redhat

Kubernetes 実践入門

2016/11/09 ver.1.0

Yoshikazu YAMADA <yyamada@redhat.com>
Red Hat K.K. DevOps Lead Senior Architect

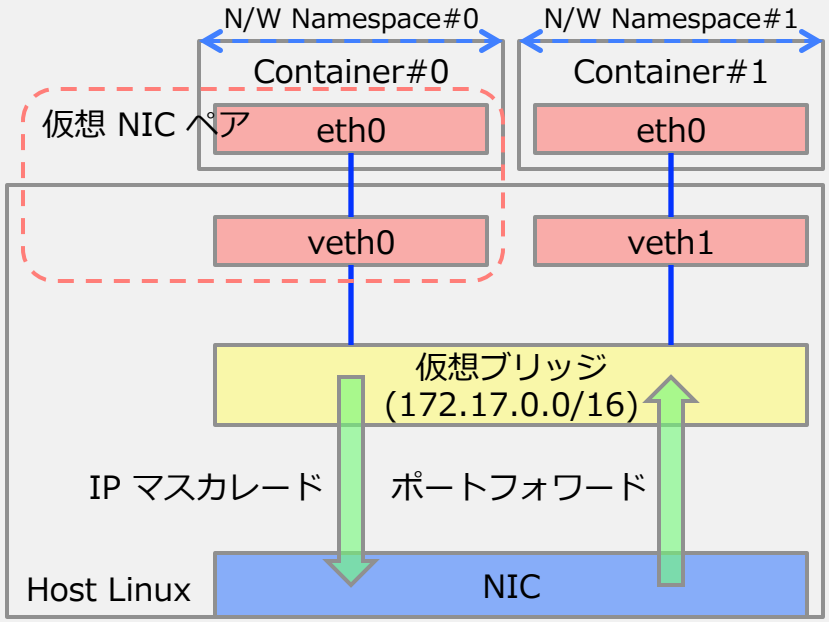
Agenda

1. Kubernetes クラスタの N/W の仕組み
2. Kubernetes の仕組み
3. Kubernetes クラスタの構築

Kubernetes クラスタの N/W の仕組み

1. コンテナの N/W の仕組み
2. 単一 Linux Host で稼働するコンテナの N/W 構成例
3. 複数 Linux Host で稼働するコンテナの N/W 構成例
4. Flannel – 概要
5. Flannel による N/W 構成例

コンテナの N/W の仕組み



仮想 NIC ペア

コンテナ内部に設定された仮想 NIC (eth0) とコンテナ毎に設定された仮想 NIC (veth) のペアで veth は仮想ブリッジに接続

仮想ブリッジ

Linux が提供する仮想 N/W スイッチ (デフォルト 172.17.42.1) で Host Linux 上で稼働する異なるプロセス間の通信に使用

デフォルトで 172.17.0.0/16 のサブネットが割り当てられコンテナ内部の仮想 NIC (eth0) には同サブネットの IP を割り当て

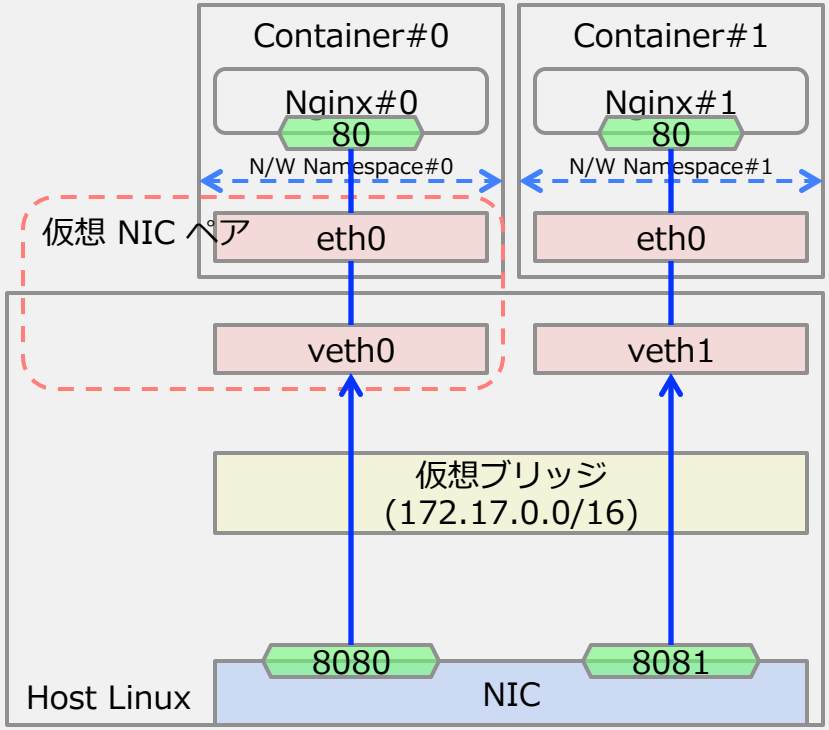
IP マスカレード

Host Linux の iptables によるコンテナから Host Linux 外部への通信

ポートフォワード

Host Linux の iptables による Host Linux からコンテナへの通信

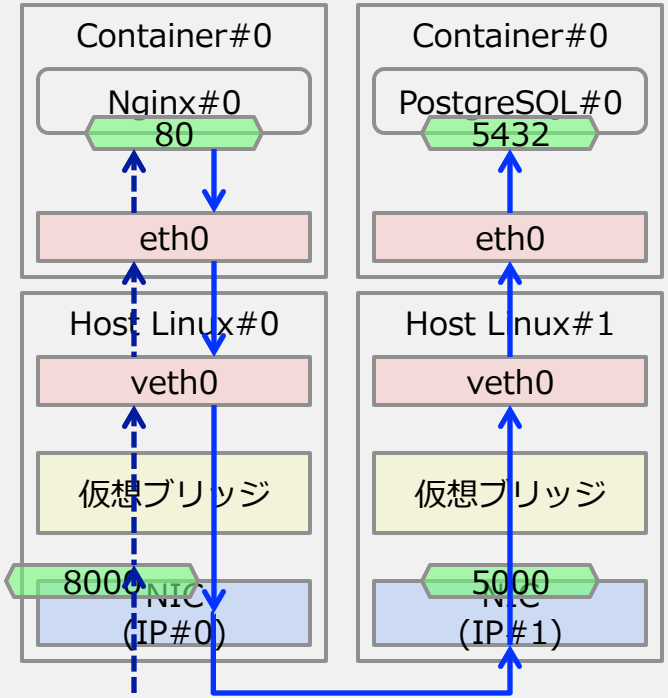
単一 Linux Host で稼働するコンテナの N/W 構成例



単一の Host Linux 上で同一のポート番号にバインドされたコンテナを複数起動する場合、コンテナ毎に異なる転送元ポート番号を設定することで対応可能。

Host Linux の外部からは Host Linux の IP アドレス：転送元ポート番号 でアクセスし、対応するコンテナに転送される。

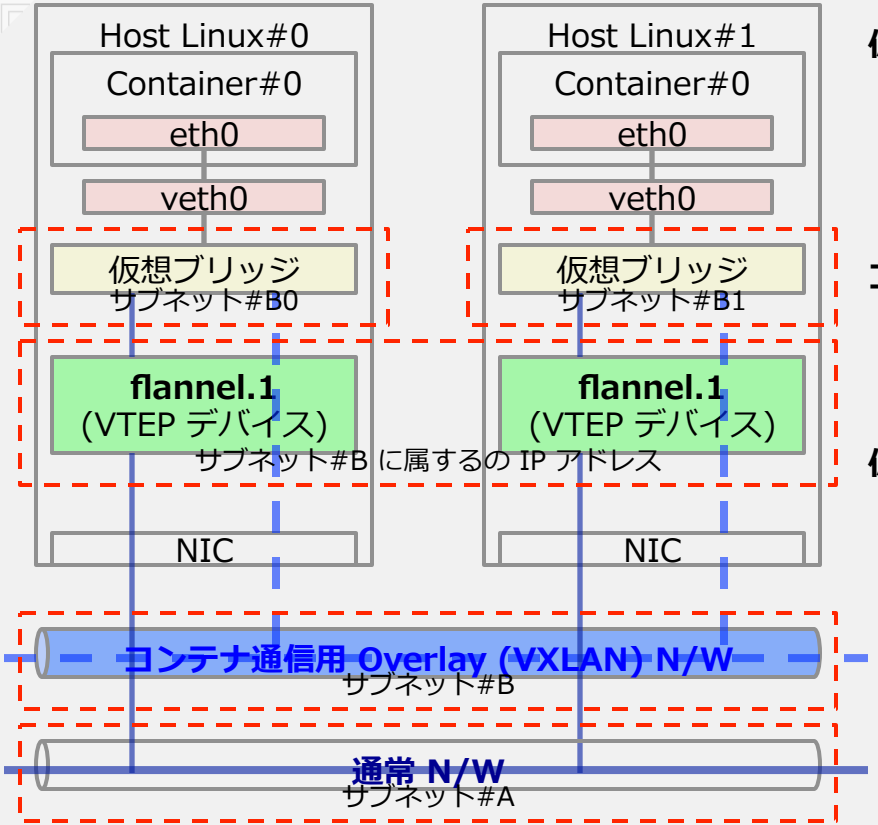
複数 Linux Host で稼働するコンテナの N/W 構成例



異なる Linux Host で稼働するコンテナを連携する場合、接続元コンテナは接続先コンテナが稼働する Linux Host の IP アドレス:転送元ポート番号 (※ 左記の例では IP#1:5000) に対して接続を行う。

