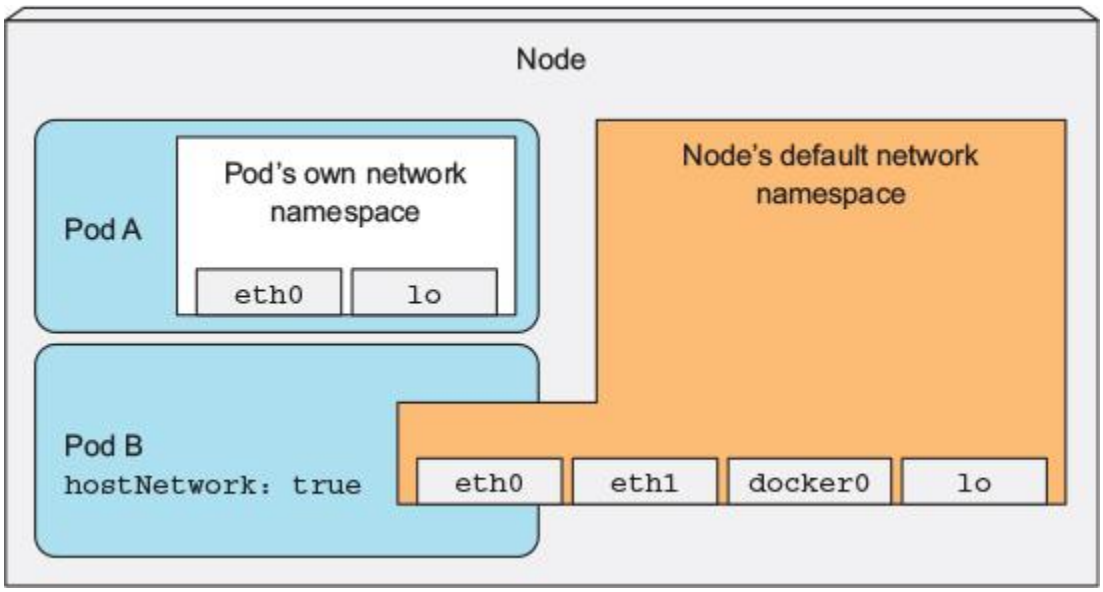
13. Security

노드의 리소스와 디바이스를 읽고 조작하기 위해 호스트 네임스페이스를 사용한다.

파드에서 호스트의 네트워크 네임스페이스 사용

예를 들어, 마스터 노드의 팟은 spec.hostNetwork: true 를 설정함으로 노드의 포트를 사용하는것이 가능하다.

