

# 쿠버네티스 CNI 설명



쿠버네티스 cni에 대해서 비기너 입장에서 설명 자세히 부탁해.



52s 동안 생각함

- Kubernetes CNI는 컨테이너 네트워크 인터페이스의 약자입니다. CNI는 컨테이너 네트워킹을 설정하는데 사용되고, kube-proxy와는 다른 역할을 합니다. kube-proxy는 클러스터 내부 서비스들을 연결하고, CNI는 포드를 네트워크와 연결합니다. 예를 들어 Cilium과 Calico는 CNI의 구현체로, Dataplane에서는 eBPF를 사용하는 차이점이 존재합니다. Minikube의 기본 CNI는 'bridge'입니다.
- Minikube는 기본적으로 'bridge' CNI 플러그인을 사용합니다. CNI 플러그인들을 /etc/cni/net.d에 설정하며, 'flannel', 'cilium', 'calico'와 같은 다양한 플러그인도 추가 가능합니다. CNI의 역할은 각 Pod에 고유 IP를 할당하고, NAT 없이 통신할 수 있도록 설정하는 것입니다. CNI 플러그인은 IPAM, veth 페어를 만들고, 라우팅 및 Host-level iptables등의 설정을 처리합니다.
- 먼저, L2/L3 네트워크, 오버레이(VXLAN, IP-in-IP), BGP 기반 라우팅 네트워크, eBPF dataplane에 대해 설명하고, 주요 CNI 플러그인인 Flannel, Calico, Cilium, Weave Net, Canal의 기능 차이점을 비교할게요. 특히 네트워크 정책, 암호화, eBPF, BGP, Windows 지원 등을 살펴보겠습니다.  
Minikube에서 CNI를 활성화하고 비활성화하는 간단한 명령어를 제공하고, CNI 확인 방법도 알려드릴게요: ``kubectl get pods -n kube-system`, `node annotations 확인`, `CNI가 설치된 위치(/etc/cni/net.d)` 확인, `ip addr`, `ip route` 등.`  
기본적인 네트워크 정책 예시도 포함할 게요. 예를 들어, 두 개의 네임스페이스를 만들고, 네트워크 정책을 적용해 네임스페이스 간의 통신을 차단하는 실습도 할 수 있어요. ``busybox` pods로 실험을 진행하며, 그에 필요한 명령어와 YAML 파일도 안내드릴 거예요.`  
이 모든 내용을 따르도록 구성할게요.
- - **한 줄 정의:** CNI는 쿠버네티스에서 컨테이너 네트워크를 설정하는 표준.
  - **왜 필요한가:** 쿠버네티스 네트워크 모델에서는 Pod 간의 통신과 네트워크 관리가 필수.
  - **CNI가 하는 일/안 하는 일:** Pod의 IP 할당, 네트워크 연결 설정 담당, Service LB는 kube-proxy/eBPF가 처리.
  - **내부 동작 순서:** Pod 생성 시 ADD 호출, 삭제 시 DEL 호출.
  - **CNI 종류:** Flannel, Calico, Cilium, Weave, Canal, Kube-router 등 다양한 플러그인과 특징.
  - **Minikube에서 CNI:** 기본 상태 확인 및 설치/전환 명령어 제공. (예: ``minikube addons enable``)
  - **실습:** NetworkPolicy 예시 (deny-all, allow-specific)와 YAML 파일 활용, 테스트와 명령어 제공.

## 쿠버네티스 CNI, 처음부터 탄탄하게

편하게 훑고, 필요할 때 다시 꺼내 쓰기 좋게 정리해 드릴게요. (WSL2 + minikube 기준 팁도 같이 넣었습니다.)

### 1) 한 줄 정의

**CNI(Container Network Interface)** 는 “파드에 IP를 주고, 서로 통신되게 네트워크를 붙여주는 표준 규격 + 플러그인 생태계” 입니다.

쿠버네티스가 “파드가 서로 IP로 통신된다”는 약속을 지키도록 실제 네트워크를 구성해요.

### 2) 왜 필요한가 — 쿠버네티스 네트워크 3원칙

쿠버네티스는 다음을 전제로 합니다.