接続元コンテナでは接続先の具体的な IP アドレス:ポート番号 を管理 (通常は環境変数で管理) する必要があり、接続先コンテナの IP アドレス が変更された場合 (フェイルオーバ 等) には別途対応する必要がある。

Flannel 概要



仮想 NIC (flannel.1)

Flannel が設定する仮想 NIC でコンテナ通信用オーバレイ N/W に接続される。**TODO** ソースコード確認

同仮想 NIC はコンテナ通信用オーバレイ N/W に割り当てられたサブネット中の IP アドレスが割り当てられる。

コンテナ通信用オーバレイ N/W

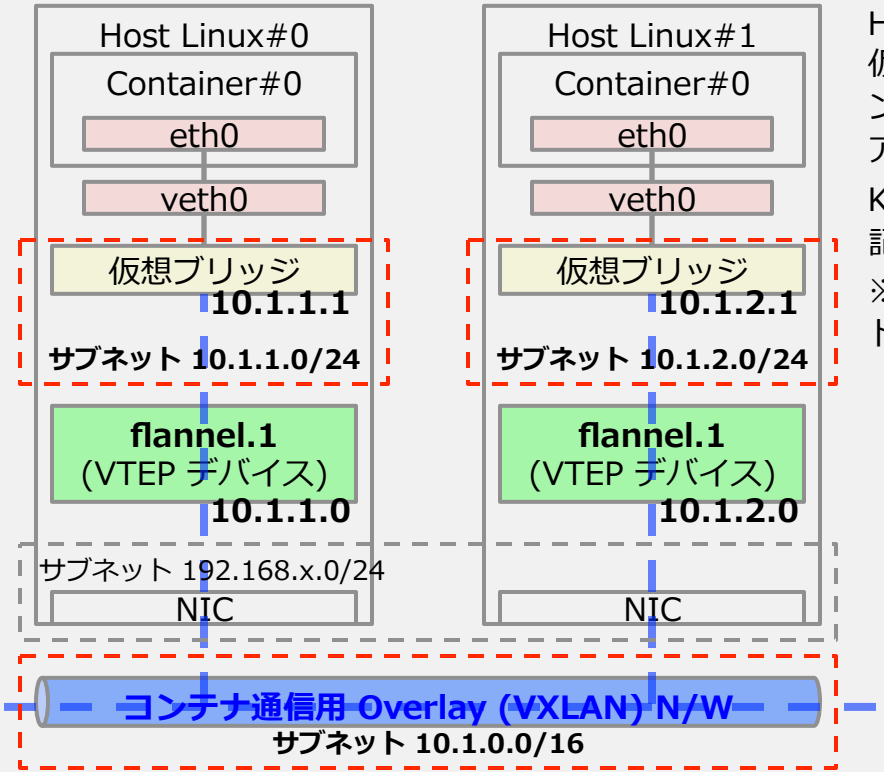
Flannel により構築された VXLAN によるオーバレイ N/W で、Host Linux が接続するサブネットとは別のサブネットが割り当てられる。

仮想ブリッジ

Flannel により構築されたコンテナ通信用オーバレイ N/W に割り当てられたサブネットを分割したサブネットが割り当てられる。

仮想 NIC (flannel.1) とのパケット転送は Linux のパケットフォワーディングが使用される。

Flannel による N/W 構成例



Host Linux のサブネット、コンテナ通信用 N/W のサブネット、仮想ブリッジのサブネットは一般的に左記のように構成され、コンテナ用 仮想ペア NIC には 仮想ブリッジ のサブネットから IP アドレスの割り当てが行われる。

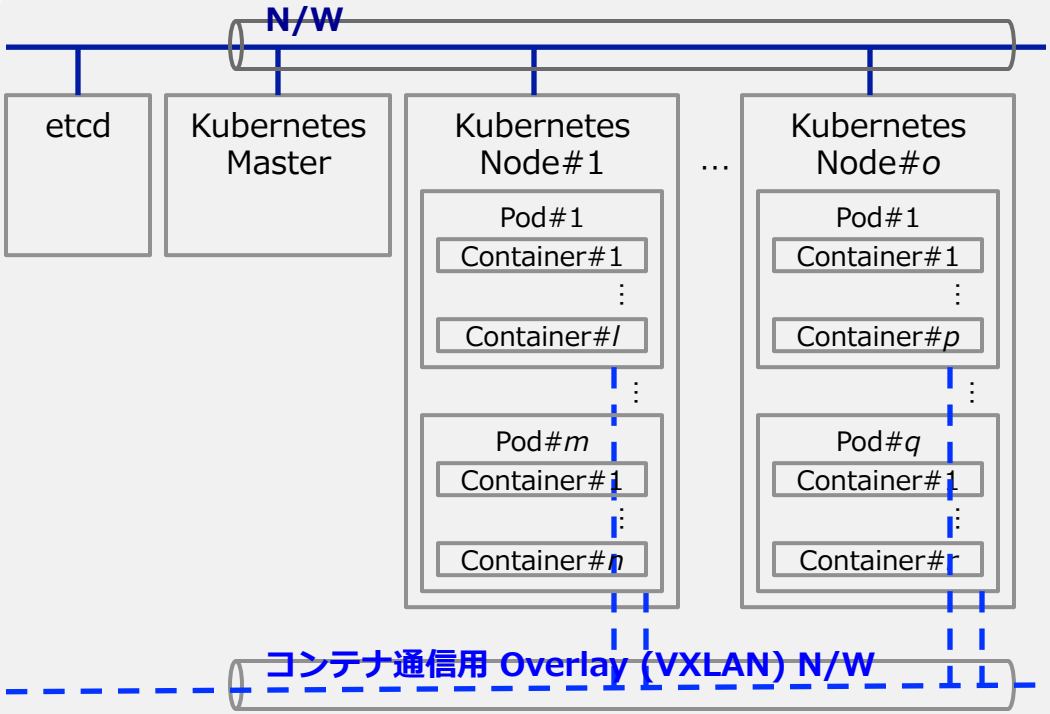
Kubernetes (※ 後述) を使用してコンテナの設定を行う場合、上記の手順は Flannel により自動的に行われる。

※ Kubernetes によるコンテナの起動時に 仮想ブリッジ の IP アドレス および サブネット が設定される。

Kubernetes の仕組み

1. アーキテクチャ概要
2. 主要コンポーネント
    1. Kubernetes Master
    2. Kubernetes Node
    3. Pod
    4. Service
    5. Proxy
3. モデルとコンフィギュレーション
    1. Replication Controller
    2. Service & Proxy
    3. Volume
    4. secret
    5. name & namespace
    6. label & selector
4. システム構成例