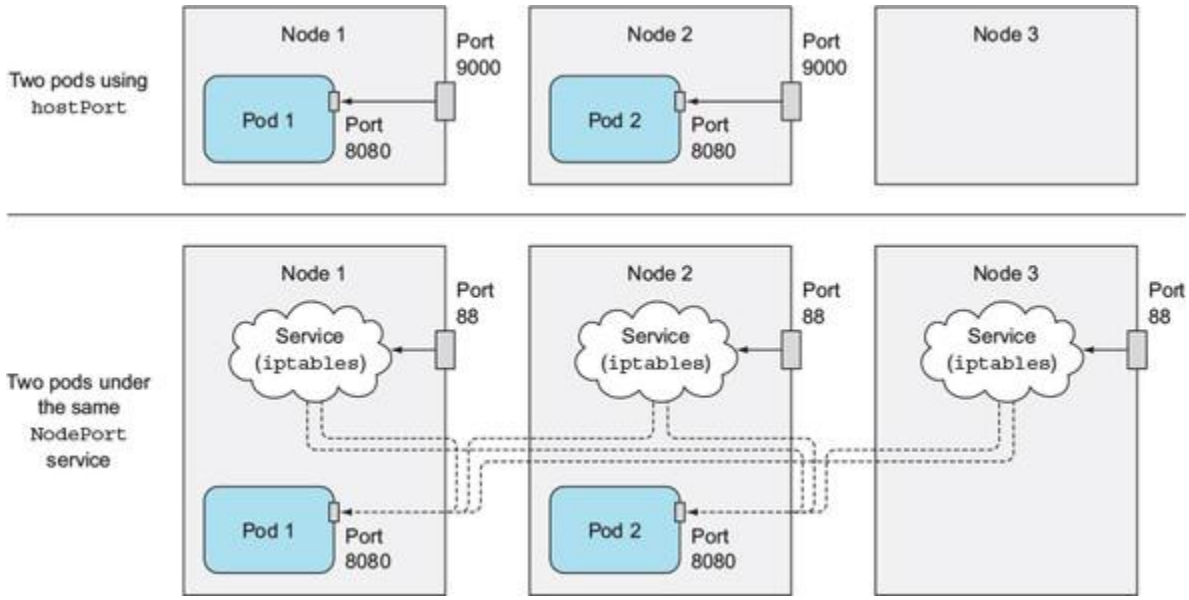


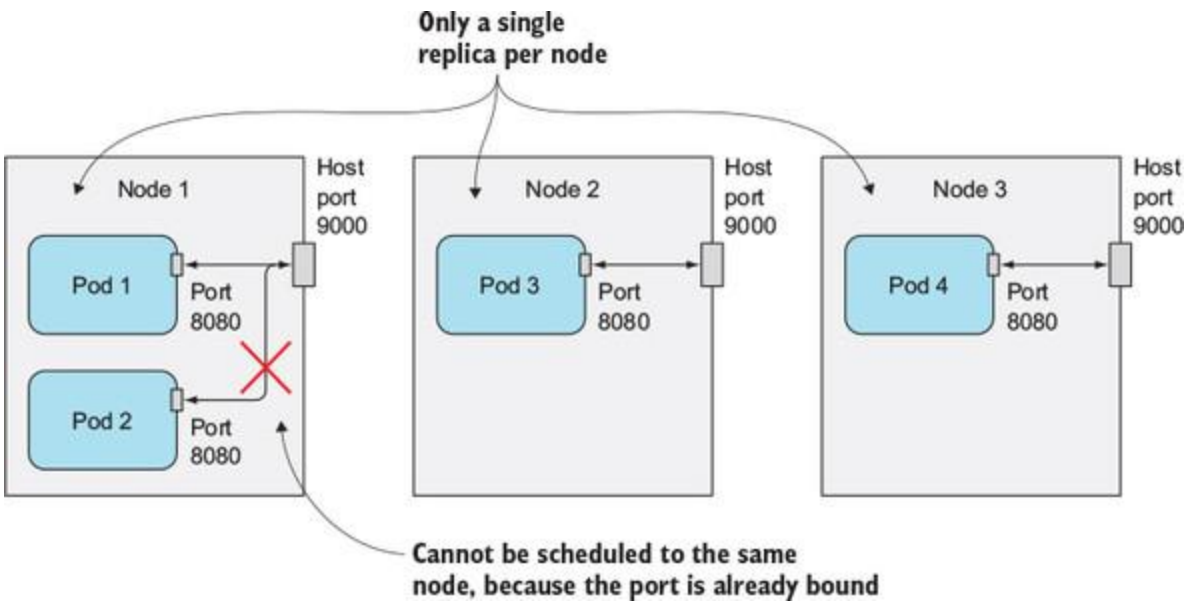

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-host-network
spec:
  hostNetwork: true
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
```

```
$ kubectl exec pod-with-host-network ifconfig
```

```
docker0   Link encap:Ethernet  HWaddr 02:42:14:08:23:47
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          ...

eth0      Link encap:Ethernet  HWaddr 08:00:27:F8:FA:4E
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          ...

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          ...

veth1178d4f Link encap:Ethernet  HWaddr 1E:03:8D:D6:E1:2C
          inet6 addr: fe80::1c03:8dff:fed6:e12c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

호스트 네임스페이스를 사용하지 않으면서 호스트 포트에 바인딩





If a host port is used, only a single pod instance can be scheduled to a node.

워커노드의 데몬셋을 사용해 모든 노드에 배포되는 시스템 서비스를 노출하는 데 사용된다.

파드에서 호스트의 PID, IPC 네임스페이스 사용

spec.hostNetwork 로 호스트 네임스페이스를 팟에서 사용한 것 처럼, spec.hostPID, spec.hostIPC 로 다른 네임스페이스도 사용할 수 있다.

```
$ kubectl exec pod-with-host-pid-and-ipc ps aux
PID   USER     TIME   COMMAND
    1 root       0:01 /usr/lib/systemd/systemd --switched-root
--system ...
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
    5 root       0:00 [kworker/0:0H]
    6 root       0:00 [kworker/u2:0]
    7 root       0:00 [migration/0]
    8 root       0:00 [rcu_bh]
    9 root       0:00 [rcu_sched]
   10 root       0:00 [watchdog/0]
```

SecurityContext (컨테이너/팟 레벨)

개별 컨테이너 또는 팟 레벨에 spec.(containers.)securityContext 를 지정할 수 있다.

security context 의 역할:

- AuthN
  - 프로세스의 user ID 지정
  - 루트로 실행되는 것 방지
  - fsGroup, supplementalGroups
    - fsGroup, supplementalGroups를 등록함으로, 하나의 팟에서 서로 다른 유저로 실행되는 컨테이너간에 마운트 된 볼륨을 사용
    - 각 팟의 effective group ID는 변함없이 0(root)이지만, group ID 에 fsGroup과 supplementalGroups에 지정된 그룹이 추가된다
    - 마운트되는 디렉터리와 각 컨테이너가 생성하는 파일은 fsGroup으로 지정된 group ID가 소유자로 지정된다.

```
/ $ echo foo > /volume/foo
/ $ ls -l /volume
total 4
-rw-r--r--    1 1111     555              4 May 29 12:25 foo
```

- AuthZ
  - 컨테이너의 privileged mode (특권모드) 활성화/비활성화
  - 특권모드보다 더욱 세분화된 권한을 capabilities 단위로 를 추가/삭제
    - 리눅스의 capabilities 이름은 CAP_ 접두어가 붙지만, 팟 스펙에는 이 접두어는 생략하고 지정한다 (예: SYS_TIME, CHOWN)
  - SELinux 옵션 설정
  - 컨테이너 (또는 컨테이너의 자식 프로세스가) 컨테이너의 파일 시스템에 쓰지 못하고 마운트된 볼륨에만 쓸 수 있도록 지정
    - PHP 파일 등은 빌드시 컨테이너 이미지에 추가되고 컨테이너의 파일 시스템에서 제공된다.