1. 모든 파드는 고유 IP를 가진다.
  2. NAT 없이 파드 ↔ 파드가 직접 통신 가능하다.
  3. 노드 ↔ 파드도 직접 통신 가능하다.
- 이걸 현실로 만드는 것이 CNI 플러그인의 역할입니다.

### 3) CNI가 “하는 일” vs “안 하는 일”

#### 하는 일

- **IPAM**: 파드에 IP 할당/회수
- **인터페이스 생성**: 파드 안/밖에 veth(가상 이더넷) 페어 생성
- **네트워크 연결**: 브리지에 붙이거나(로컬 L2), 오버레이(VXLAN/IP-in-IP)나 라우팅(BGP) 구성
- **라우팅/규칙**: 호스트 라우팅/iptables(eBPF) 세팅
- **정리(DEL)**: 파드 삭제 시 깔끔히 제거

#### 안 하는 일

- **Service 로드밸런싱** 자체는 담당 아님 (kube-proxy 또는 eBPF 데이터플레인 담당)
- **Ingress/북-남(North-South) 트래픽**은 Ingress Controller/외부 LB 영역

## 4) 파드가 뜰 때 내부 동작(요약)

1. scheduler가 노드 배정 → kubelet이 파드 네트워크 준비 요청
2. 컨테이너 런타임(CRI)이 **CNI 호출(ADD)**
3. CNI 플러그인이 **IP할당 + veth 생성 + 브리지/오버레이 연결 + 라우팅 설정**
4. 파드가 IP를 얻고 통신 시작  
삭제 시에는 **CNI DEL** 로 반대로 정리

## 5) 대표 CNI들의 성격 지도

- **Flannel**: 가장 단순/가벼운 오버레이(VXLAN). 네트워크폴리시 X → 필요하면 다른 컴포넌트와 조합. 메모리 적게 씀.
- **Calico**: L3 라우팅(BGP) + 네트워크폴리시 강력. 오버레이(IP-in-IP/VXLAN)도 가능. 대규모/온프레 많이 씀.
- **Cilium**: 커널 **eBPF** 기반. 고성능, 네트폴, 레벨7 정책/가시성(**Hubble**), 서비스 LB(eBPF)까지. 기능 풍부.
- **Canal(Flannel+Calico)**: Flannel 데이터플레인 + Calico 정책.
- **Weave Net**: 간편 오버레이 + 네트폴 지원(요즘은 사용 비중 감소 추세).

간단 기준:

- “가벼움/학습” → Flannel
- “온프레/라우팅/BGP/정교한 정책” → Calico
- “고성능·관측·eBPF” → Cilium

## 6) minikube에서 CNI 다루기 (WSL2 포함)

minikube는 여러 CNI를 애드온으로 제공합니다.

```
# 어떤 CNI가 켜져 있는지
minikube addons list | grep -E 'cilium|calico|flannel'

# 예) Cilium 켜기
minikube addons enable cilium

# 예) Flannel 켜기
minikube addons enable flannel
```

```
# 예) Calico 켜기
minikube addons enable calico

# 변경했으면, 보통 재시작/재생성이 깔끔
minikube delete
minikube start --cni=cilium # 또는 --cni=flannel / --cni=calico
```

상태 확인:

```
kubectl get pods -n kube-system
kubectl -n kube-system logs deploy/cilium-operator --tail=100 # 예: Cilium
kubectl -n kube-system get ds,cj,deploy | grep -i cilium
```

노드 내부(WSL2 내 minikube VM)에 들어가 네트워크도 볼 수 있어요:

```
minikube ssh
ip addr | grep -E 'cni|vxlan|flannel|cilium'
ip route
```

### WSL2 팁

- Windows <-> WSL2 NAT 특성상 **NodePort** 접근이 막히는 느낌이면 `minikube tunnel` (LoadBalancer용)이나 `kubectl port-forward`를 활용하세요.
- VPN 사용 중엔 **MTU** 이슈로 오버레이가 끊기는 경우가 흔합니다(증상: 간헐적 타임아웃). CNI MTU를 1450 근처로 낮추면 안정되는 편.

## 7) 바로 해보는 입문 실습: NetworkPolicy 맛보기

아래는 CNI가 네트워크폴리시를 지원할 때(예: Calico/Cilium) 효과가 보이는 실습입니다.