アーキテクチャ概要



Kubernetes Master

コンテナ管理、ノードのリソース使用状況の監視等を行う管理サーバ

Kubernetes Node

コンテナが稼働する Linux Host

Pod

複数のコンテナを束ねるコンポーネント

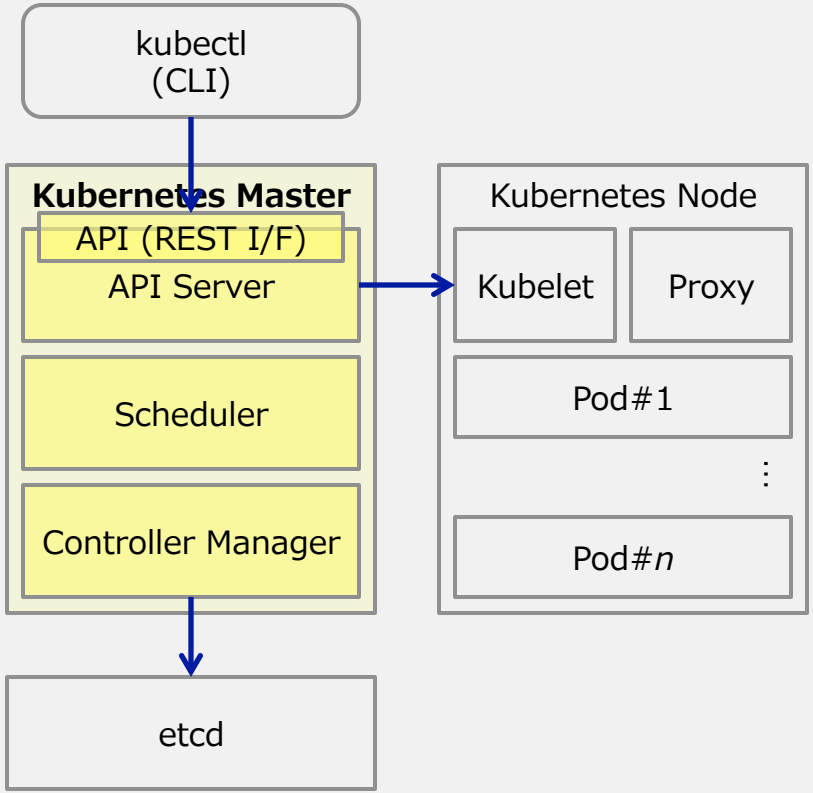
Container

コンテナ (rkt [4], docker[6] をランタイムエンジンとして使用)

etcd [5]

環境構成情報等を保持する KVS 型データ・ストレージ

Kubernetes Master



Kubernetes Master
- TODO

API Server
- TODO

API (REST I/F)
- TODO

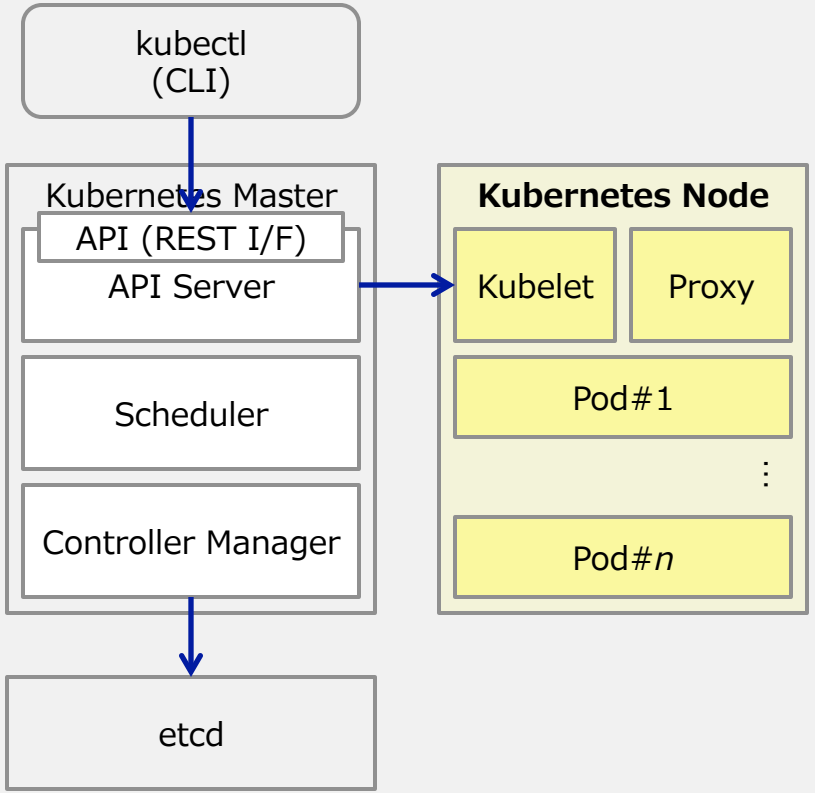
kubectl
- TODO

Scheduler
- TODO

Controller Manager
- TODO

etcd
- TODO

Kubernetes Node



Kubernetes Node

TODO

Kubelet

TODO

Proxy

TODO

Pod

TODO

Service

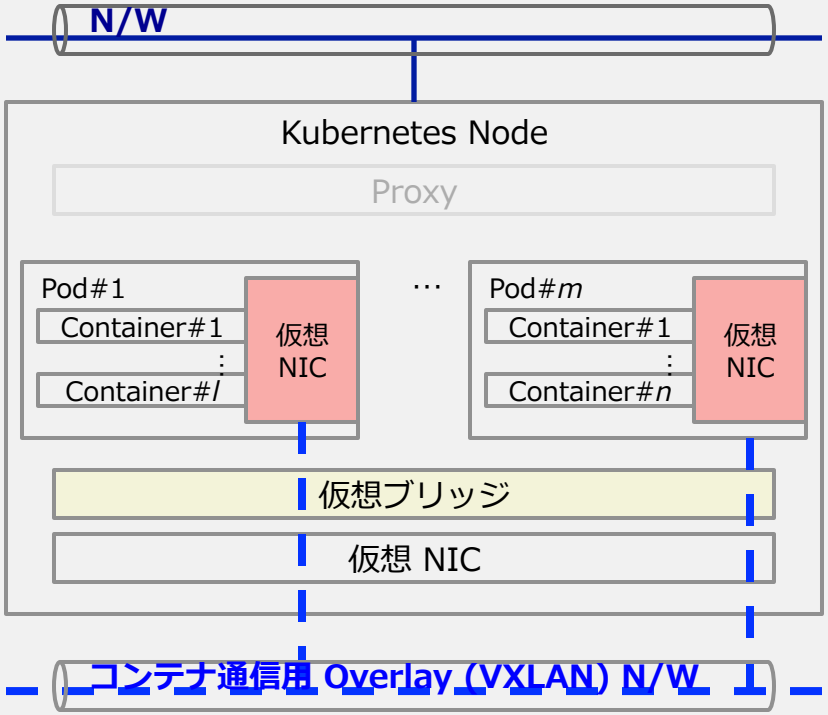
TODO1

https://docs.openshift.org/latest/architecture/core_concepts/pods_and_services.html#services

TODO2 http://kubernetes.io/docs/user-guide/services/

TODO

Pod

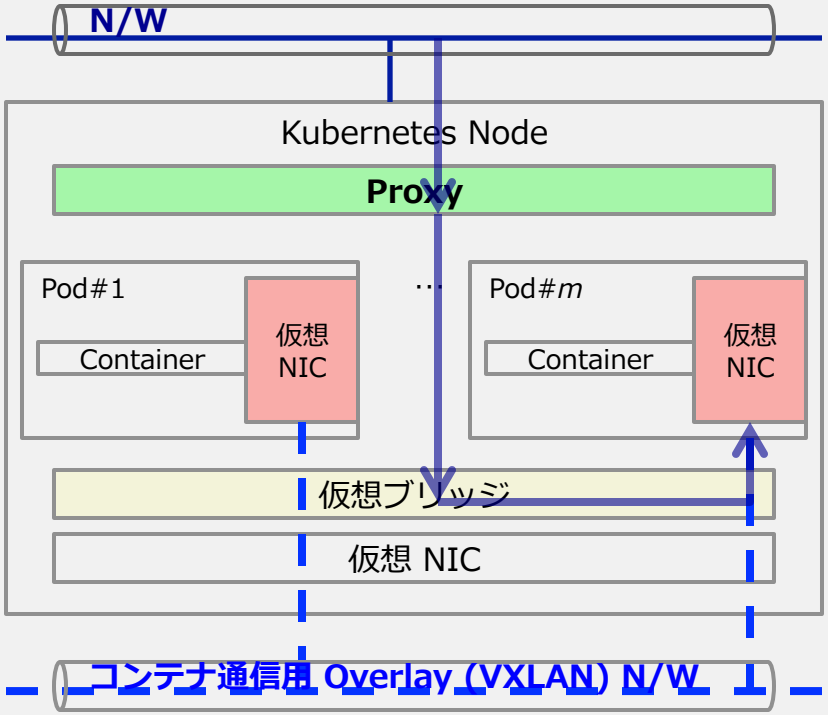


Kubernetes ではコンテナの管理を Pod の単位で行う。(コンテナを起動する場合には Pod を起動する。)

