

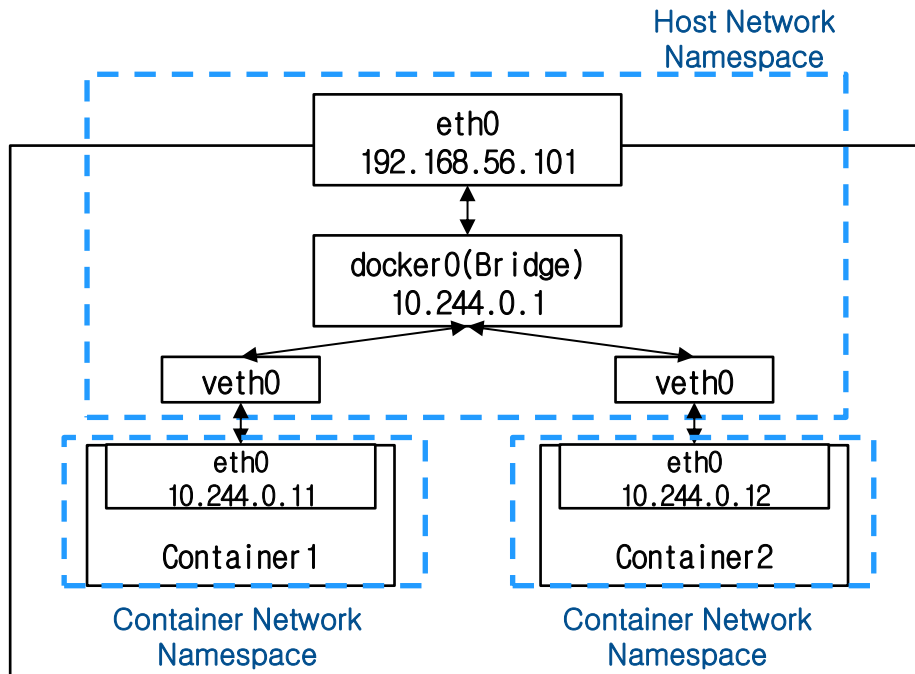
4. k8s 네트워킹

- ❖ k8s 네트워킹 개요
- ❖ CNI
- ❖ Calico CNI
- ❖ AWS VPC CNI
- ❖ Azure AKS의 CNI
- ❖ Calico를 이용한 네트워크 보안

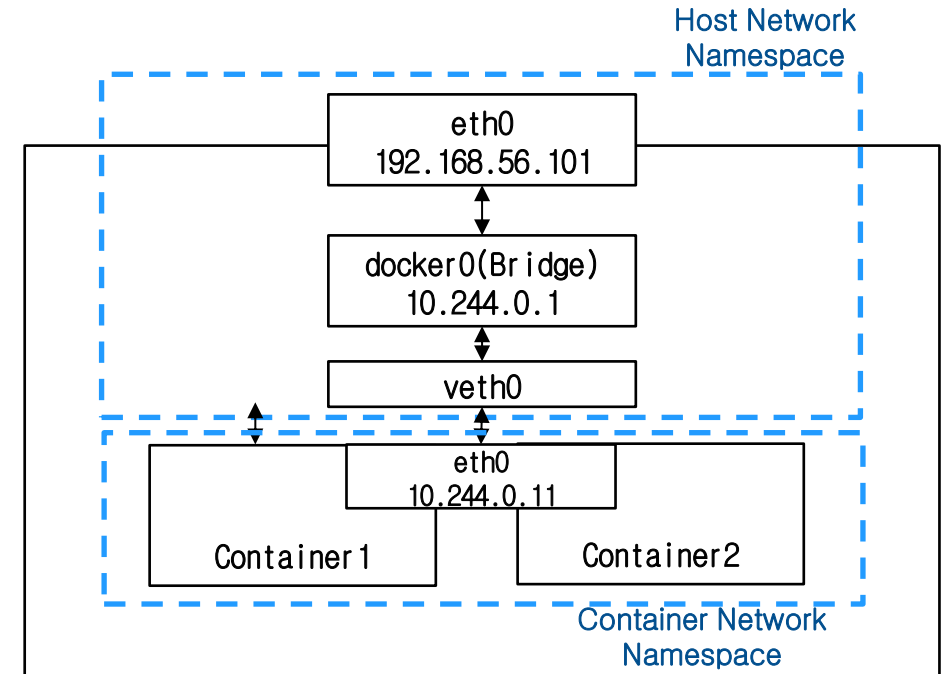
4.1 k8s 네트워킹 개요

❖ Docker Container Network

- Docker Bridge Network Mode
 - 컨테이너를 시작할 때마다 호스트에 veth 라는 가상의 네트워크 인터페이스를 생성
- Docker Container Network Mode : `--net` 옵션으로 container를 입력
 - 다른 컨테이너의 네트워크 네임스페이스를 공유할 수 있음. 내부 IP, MAC 주소