### (1) 네임스페이스 및 샘플 앱

```
kubectl create ns team-a
kubectl create ns team-b

# team-a에 NGINX 배포
kubectl -n team-a apply -f - <<'YAML'
apiVersion: apps/v1
kind: Deployment
metadata: { name: web }
```

```
spec:
  replicas: 1
  selector: { matchLabels: { app: web } }
  template:
    metadata: { labels: { app: web } }
    spec:
      containers:
      - name: nginx
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
  ---
apiVersion: v1
kind: Service
metadata: { name: web-svc }
spec:
  selector: { app: web }
  ports:
  - port: 80
    targetPort: 80
```

YAML

*# team-a, team-b 각각 테스트용 파드(무한 대기)*

```
kubectl -n team-a run t1 --image=busybox:1.36 --restart=Never -- /bin/sh -c
'sleep 1d'
kubectl -n team-b run t2 --image=busybox:1.36 --restart=Never -- /bin/sh -c
'sleep 1d'
```

## (2) 정책 적용 전 통신 확인

*# team-a 내부에서 접근 (성공)*

```
kubectl -n team-a exec t1 -- wget -qO- http://web-svc
```

*# team-b 에서 접근 (기본은 성공; CNI/정책 엔진에 따라 다름)*

```
kubectl -n team-b exec t2 -- wget -qO- http://web-svc
```

## (3) team-a 네임스페이스 기본 거부(deny all)

```
kubectl -n team-a apply -f - <<'YAML'
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: { name: default-deny }
spec:
  podSelector: {}          # team-a의 모든 파드 대상
  policyTypes: ["Ingress"]
YAML
```

#### (4) 같은 네임스페이스에서만 허용

```
kubectl -n team-a apply -f - <<'YAML'
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: { name: allow-same-ns }
spec:
  podSelector:
    matchLabels: { app: web }
  policyTypes: ["Ingress"]
  ingress:
    - from:
      - podSelector: {}    # team-a 내부 파드만
YAML
```

테스트:

```
# team-a → 허용
kubectl -n team-a exec t1 -- wget -qO- --timeout=2 http://web-svc

# team-b → 차단 (타임아웃 예상)
kubectl -n team-b exec t2 -- wget -qO- --timeout=2 http://web-svc || echo
"blocked"
```

## 8) 자주 부딪히는 오류 & 디버깅 포인트

- 파드가 **Pending/ContainerCreating**에서 멈춤 + **“FailedCreatePodSandBox / network plugin is not ready”**  
→ CNI 데몬셋(예: cilium, calico-node, flannel) 상태/로그 확인

- 간헐적 타임아웃  
→ MTU/오버레이/VPN 충돌 의심. CNI MTU 조정, VPN 끄고 비교
- **NodePort/외부접속 불가(WSL2/회사망)**  
→ ``kubectl port-forward`, `minikube service ... --url`, `minikube tunnel`` 활용
- 정책이 안 먹는 느낌  
→ 쓰는 CNI가 네트워크폴리시 지원하는지 먼저 확인(Calico/Cilium OK, Flannel 단독은 X)

현장 체크 커맨드:

```
kubectl describe pod <이름>
kubectl get events -A --sort-by=.lastTimestamp | tail
kubectl -n kube-system logs -l k8s-app=flannel --tail=200
kubectl -n kube-system logs -l k8s-app=calico-node --tail=200
kubectl -n kube-system logs -l k8s-app=cilium --tail=200
```

## 9) 용어 빠르게 정리

- **veth pair**: 파드 네임스페이스 ↔ 호스트를 잇는 가상 케이블
- **브리지(cni0 등)**: 여러 veth를 스위치처럼 묶는 가상 L2
- **오버레이**: 다른 네트워크 위에 가상 네트워크를 얹음(VXLAN, IP-in-IP, WireGuard 등)
- **BGP**: 노드 간 라우팅 정보를 광고해 오버레이 없이 L3로 통신
- **eBPF**: 커널에서 고성능 패킷 처리/가시성 제공(Cilium의 핵심)
- **IPAM**: IP 할당 관리

## 10) 어떤 걸 고를까? (빠른 감)

- 학습/가볍게: Flannel
- 온프레/정교한 제어/**BGP**: Calico
- 고성능·관측·정책 풍부: Cilium(Hubble로 트래픽 보기도 좋음)

## 11) 당신 환경에 딱 맞는 팁(WSL2 + minikube)

- 회사 VPN 쓰면 **MTU**부터 의심하세요(1450 전후).
- Windows에서 서비스 접근이 안 될 땐 ``minikube service ... --url`, `kubectl port-forward`, `minikube tunnel`` 조합이 현실적입니다.