Pod 中で稼働するコンテナは 仮想 NIC を共有し、同一コンテナ中に含まれるコンテナ同士は localhost (127.0.0.1) を経由して通信することが可能。

単一のノードに含まれる複数の Pod は 仮想ブリッジ、コンテナ通信用 N/W に接続する仮想 NIC を共有する。

Proxy



各 Kubernetes Node で動作し、 外部 N/W から Service (※ 外部 N/W へ公開するコンテナ中で稼働するアプリケーション、詳細は後述) へのパケット (TCP, UDP) を 転送 (※1)・ロードバランス (※2) する。

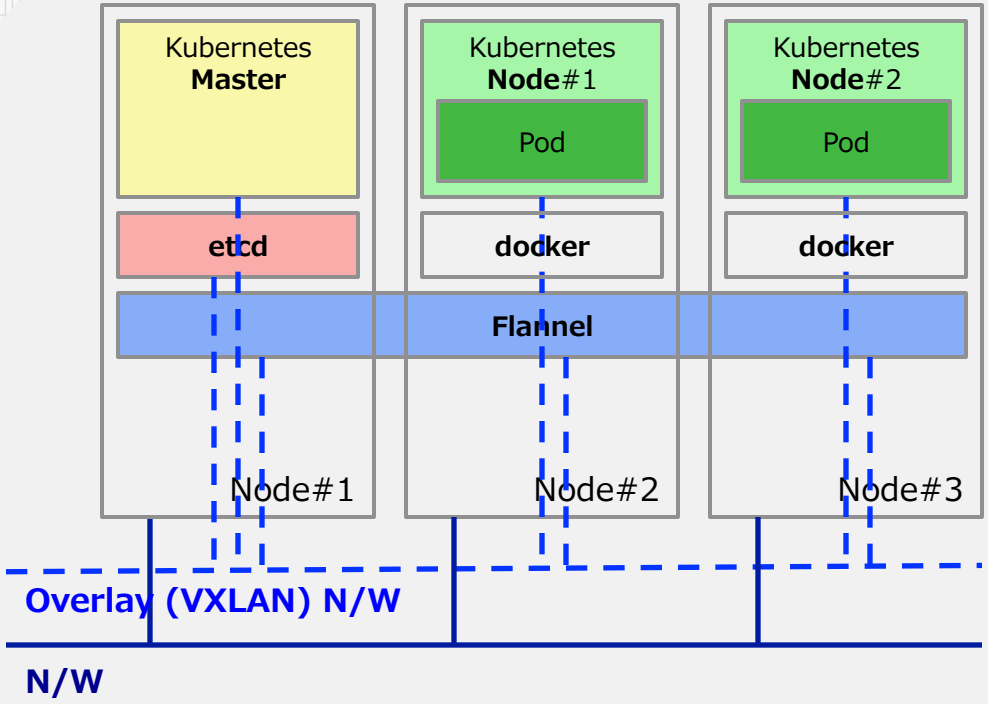
※1 デフォルトで iptables (NAT テーブルエントリ) を使用してパケット転送を行う。

※2 同一のコンテナイメージから起動した複数 Service へのラウンドロビンを行う。

Kubernetes クラスタの構築

1. クラスタの構成
2. H/W, N/W, S/W 環境
3. 環境準備
4. データストア (etcd) の構築
5. Docker の構築
6. Overlay N/W の構築
7. Docker の統合
8. Kubernetes Master の構築
9. Kubernetes Node の構築
10. サンプルコンテナ (Pod) の起動

クラスタの構成



Node#1

Kubernetes Master + etcd (Kubernetes データストア) が稼働する。

Flannel はオプショナルだが、Kubernetes UI を使用する場合は必要となる。

Node#2, Node#3

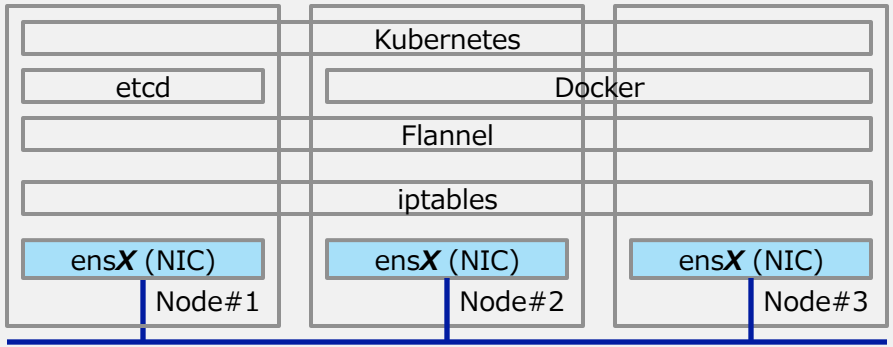
Kubernetes Node (+ docker) + Flannel が稼働する。

その他

- プライベートレジストリの構築は行わない。
- 追加 Node を用意し etcd を同ノードで稼働させる場合も手順は基本的に同じ。

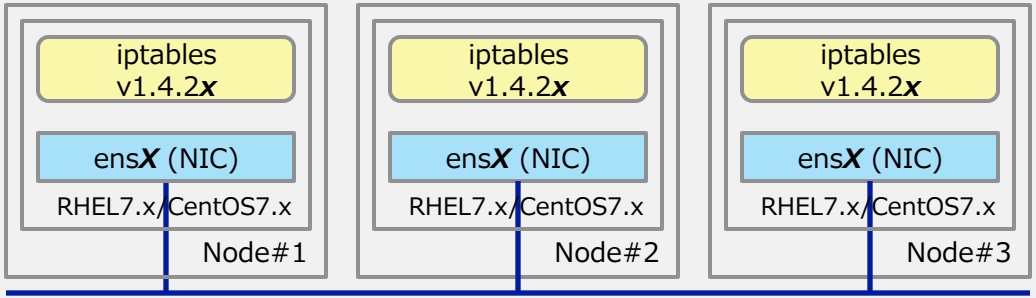
H/W, N/W, S/W 環境

		Node#1	Node#2	Node#3
H/W Spec	CPU	1 Core 以上		
	Memory	2 GB 以上		
OS (Host Linux)		• Red Hat Enterprise Linux 7 以降 ※ 下記 ※1, ※2 の yum リポジトリ設定が必要 • CentOS 7 以降		
S/W		• Kubernetes • Flannel (※ オプション) • etcd 2.3.7 以上 • iptables 1.4.21 以上	• Kubernetes • Flannel • Docker • iptables 1.4.21 以上	



※ 1 subscription-manager repos --enable=rhel-7-server-extras-rpms
※ 2 subscription-manager repos --enable=rhel-7-server-optional-rpms

環境準備



1. パッケージのアップデート

```
$ sudo yum update -y
```

2. iptables の確認

```
$ iptables –version
iptables v1.4.21
```

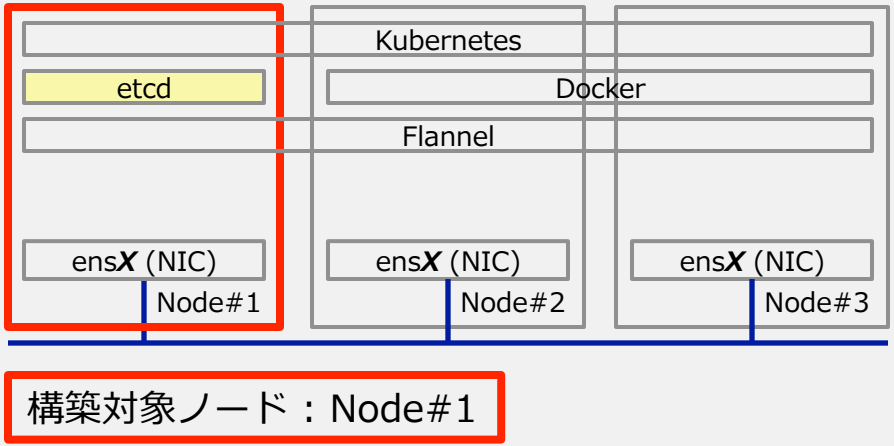
3. ネットワークインターフェースの確認

```
$ ip addr show up && nmcli device show
```