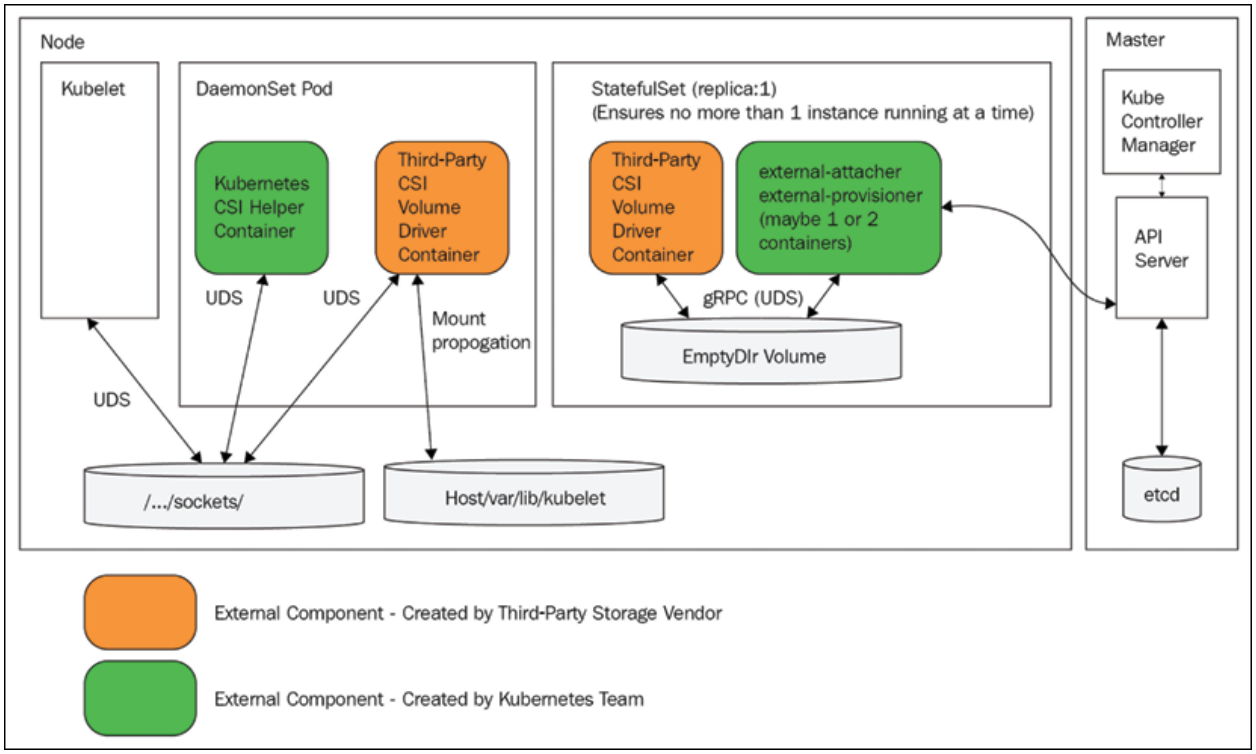
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-privileged
spec:
  containers:
  - name: main
    image: alpine
```

```
    command: ["/bin/sleep", "999999"]
    securityContext:
      privileged: true
```

privileged 모드에서 컨테이너가 실행되므로, 호스트의 /dev 디렉터리에 연결된 장치목록을 볼 수 있다