- CNI 바꾼 후 꼬이면 ``minikube delete && minikube start --cni=<선택>`` 가 시간 절약.

필요하면, **Calico**나 **Cilium** 기준으로 “설치 → 기본 튜닝(MTU/정책) → Hubble/Policy 시각화”까지 바로 실습 흐름도 만들어 드릴게요. 어느 쪽이 궁금하세요?



cilium기준으로 최대한 기초적일것 부터해서 모두 설명해줘



좋습니다 👍 이제부터는 **Cilium**만 딱 잡고, “아주 기초 → 조금 깊이 → 실제 설치/테스트 → 문제 상황 디버깅” 흐름으로 정리해 드릴게요.

## Cilium, 처음부터 끝까지

### 1. Cilium이 뭐냐?

- 쿠버네티스에서 **네트워크(CNI)** 역할을 하는 플러그인입니다.
- 보통 CNI는 iptables, 라우팅 같은 **리눅스 커널 네트워킹** 기능을 이용하는데,  
👉 **Cilium은 eBPF** 라는 최신 커널 기술을 씁니다.

### 2. eBPF 한 줄 설명

- “리눅스 커널 안에서 작고 안전한 프로그램을 실행할 수 있게 해주는 기술.”
- 네트워크 패킷을 잡아 처리하거나, 보안 규칙/관측(log, metrics)을 고성능으로 처리 가능.
- iptables보다 빠르고, 더 세밀한 제어가 가능함.

### 3. Cilium의 역할 (쿠버네티스에서)

- **CNI**: 파드에 IP 주고, 라우팅 세팅.
- **Service LB**: kube-proxy 없이도 서비스 로드밸런싱 가능.
- **네트워크 정책**: 파드 간 “누가 누구랑 통신 가능한지” 제어.



- **가시성(Hubble)**: 어떤 파드가 누구랑 통신했는지, 정책에 막혔는지 시각화.
- 

## 4. 쿠버네티스에서 동작 흐름

1. 파드가 생성 → kubelet이 Cilium 호출
  2. Cilium이 파드에 IP 할당, veth pair 생성, 라우팅 설정
  3. 모든 노드에 깔린 Cilium Agent가 eBPF로 통신을 처리
  4. 서비스 접근 시 kube-proxy 대신 Cilium이 로드밸런싱 수행
  5. 네트워크 정책 적용 시, eBPF 레벨에서 허용/차단
- 

## 5. 왜 Cilium인가?

- 성능: iptables 대신 eBPF로 직접 처리 → 지연 적음.
  - 확장성: 노드/파드 많아도 룰 관리가 깔끔.
  - 기능: L7(HTTP, gRPC) 수준까지 정책 제어 가능.
  - 관측성: Hubble로 트래픽 흐름을 눈으로 확인.
- 

## 6. 설치 (minikube + WSL2 기준)

### 6-1) minikube 시작할 때

```
minikube start --cni=cilium
```

혹은 애드온으로 켜기:

```
minikube addons enable cilium
```

### 6-2) 상태 확인

```
kubectl get pods -n kube-system -l k8s-app=cilium
kubectl get ds -n kube-system cilium
kubectl -n kube-system logs -l k8s-app=cilium --tail=50
```

---

## 7. Hubble (Cilium 가시성 툴)

```
minikube addons enable hubble
kubectl -n kube-system get pods -l k8s-app=hubble
```

UI 띄우기:

```
minikube addons open hubble
```

CLI 사용:

```
cilium hubble enable
cilium hubble port-forward&
hubble status
hubble observe --last 5
```

---

## 8. 네트워크 정책 (기본 실습)

### (1) 파드 만들기

```
kubectl create ns demo

kubectl -n demo run client --image=busybox:1.36 --restart=Never -- sleep 1d
kubectl -n demo run server --image=nginx:1.25-alpine --restart=Never --port=80
kubectl -n demo expose pod server --port=80
```

### (2) 통신 확인

```
kubectl -n demo exec client -- wget -qO- http://server:80
```

### (3) 정책으로 막기

```
# deny-all.yaml
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: deny-all
  namespace: demo
```

```
spec:
  endpointSelector: {}
  ingress: []
```

적용:

```
kubectl apply -f deny-all.yaml
```

테스트 → 타임아웃 나와야 정상.