4. 1. ～ 3. の手順について node#1 ～ node#3 で実施。

データストア (etcd) の構築概要

1. etcd の インストール と 動作確認
2. 自動起動設定
3. コンフィギュレーション



etcd の インストール と 動作確認

1. etcd のインストール

```
$ sudo yum install -y etcd
```

2. インストールの確認

```
$ etcd –version
etcd Version: 2.3.7
Git SHA: fd17c91
Go Version: go1.6.2
Go OS/Arch: linux/amd64
```

3. etcd の動作確認
   1. etcd の起動

```
$ etcd --name sample-etcd --data-dir /tmp/sample.etcd &
```

   2. etcd の動作確認 (プロセス、ポート番号) ※ etcd プロセスが TCP 2379 ポートを listen していることを確認

```
$ ss – natup | grep etcd
```

   3. etcdctl による etcd の動作確認

```
$ etcdctl set key0 value0 && etcdctl get key0
value0
value0
```

   4. curl による REST API 経由での etcd の動作確認

```
$ curl -L http://localhost:2379/v2/keys/key0
{"action":"get","node":{"key":"/key0","value":"value0","modifiedIndex":4,"createdIndex":4}}
```

   5. etcd プロセスの停止

```
$ ps -e | grep etcd | awk '{print $1}' | xargs kill
```

etcd の 自動起動設定

1. Unit 設定ファイル (systemd) の作成 – 以下の URL からファイルを取得

   https://github.com/yyamada-redhat/training_kubernetes/blob/master/env/etcd.service

2. Unit 設定ファイルの配置

```
$ sudo cp etcd.service /usr/lib/systemd/system/
```

3. 自動起動の有効化

```
$ sudo systemctl enable etcd
```

4. etcd の起動

```
$ sudo systemctl start etcd.service
```

5. etcd プロセスの確認

```
$ systemctl status etcd.service
```

6. etcd プロセスの確認 ※ etcd プロセスが TCP 2379 ポートを listen していることを確認

```
$ sudo ss -nutap | grep etcd
```

etcd の コンフィギュレーション

1. 以下の etcd 設定ファイルを確認

```
/etc/etcd/etcd.conf
```

2. 以下の項目を設定

設定項目	意味	設定値	備考
ETCD_NAME	インスタンス名	default (デフォルト)	
ETCD_DATA_DIR	データディレクトリ	“/var/lib/etcd/default.etcd” (デフォルト)	
ETCD_LISTEN_CLIENT_URLS	リスン URL	http://0.0.0.0:2379	変更必須
ETCD_ADVERTISE_CLIENT_URLS	Advertise URL	http://0.0.0.0:2379	変更必須

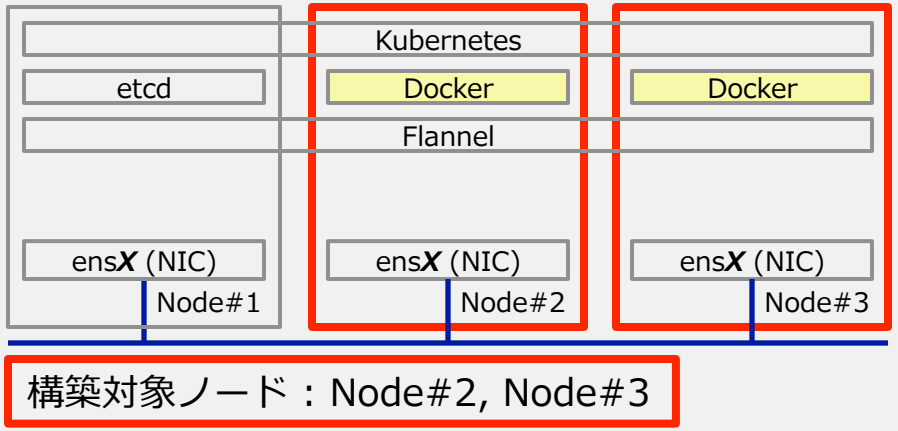
3. 設定変更後 etcd を再起動

```
$ systemctl restart etcd.service
```

※ 本番環境で etcd を使用する場合、上記以外に セキュリティ、冗長構成 の設定が必要

Docker の構築概要

1. インストールと動作確認



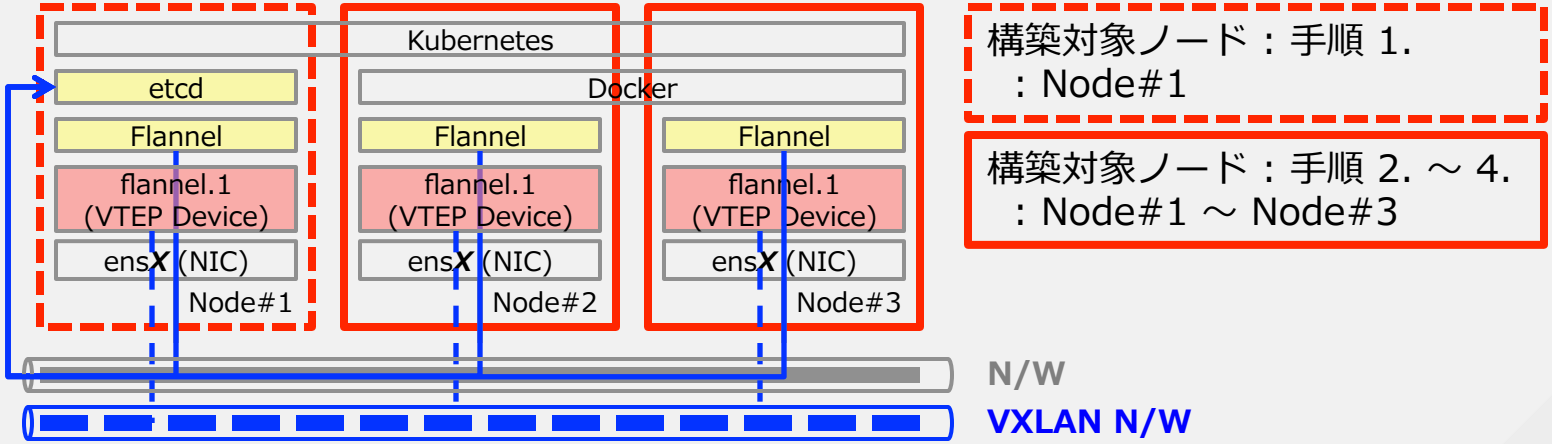
Docker の インストール と 動作確認

1. docker のインストール
```
$ sudo yum install -y docker
```
2. インストールの確認
```
$ docker --version
Docker version 1.10.3, build 79ebcd8-unsupported
```
3. 自動起動の有効化
```
$ sudo systemctl enable docker.service && systemctl is-enabled docker.service
```
4. docker の起動
```
$ sudo systemctl start docker.service && systemctl status docker.service
```
5. 仮想 Bridge (docker0) の起動確認
```
$ ip link show type bridge
```
6. docker の動作確認
```
$ docker run hello-world
```
7. docker の停止
```
$ sudo systemctl stop docker.service && systemctl status docker.service
```

※ 上記 1. ～ 6. の手順を Node#2, Node#3 で実施

Overlay N/W の構築概要

1. Flannel コンフィギュレーション の etcd への登録
2. Flannel のインストールとコンフィギュレーション
3. 自動起動設定
4. N/W 構成の確認



Flannel コンフィギュレーションの etcd への登録

1. docker 仮想ブリッジ (docker0) の削除

```
$ sudo ip link delete docker0
```

2. Flannel コンフィギュレーション (.json) の取得 – 以下の URL からファイルを取得

```
https://github.com/yyamada-redhat/training_kubernetes/blob/master/env/flannel-config-vxlan.json
```

3. コンフィギュレーションの確認

```
$ cat flannel-conig-vxlan.json
{
 "Network": "10.1.0.0/16",
 "SubnetLen": 24,
 "Backend": {
  "Type": "vxlan",
  "VNI": 1
 }
}
```

設定項目	意味	設定値
Network	Flannel が Overlay N/W に使用する IPv4 ネットワーク	10.1.0.0
SubnetLen	各 Linux Host に割り当てられるサブネットマスクの長さ	24
Backend:Type	パケット転送方式　(udp, vxlan, etc.)	vxlan
Baclemd:VNI	VXLAN Network Identifier：24 ビッド VXLAN ID	1 (デフォルト)