```
$ kubectl exec -it pod-privileged ls /dev
autofs              snd                 tty46
bsg                 sr0                 tty47
btrfs-control       stderr              tty48
core                stdin               tty49
cpu                 stdout              tty5
cpu_dma_latency     termination-log     tty50
fd                  tty                 tty51
full                tty0                tty52
fuse                tty1                tty53
hpet                tty10               tty54
hwrng               tty11               tty55
...                 ...                 ...
```

privileged 모드에서 컨테이너가 실행되므로, 호스트의 /dev 디렉터리에 연결된 장치목록을 볼 수 있다

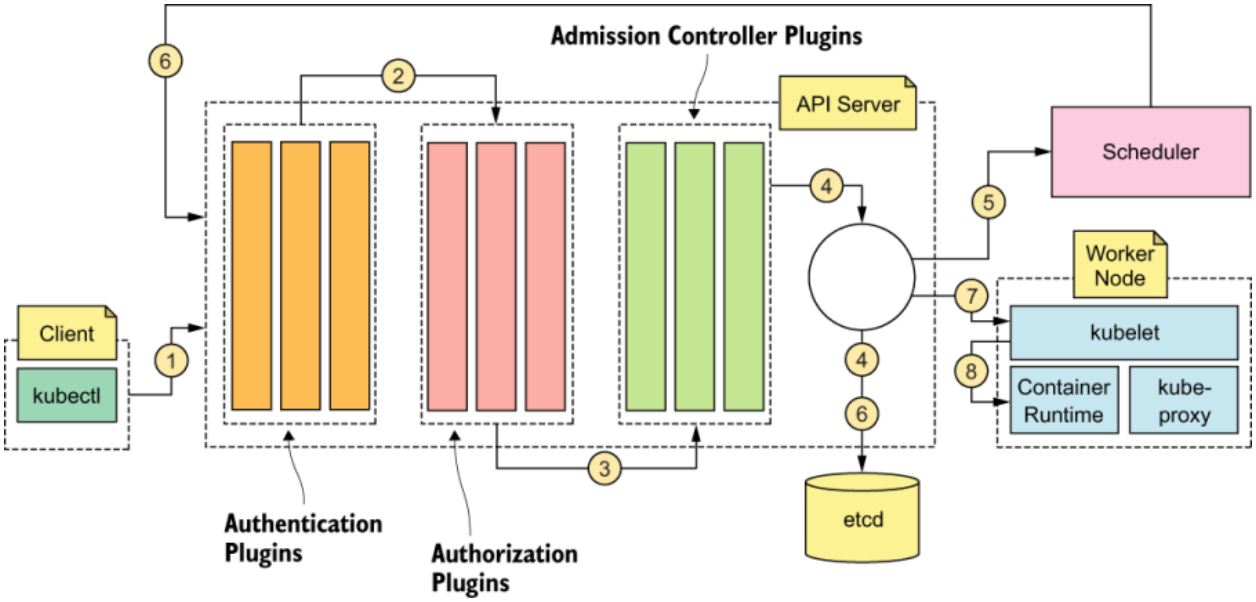


PodSecurityPolicy로 클러스터 레벨에서 팟의 AuthZ 제어

SecurityContext가 팟 또는 컨테이너 레벨에서 리소스의 AuthZ / AuthN을 제어하는 반면, PodSecurityPolicy는 클러스터 레벨의 리소스로서 이를 달성한다

예:

- 호스트 네임스페이스 사용 가능 여부
- 바인딩 가능한 호스트 포트
- 컨테이너각 실행할 수 있는 user ID
- privileged containers가 있는 팟을 만들수 있는지에 대한 여부
- capabilities 사용 여부
    - allowedCapabilities
    - defaultAddCapabilities
    - requiredDropCapabilities
- Whether a container can use a writable root filesystem or not
- Which filesystem groups the container can run as
- Which volume types a pod can use



PodSecurityPolicy 는 Admission Controller Plugin으로써 실행된다.

```yaml
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: default
spec:
  hostIPC: false                  ❶
  hostPID: false                  ❶
  hostNetwork: false              ❶
  hostPorts:                      ❷
  - min: 10000                    ❷
    max: 11000                    ❷
  - min: 13000                    ❷
    max: 14000                    ❷
  privileged: false               ❸
  readOnlyRootFilesystem: true    ❹
  runAsUser:                      ❺
    rule: RunAsAny                ❺
  fsGroup:                        ❺
    rule: RunAsAny                ❺
  supplementalGroups:             ❺
    rule: RunAsAny                ❺
  seLinux:                        ❻
    rule: RunAsAny                ❻
  volumes:                        ❼
```

```
  - '*'                                    ❼
```

PodSecurityPolicies는 클러스터 리소스이지만, ClusterRole에 attach 하는 것이 가능하다. 이렇게 attach된 role 에 identity 가 바인딩 되면, 해당 리소스를 기반으로 Admission controller plugin이 동작하게 된다.

아래 커맨드로 새로운 kubectl 유저를 만든다