Bridge Network Mode

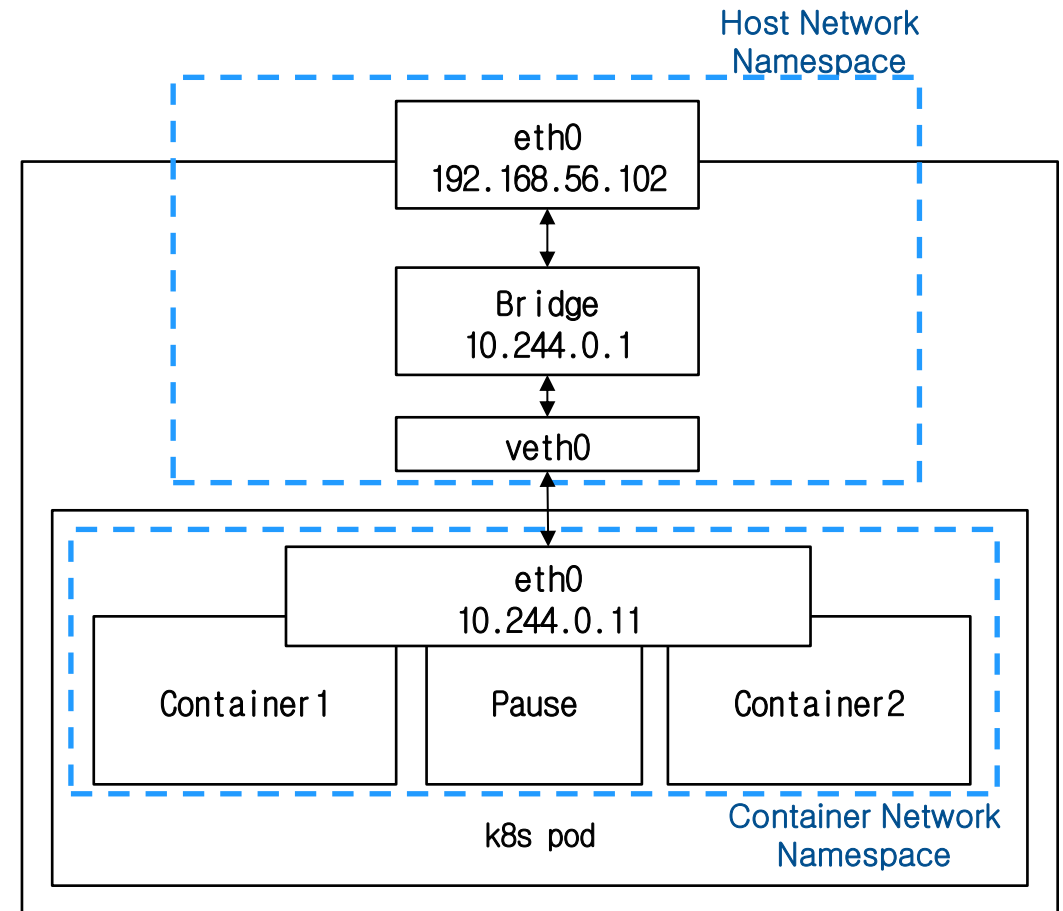


Container Network Mode

4.1 k8s 네트워킹 개요

❖ k8s의 한 Pod 내에서의 컨테이너간 통신

- Docker Container Network Mode 개념을 k8s에 도입
- pause 컨테이너를 이용해 통신함
- pause 컨테이너의 역할
 - 네트워크 네임스페이스 생성 (-- 하나의 IP 주소 부여)



4.2 CNI

❖CNI(Container Network Interface)란?

- 컨테이너 간의 네트워킹을 제어할 수 있는 플러그인이 지켜야하는 표준
- 기본 제공하는 CNI인 kubenet은 기능이 매우 제한적임.
- 일반적으로 추가적인 CNI 플러그인을 설치함

❖CNI 플러그인 종류

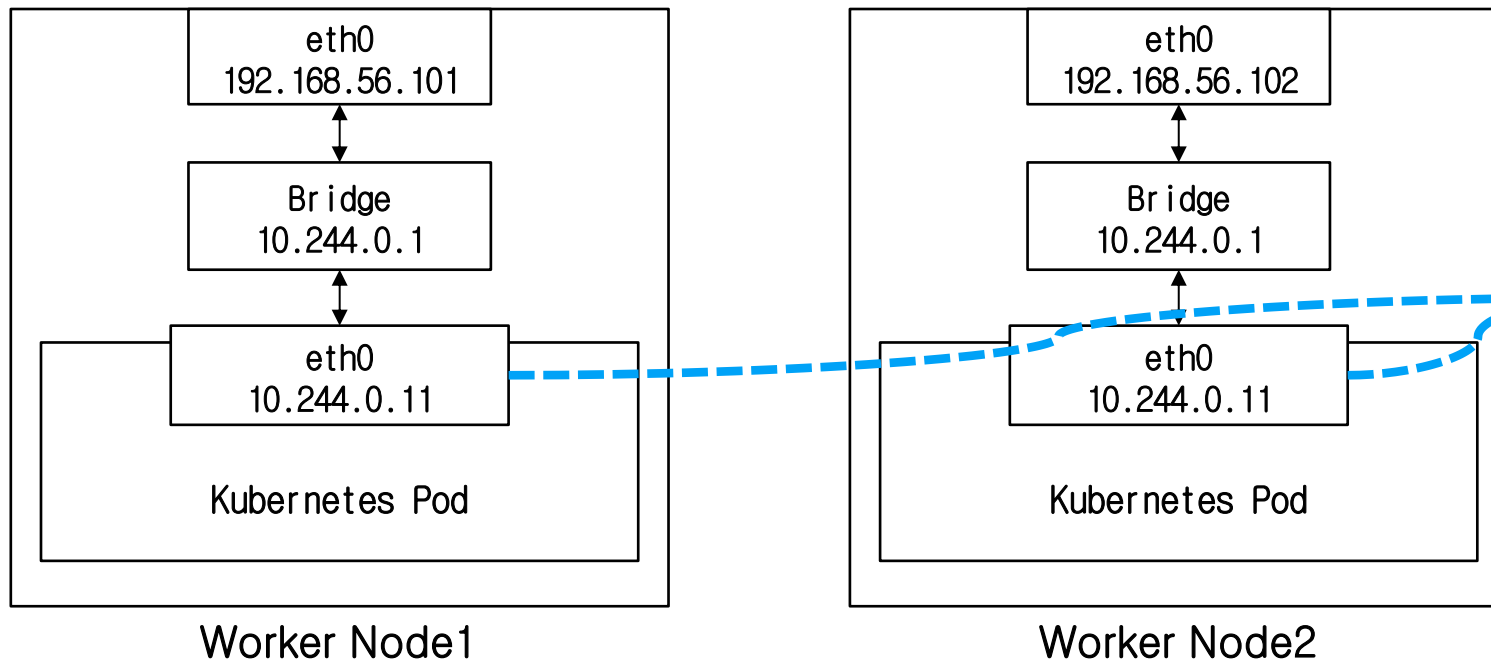
CNI	성능	네트워크 정책	설치 난이도	추가 기능
Calico	 높음 (BGP 사용 가능)	 지원	 중간	BGP, VXLAN, IP-in-IP
Flannel	 낮음 (VXLAN 사용)	 없음	 쉬움	간단한 오버레이 네트워크
Cilium	  매우 높음 (eBPF)	 강력한 정책	 중간~어려움	eBPF, API-aware 정책
Weave Net	 낮음~중간	 지원	 쉬움	암호화 지원

- 초보자, 간단한 네트워크 : Flannel
- 보안, 성능, 확장성 : Calico
- 보안 : Weave Net
- 최고 성능, 최신 기술 : Cilium

4.2 CNI

❖ CNI가 필요한 이유

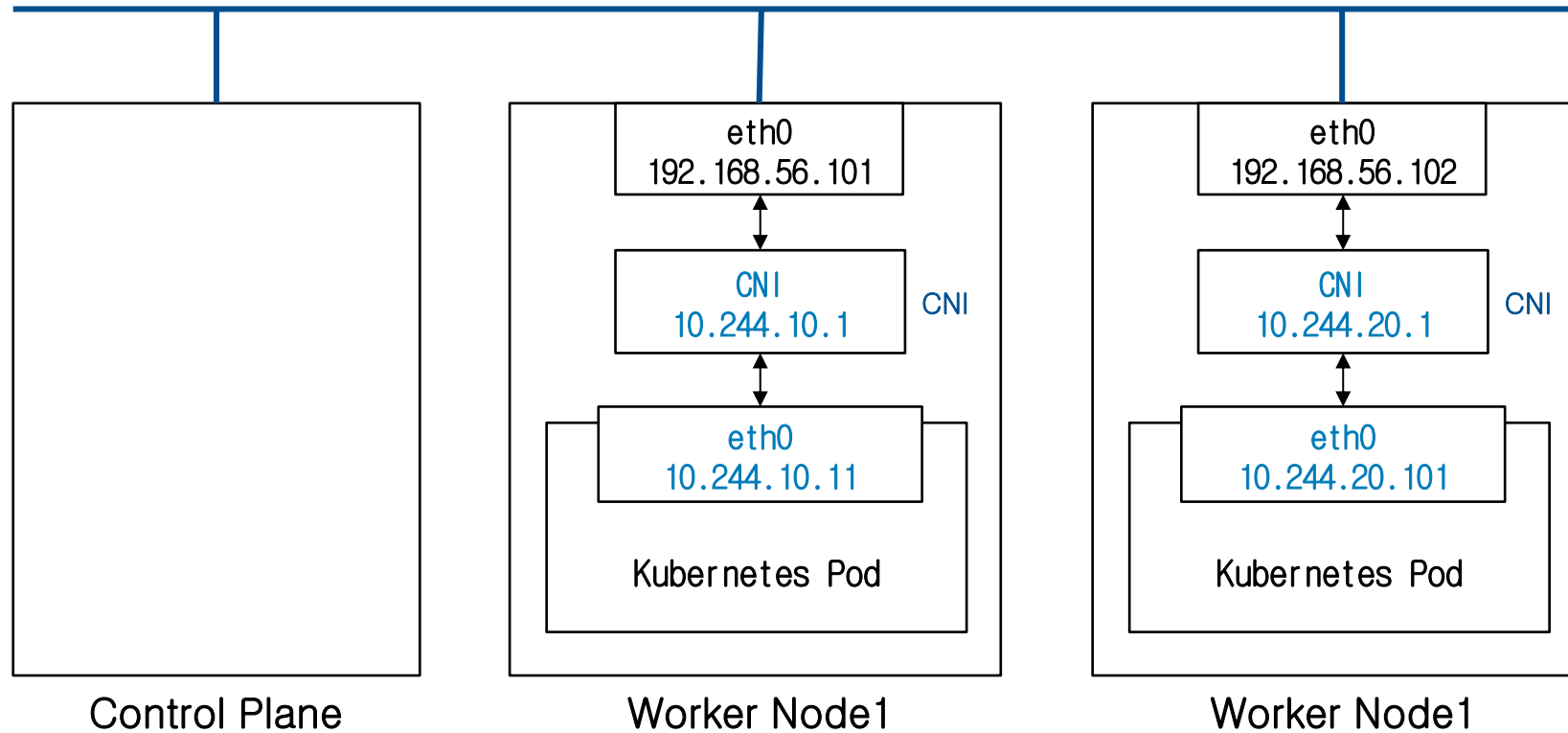
- 서로 다른 노드에서 여러 Pod가 생성될 때 동일한 IP 주소를 가진 Pod가 존재하게 되면 Pod간 통신에 문제가 발생함
- 따라서 CNI를 이용해 Bridge Interface의 네트워크 IP 대역과 라우팅 테이블의 정보를 다르게 설정
 - 서로 다른 노드 상의 Pod끼리 통신이 가능하도록 함
- CNI를 사용하지 않은 경우