4. Flannel コンフィギュレーションの etcd への登録

```
$ etcdctl set /atomic.io/network/config < flannel-config-vxlan.json
```

5. etcd 上の Flannel コンフィギュレーション の確認

```
$ etcdctl get atomic.io/network/config
{
 "Network": "10.1.0.0/16",
 "SubnetLen": 24,
 "Backend": {
  "Type": "vxlan",
  "VNI": 1
 }
}
```

Flannel のインストールとコンフィギュレーション

1. flannel のインストール

```
$ sudo yum install –y flannel
```

2. 以下の flannel 設定ファイルを確認

```
/etc/sysconfig/flanneld
```

3. 以下の項目を設定

設定項目	意味	設定値
FLANNEL_ETCD_ENDPOINTS	etcd URL	(例) =“http://192.168.140.144:2379”
FLANNEL_ETCD_PREFIX	etcd 上のコンフィギュレーション key	(例) =“/atomic.io/network” **※ /atomic.io/network/config ではないことに注意**
FLANNEL_OPTIONS	オプション	

4. 以下のファイルを確認

```
/usr/lib/systemd/system/flanneld.service
```

```
…
ExecStart=/usr/bin/flanneld -etcd-endpoints=${FLANNEL_ETCD_ENDPOINTS} -etcd-prefix=${FLANNEL_ETCD_PREFIX} $FLANNEL_OPTIONS
…
```

/usr/lib/systemd/system/flanneld.service

flannel の 自動起動設定

1. 自動起動の有効化
```
$ sudo systemctl enable flanneld
```
2. flanneld の起動
```
$ sudo systemctl start flanneld.service
```
3. N/W インターフェースの確認
```
$ ip addr show up
```

N/W 構成の確認

1. flannel が構成する仮想ブリッジ (flannel.1) の確認

```
$ ip link show flannel.1
```

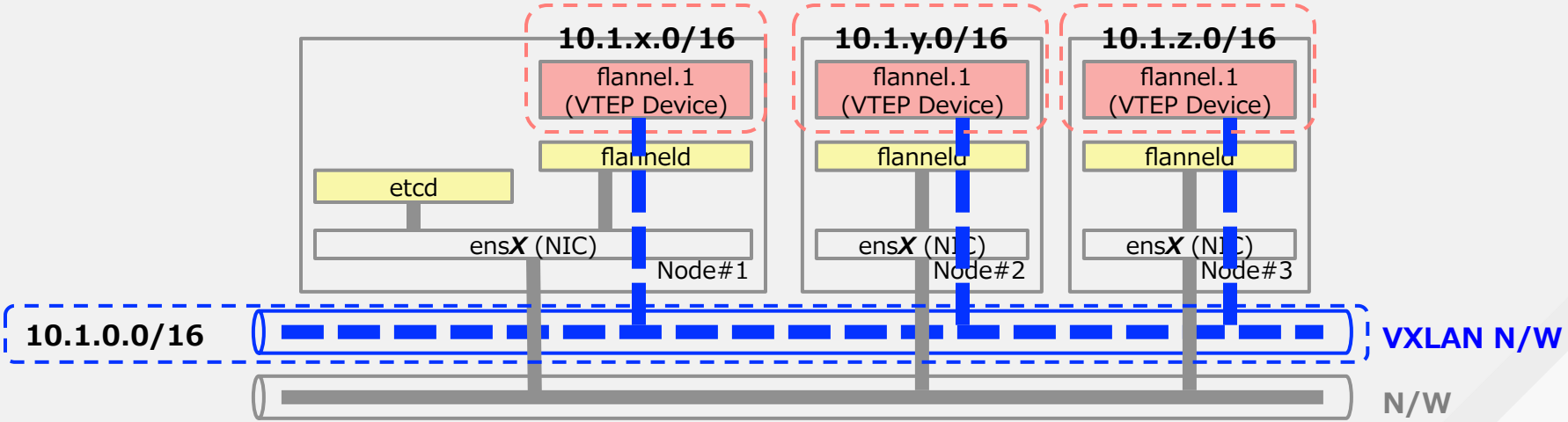
2. flannel の サブネット環境設定ファイル の確認

```
/run/flannel/subnet.env
```

3. etcd コンフィギュレーションの確認

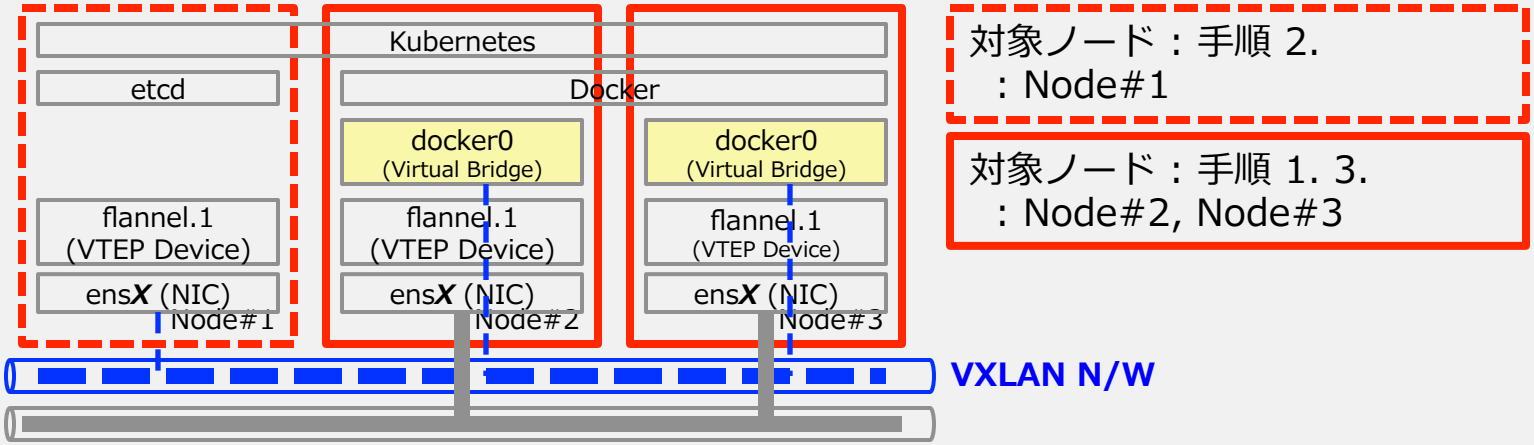
```
$ etcdctl get /atomic.io/network/config
```

4. N/W 構成が下図の通りになっていることを確認



Docker の統合概要

1. コンテナ仮想 bridge (docker0) の設定
2. etcd 上の flannel ランタイムデータの確認
3. サンプルコンテナの起動と N/W の確認



コンテナ仮想 bridge (docker0) の設定

1. 以下のファイルを確認

/run/flannel/subnet.env

```
FLANNEL_SUBNET=10.1.16.1/24
FLANNEL_MTU=1450
FLANNEL_IPMASQ=false
```

/etc/sysconfig/docker

2. 以下の flanneld Unit 設定ファイルの確認

/usr/lib/systemd/system/flanneld.service

```
…
[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/flanneld
EnvironmentFile=-/etc/sysconfig/docker-network
ExecStart=/usr/bin/flanneld -etcd-endpoints=${FLANNEL_ETCD_ENDPOINTS} -etcd-prefix=${FLANNEL_ETCD_PREFIX}
$FLANNEL_OPTIONS
ExecStartPost=/usr/libexec/flannel/mk-docker-opts.sh -k DOCKER_NETWORK_OPTIONS –d /run/flannel/docker
…
```

/usr/lib/systemd/system/flanneld.service

DOCKER_NETWORK_OPTIONS の設定が行われいる場合は手順 6. に進む

3. 以下のファイルを確認

/run/flannel/subnet.env

```
FLANNEL_SUBNET=10.1.16.1/24
FLANNEL_MTU=1450
FLANNEL_IPMASQ=false
```

/etc/sysconfig/docker

4. 以下の docker 起動パメータファイルの確認

/etc/sysconfig/docker

```
OPTIONS='--selinux-enabled --log-driver=journald  -G dockerroot'
```

/etc/sysconfig/docker

5. OPTIONS (/etc/sysconfig/docker) に以下の docker0 (仮想 bridge) 設定項目を追加

設定項目	意味	設定値
--bip=“”	コンテナ仮想 bridge (docker0) の IP	${FLANNEL_SUBNET}
--mtu=o	コンテナ N/W (docker0, veth) の MTU (Maximum Transmission Unit)	${FLANNEL_MTU}
--ip-masq=true	(オプション) IP マスカレードの有効/無効	${FLANNEL_IPMASQ}

```
OPTIONS='--selinux-enabled --log-driver=journald  -G dockerroot --bip=${FLANNEL_SUBNET} --mtu=${FLANNEL_MTU} --ip-masq=${FLANNEL_IPMASQ}'
```