```
$ kubectl config set-credentials alice --username=alice
--password=password
```

생성된 유저 credential 로 kubectl커맨드를 실행해 팟을 생성한다

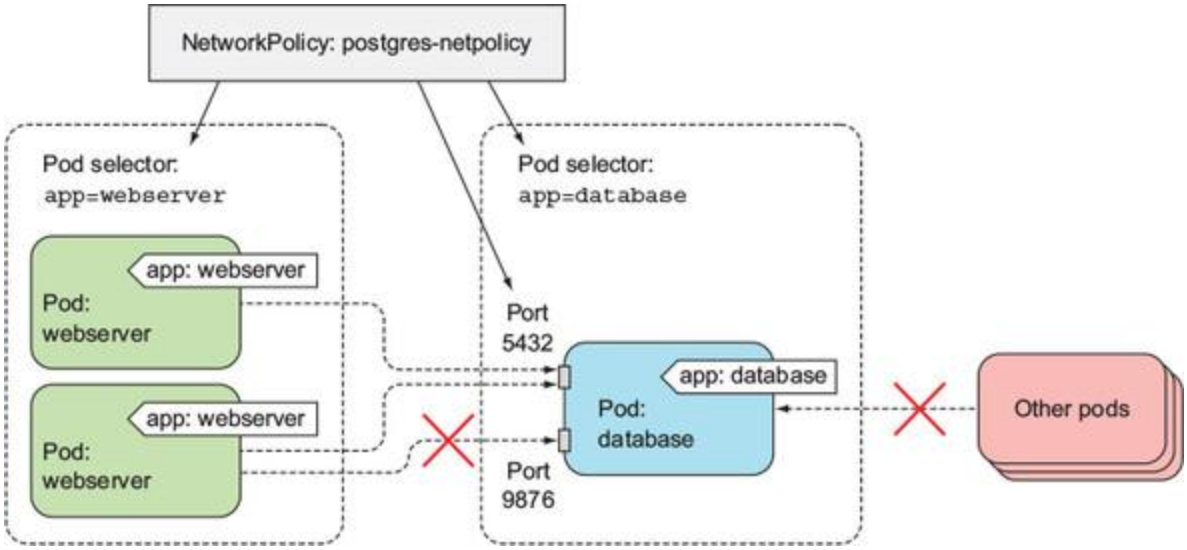
```
$ kubectl --user alice create -f pod-privileged.yaml
Error from server (Forbidden): error when creating
"pod-privileged.yaml":
     pods "pod-privileged" is forbidden: unable to validate against
any pod
     security policy: [spec.containers[0].securityContext.privileged:
Invalid
     value: true: Privileged containers are not allowed]
```

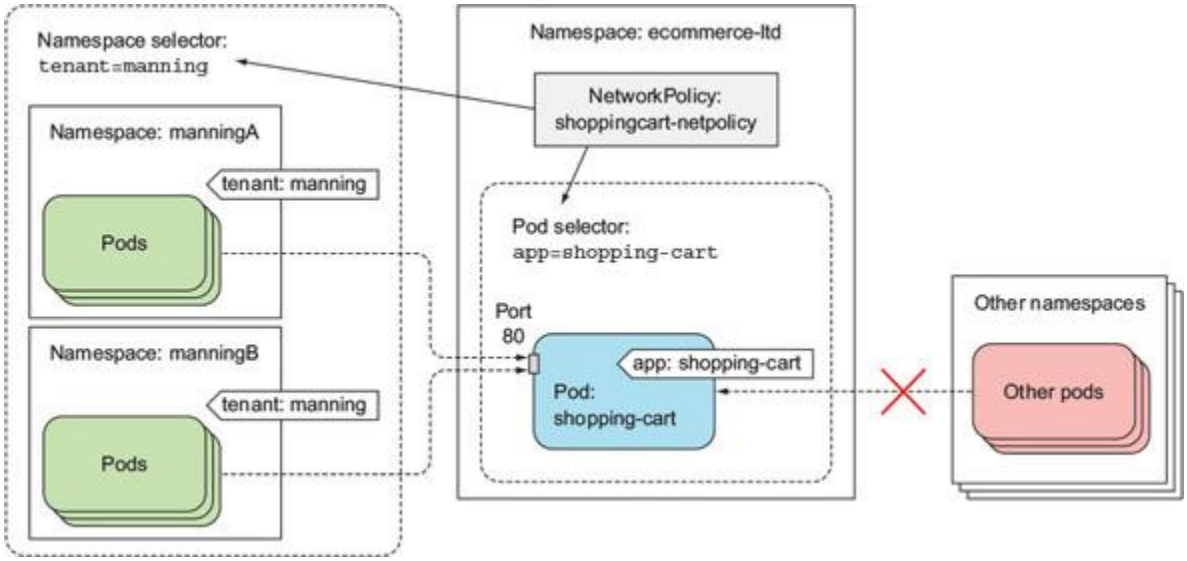
NetworkPolicy 로 inter Pod 네트워크 제어

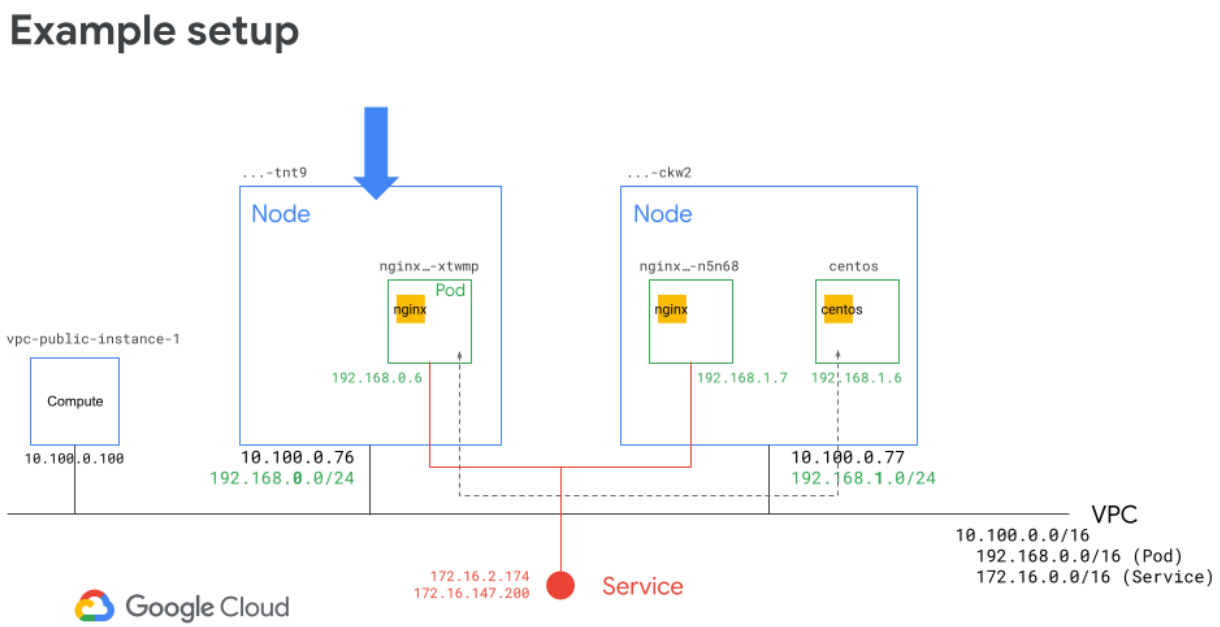