#### (4) 다시 열어주기

```
# allow-client.yaml
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-client
  namespace: demo
spec:
  endpointSelector:
    matchLabels:
      run: server
  ingress:
    - fromEndpoints:
        - matchLabels:
            run: client
      toPorts:
        - ports:
            - port: "80"
          protocol: TCP
```

적용 후 다시 wget 하면 성공.

## 9. 자주 겪는 문제

- **Cilium 파드 CrashLoopBackOff** → 커널 버전/eBPF 지원 확인 (`uname -r`). WSL2는 보통 최신이라 괜찮음.
- **통신 타임아웃** → VPN/MTU 이슈. MTU 1450 근처로 조정 필요할 수 있음.

- 서비스 안 됨 → kube-proxy 끄고 Cilium LB 쓰는지 확인 (``kubectl -n kube-system get cm cilium-config``).

## 10. 핵심 용어 빠르게

- **cilium-agent**: 각 노드에서 실행, eBPF 관리.
- **cilium-operator**: IPAM 같은 클러스터 레벨 관리.
- **CiliumNetworkPolicy(CNP)**: Calico의 NetworkPolicy보다 세밀한 정책 리소스.
- **Hubble**: 트래픽 흐름 관측 도구.

정리하면:

- 👉 Cilium은 “쿠버네티스 네트워크 + 보안 + 가시성” 3박자를 eBPF로 해결하는 CNI.
- 👉 minikube에서는 ``--cni=cilium``으로 간단히 켤 수 있고, Hubble로 시각화까지 체험 가능.
- 👉 초보 단계에서는 “파드 간 통신 막고 열어보기 → Hubble로 확인하기” 실습이 가장 직관적이에요.

원하시면, 제가 **“Cilium 실습 시나리오 PDF 버전(예제 YAML 포함)”**으로 정리해서 드릴 수도 있습니다. 그렇게 해드릴까요?



응 제공해줘(설명도 자세히 달아줘, 내가 설명할 수 있을 정도로)



1m 37s 동안 생각함

- 사용자가 PDF 제공을 요청했으므로 세부 설명과 예시를 포함해 보고서를 작성할 예정입니다. Markdown 방식으로 작성해 PDF로 변환해야 하는데, 인터넷 없이 ReportLab을 사용하여 PDF를 간단히 생성하는 방법을 선택할 거예요. 코드 블록과 차트 관련 제약을 고려해 가며 PDF를 준비하겠습니다.
- 한글을 지원하는 폰트가 없는 문제가 있어요. ReportLab의 기본 글꼴은 한글을 지원하지 않아서, 한글이 네모로 표시될 수 있어요. NotoSansCJK나 NanumGothic과 같은 한글 폰트를 등록해야 하지만 기본 환경에서는 이를 지원하지 않아요. 그러므로 PDF를 만드는 다른 방법을 고려해야 해요.

- 사용자가 예의 바른 한국어를 선호하므로 PDF는 한국어로 제공해야 해요. 그러나 인터넷 없이 한글을 지원하는 폰트를 추가할 수 없고, 기본 폰트로는 한글이 잘 나오지 않아요. 그래서 DOCX 파일을 생성하는 방법도 고려해볼 수 있어요. 둘 다 제공하는 방법도 가능할 듯해요.
- 한글 폰트 문제로 PDF 생성이 어려운 경우, DOCX와 Markdown 파일을 제공하는 방법을 고려할 수 있어요. PDF는 폰트가 없으면 제대로 표시되지 않지만, 이를 명확히 안내하고 동일 자료를 다른 형태로 제공하는 것이 좋겠어요. 폰트가 없으면 PDF에서 빈 사각형이 나타날 수 있어요.
- 그래서 사용자님이 요청하신 PDF 대신, 캔버스를 활용하여 "Cilium 실습 가이드" 자료를 만들면 좋겠어요. 이 가이드를 캔버스에서 제공하고, 사용자가 이 내용을 PDF로 변환할 수 있게 하면 될 것 같아요. PDF에 대한 폰트 문제를 피할 수 있으니까요.

Cilium 실습 가이드 (초심자용) v1

복사 편집 다운로드

# Cilium 실습 가이드 (초심자용)

**대상:** 쿠버네티스 입문자(3개월 내), WSL2 + Minikube 환경에서 Cilium을 처음 써보려는 분

**목표:** ① Cilium과 eBPF의 기본 개념 이해 ② Minikube에 설치/검증 ③ 네트워크 정책 (CNP) 실습 ④ Hubble로 트래픽 가시화 ⑤ 자주 겪는 문제와 해결법 숙지

## 0. 준비물 & 환경

- OS: Windows 10/11 + **WSL2 (Ubuntu 22.04/24.04)**
- 도구: `kubectl`, `minikube`, (선택) `helm`
- 리소스: CPU 4+, RAM 8GB 이상 권장
- 네트워크: 회사 VPN 연결 시 **MTU** 이슈가 있을 수 있음(뒤에서 다룸)

**명령어 표기:** 특별히 밝히지 않으면 **WSL2 리눅스 셸**에서 실행합니다.

# 1. Cilium 한눈에 보기

## 1.1 Cilium이란?

- 쿠버네티스의 **CNI(Container Network Interface)** 플러그인.
- 파드에 IP를 부여하고, 파드/노드 간 통신을 **eBPF** 기반으로 고성능 처리.
- **kube-proxy** 없이 서비스 로드밸런싱도 가능(환경 설정에 따라 다름).

## 1.2 eBPF 한 줄 정의

- 리눅스 커널에서 안전한 작은 프로그램을 실행하는 기술.
- 패킷 처리/관측/보안을 **커널 레벨**에서 고속으로 수행할 수 있음.

## 1.3 왜 Cilium인가?

- 성능: iptables 체인보다 낮은 지연과 높은 확장성
- 기능: L3~L7(HTTP/gRPC) 정책, 서비스 LB, NAT 세밀 제어
- 가시성: **Hubble**로 흐름(Flow) 관찰, 정책 드롭 원인 파악이 쉬움

# 2. 아키텍처 빠르게 훑기

- **cilium-agent (DaemonSet)**: 각 노드에서 eBPF 설치/관리, 정책 적용, 서비스 LB 등 데이터플레인 역할
- **cilium-operator (Deployment)**: IPAM 등 클러스터 레벨 관리
- **Hubble**: 트래픽 흐름 수집/조회(리레이 + UI)
- 데이터플레인: eBPF 프로그램 + BPF Maps (정책/서비스/연결 추적 등 상태 저장)

핵심 흐름: (1) 파드 생성 → (2) Cilium이 veth + IP 할당 → (3) eBPF가 라우팅, 정책, 로드밸런싱을 처리

## 3. Minikube에 Cilium 설치

### 3.1 가장 간단한 방법

*# 기존 클러스터가 있다면 지우고 시작(선택)*

```
minikube delete
```

*# Cilium을 CNI로 지정해 클러스터 시작*

```
minikube start --cni=cilium
```

또는 애드온 사용:

```
minikube addons enable cilium
```

*# 필요 시 재시작*

```
minikube delete && minikube start
```

### 3.2 정상 동작 확인

```
kubectl -n kube-system get pods -l k8s-app=cilium
```

```
kubectl -n kube-system get ds cilium
```

```
kubectl -n kube-system get deploy cilium-operator
```

```
kubectl -n kube-system logs -l k8s-app=cilium --tail=50
```

``cilium`` CLI가 필요하다면 노드에서 실행:

*# 노드에 접속하여(미니쿠브 VM 내부)*

```
minikube ssh
```

```
sudo cilium status
```

```
sudo cilium bpf map list | head
```

**팁:** 클러스터 안의 에이전트 컨테이너에서 직접 실행할 수도 있습니다.

```
kubectl -n kube-system exec -it ds/cilium -c cilium-agent -- cilium status
```

### 3.3 MTU 이슈(WSL2/VPN)

- 오버레이(VXLAN) + VPN 환경에서 MTU가 커서 **간헐 타임아웃/끊김** 발생 가능.
- 해결: Cilium/노드 MTU를 **1450** 전후로 맞추기(환경에 따라 최적값 상이).

- Minikube 애드온은 일반적으로 적절한 MTU를 자동 설정하지만, 문제시 문서화된 옵션으로 조정.

## 4. 첫 실습: 파드 통신과 서비스

### 4.1 샘플 배포

```
kubectl create ns demo
```

```
kubectl -n demo apply -f - <<'YAML'
apiVersion: apps/v1
kind: Deployment
metadata: { name: web }
spec:
  replicas: 1
  selector: { matchLabels: { app: web } }
  template:
    metadata: { labels: { app: web } }
    spec:
      containers:
      - name: nginx
        image: nginx:1.25-alpine
        ports:
        - containerPort: 80
```

```
---
apiVersion: v1
kind: Service
metadata: { name: web-svc }
spec:
  selector: { app: web }
  ports:
  - port: 80
    targetPort: 80
```

YAML

*# 테스트용 클라이언트 파드*

```
kubectl -n demo run client --image=busybox:1.36 --restart=Never -- /bin/sh
-c 'sleep 1d'
```



## 4.2 통신 확인

```
kubectl -n demo exec client -- wget -qO- http://web-svc
```

정상이라면 NGINX 기본 HTML이 출력됩니다.

**참고(서비스 LB):** Cilium은 환경에 따라 **kube-proxy 없이** 서비스 로드밸런싱을 수행합니다. `cilium status` 출력에서 `KubeProxyReplacement: Strict/Probe` 항목으로 상태를 확인하세요.

## 5. 네트워크 정책(CiliumNetworkPolicy) 기본

쿠버네티스 표준 `NetworkPolicy`도 지원하지만, Cilium 고유 리소스인 **CiliumNetworkPolicy(CNP)**가 더 세밀(L7 등)합니다.

### 5.1 기본 차단(deny all)

```
# deny-all.yaml
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: deny-all
  namespace: demo
spec:
  endpointSelector: {}
  ingress: []
```

적용:

```
kubectl apply -f deny-all.yaml
```

테스트:

```
kubectl -n demo exec client -- wget -qO- --timeout=2 http://web-svc || echo "blocked"
```

### 5.2 특정 소스만 허용

```
# allow-client-to-web.yaml
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-client
  namespace: demo
spec:
  endpointSelector:
    matchLabels:
      app: web
  ingress:
    - fromEndpoints:
        - matchLabels:
            run: client
      toPorts:
        - ports:
            - port: "80"
          protocol: TCP
```

적용 후 다시 접근하면 성공해야 합니다.

## 5.3 L7(HTTP) 정책 맛보기

특정 **HTTP** 메서드/경로만 허용합니다.

```
# allow-http-get-root.yaml
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-http-get-root
  namespace: demo
spec:
  endpointSelector:
    matchLabels:
      app: web
  ingress:
    - fromEndpoints:
        - matchLabels: { run: client }
      toPorts:
        - ports:
            - port: "80"
```

```
protocol: TCP
rules:
  http:
    - method: GET
      path: "/"
```

**주의:** L7 정책은 앱이 **HTTP**로 통신해야 의미가 있습니다(암호화된 HTTPS는 L7 파싱 불가).

## 6. Hubble로 가시화

### 6.1 활성화

```
minikube addons enable hubble
kubectl -n kube-system get pods -l k8s-app=hubble
```

### 6.2 UI 접속

# 미니쿠브가 자동으로 포트 오픈(기본 브라우저)  
minikube addons open hubble

### 6.3 관찰 포인트

- `demo` 네임스페이스 필터 후, `client` → `web` 요청 플로우 확인
- 허용/차단 이벤트(Policies)와 드롭 이유
- L7 탭에서 HTTP 메서드/경로 관찰

**CLI 참고:** 로컬에 `hubble` CLI가 없다면, UI만으로도 충분히 시연 가능합니다.

## 7. 운영에 가까운 팁

- **MTU 튜닝:** VPN/클라우드 혼재 환경에서  $1450 \pm 10$  수준 권장(환경별 상이). 증상은 간헐 타임아웃.
- **kube-proxy 대체 모드:** ``cilium status``에서 ``KubeProxyReplacement`` 확인. 서비스 문제가 있으면 이 부분을 먼저 점검.
- **DNS 문제:** ``kube-dns/CoreDNS`` 파드와의 정책 충돌 혼함 → DNS로의 egress를 허용하는 정책 추가 검토.
- **BPF 상태 점검:** ``cilium bpf`` 하위 명령으로 맵 상태를 확인(연결 추적, 서비스, 정책 등).

## 8. 자주 겪는 오류와 해결

증상	원인 후보	해결책
파드가 <code>`ContainerCreating/FailedCreatePodSandBox`</code>	CNI 준비 안 됨	<code>`kubectl -n kube-system get pods -l k8s-app=cilium`</code> , 로그 확인
통신 타임아웃(간헐)	MTU 불일치, VPN	MTU 1450 전후로 조정, VPN 해제 후 비교
Service 접근 문제	kube-proxy 대체 모드/설정 충돌	<code>`cilium status`</code> 확인, 모드 조정, 서비스/Endpoint 상태 재확인
정책이 안 먹음	CNI가 정책 미지원/라벨 불일치	Cilium 사용 확인, <code>`pod&lt;labels&gt;`</code> 와 정책의 <code>`matchLabels`</code> 정합성 점검
NodePort 외부접속 불가(WSL2)	NAT 구조/방화벽	<code>`kubectl port-forward`</code> , <code>`minikube service ... --url`</code> , <code>`minikube tunnel`</code> 활용

빠른 디버깅 명령:

```
kubectl describe pod <이름>
kubectl get events -A --sort-by=.lastTimestamp | tail
kubectl -n kube-system logs -l k8s-app=cilium --tail=200
kubectl -n kube-system logs deploy/cilium-operator --tail=200
```

---

## 9. 부록 A: 명령 모음

*# 설치*

```
minikube start --cni=cilium
minikube addons enable cilium
minikube addons enable hubble
```

*# 점검*

```
kubectl -n kube-system get pods -l k8s-app=cilium
kubectl -n kube-system exec -it ds/cilium -c cilium-agent -- cilium status
```

*# 서비스 확인*

```
kubectl -n demo exec client -- wget -qO- http://web-svc
```

*# Hubble UI*

```
minikube addons open hubble
```

---

## 10. 부록 B: 전체 YAML 모음

### 10.1 web + service

```
apiVersion: apps/v1
kind: Deployment
metadata: { name: web }
spec:
  replicas: 1
  selector: { matchLabels: { app: web } }
  template:
    metadata: { labels: { app: web } }
    spec:
```

```
    containers:
      - name: nginx
        image: nginx:1.25-alpine
        ports:
          - containerPort: 80
---
apiVersion: v1
kind: Service
metadata: { name: web-svc }
spec:
  selector: { app: web }
  ports:
    - port: 80
      targetPort: 80
```

## 10.2 deny-all

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: deny-all
  namespace: demo
spec:
  endpointSelector: {}
  ingress: []
```

## 10.3 allow client → web:80

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-client
  namespace: demo
spec:
  endpointSelector:
    matchLabels:
      app: web
  ingress:
    - fromEndpoints:
        - matchLabels:
            run: client
```

```
toPorts:
- ports:
  - port: "80"
    protocol: TCP
```

## 10.4 L7 HTTP: GET /

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-http-get-root
  namespace: demo
spec:
  endpointSelector:
    matchLabels:
      app: web
  ingress:
  - fromEndpoints:
    - matchLabels: { run: client }
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
      rules:
        http:
        - method: GET
          path: "/"
```

---

## 11. 마무리 요약(발표용 포인트)

- Cilium = eBPF 기반 CNI + L7 정책 + 서비스 LB + Hubble 가시성
- Minikube에서 ``--cni=cilium``이면 쉽게 체험 가능
- 실습 골자: (1) 배포 (2) 통신확인 (3) 정책으로 차단/허용 (4) Hubble로 눈으로 검증
- 운영 팁: MTU/서비스 모드/라벨 정합성/코어DNS 정책을 우선 점검

다음 단계: Hubble Metrics, L7 정책 심화(HostPolicies, FQDN), WireGuard 암호화, BGP/Direct Routing 모드 비교

완료했습니다. 캔버스에 \*\*PDF로 바로 내보내기 좋은 “Cilium 실습 가이드 (초심자용)”\*\*를 올려두었어요.

필요한 만큼 바로 써먹고, 내용 추가·수정할 부분 있으면 편하게 말씀 주세요.