- * 중복된 IP
- * 이 문제를 막으려면 클러스터 수준에서 IP 주소를 관리할 수 있는 기능이 필요함

4.2 CNI

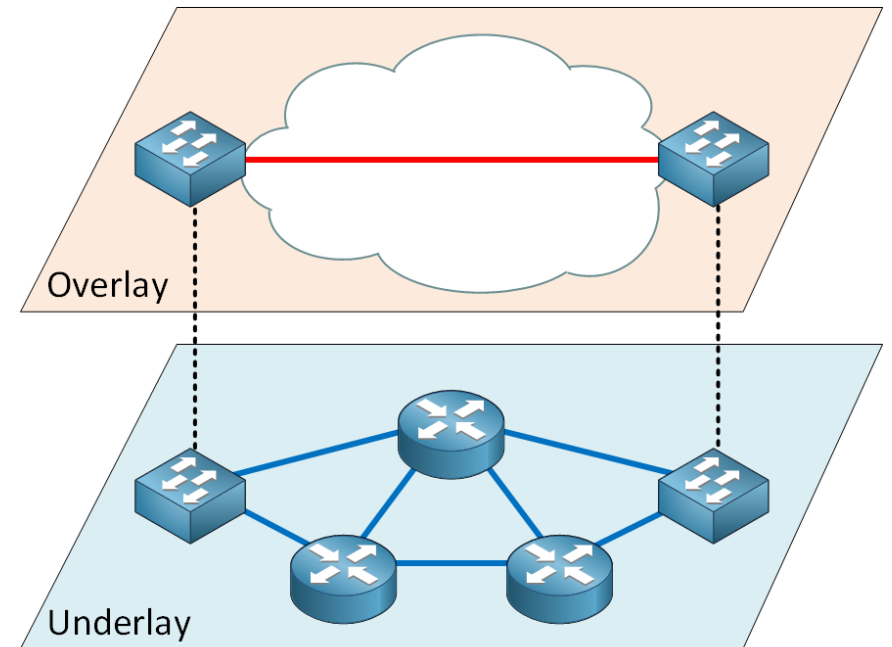
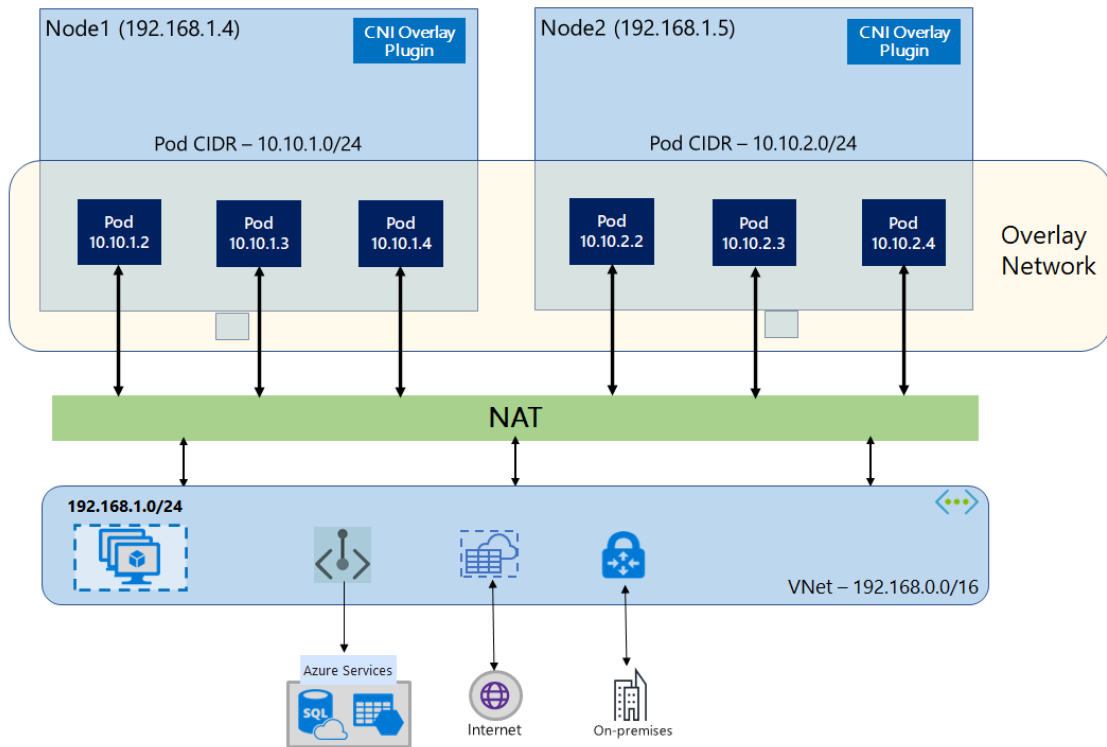
❖CNI를 사용한 경우



4.2 CNI

❖ CNI 플러그인의 네트워크 모델

- 오버레이(Overlay) 네트워크 모델 : VXLAN(Virtual Extensible LAN), IP-in-IP
- 비 오버레이(Non-overlay) 네트워크 모델 : BGP(Border Gateway Protocol)



오버레이 네트워크 개요도 : 참조

– <https://learn.microsoft.com/ko-kr/azure/aks/azure-cni-overlay>

– <https://community.cisco.com/t5/data-center-switches/understanding-underlay-and-overlay-networks/td-p/4295870>

4.2 CNI

❖ 오버레이 네트워크

- Calico(VXLAN/IP-in-IP 모드), Flannel, Cilium(VXLAN 모드), Weave Net
- 장점
 - 클러스터 네트워크와 물리 네트워크를 분리
 - IP 충돌 가능성이 줄어듦
- 단점
 - 터널링으로 인한 패킷의 캡슐화/디캡슐화 과정에서 성능 저하가 발생할 수 있음

❖ 비 오버레이 네트워크의 장단점

- aws-vpc-cni, Calico(BGP 모드), Cilium (Native Routing 모드)
- 장점
 - overlay가 없으므로 터널링으로 인한 패킷 캡슐화가 필요없으므로 네트워크 상의 지연 시간이 줄어듦
- 단점
 - 네트워크 구성의 복잡성 증가
 - Overlay Network보다 네트워크 격리 수준이 낮아서 보안 정책 적용이 더 어려움
 - 물리 네트워크 설정(BGP 라우팅 등)에 따라 제약이 발생함

4.3 Calico CNI

❖ Calico CNI

- kubernetes 클러스터에서 네트워크 정책과 보안 기능을 제공하는 CNI 플러그인
- 특징
 - 다양한 네트워크 연결 방식 지원
 - VXLAN (Overlay) : 네트워크 경로가 정해지지 않은 환경에서 사용하는 가상 네트워크 방식
 - Direct Routing(Non-Overlay) : L3모드. 비오버레이. 각 노드가 직접 라우팅테이블을 구성하여 Pod 통신을 처리함
 - BGP(Non-Overlay)
 - » BGP(Border Gateway Protocol)를 사용하여 클러스터 노드 간에 네이티브 IP 라우팅을 수행
 - » Overlay 네트워크 없이 직접 네트워크 경로를 설정하여 성능이 뛰어남
 - IP-in-IP (Overlay)
 - » 네트워크 노드가 서로 다른 서브넷에 있을 경우, IP-in-IP 터널링을 사용하여 패킷을 전달
 - » BGP 라우팅을 사용할 수 없는 경우 유용하며, 기본 설정에서 활성화
 - 네트워크 정책(Network Policy) 지원
 - Kubernetes 기본 NetworkPolicy API와 완벽하게 호환됨
 - IP 주소 기반, Namespace 기반, Service 기반으로 트래픽을 제어 가능
 - Ingress(수신) 및 Egress(송신) 정책을 모두 적용할 수 있음

4.3 Calico CNI

■ 특징(이어서)

- 두가지 IPAM(IP Address Management) 지원
 - Calico IPAM
 - » Calico 자체 IPAM을 사용하여 Pod IP 주소를 관리
 - » 각 노드에 일정한 블록 크기의 IP를 할당하여 관리 효율성을 높임
 - Kubernetes IPAM
 - » Kubernetes에서 제공하는 기본 IPAM을 사용 가능
 - » Calico의 고급 기능과 함께 사용할 수 없으므로 권장되지 않음
- 서비스 메시와 통합 가능
 - Calico는 Istio, Linkerd와 같은 서비스 메시와 쉽게 통합 가능
 - Application Layer 네트워크 정책 적용 가능
- 로깅 및 모니터링
 - Calico는 네트워크 트래픽을 모니터링하고 로깅할 수 있는 기능을 제공
 - Felix 에이전트가 네트워크 정책을 실행하고 로깅 기능을 제공.
 - Prometheus, Grafana와 통합하여 네트워크 상태를 시각적으로 모니터링 가능.

4.4 AWS VPC CNI

❖ AWS EKS에 기본 탑재되는 CNI

❖ 특징

- 비 오버레이 네트워크 모델
- 각 노드의 IP 대역과 Pod에 할당되는 IP 대역이 동일함
 - VPC(Virtual Private Cloud)에 할당되는 IP 대역의 IP 주소를 Pod에 직접 할당함
 - 이를 위해 각 Worker Node(EC2)에 ENI(Elastic Network Interface: 가상 네트워크 인터페이스)를 연결함
- 네트워크 성능에는 장점이 있지만 VPC의 IP 주소가 빠르게 고갈될 수 있음

❖ 참고

- VPC와 EKS에 대해...
 - AWS에 구성하는 가상의 사설 네트워크
 - AWS에서 EC2와 같은 가상머신을 생성하기 위해 VPC를 생성하고 내부에 가용영역(Availability Zone) 별로 서브넷이라는 네트워크를 구성한 다음 서브넷에 EC2 가상머신을 생성함
 - AWS EKS에서의 Worker Node 역할을 EC2 가상머신이 담당함

4.5 Azure AKS의 CNI

❖ 지원하는 CNI 종류

- Azure CNI
 - 비 오버레이 네트워크 방식
 - Pod에 VNET 의 IP 할당
- Azure CNI Overlay
 - VXLAN 기반의 오버레이 네트워크 방식
 - 약간의 성능 저하 있을 수 있음
- Azure CNI with Calico
 - Azure CNI 또는 Azure CNI Overlay와 Calico를 결합하여 네트워크 정책을 추가하는 네트워크 방식
- Kubenet
 - 기본 Kubernetes 네트워크 방식

4.6 Calico를 이용한 네트워크 보안

❖ 일반적인 방화벽

- Node 수준의 접근 제어
- IP, TCP 포트 정보를 이용한 접근제어

❖ k8s 환경에서 필요한 보안

- Pod 수준의 접근제어
- Label을 이용한 접근 제어
- 이를 위해 Calico Network Policy를 사용할 수 있음

4.6 Calico를 이용한 네트워크 보안

❖ Calico Network 보안 예제 실습

■ 예제 파일 준비

```
# 예제 git clone 후 backend, frontend, test용도의 pod(nodeapp-pod) 준비
git clone https://github.com/stepanowon/calico-policy-sample
cd calico-policy-sample
kubectl apply -f backend-web/k8s-backend.yaml
kubectl apply -f frontend-web/k8s-frontend.yaml
kubectl apply -f backend-web/nodeapp-pod.yaml
```

■ 생성된 리소스 확인

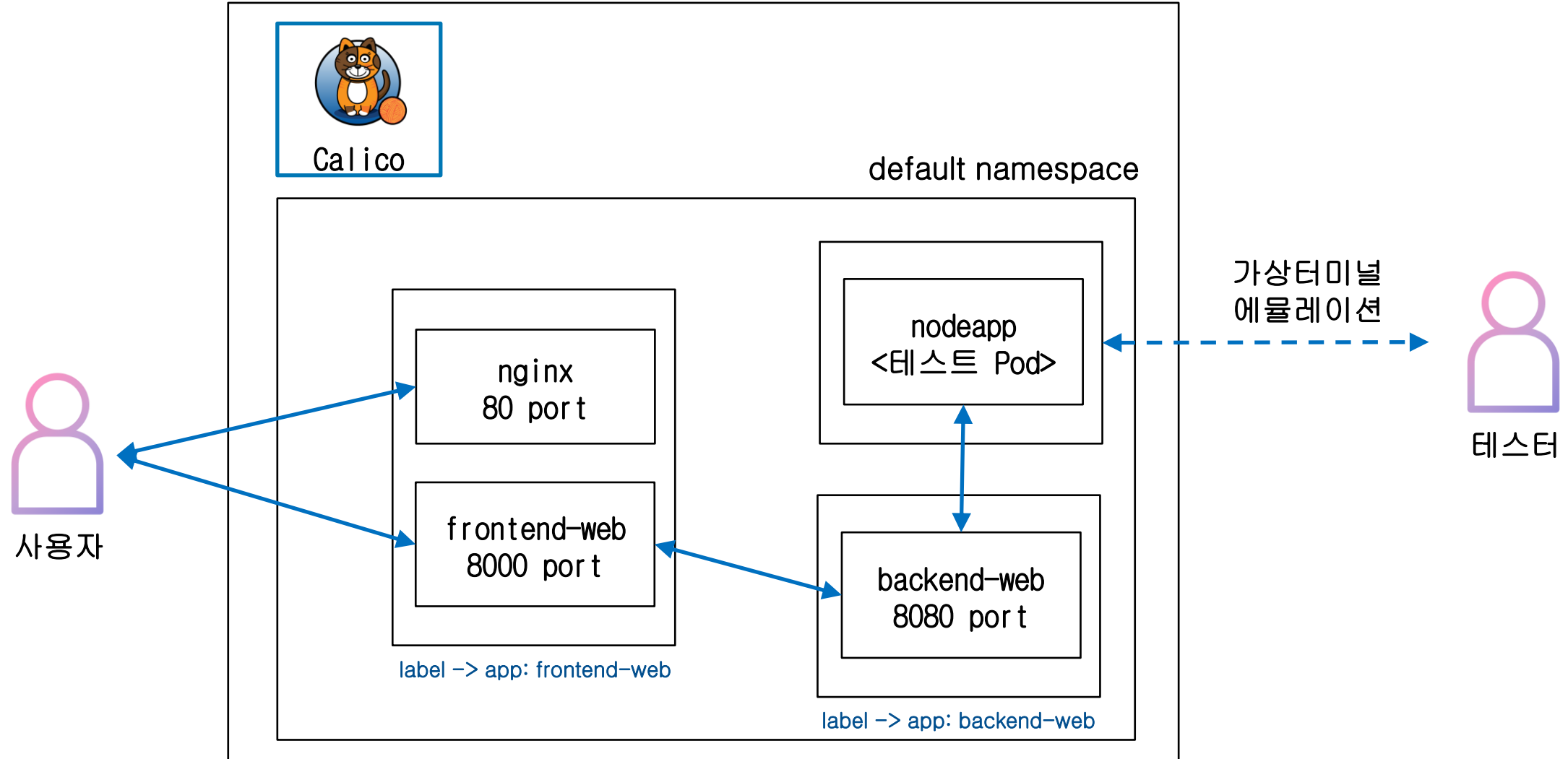
```
$ kubectl get pods,services
```

NAME	READY	STATUS	RESTARTS	AGE
pod/backend-web-68bdf65dc5-wnmcr	1/1	Running	0	49s
pod/frontend-web-644c49c747-xs5lf	2/2	Running	0	49s
pod/nodeapp	1/1	Running	0	48s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/backend-web	ClusterIP	10.100.24.205	<none>	8080/TCP	49s
service/frontend-web	LoadBalancer	10.101.224.240	192.168.56.51	8000:31049/TCP, 80:31392/TCP	49s
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	9d

4.6 Calico를 이용한 네트워크 보안

❖예제 아키텍처



4.6 Calico를 이용한 네트워크 보안

■ 테스트용 Pod에서 backend-web 접근 테스트

```
# nodeapp Pod에 대해 가상 터미널을 에뮬레이션해서 backend-web으로 직접 요청해봄
$ kubectl exec -it nodeapp -- /bin/bash
root@nodeapp:/usr/src/app# curl http://backend-web.default.svc:8080/countries
[{"country":"japan","region":"asia","visited":true}, {"country":"laos","region":"asia","visited":false}, {"country":"france","region":"europe","visited":false}, {"country":"spain","region":"europe","visited":true}, {"country":"egypt","region":"africa","visited":false}, {"country":"morocco","region":"africa","visited":true}]
```

■ frontend-web으로 직접 요청해 봄

```
# 앞서 확인된 리소스에서 LoadBalancer의 External IP 를 이용해 접근
$ curl http://192.168.56.51
.....(생략: nginx 웹서버)

# 백엔드로부터 응답받은 데이터로 HTML 데이터 응답
$ curl http://192.168.56.51:8000
<div>
  <h2>frontend-web sample</h2>
  <hr />
  .....(생략)
</div>
```

4.6 Calico를 이용한 네트워크 보안

❖ all-deny.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: all-deny
  namespace: default
spec:
  podSelector: {}           # 모든 Pod 선택
  policyTypes:              # Ingress, Egress 모두 접근이 불가능함
  - Ingress
  - Egress
```

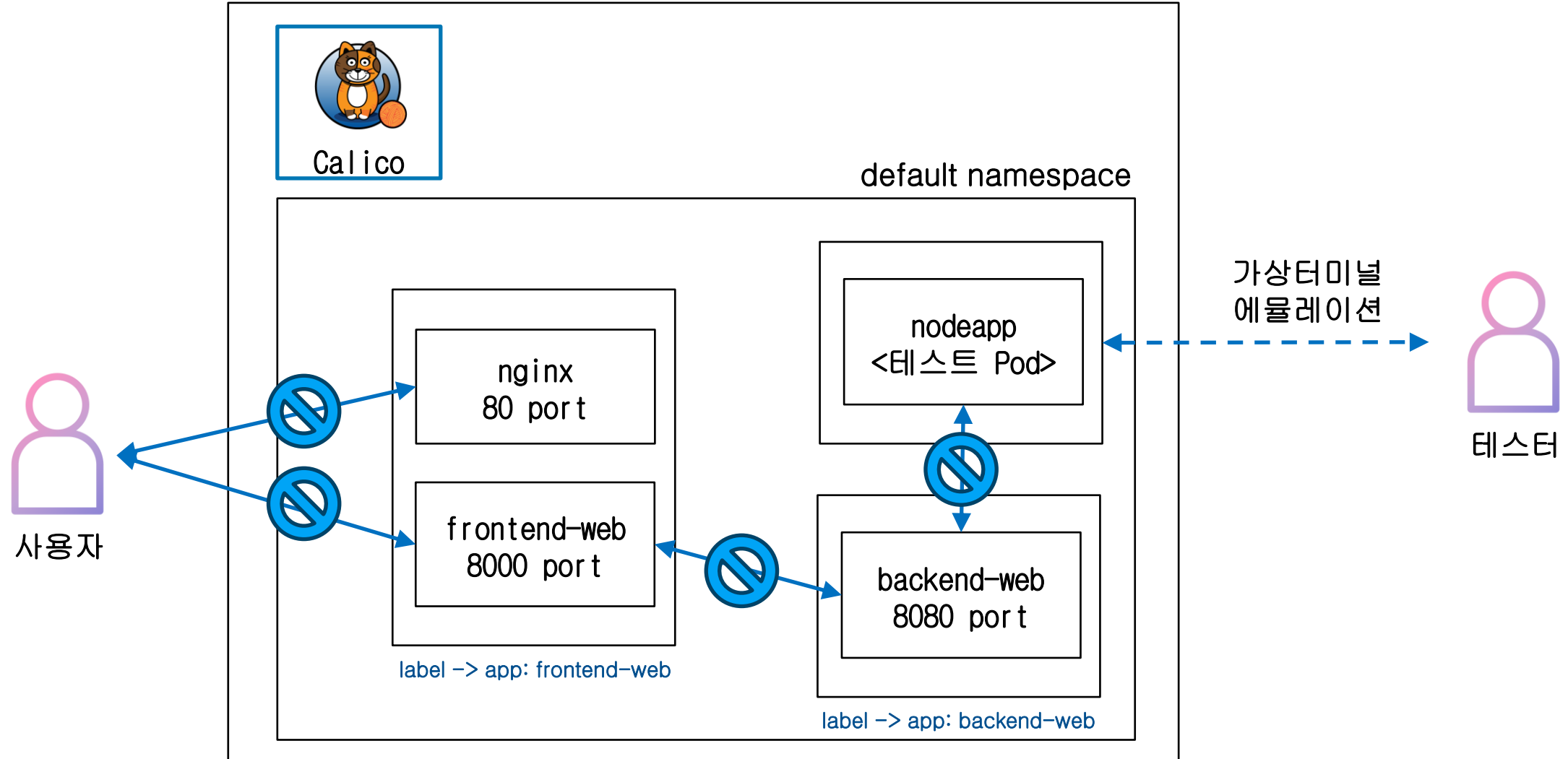
❖ all-deny.yaml 적용

```
# 모두 거부 정책 적용
kubectl apply -f calico-policy/all-deny.yaml

# 테스트 후 반드시 제거할 것
kubectl delete -f calico-policy/all-deny.yaml
```

4.6 Calico를 이용한 네트워크 보안

❖all-deny.yaml 적용 결과



4.6 Calico를 이용한 네트워크 보안

❖ 원하는 네트워크 보안 정책의 기능

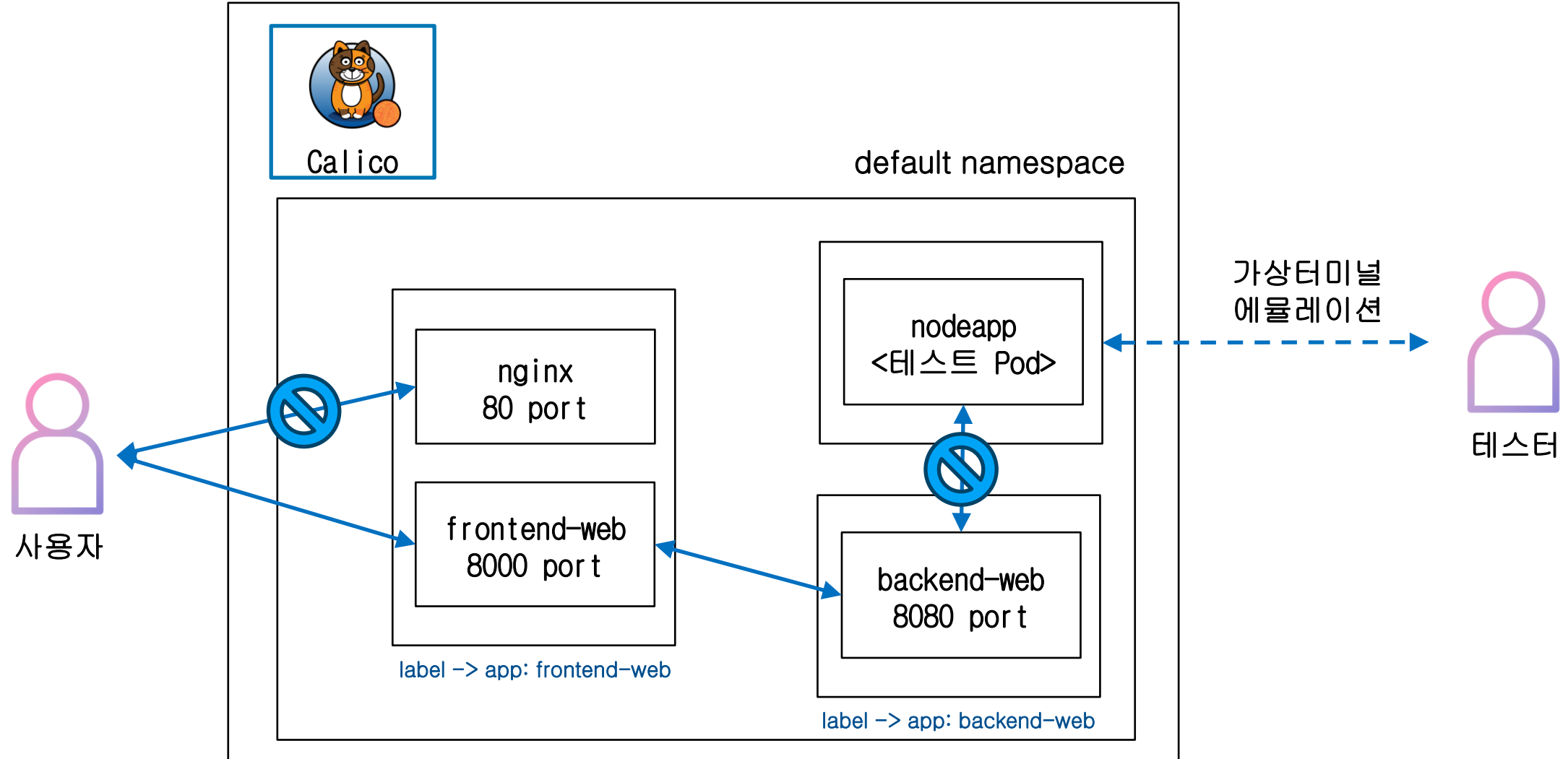
- frontend-web
 - 외부에서 8000 포트만 접근 가능하도록 설정
- backend-web
 - app: frontend-web 레이블을 가진 Pod만 app: backend-web 레이블을 가진 Pod에 접근할 수 있도록 함

```
# kubectl apply -f calico-policy/frontend-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: frontend-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: frontend-web
  ingress:
    - ports:
        - port: 8000
      from: []
```

```
# kubectl apply -f calico-policy/backend-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend-web
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: frontend-web
```

4.6 Calico를 이용한 네트워크 보안

❖ frontend-policy.yaml, backend-policy.yaml 적용 결과



4.6 Calico를 이용한 네트워크 보안

❖테스트 완료 후 리소스 정리

정책 삭제

```
kubectl delete -f calico-policy/backend-policy.yaml  
kubectl delete -f calico-policy/frontend-policy.yaml
```

리소스 삭제

```
kubectl delete -f backend-web/k8s-backend.yaml  
kubectl delete -f backend-web/nodeapp-pod.yaml  
kubectl delete -f frontend-web/k8s-frontend.yaml
```