/etc/sysconfig/docker

6. docker の再起動

```
$ sudo systemctl restart docker.service && systemctl status docker.service
```

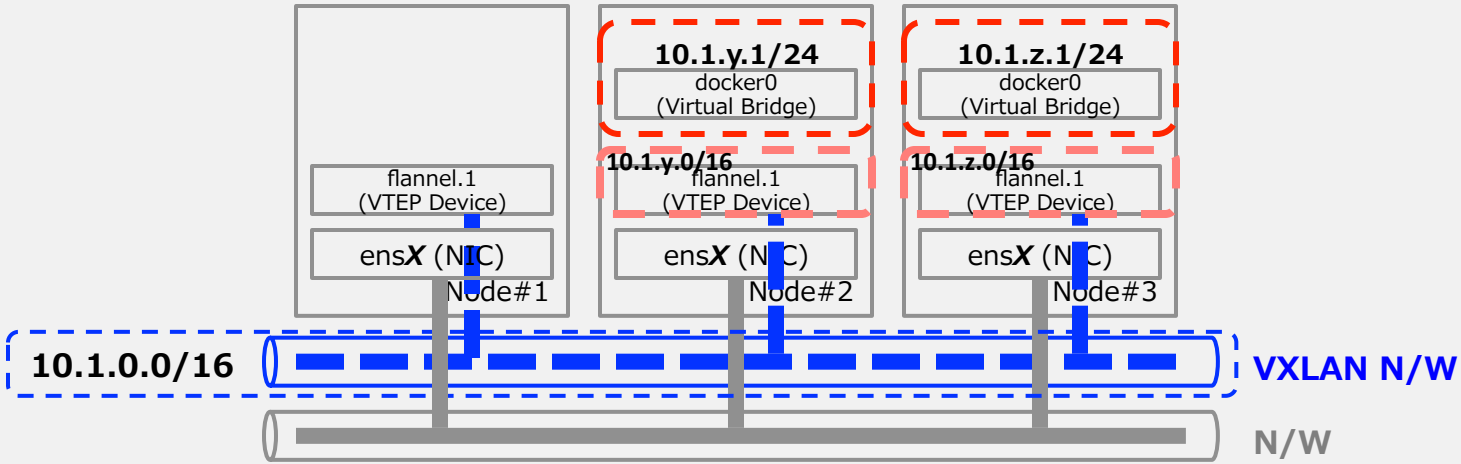
7. docker サービスの起動確認

```
$ systemctl status docker.service
```

8. docker0 (仮想ブリッジ)、flannel.1 (VXLAN デバイス) の状態確認

```
$ systemctl status ip addr show up type bridge && ip addr show up type vxlan && ip -4 addr show up | grep inet
```

下図のような N/W インターフェース、サブネット の構成になっていることを確認

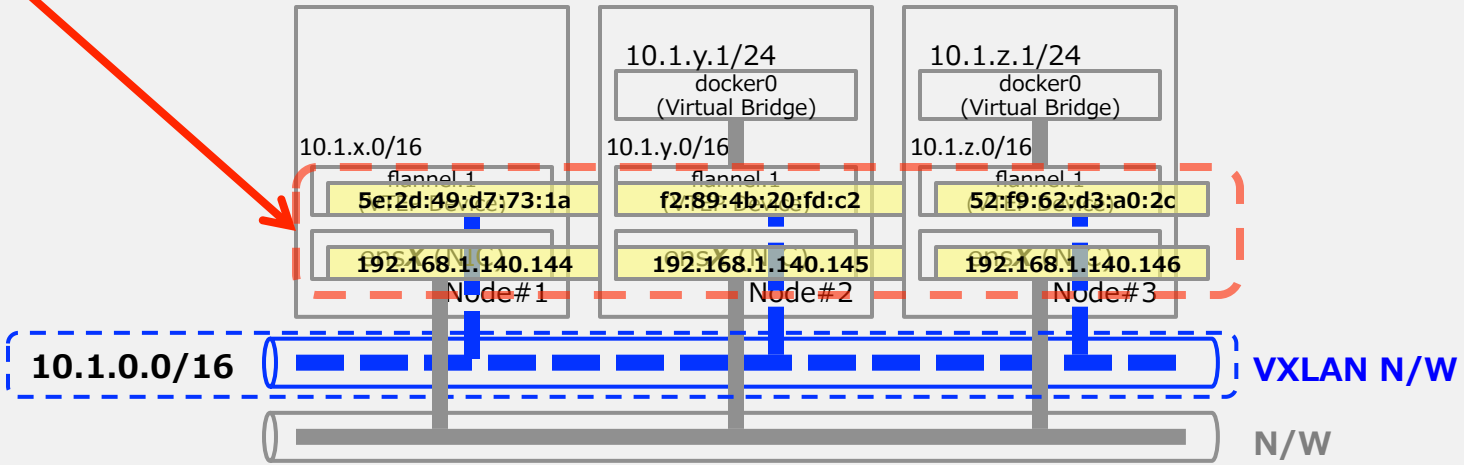


etcd 上の flannel ランタイムデータの確認

1. 各 Node 毎の VTEP MAC を確認

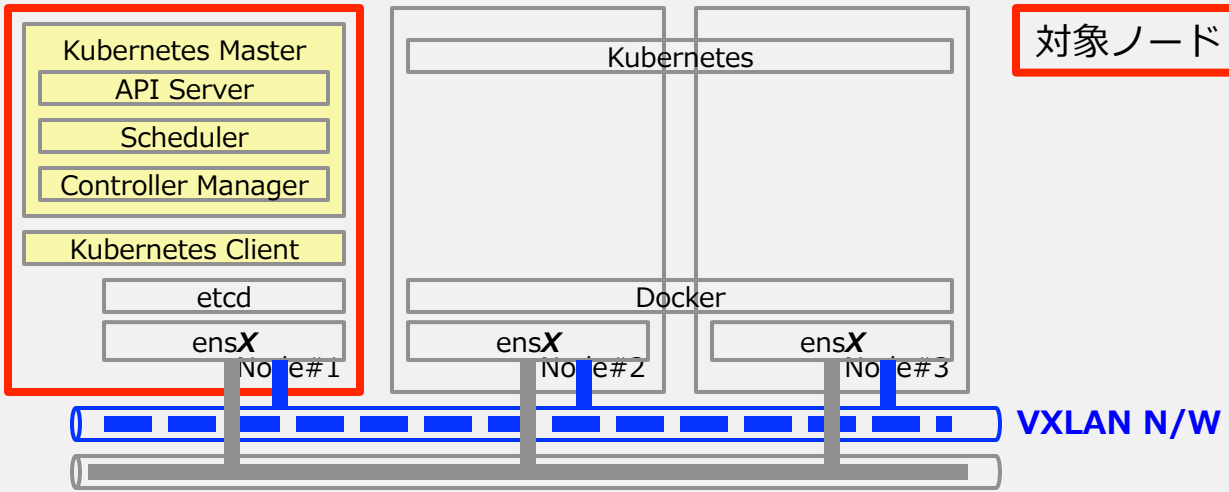
```
$ etcdctl ls /atomic.io/network/subnets | while read line; do etcdctl get $line; done

{"PublicIP":"192.168.140.145","BackendType":"vxlan","BackendData":{"VtepMAC":"f2:89:4b:20:fd:c2"}}
{"PublicIP":"192.168.140.144","BackendType":"vxlan","BackendData":{"VtepMAC":"5e:2d:49:d7:73:1a"}}
{"PublicIP":"192.168.140.146","BackendType":"vxlan","BackendData":{"VtepMAC":"52:f9:62:d3:a0:2c"}}
```

下図のような N/W インターフェース、サブネット の構成になっていることを確認



Kubernetes Master の構築

1. インストール
2. API Server の設定
3. サービスの起動
4. 動作確認



Kubernetes Master 関連パッケージのインストール

1. Kubernete Master 関連パッケージのインストール

```
$ sudo yum install -y kubernetes-master kubernetes-client
```

kubernetes-master：Kubernetes Master Daemon

Kubernetes-client：Kubernetes CLI (kubectl)

2. インストールファイルの存在確認
    1. Kubernetes Master (関連サービス) の環境設定ファイルの確認

```
$ ls /etc/kubernetes
apiserver  config  controller-manager  scheduler
```