Container Network Plugin (CNI Plugin) 중에는, NetworkPolicy가 사용 가능한 것들이 있다. NetworkPolicy에는 팟간의 ingress 이나 egress whitelist 를 설정 할 수 있다.

설정 대상은 아래 세가지 이다:

- 팟 셀렉터
- 네임스페이스 셀렉터
- CIDR notation

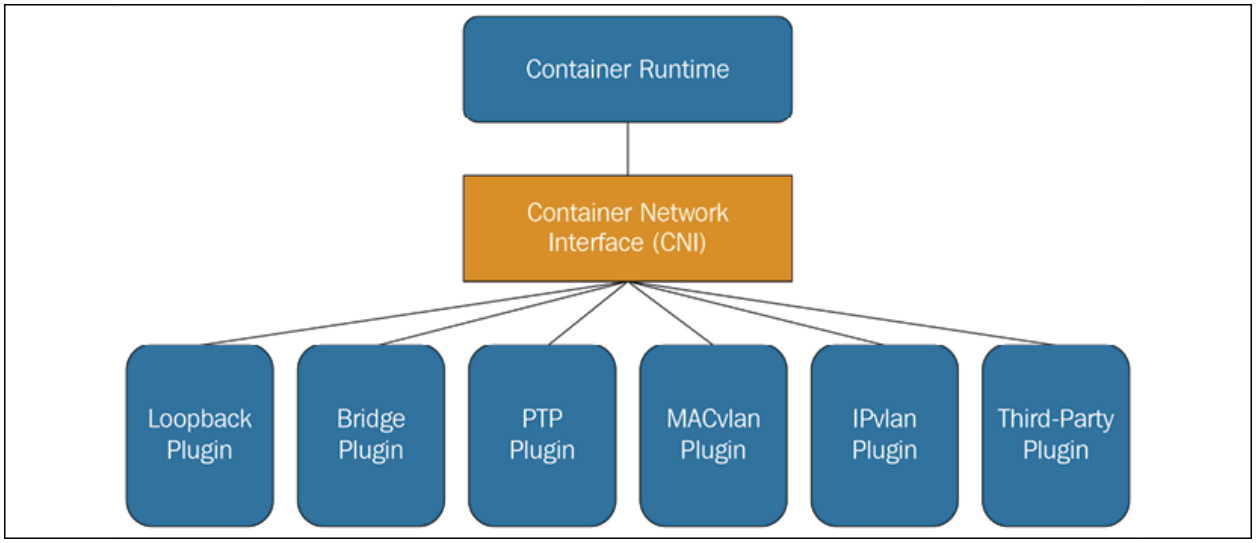

NetworkPolicy: postgres-netpolicy
Pod selector:
app=webserver
Pod selector:
app=database
app: webserver
Pod:
webserver
app: webserver
Pod:
webserver
Port
5432
app: database
Pod:
database
Port
9876
Other pods


Namespace selector:
tenant=manning
Namespace: ecommerce-ltd
NetworkPolicy:
shoppingcart-netpolicy
Namespace: manningA
tenant: manning
Pods
Pod selector:
app=shopping-cart
Port
80
Other namespaces
Namespace: manningB
app: shopping-cart
Pod:
shopping-cart
Other pods
tenant: manning
Pods

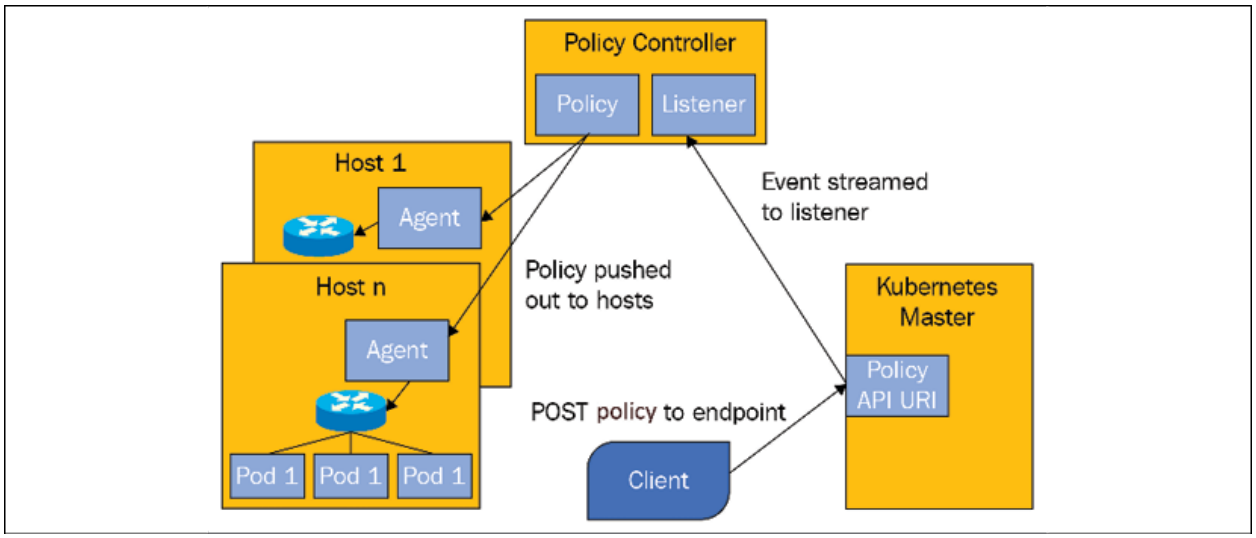


여담: CNI와 NetworkPolicy

- CNI 는 container runtime에 plug 되어야 한다.
- CNI 컨텍스트에서, 컨테이너랑 network-addressable entity이다
  - 예: docker 컨테이너
  - 팟도 컨테이너로 취급된다. rkt 컨테이너의 경우 컨테이너 안에 여러 컨테이너를 가지는 구성을 할 때가 있다는 점에서 팟과 유사하다.



- CNI plugin에도 네트워크 policy와 같은 기능이 있고, 일부 Calico & Flannel을 포함한 일부 plugin들은 다른 plugin과 병용해서도 사용할 수 있다.
    - 각 노드에는 agent 또는 gatekeeper가 존재해서,
        i. 노드로 오는 모든 트래픽을 intercept
        ii. network policy를 적용
        iii. 리퀘스트를 forward 또는 reject
    - 하는 역할을 한다
- 쿱에는 network policy를 정의하고 제공하는 API가 있어서, 실제 CNI 플러그인의 controller 와 연동되어 사용된다.



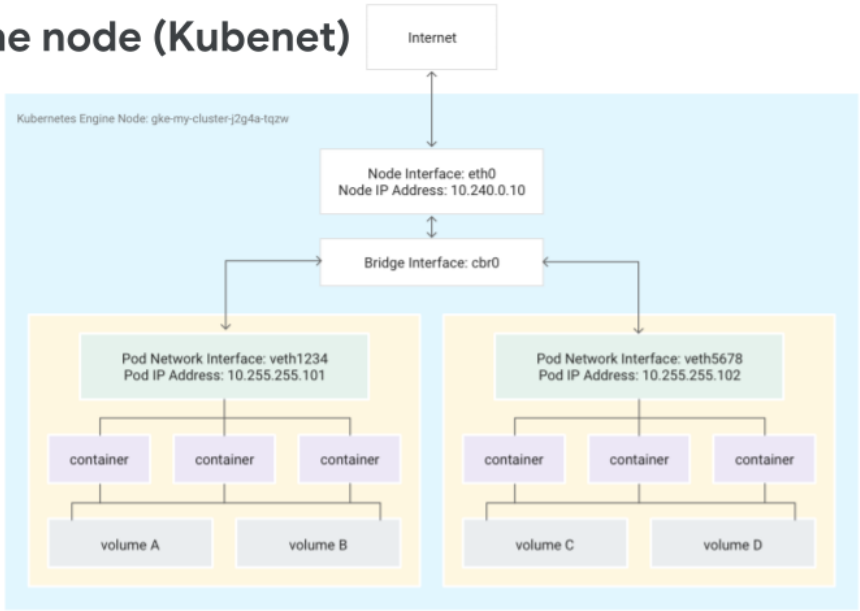
- CNI plugin의 참조구현 인 PTP CNI plugin : https://www.cni.dev/plugins/current/main/ptp/
- GKE의 CNI plugin 은 gke, calico 를 선택하는 것도 가능

Networking in the node (Kubenet)

It uses the same implementation for both routes-based and VPC native.

However, it differs when Network Policy is enabled, which uses Calico instead of default GKE's CNI.

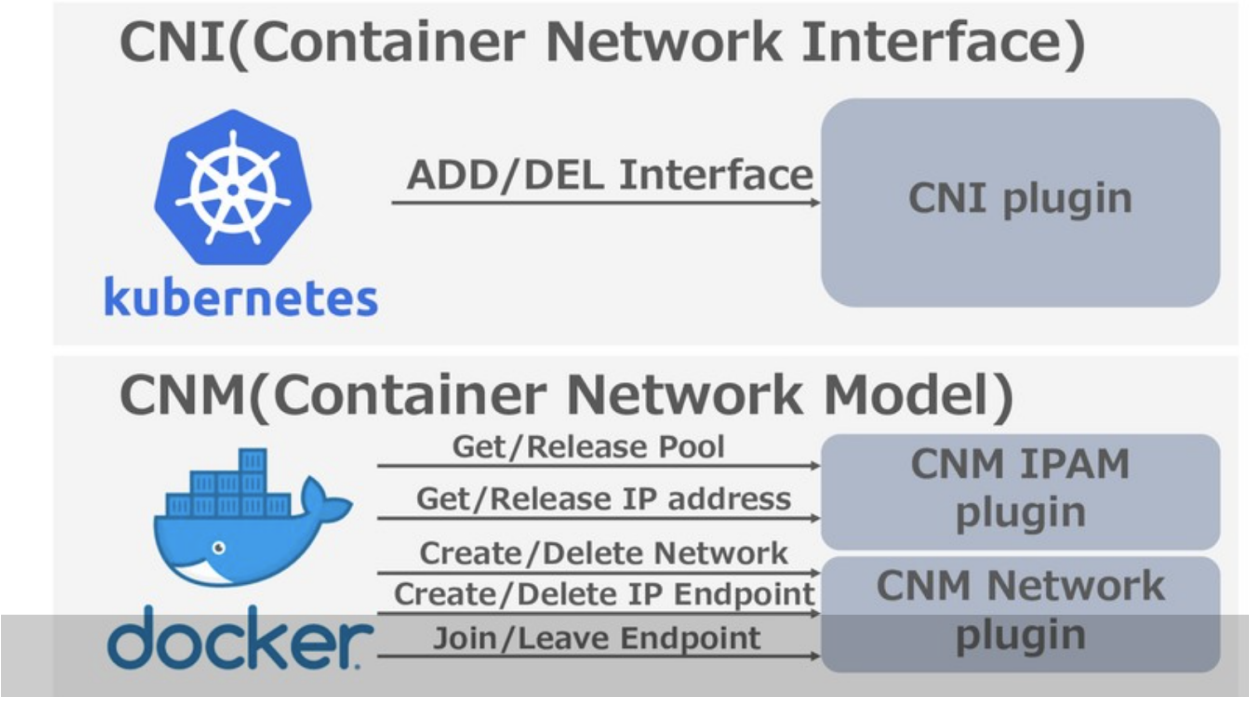
Intranode visibility feature also changes default routing policy. (PTP CNI)



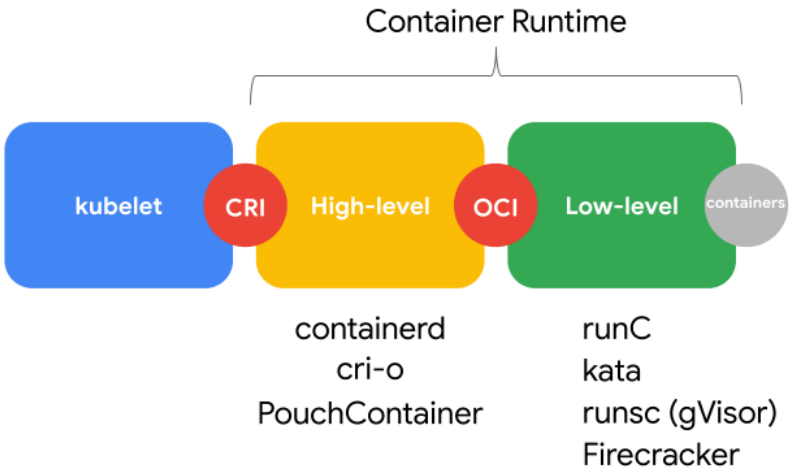
Google Cloud

https://cloud.google.com/kubernetes-engine/docs/concepts/network-overview#pods

CNI vs CNM



Container Runtime positioned



Google Cloud