4096 lladdr 00:15:5d:3c:3e:3c PERMANENT ③
(以下略)
# bridge fdb | grep flannel
00:15:5d:3c:3e:3c dev flannel.4096 dst 192.168.1.51 self permanent ④
(以下略)
```

cni0～vethまでのBridge接続設定

① Bridge InterfaceとしてCni0が作成される。
② PodネットワークのデフォルトGWのIPがcni0に設定される。
③ vethはcni0をマスターとするBridge Interfaceとして定義される。
④ Bridge Slaveとして各種設定がされる。

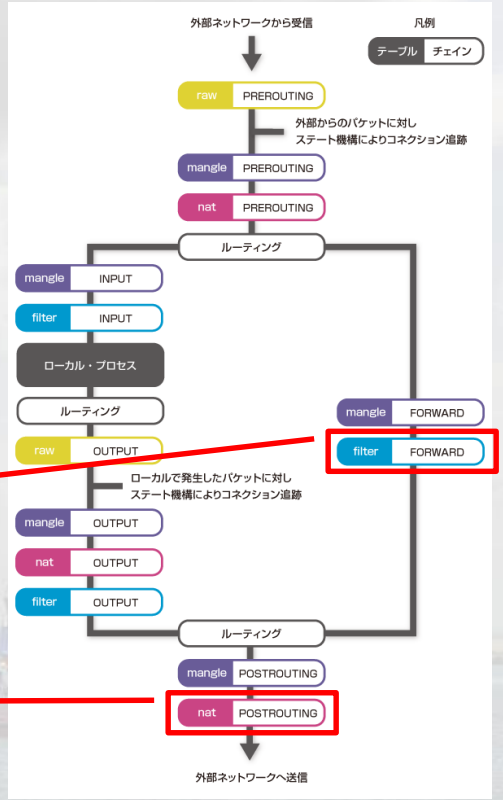
```
# ip -d addr
5: cni0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP mode DEFAULT group default qlen 1000
    link/ether ae:5a:65:7b:a8:6c brd ff:ff:ff:ff:ff:ff promiscuity 0
    bridge ・・・ (以下略) ①
    inet 10.244.0.1/24 scope global cni0 ②
       valid_lft forever preferred_lft forever
                                                                                                    ③
6: veth88b72f0a@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master cni0 state UP mode DEFAULT group default
    link/ether a6:58:7b:1e:c3:00 brd ff:ff:ff:ff:ff:ff link-netnsid 0 promiscuity 1
    veth
    bridge_slave ・・・ (以下略) ④
                                                                                                    ③
7: veth8ccd5a3e@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master cni0 state UP mode DEFAULT group default
    ・・・ (以下略)
```

Node内のパケット転送設定

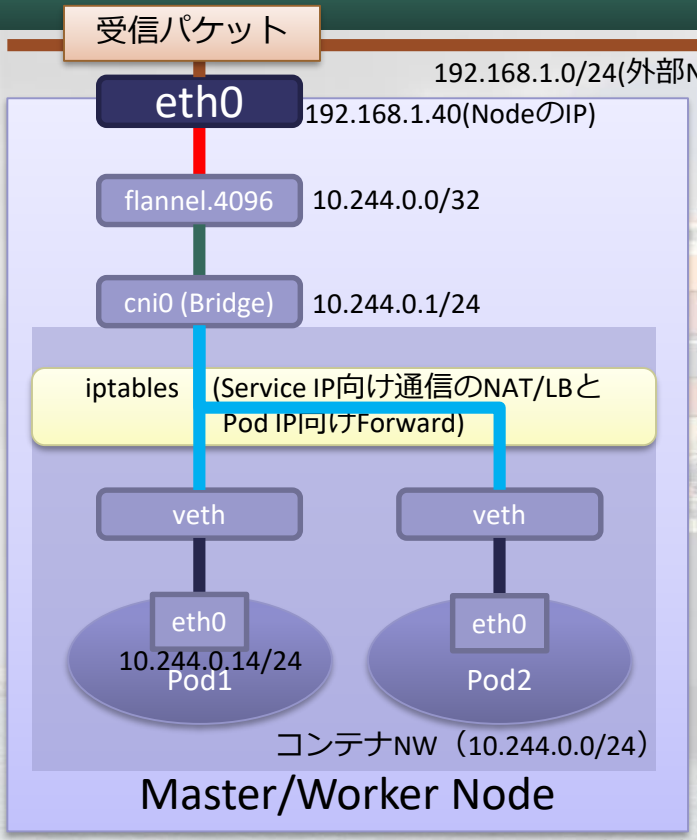
- Kubernetes Cluster構築時に指定したCluster CIDR向けの通信はForwarding許可
- NATテーブルのPOSTROUTINGで自Node内Pod IP（10.244.0.0/24）か別Nodeか等に応じてIPマスカレードを実施。

出典：コピペから脱出！iptablesの仕組みを理解して環境に合わせた設定をしよう
https://oxynotes.com/?p=6361

```
# iptables-save -t filter| grep -v "KUBE" | grep 10.244
-A FORWARD -s 10.244.0.0/16 -j ACCEPT
-A FORWARD -d 10.244.0.0/16 -j ACCEPT
# iptables-save -t nat| grep -v "KUBE" | grep 10.244
-A POSTROUTING -s 10.244.0.0/16 -d 10.244.0.0/16 -j RETURN
-A POSTROUTING -s 10.244.0.0/16 ! -d 224.0.0.0/4 -j MASQUERADE
-A POSTROUTING ! -s 10.244.0.0/16 -d 10.244.0.0/24 -j RETURN
-A POSTROUTING ! -s 10.244.0.0/16 -d 10.244.0.0/16 -j MASQUERADE
```



Cluster内別ノードからPodへの受信①

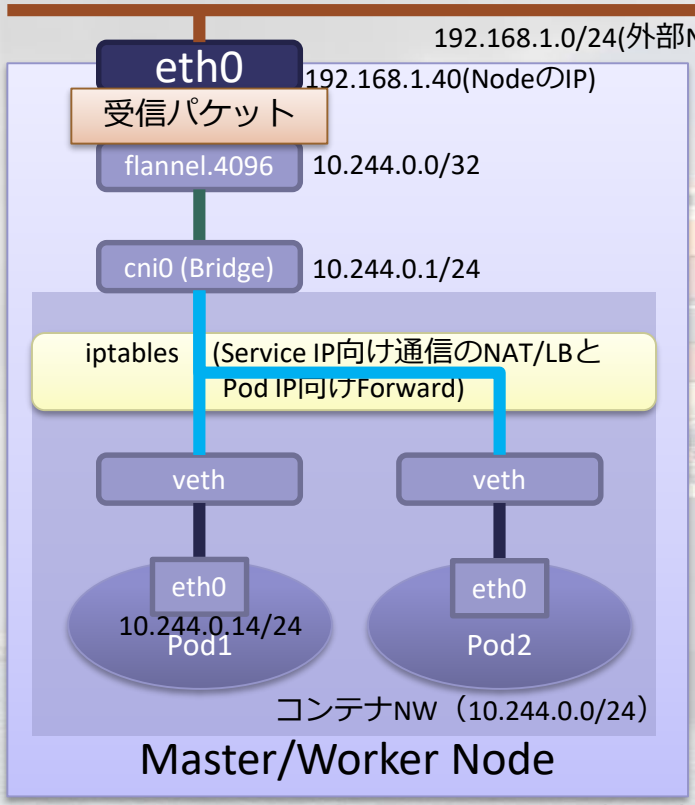


①外部からVXLANカプセル化されたUDPパケット到着

受信パケット

Outer Header（宛先） MAC:Nodeのeth0 MAC IP:192.168.1.40 UDP:4789	Inner Header（宛先） MAC:flannel.4096のMAC IP:10.244.0.14 UDP:53	実際の通信データ

Cluster内別ノードからPodへの受信②



①外部からVXLANカプセル化されたUDPパケット到着

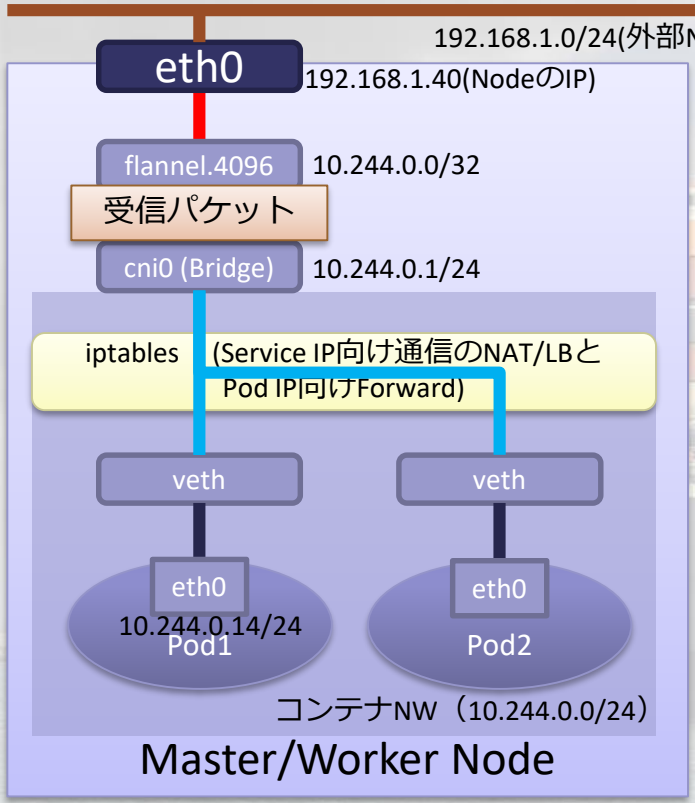
②flannel.4096 VTEPでOuter Header解除

```
# ip -d addr
4: flannel.4096: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN
group default
vxlan id 4096 local 192.168.1.40 dev eth0 srcport 0 0 dstport 4789 ・・・
   inet 10.244.0.0/32 scope global flannel.4096
```

受信パケット

Inner Header（宛先） MAC:flannel.4096のMAC IP:10.244.0.14 UDP:53	実際の通信データ

Cluster内別ノードからPodへの受信③



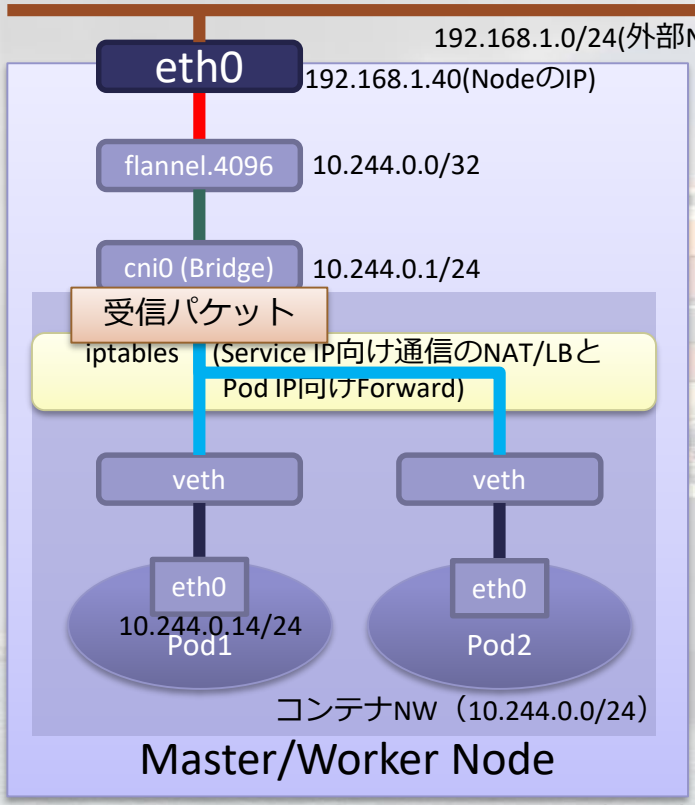
①外部からVXLANカプセル化されたUDPパケット到着
②flannel.4096 VTEPでOuter Header解除
③Inner Headerの宛先IPに従いcni0へルーティング

```
# ip route | grep 10.244
10.244.0.0/24 dev cni0 proto kernel scope link src 10.244.0.1
```

受信パケット

Inner Header（宛先） MAC:cni0のMAC IP:10.244.0.14 UDP:53	実際の通信データ

Cluster内別ノードからPodへの受信④

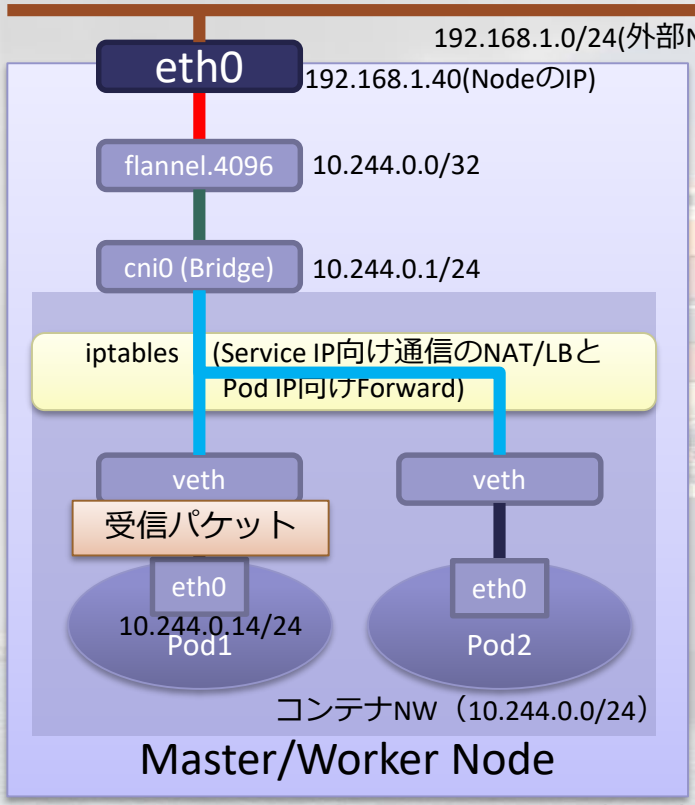


①外部からVXLANカプセル化されたUDPパケット到着
②flannel.4096 VTEPでOuter Header解除
③Inner Headerの宛先IPに従いcni0へルーティング
④cni0のARPテーブルで宛先IPに対応するMACを宛先に設定

受信パケット

Inner Header（宛先） MAC:**Pod1 eth0のMAC** IP:10.244.0.14 UDP:53	実際の通信データ

Cluster内別ノードからPodへの受信⑤



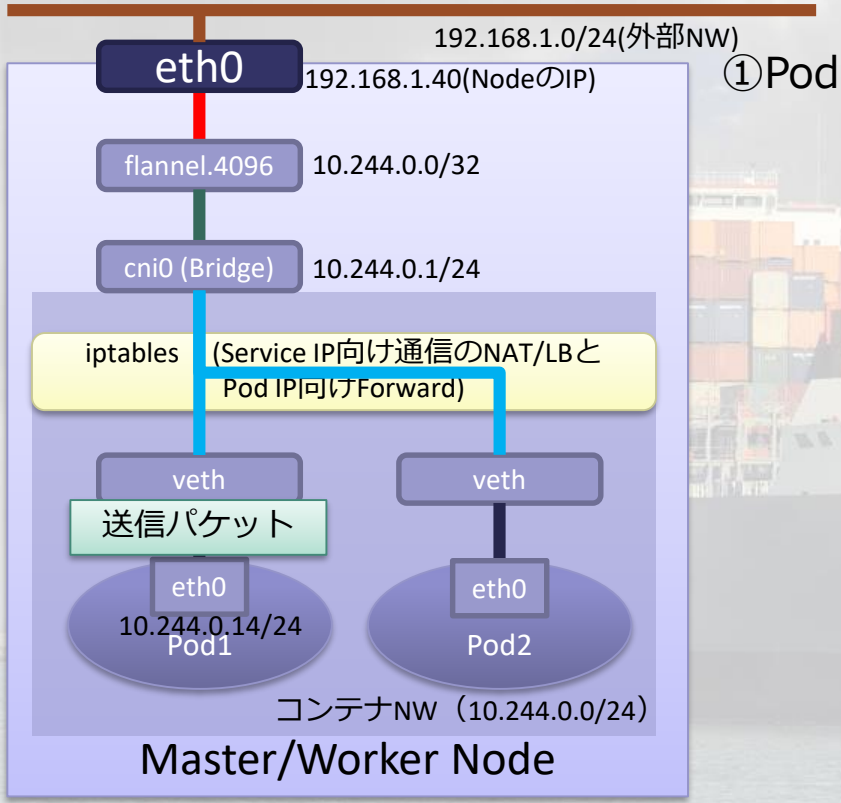
①外部からVXLANカプセル化されたUDPパケット到着
②flannel.4096 VTEPでOuter Header解除
③Inner Headerの宛先IPに従いcni0へルーティング
④cni0のARPテーブルで宛先IPに対応するMACを宛先に設定
⑤iptablesのForwardingルールに従い転送を許可

```
# iptables-save -t filter| grep -v "KUBE" | grep 10.244
-A FORWARD -s 10.244.0.0/16 -j ACCEPT
-A FORWARD -d 10.244.0.0/16 -j ACCEPT
```

受信パケット

Inner Header（宛先） MAC:Pod1 eth0のMAC IP:10.244.0.14 UDP:53	実際の通信データ

ノード内Podからの送信①

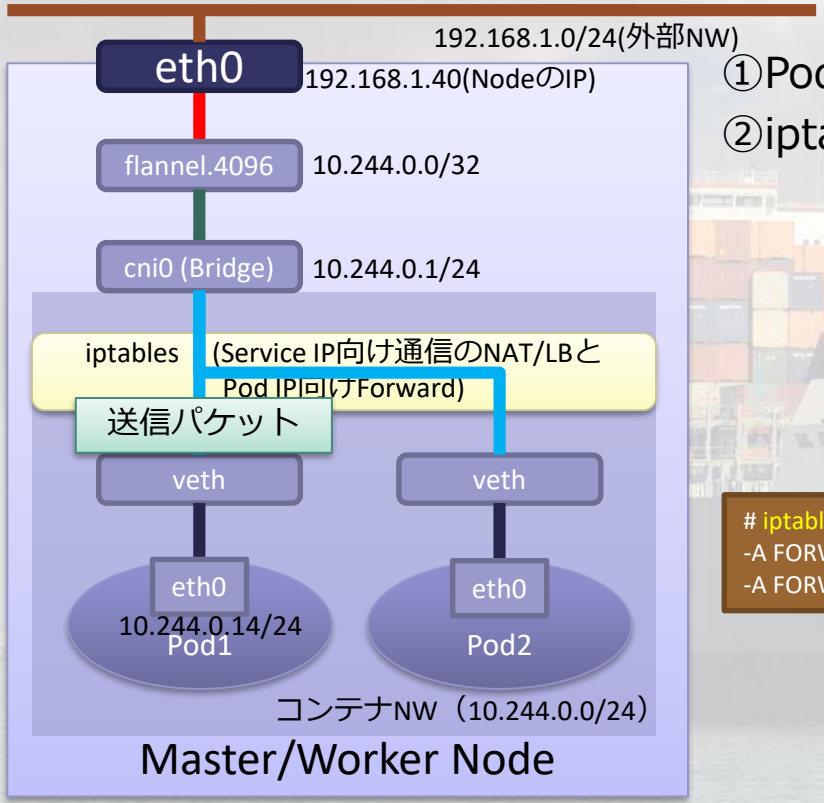


①Pod1で生成されたパケットがNodeに届く

送信パケット

Inner Header（宛先） MAC:cni0のMAC IP:10.244.2.8 TCP:80	実際の通信データ

ノード内Podからの送信②



①Pod1で生成されたパケットがNodeに届く

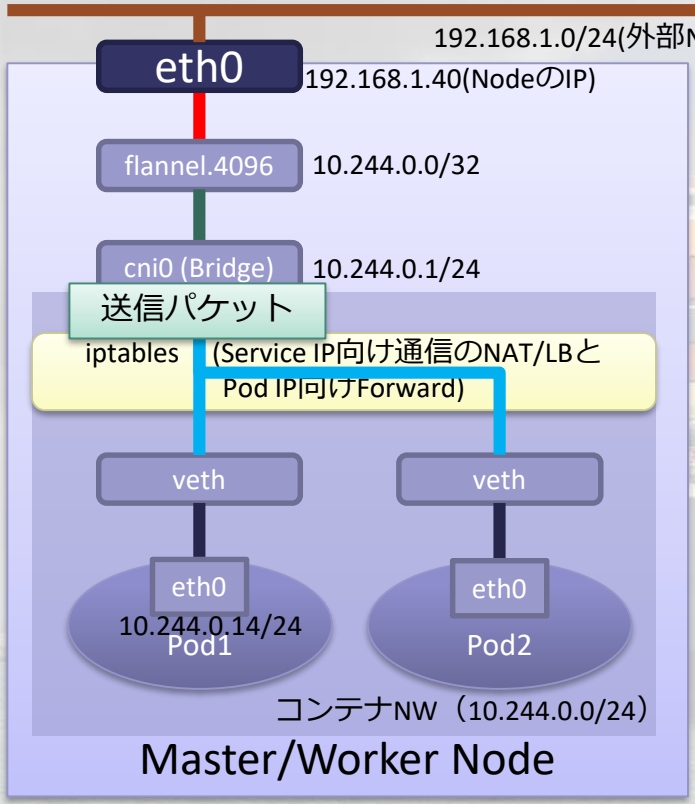
②iptablesのForwarding許可ルールに従い転送許可

```
# iptables-save -t filter| grep -v "KUBE" | grep 10.244
-A FORWARD -s 10.244.0.0/16 -j ACCEPT
-A FORWARD -d 10.244.0.0/16 -j ACCEPT
```

送信パケット

Inner Header（宛先） MAC:cni0のMAC IP:10.244.2.8 TCP:80	実際の通信データ

ノード内Podからの送信③



①Pod1で生成されたパケットがNodeに届く

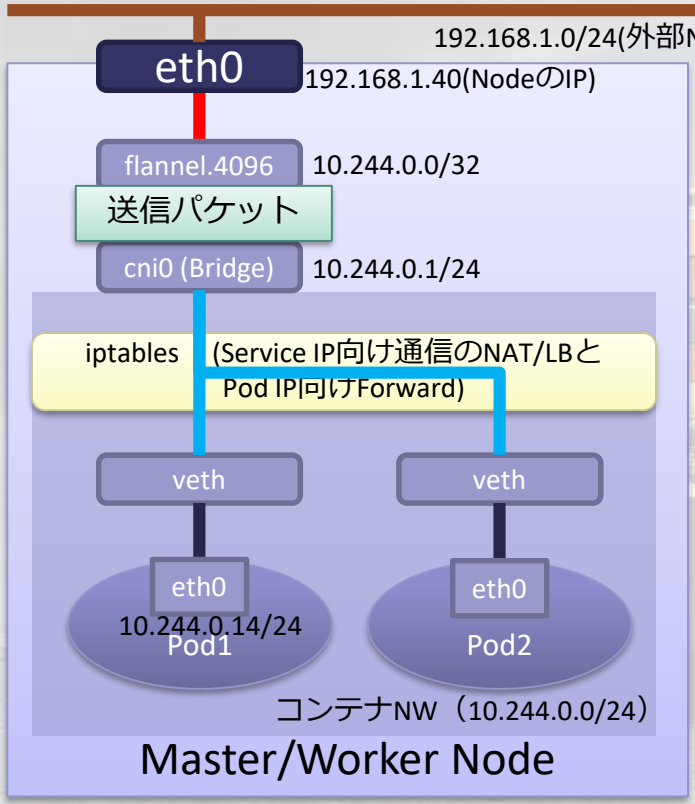
②iptablesのForwarding許可ルールに従い転送許可

③デフォルトGWであるcni0に到着

送信パケット

Inner Header（宛先） MAC:cni0のMAC IP:10.244.2.8 TCP:80	実際の通信データ

ノード内Podからの送信④



①Pod1で生成されたパケットがNodeに届く

②iptablesのForwarding許可ルールに従い転送許可

③デフォルトGWであるcni0に到着

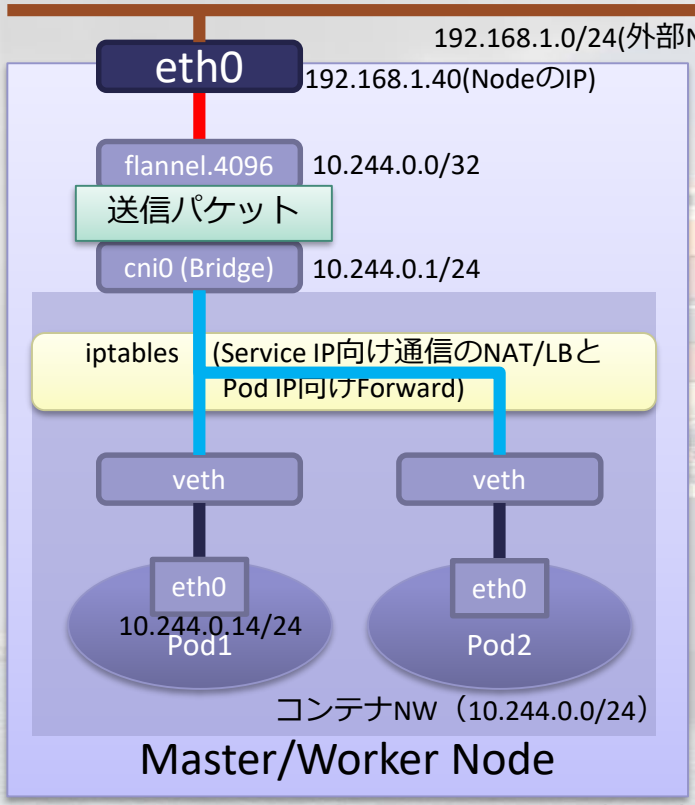
④Static Route設定に従いflannel.4096に転送されつつ、Next Hopが10.244.2.0/32に指定される

```
# ip route | grep 10.244
10.244.2.0/24 via 10.244.2.0 dev flannel.4096 onlink
```

送信パケット

Inner Header（宛先） MAC:cni0のMAC IP:10.244.2.8 TCP:80	実際の通信データ

ノード内Podからの送信⑤



①Pod1で生成されたパケットがNodeに届く

②iptablesのForwarding許可ルールに従い転送許可

③デフォルトGWであるcni0に到着

④Static Route設定に従いflannel.4096に転送されつつ、Next Hopが10.244.2.0/32に指定される

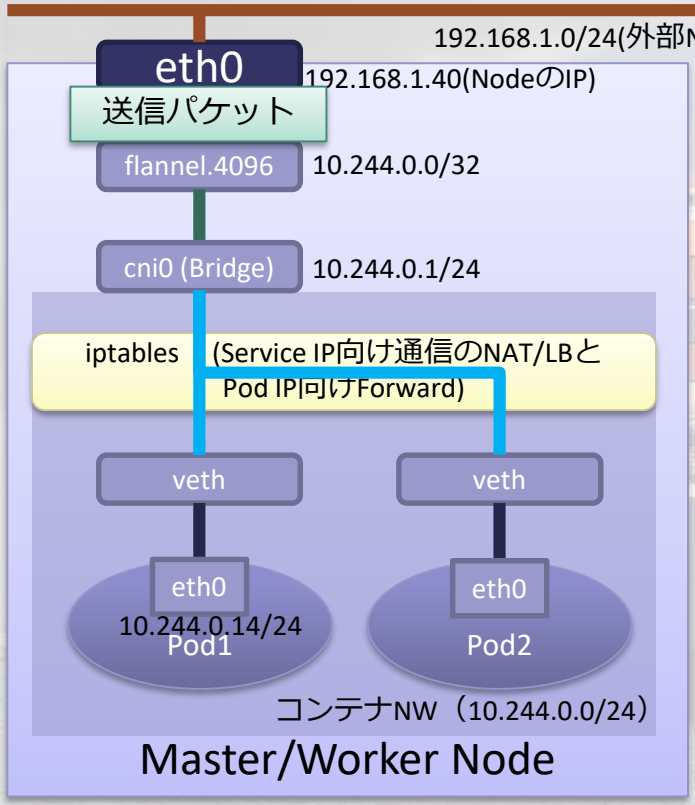
⑤flannel.4096上の静的MACアドレステーブルによりInner HeaderのMACアドレスを宛先VTEPに更新。

```
# ip neigh | grep flannel
10.244.2.0 dev flannel.4096 lladdr 00:15:5d:3c:3e:3c PERMANENT
```

送信パケット

Inner Header（宛先） MAC:**宛先VTEPのMAC** IP:10.244.2.8 TCP:80	実際の通信データ

ノード内Podからの送信⑥



①Pod1で生成されたパケットがNodeに届く

②iptablesのForwarding許可ルールに従い転送許可

③デフォルトGWであるcni0に到着

④Static Route設定に従いflannel.4096に転送されつつ、Next Hopが10.244.2.0/32に指定される

⑤flannel.4096上の静的MACアドレステーブルによりInner HeaderのMACアドレスを宛先VTEPに更新。

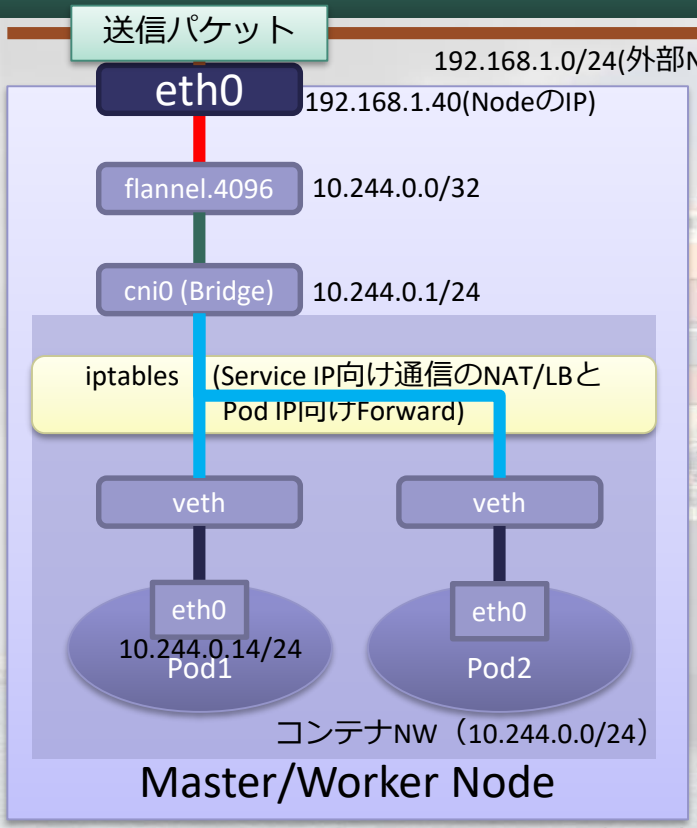
⑥”bridge fdb”に従いVXLANカプセル化される。

```
# bridge fdb | grep flannel
00:15:5d:3c:3e:3c dev flannel.4096 dst 192.168.1.51 self permanent
```

送信パケット

Outer Header（宛先） MAC:eth0のMAC IP:192.168.1.42 UDP:4789	Inner Header（宛先） MAC:宛先VTEPのMAC IP:10.244.2.8 TCP:80	実際の通信データ

ノード内Podからの送信⑦



①Pod1で生成されたパケットがNodeに届く

②iptablesのForwarding許可ルールに従い転送許可

③デフォルトGWであるcni0に到着

④Static Route設定に従いflannel.4096に転送されつつ、Next Hopが10.244.2.0/32に指定される

⑤flannel.4096上の静的MACアドレステーブルによりInner HeaderのMACアドレスを宛先VTEPに更新。

⑥”bridge fdb”に従いVXLANカプセル化される。

⑦eth0のMACテーブルに従いOuter HeaderのMACアドレスを宛先IPのものに更新。

送信パケット

Outer Header（宛先） MAC:**宛先NodeのMAC** IP:192.168.1.42 UDP:4789	Inner Header（宛先） MAC:宛先VTEPのMAC IP:10.244.2.8 TCP:80	実際の通信データ

Service IPの利用状況確認

- Service IP利用状況をkubectlで確認。
- 下記にある10.106.228.152宛通信を例にNAT・LB設定を見ていく。
- 設定は全て「iptables-save」コマンドで見ることができる。

```
$ kubectl get service --all-namespaces
NAMESPACE   NAME          TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
default     kubernetes    ClusterIP     10.96.0.1       <none>        443/TCP          153d
default     win-webserver LoadBalancer  10.106.228.152  <pending>     80:31992/TCP     89d
kube-system kube-dns      ClusterIP     10.96.0.10      <none>        53/UDP,53/TCP    153d
```

Service IP向け通信のNAT/LB設定

- Natテーブル上でPREROUTING＞KUBE-SERVICESとチェーンが続く。

```
-A PREROUTING -m comment --comment "kubernetes service portals" -j KUBE-SERVICES
```

- Cluster CIDR内からならKUBE-SVC-34・・・チェーンに続く。その他のソースからの場合はマスカレードする。

```
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.106.228.152/32 -p tcp -m comment --comment "default/win-webserver: cluster IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.106.228.152/32 -p tcp -m comment --comment "default/win-webserver: cluster IP" -m tcp --dport 80 -j KUBE-SVC-34RXOMW5IY6AEBVE
```

- 50%の確率で”KUBE-SEP-3U・・・”か”KUBE-SEP-Y5・・・“に続く。（ロードバランシング）

```
-A KUBE-SVC-34RXOMW5IY6AEBVE -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-3UI7KTYFEJOGRG25
-A KUBE-SVC-34RXOMW5IY6AEBVE -j KUBE-SEP-Y5YCUEFVTEI2Q4C7
```

- “KUBE-SEP-・・・”で実際のPod宛にIPが変換される。以後は通信ルールに従う。

```
-A KUBE-SEP-3UI7KTYFEJOGRG25 -s 10.244.2.8/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-3UI7KTYFEJOGRG25 -p tcp -m tcp -j DNAT --to-destination 10.244.2.8:80
```

Section 3

Section 3：VXLAN Overlay構成でのコンテナ間通信（Windows）

Windowsでは

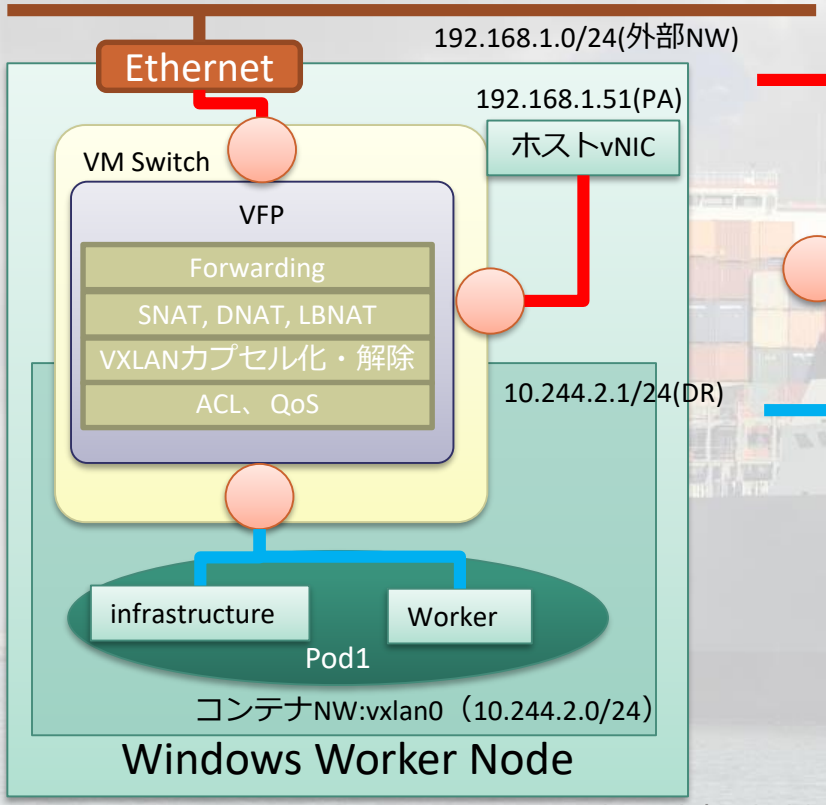
WindowsでVXLAN Overlayと言ったら、
もちろんみんな大好き

Microsoft SDN v2

前提知識（Microsoft SDN v2版）

- VFPの仕組み
  - 物理NWとVNETの隔離（PAとCA）
  - PAのルーティングポリシー
  - NATポリシー
  - HNS Endpoint設定
- VXLANの仕様　　（RFC7348）
  https://tools.ietf.org/html/rfc7348

Windows Node内ネットワーク概要

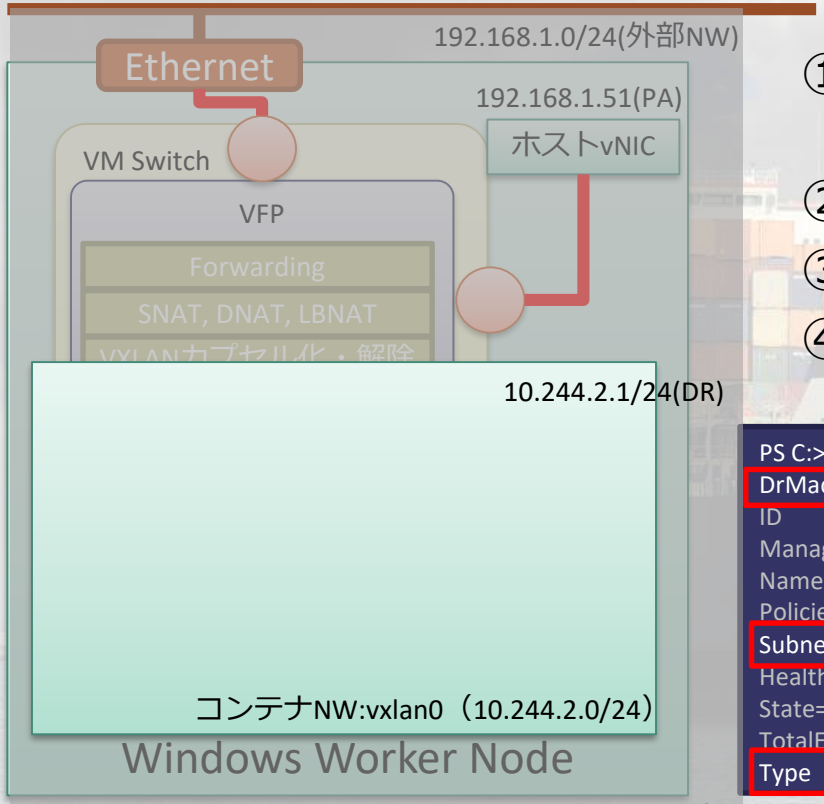


VXLANカプセル化された後の外部向け、もしくはService IP向けLB通信（MTU 1,500）

VM SwitchのPort(もしくはVFP Endpoint)

Pod向けの内部通信（MTU 1,450）

仮想ネットワーク定義（vxlan0）



① このNW向けの分散ルーターMACアドレスは「00:15:5D:3C:3E:3C」

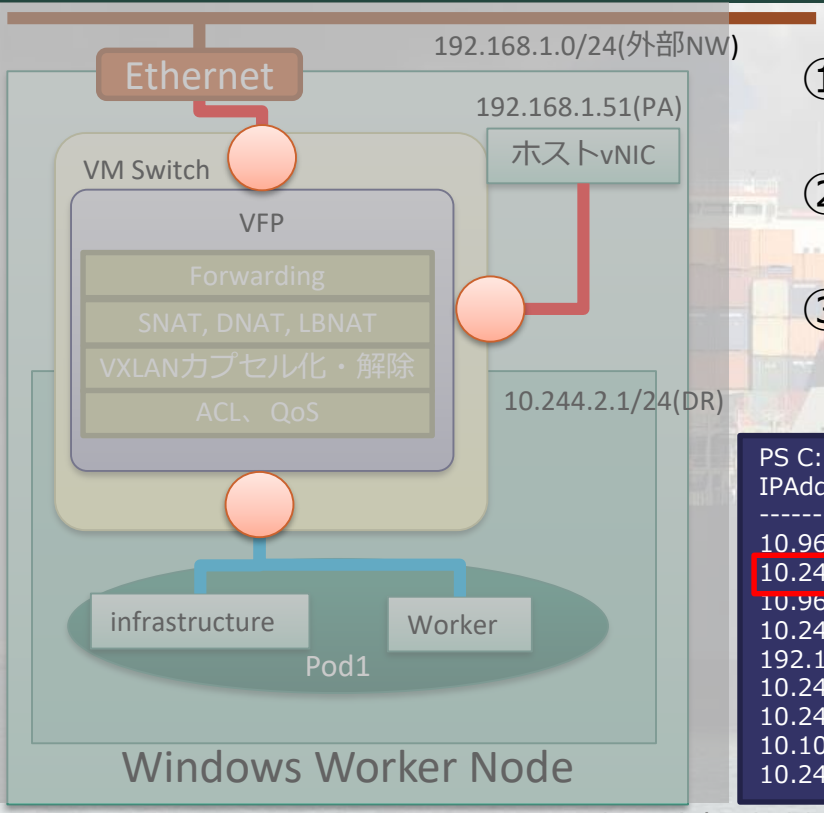
② 内部Subnetは「10.244.2.0/24」

③ GWは「10.244.2.1」

④ 仮想ネットワークを「Overlay」として定義

```
PS C:> Get-HnsNetwork |where{$_.name -eq "vxlan0"}
DrMacAddress        : 00-15-5D-3C-3E-3C   ①
ID                  : 8F43A6E1-78A0-4B11-B580-CF9A1C8CF5D9
ManagementIP        : 192.168.1.51
Name                : vxlan0
Policies            : （後述のため省略）
Subnets             : {@{AdditionalParams=; AddressPrefix=10.244.2.0/24; GatewayAddress=10.244.2.1;   ② ③
Health=; ID=D0553048-8B77-4DF6-B3FE-A3E953573369; ObjectType=5; Policies=System.Object[];
State=0}}
TotalEndpoints      : 3
Type                : Overlay   ④
```

VFP Endpoint定義



① Service IP(10.96.0.0/12)向けLBNAT用Source IPとして「10.244.2.2」を定義

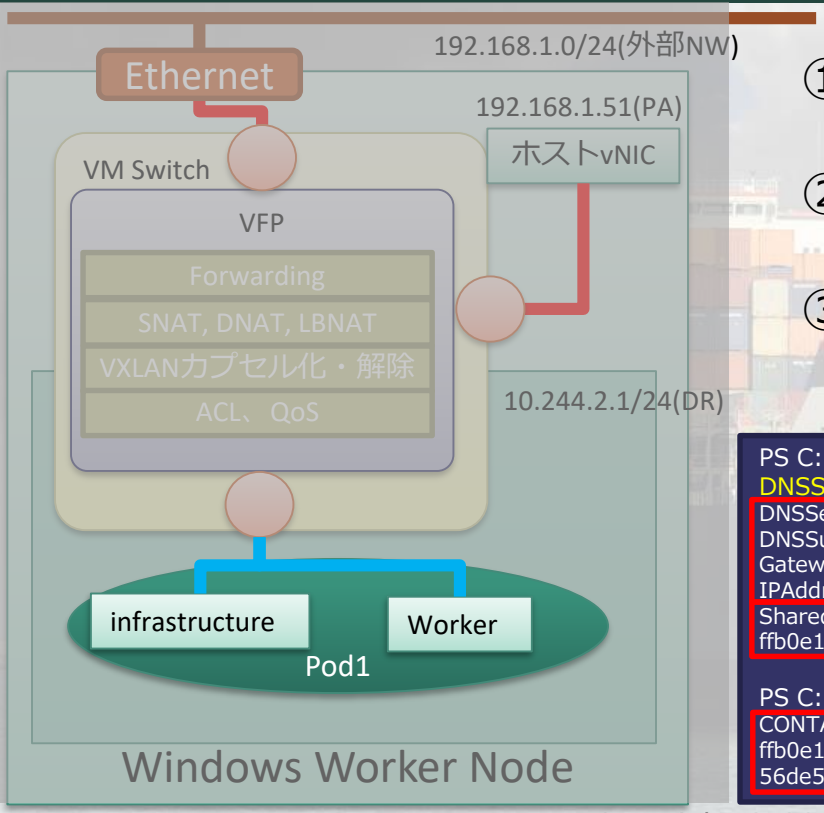
② Service IP向けEndpointは全て①と同じMACアドレスを持つ

③ 別NodeにあるEndpointについては宛先PAを定義。Service IPについては自PAを指定。

```
PS C:> Get-HnsEndpoint |ft IPAddress,MACAddress,IsRemoteEndpoint,Policies
IPAddress       MacAddress        IsRemoteEndpoint Policies
---------       ----------        ---------------- --------
10.96.0.10      00:15:5d:01:0b:11 True             {@{PA=192.168.1.51; Type=PA}}
10.244.2.2 (1)  00:15:5d:01:0b:11                  {}
10.96.0.1       00:15:5d:01:0b:11 True             {@{PA=192.168.1.51; Type=PA}}
10.244.0.14     02-11-0a-f4-00-0e True             {@{PA=192.168.1.40; Type=PA}}
192.168.1.40    02-11-c0-a8-01-28 True             {@{PA=192.168.1.40; Type=PA}}
10.244.0.15     02-11-0a-f4-00-0f True             {@{PA=192.168.1.40; Type=PA}}
10.244.2.8      0E-2A-0a-f4-02-08                  (後述のため省略)
10.106.228.152  00:15:5d:01:0b:11 True             {@{PA=192.168.1.51; Type=PA}}
10.244.3.20     02-11-0a-f4-03-14 True             {@{PA=192.168.1.52; Type=PA}}
```

(2) (3)

Pod内の複数コンテナでのEndpoint共有



① PodのIP, GW, DNSの設定はVFP Endpoint側で定義される。

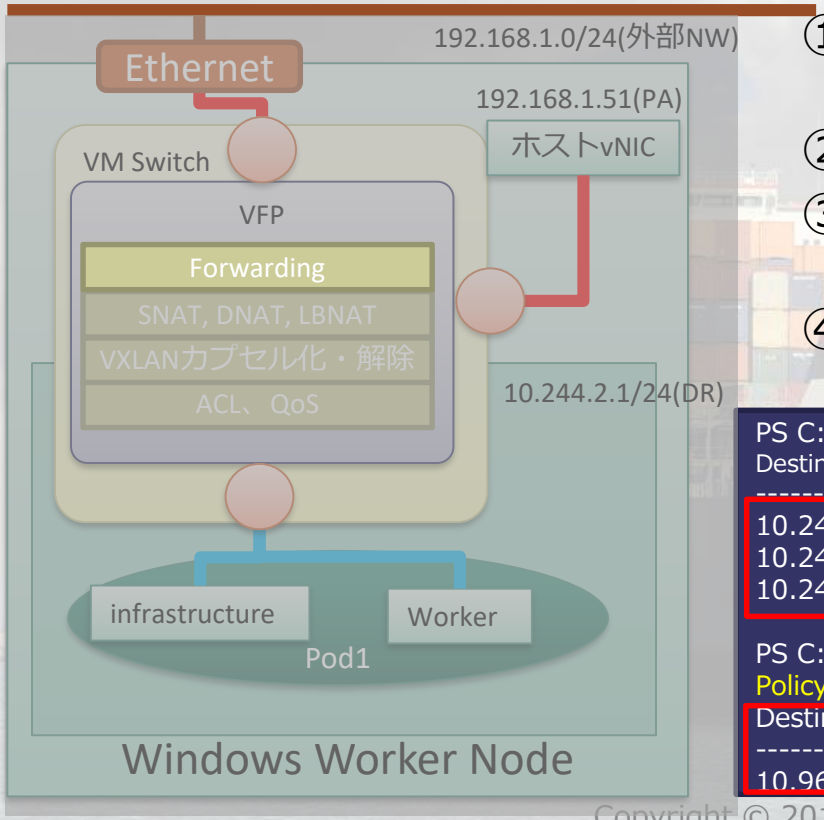
② Endpointを使うコンテナの一覧を指定（複数指定可能、同一Pod内のコンテナを指定）

③ ②で指定するコンテナのIDはdockerコマンドで見えるもの。

```
PS C:> (Get-HnsEndpoint)[6] |fl
DNSServerList,DNSSuffix,GatewayAddress,IPAddress,SharedContainers
DNSServerList      : 10.96.0.10
DNSSuffix          : default.svc.cluster.local   ①
GatewayAddress     : 10.244.2.1
IPAddress          : 10.244.2.8
SharedContainers   : {56de574d562468bf91e8aa7d35509eaa1c73bbef02d432d427b76a7e40c69538,
ffb0e1693332953d3226999d116bb2e669189e582dd1c1c50e04bb76cfd29dfa}   ②

PS C:> docker ps
CONTAINER ID        (以下省略)
ffb0e1693332        (以下省略)   ③
56de574d5624        (以下省略)
```

Remote EndpointへのForwarding定義



① 別NodeにあるコンテナNW（CA）向けのパケットの宛先VTEPのMACアドレスを指定。

② VNIはすべて4096。

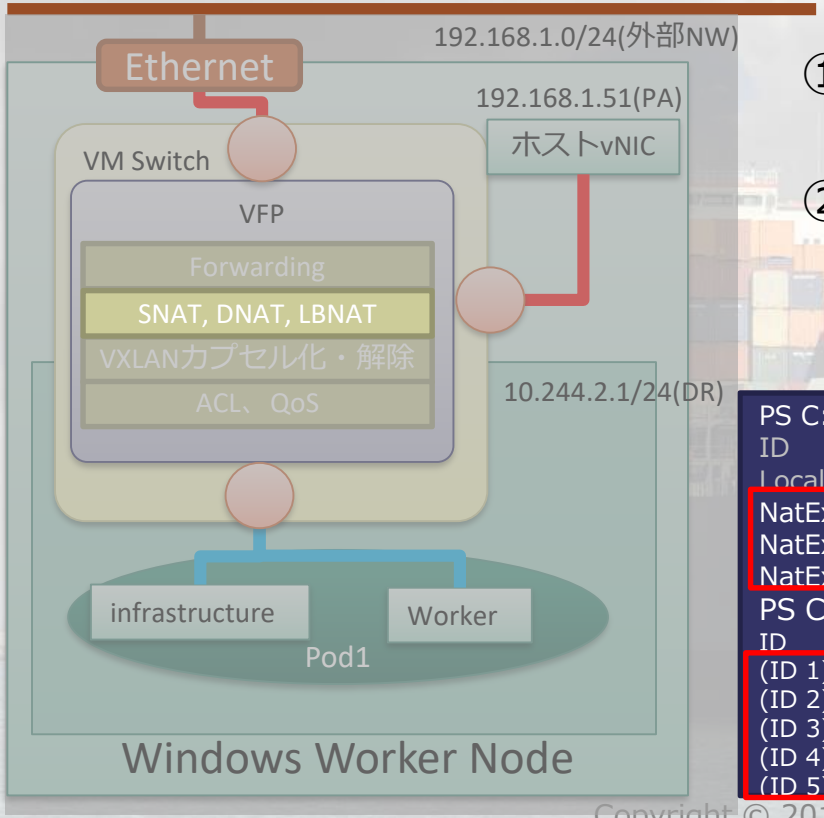
③ VXLANカプセル化後のOuter宛先IP（PA）を指定。

④ Service IP向けは自NodeのPAを指定し、カプセル化必要と定義。

```
PS C:> (Get-HnsNetwork).policies |ft
DestinationPrefix DistributedRouterMacAddress IsolationId ProviderAddress Type
----------------- ------------------------- ----------- --------------- ----
10.244.0.0/24  ce:7a:e5:af:6a:e9  4096  192.168.1.40 RemoteSubnetRoute
10.244.1.0/24  06:cd:30:81:7a:0d  4096  192.168.1.41 RemoteSubnetRoute
10.244.3.0/24  00:15:5d:91:50:41  4096  192.168.1.52 RemoteSubnetRoute

PS C:> (Get-HnsEndpoint).Resources.Allocators |where{$_.Tag -eq "VFP Route
Policy"}|ft DestinationPrefix,NeedEncap
DestinationPrefix NeedEncap
----------------- ---------
10.96.0.0/12          True
```

NAT、LBルール定義



① コンテナNW、Service IPは外向けNATを直接しないでカプセル化する設定。

② Service IP、NodePortについて負荷分散先Endpoint（Local/Remote問わず）を指定。

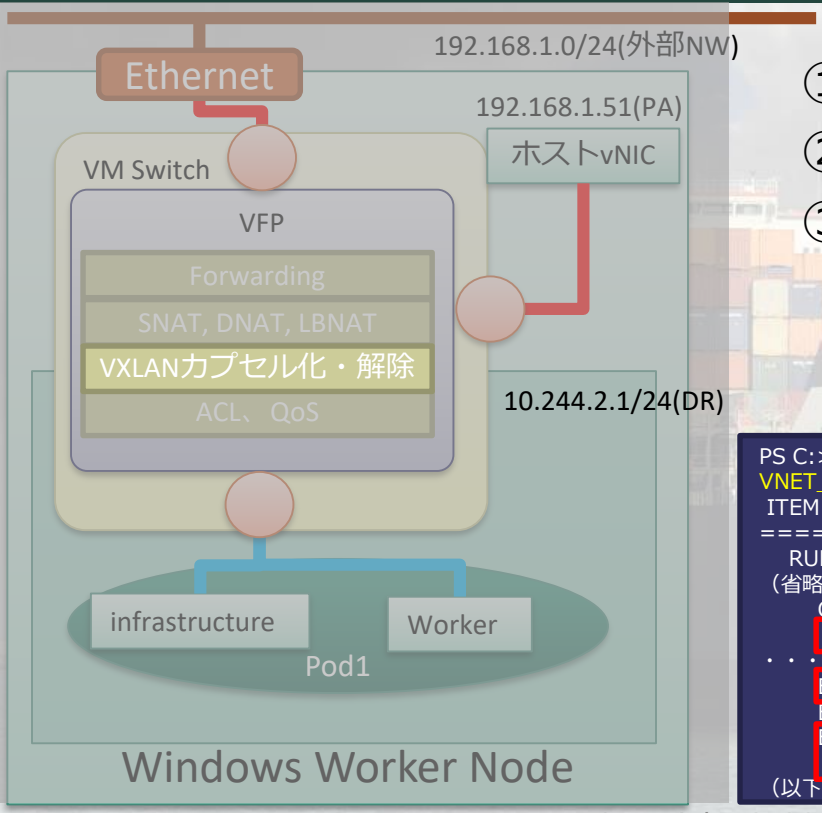
```
PS C:> (Get-HnsEndpoint).Resources.Allocators|where {$_.Tag -like "*NAT*"}
ID                : B672158D-754E-4D90-9513-61F7E72753CE
LocalRoutedVip    : False
NatExceptions_0   : 10.244.0.0/16
NatExceptions_1   : 10.96.0.0/12
NatExceptions_2   : 10.244.2.0/24
PS C:> hnsdiag list loadbalancers
ID      | Virtual IPs          | Direct IP IDs
(ID 1) |  10.96.0.10          | (VFP Endpoint ID A) (VFP Endpoint ID B)
(ID 2) |  10.96.0.10          | (VFP Endpoint ID A) (VFP Endpoint ID B)
(ID 3) |  10.96.0.1           | (VFP Endpoint ID C)
(ID 4) |  10.106.228.152      | (VFP Endpoint ID D) (VFP Endpoint ID E)
(ID 5) |                      | (VFP Endpoint ID D) (VFP Endpoint ID E)
```

①

②

←NodePort用LB設定

VXLANカプセル化（テナント分離）定義



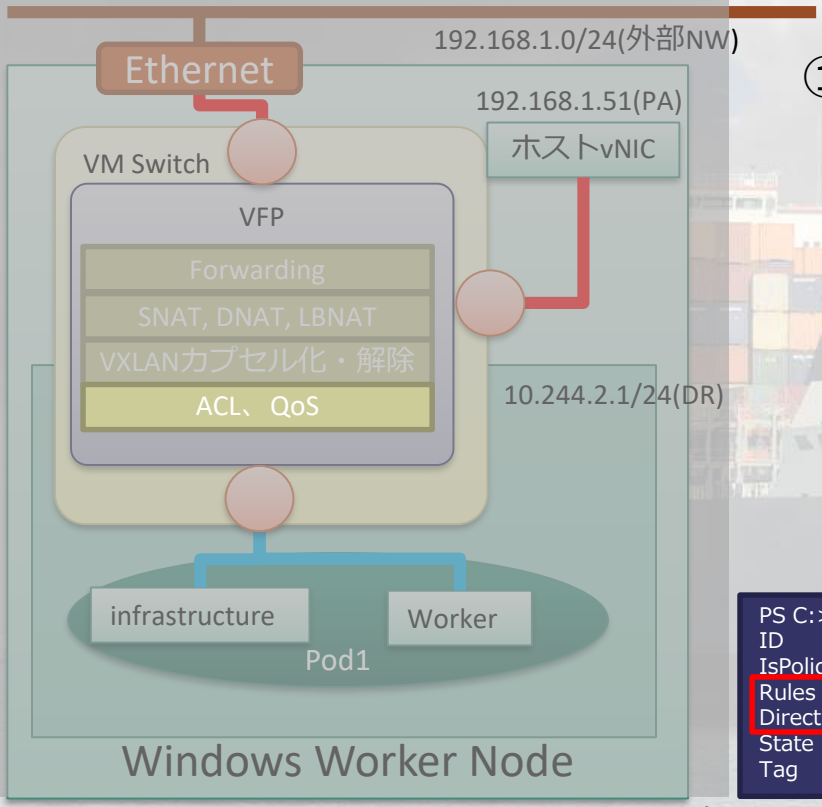
① 宛先Subnetを定義

② VXLANでカプセル化する定義

③ VXLANカプセル化した後のOuter Headerの宛先PA、宛先MACアドレス、VNIを指定。

```
PS C:> vfpctrl /port d7659331-344c-407c-898d-c68f71527484 /layer VNET_ENCAP_LAYER /group
VNET_GROUP_ENCAP_IPV4_OUT /list-rule
 ITEM LIST
============
   RULE：（省略）
（省略）
      Conditions:
         Destination IP：10.244.0.0-10.244.0.255 ①
・・・
      Encap Type: VXLAN ②
      Encap Source IP：192.168.1.51
      Encap Destination(s)：
         { IP address=192.168.1.40, MAC address=CE-7A-E5-AF-6A-E9, GRE key=4096 } ③
（以下、他Subnet向けのVXLANカプセル化ルール）
```

ACLの設定

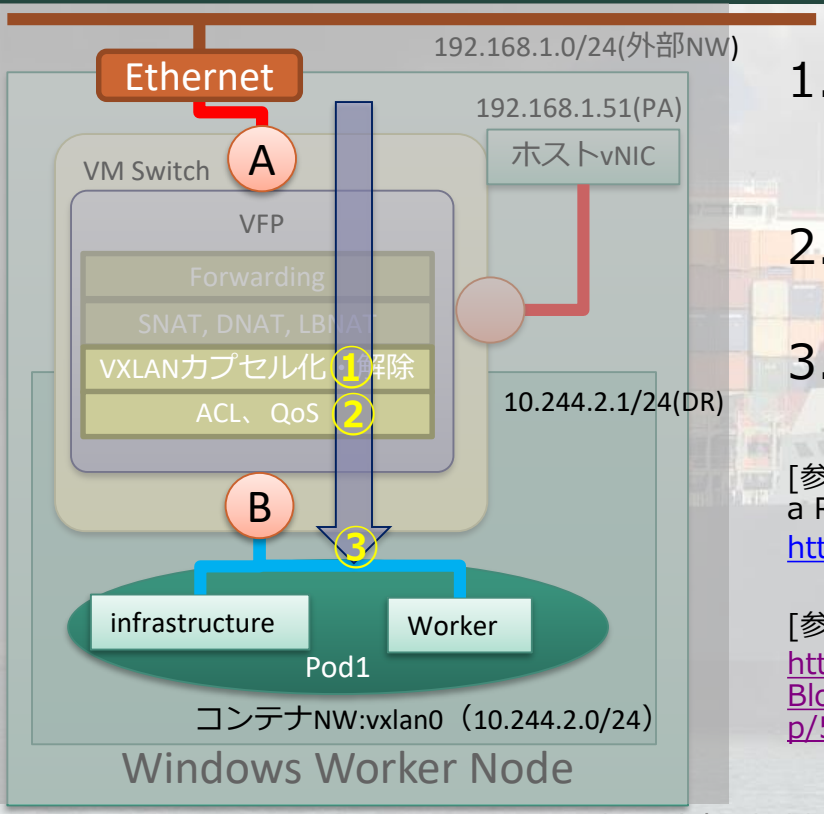


① 内向き、外向きでANYで通信を許可

```
PS C:> (Get-HnsEndpoint).Resources.Allocators|where {$_.Tag -like "*ACL*"}
ID          : 47714056-E296-4DA0-824F-D14A49D2CD46
IsPolicy    : True
Rules       : {@{Action=Allow; Direction=In; Priority=65500; Type=ACL}, @{Action=Allow;
Direction=Out; Priority=65500; Type=ACL}}
State       : 3
Tag         : ACL Policy
```

①

自ノード内PodのIP向け通信（Inbound）

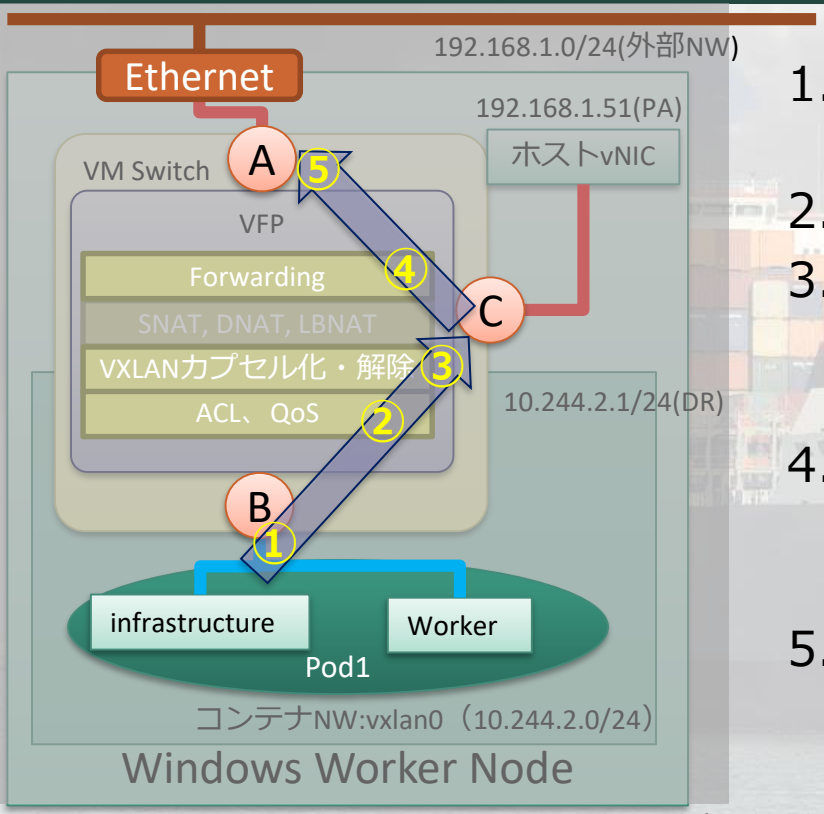


1. 物理NICから届いたVXLANパケットを VNI, 宛先PA, 宛先MACを見てカプセル化解除
2. ACL（Anyで許可）でPodのつながるEndpointへ転送
3. Podのプロセス群にパケットが到着

[参考] Understanding Windows Container Network in Kubernetes Using a Real Story (@ KubeCon CN 2018)
https://www.youtube.com/watch?v=tTZFoiLObX4&feature=youtu.be

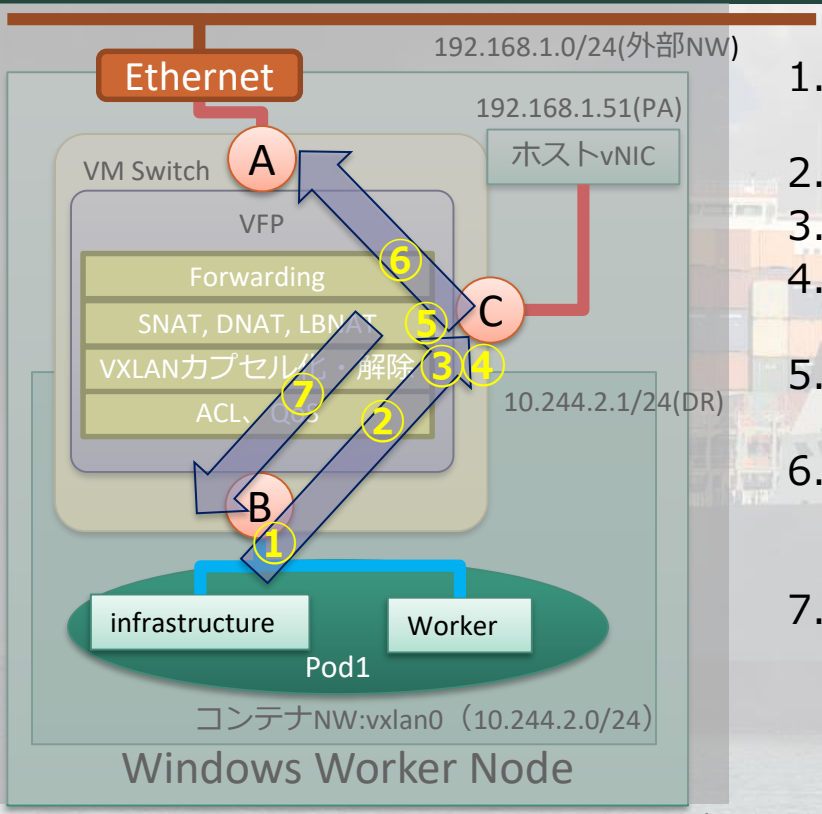
[参考] Troubleshooting Kubernetes Networking on Windows: Part 1
https://techcommunity.microsoft.com/t5/Networking-Blog/Troubleshooting-Kubernetes-Networking-on-Windows-Part-1/ba-p/508648

他ノードのPodのIP向け通信（Outbound）



1. コンテナから送信したパケットがVFP Endpoint（B）に到着する。
2. ACKでAny許可される。
3. 宛先IPに対応するVNI, VTEP向けにVXLANカプセル化して管理Endpoint（C）に送信する。
4. Endpoint（C）のForwarding設定に従い宛先VTEPに対応するPAをOuter Headerの宛先IPに設定する。
5. 作成したパケットを物理NIC向けVM Switchポート（A）から外部へ送信する。

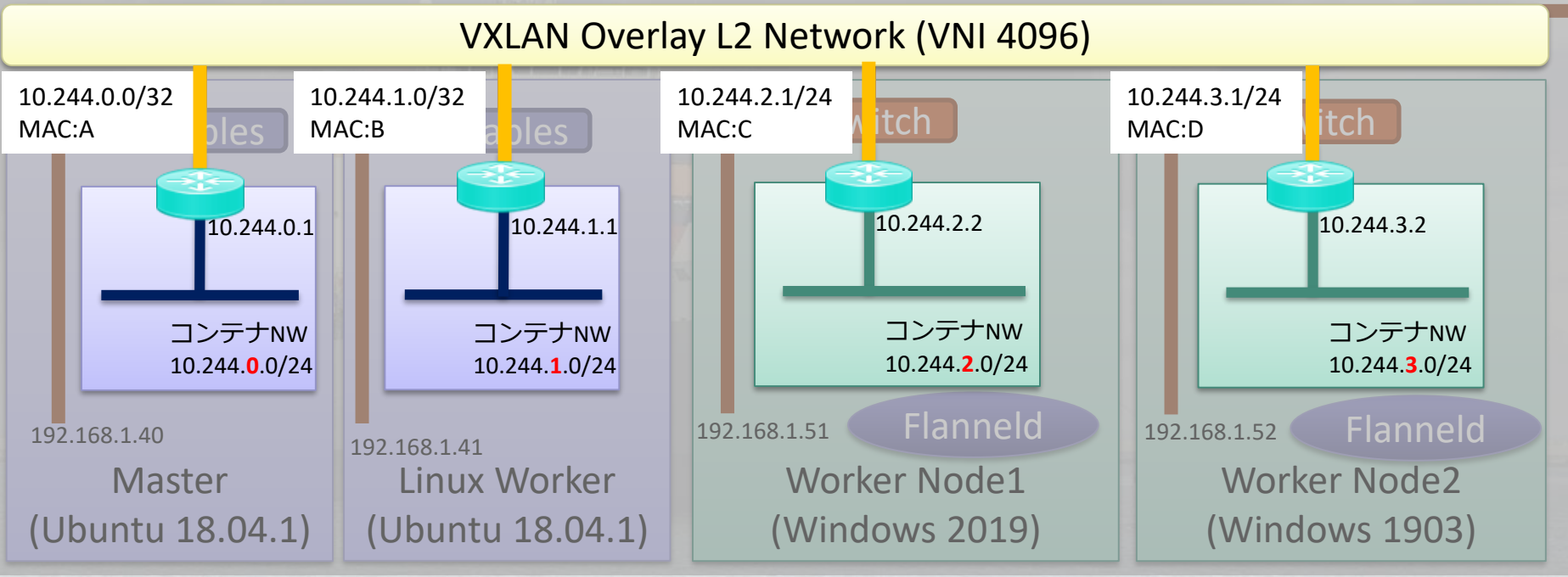
Service IP向け通信



1. コンテナから送信したパケットがVFP Endpoint（B）に到着する。
2. ACKでAny許可される。
3. 自ノードのVTEP向けにVXLANカプセル化する。
4. ホストvNIC側Endpoint（C）でカプセル化を解除した後、LBNATルールを適用
5. LBの分散先Endpoint（Podの実IPに対応するEndpoint）に合わせて再度VXLANカプセル化する。
6. 他ノード向けの場合はForwarding設定に従い宛先VTEPに対応するPAをOuter Headerの宛先IPに設定して外部Endpoint（A）から送信。
7. 自ノード向けの場合は外部からの受信時と同じ処理を行い、Endpoint（B）にパケットを転送。

ここまでのまとめ

各Node上のルーター間をVXLAN Overlay L2接続



Microsoft SND v2とKubernetesの対応表

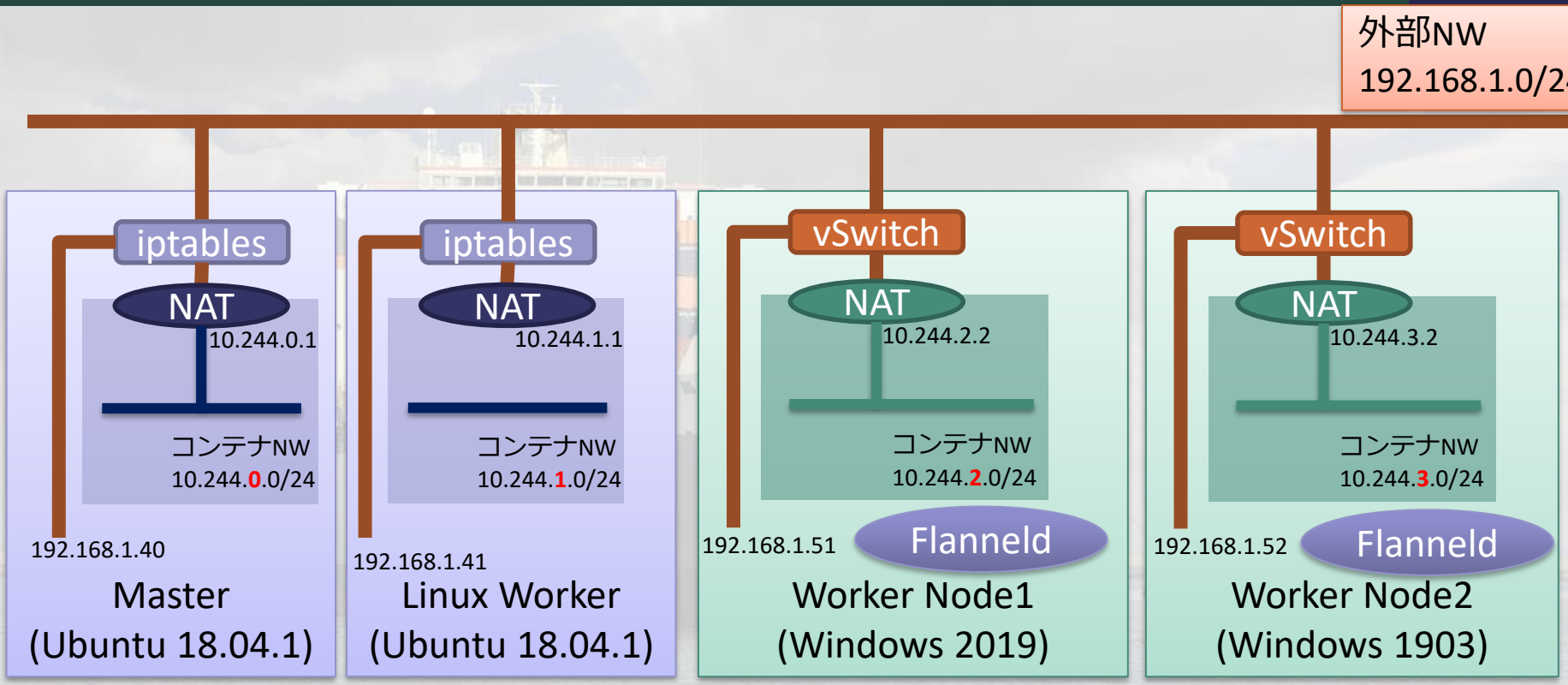
SNDのレイヤー	Kubernetes	Microsoft SDN v2
Management Plane	Kubernetes	SCVMM, API, PowerShellなど
Mgmt/Ctrl間のやり取り	CNI	NC Northband API
Control Plane	Flannel、 Calicoなど	Network Controller (NC)
Ctrl/Data間のやり取り	Kubelet＋Flanneld	NC Host Agentなど
Data Plane	VM Switch, VFP	VM Switch, VFP, SLBMUXなど

Microsoft SDN v2の機能のうち、 Data Planeの機能のみをKubernetesでは利用している。
そのため、 Standard EditionでもVXLAN Overlay Networkを作ることができる。

Section 4

FlannelによるVXLAN Overlay構成の構築手順紹介

本日構築するデモ環境



構築の全体的なステップ

1. Kubeadmコマンドを使ったKubernetes Master Nodeのインストール(今回省略)
2. Master NodeでのFlannelのインストール
3. Windows Worker Nodeの構築
4. Kubernetesのサービス化

1-1. Flannelのインストール準備①

- iptablesでのBridge許可（デフォルトのはず）

```
$ sudo sysctl net.bridge.bridge-nf-call-iptables=1
```

- Flannelインストール用YAMLダウンロード

```
$ wget https://raw.githubusercontent.com/coreos/flannel/v0.11.0/Documentation/kube-flannel.yml
```

1-2. Flannelのインストール準備②

- Kube-flannel.ymlの変更

```
$ vi kube-flannel.yml
 net-conf.json: |
 {
     "Network": "10.244.0.0/16",
     "Backend": {
       "Type": "vxlan",
       "VNI" : 4096,
       "Port": 4789
     }
   }
(中略)
 nodeSelector:
       beta.kubernetes.io/arch: amd64
       beta.kubernetes.io/os: linux
```

Cluster-CIDRと同じであることを確認

VNIとPortを明示的に指定する必要あり。

追記

1-3. Flannelのインストール

- Kube-flannel.ymlのapply

```
$ kubectl apply -f kube-flannel.yml
```

- （数分後）Flannel用Podが起動したことを確認

```
$ kubectl get pods -o wide --all-namespaces | egrep "NAMESPACE|kube-flannel-ds"
NAMESPACE     NAME                    READY   STATUS    RESTARTS  AGE
kube-system   kube-flannel-ds-zxtpm   1/1     Running   4         33d
```

2-1. Windows Worker設定情報取得

- Master NodeでService CIDR情報取得

```
$ kubectl cluster-info dump | grep -i service-cluster-ip-range
                    "--service-cluster-ip-range=10.96.0.0/12"
```

- Master Nodeでkube-dnsのService IP取得

```
$ kubectl describe svc/kube-dns -n kube-system
Name:              kube-dns
  (中略)
IP:                10.96.0.10
  (以下略)
```

2-2. Windowsコンテナイメージの準備

- WindowsコンテナImageの取得

```
PS C:¥> docker pull microsoft/windowsservercore:ltsc2019
PS C:¥> docker pull microsoft/nanoserver:ltsc2019
```

- Dockerイメージへのタグ付け

```
PS C:¥> docker tag microsoft/windowsservercore:ltsc2019
microsoft/windowsservercore:latest
PS C:¥> docker tag microsoft/nanoserver:ltsc2019 microsoft/nanoserver:latest
```

2-3. Windows Worker Nodeセットアップ

Microsoftの手順そのままでセットアップできますので、解説省略！

[参考] Joining Windows Server Nodes to a Cluster
https://docs.microsoft.com/en-us/virtualization/windowscontainers/kubernetes/joining-windows-workers?tabs=ManagementIP

3-1. Kubernetesのサービス化

- これまではPowershell上でKubernetesのプロセスを動かしてきた。
- KubernetesでのWindows Node GA化に伴い、サービスとして起動する手順も公開。

[参考] Joining Windows Server Nodes to a Cluster
https://docs.microsoft.com/en-us/virtualization/windowscontainers/kubernetes/kube-windows-services?tabs=ManagementIP

3-2. サービス化に必要なファイルの入手

- https://nssm.cc/downloadからnssm.exeをダウンロード
- GithubのMicrosoft/SDNリポジトリからサービス登録用Powershellスクリプトをダウンロード
  https://github.com/microsoft/SDN/blob/master/Kubernetes/flannel/register-svc.ps1

- nssm.exeとregister-svc.ps1をC:¥kフォルダに置く。

3-3. サービス化Scriptの実行

- 各引数に入れるパラメータを確認した上で、サービス化Scriptを実行

```
PS C:¥> cd C:¥k¥
PS C:¥k¥> C:¥k¥register-svc.ps1 -NetworkMode overlay -ManagementIP <NodeのIP> -ClusterCIDR 10.244.0.0/16 -KubeDnsServiceIP 10.96.0.10 -LogDir <Log保存Directory>
```

デモ（時間があれば・・・）

- Linux NodeからService IPへの疎通確認
- Windows NodeからService IPへの疎通（できないこと）の確認

最後に

- ついにKubernetesでWindowsコンテナー対応がGAとなった（長かった・・・）
- 実装の詳細を見るほど、確かに大変そう
- 今後の機能追加にも期待

参考文献

- Kubernetes on Windows
  - https://docs.microsoft.com/en-us/virtualization/windowscontainers/kubernetes/getting-started-kubernetes-windows
- Understanding Windows Container Network in Kubernetes Using a Real Story
  - https://www.youtube.com/watch?v=tTZFoiLObX4&feature=youtu.be
- Troubleshooting Kubernetes Networking on Windows: Part 1
  - https://techcommunity.microsoft.com/t5/Networking-Blog/Troubleshooting-Kubernetes-Networking-on-Windows-Part-1/ba-p/508648
- Hyper-V Network Virtualization Technical Details in Windows Server 2016
  - https://docs.microsoft.com/ja-jp/windows-server/networking/sdn/technologies/hyper-v-network-virtualization/hyperv-network-virtualization-technical-details-windows-server#hyper-v-network-virtualization-concepts
- RFC 7348（VXLAN）
  - https://tools.ietf.org/html/rfc7348
- Windowsコンテナーネットワーク
  - https://docs.microsoft.com/ja-jp/virtualization/windowscontainers/container-networking/architecture
- Container Networking Deep Dive
  - https://www.slideshare.net/hichihara/container-networking-deep-dive-94307233
- コンテナネットワーキング（CNI）最前線
  - https://www.slideshare.net/motonorishindo/cni-124981353
- コピペから脱出！iptablesの仕組みを理解して環境に合わせた設定をしよう
  - https://oxynotes.com/?p=6361
- Interact2016：Introduction of Hyper-V Network Virtualization ver.2
  - https://www.slideshare.net/wind06106/interact2016introduction-of-hyperv-network-virtualization-ver2

セッションアンケートにご協力ください

5セッション目アンケート⇒

END

- ご清聴、ありがとうございました