    2. Kubernetes Master (関連サービス) の (Systemd) Uint 設定ファイルの確認

```
$ ls /usr/lib/systemd/system/kube-*
/usr/lib/systemd/system/kube-apiserver.service  /usr/lib/systemd/system/kube-controller-manager.service  /usr/lib/
systemd/system/kube-scheduler.service
```

API Server の設定

1. 以下の flannel 設定ファイルを確認

   /etc/kubernetes/apiserver

2. 以下の項目を設定

設定項目	意味	設定値
KUBE_API_ADDRESS	API Server のリスンアドレス	"--insecure-bind-address=0.0.0.0”
KUBE_API_PORT	API Server のリスンポート	"--port=8080” (デフォルト)
KUBELET_PORT	kublet (Kubernetes Node で稼働) のリスンポート	"--kubelet-port=10250” (デフォルト)
KUBE_ETCD_SERVERS	etcd (クラスタ) サーバ の アドレス:ポート	"--etcd-servers=http://127.0.0.1:2379” (デフォルト)
KUBE_SERVICE_ADDRESSES	Service のアドレスレンジ	"--service-cluster-ip-range=**10.1.0.0/16**” ※ 環境毎に Linux Host に割り当てられるサブネットを設定

```
KUBE_API_ADDRESS="--insecure-bind-address=0.0.0.0"
KUBE_API_PORT="--port=8080"
KUBELET_PORT="--kubelet-port=10250"
KUBE_ETCD_SERVERS="--etcd-servers=http://127.0.0.1:2379"
KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.1.0.0/16”
```

/etc/kubernetes/apiserver

Kubernetes Master 関連サービスの起動

1. Kubernete Master 関連サービスの起動

```
$ sudo systemctl start kube-apiserver
$ sudo systemctl start kube-scheduler
$ sudo systemctl start kube-controller-manager
```

2. 自動起動の有効化

```
$ sudo systemctl enable kube-apiserver
$ sudo systemctl enable kube-scheduler
$ sudo systemctl enable kube-controller-manager
```

動作確認

1. Kubernete Master 関連サービスのステータス確認

```
$ systemctl status kube-apiserver
$ systemctl status kube-scheduler
$ systemctl status kube-controller-manager
```

2. 詳細ログの確認

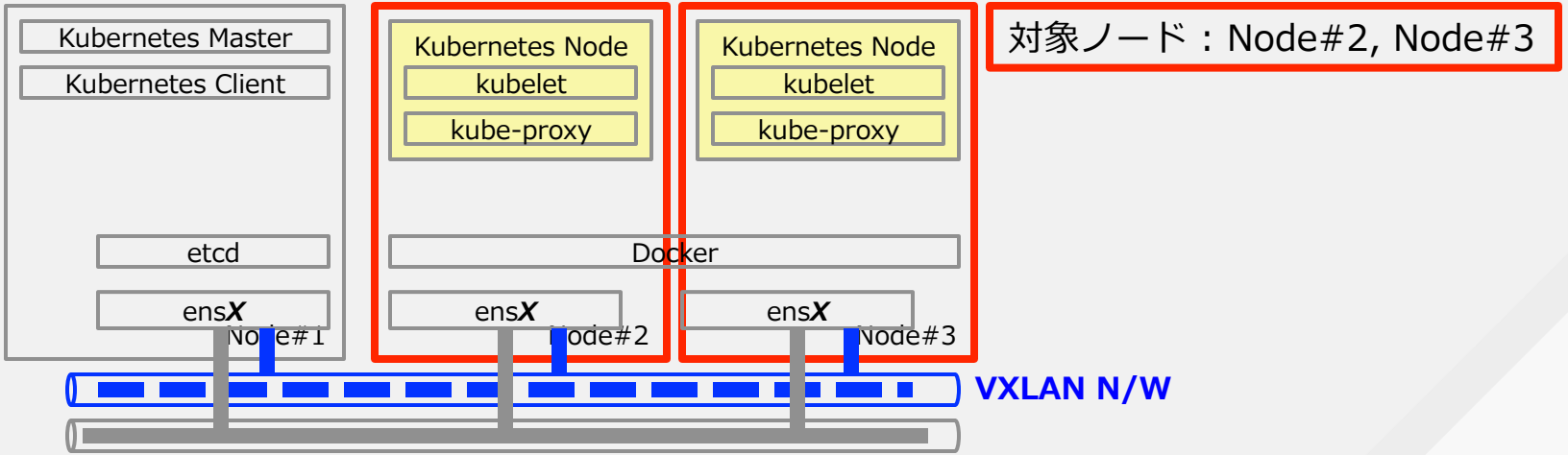
```
$ journalctl -u kube-apiserver --no-pager --full
$ journalctl -u kube-scheduler --no-pager --full
$ journalctl -u kube-controller-manager --no-pager --full
```

3. Kubernetes CLI (kubectl) の動作確認

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"3", GitVersion:"v1.3.0",
GitCommit:"c5ee292ce8f0df382b52096902778274c7eab945", GitTreeState:"clean", BuildDate:"2016-09-07T16:26:49Z",
GoVersion:"go1.6.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"3", GitVersion:"v1.3.0",
GitCommit:"c5ee292ce8f0df382b52096902778274c7eab945", GitTreeState:"clean", BuildDate:"2016-09-07T16:26:49Z",
GoVersion:"go1.6.2", Compiler:"gc", Platform:"linux/amd64"}
```

Kubernetes Node の構築

1. インストール
2. kubelet の設定
3. kubelet の起動
4. 動作確認



Kubernetes Node パッケージのインストール

1. Kubernete Master 関連パッケージのインストール

```
$ sudo yum install -y kubernetes-node
```

kubernetes-master：Kubernetes Master Daemon

Kubernetes-client：Kubernetes CLI (kubectl)

2. インストールファイルの存在確認
    1. Kubernetes Master (関連サービス) の環境設定ファイルの確認

```
$ ls /etc/kubernetes
apiserver  config  controller-manager  scheduler
```

    2. Kubernetes Master (関連サービス) の (Systemd) Uint 設定ファイルの確認

```
$ ls /usr/lib/systemd/system/kube-*
/usr/lib/systemd/system/kube-apiserver.service  /usr/lib/systemd/system/kube-controller-manager.service  /usr/lib/
systemd/system/kube-scheduler.service
```

kubelet の設定

1. 以下の flannel 設定ファイルを確認

   /etc/kubernetes/kubelet

2. 以下の項目を設定

設定項目	意味	設定値
KUBELET_ADDRESS	kubelet のリスンアドレス	“—address=0.0.0.0"
KUBELET_PORT	kubelet のリスンポート	"--port=10250"
KUBELET_HOSTNAME	kublet のホスト名	"--hostname-override=node***X***"
KUBELET_API_SERVER	API Server のロケーション	"--api-servers=http://192.168.140.144:8080” ※ 環境毎に Kubernetes Master のロケーションを設定

```
KUBELET_ADDRESS="--address=0.0.0.0"
KUBELET_PORT="--port=10250"
KUBELET_HOSTNAME="--hostname-override=nodeX"
KUBELET_API_SERVER="--api-servers=http://192.168.140.144:8080”
```

/etc/kubernetes/config

kubelet の起動

1. Kubernete Master 関連サービスの起動

```
$ sudo systemctl start kubelet
```

2. 自動起動の有効化

```
$ sudo systemctl enable kubelet
```

動作確認

1. Kubernetes Node の確認

```
$ kubelet get nodes
```

Appendix

1. 一般ユーザ権限による docker の実行

一般ユーザ権限による docker の実行

1. /etc/sysconfig/docker の設定

```
OPTIONS='--selinux-enabled --log-driver=journald -G dockerroot'
```

2. dockerroot グループへの一般ユーザの追加

```
$ sudo usermod dockerroot $USER_NAME
```

3. docker (daemon) の再起動

```
$ sudo systemctl restart docker.service
```

4. 変更対象ユーザのシステムの login & logout
5. docker の動作確認

```
$ docker run hello-world
```

※ 上記の設定は開発環境のみで行うこと。

References

[1] [kubernetes] (http://kubernetes.io)

[2] [CoreOS] (https://coreos.com)

[3] [flannel] (https://github.com/coreos/flannel/)

[4] [rkt] (https://coreos.com/rkt/)

[5] [etcd] (https://coreos.com/etcd/)

[6] [docker] (https://www.docker.com)

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos