

Cilium과 Istio Ambient 통합 전략

차세대 서비스 메시 아키텍처



eBPF 기반 네트워킹



Zero-Trust 보안



통합 관측성

2025-08-26

이기하

dasomell@gmail.com

목차

1. 왜 지금 서비스 메시인가?
2. 배경 기술 개요
3. 아키텍처 통합 시나리오
4. 테스트 환경 구성
5. 보안 설계
6. 테스트 App 배포
7. 성능/운영 비용 비교
8. 단계별 도입 로드맵
9. 데모 시나리오
10. 운영 가이드 체크리스트
11. 위험 완화 전략
12. 결론 및 Next Steps

1. 왜 지금 서비스 메시인가?

마이크로서비스의 도전과제

- ⚙️ 복잡성 증가: 서비스 간 통신 경로 기하급수적 증가
- 🛡️ 보안 요구사항: Zero Trust, mTLS, 세밀한 접근 제어
- 📈 관측성 필요: 분산 추적, 메트릭, 로깅의 일관성
- ⚙️ 운영 부담: 라이브러리 업데이트, 설정 관리 복잡도

해결책: eBPF + Ambient Mesh

- ⚡ 성능: 커널 레벨 처리로 오버헤드 최소화
- ☁️ 운영성: 사이드카 없는 투명한 메시
- 🏠 점진적 도입: 네임스페이스/서비스 단위 적용

기존 사이드카 패턴의 한계

- 🚧 리소스 오버헤드: Pod당 프록시 컨테이너 필요
- 🔑 배포 복잡성: 프록시 버전 관리, 설정 동기화
- 🌐 성능 영향: 추가 네트워크 홉, 메모리/CPU 사용량

서비스 메시가 필요한 이유

보안 및 연결성 관리

- 🔒 **Zero Trust 보안**: 모든 서비스 간 통신에 상호 인증 적용
- 🔑 **mTLS 암호화**: 서비스 간 통신 암호화로 데이터 보호
- 🛡️ **접근 제어 정책**: 서비스 ID 기반 세밀한 권한 관리
- 📦 **트래픽 라우팅**: 고급 로드밸런싱, 카나리/블루그린 배포

운영 및 개발 관점

- 📈 **분산 추적**: 서비스 간 호출 흐름 추적 및 병목점 발견
- 📊 **통합 메트릭**: 일관된 방식의 성능/사용률 모니터링
- 🔧 **장애 주입 테스트**: 회복력과 내결함성 검증
- 🔗 **개발 부담 감소**: 네트워킹/보안 로직의 비즈니스 코드 분리

다양한 조직 관점에서의 가치



보안팀

표준화된 인증/인가 정책 적용, 통신 암호화, 보안 감사



운영팀

중앙화된 정책 관리, 이상 감지, 트래픽 제어, 장애 격리



개발팀

비즈니스 로직 집중, 인프라 문제 추상화, 빠른 서비스 개발

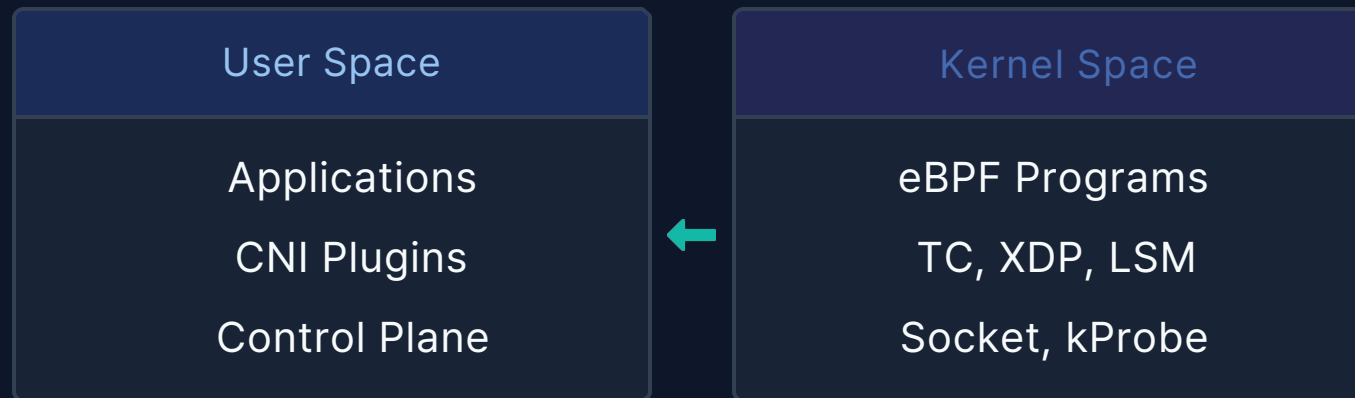


2. 배경 기술 개요

eBPF, Cilium, Istio Ambient의 핵심 원리와 장점



eBPF란?



핵심 장점



커널 수정 없는 확장성

커널 재컴파일 없이 동적 프로그램 로드



통합 플랫폼

네트워크, 보안, 관측성의 통합 지원



고성능 처리

고성능 패킷 처리와 정책 적용



Cilium: eBPF 네트워킹

CNI 확장 기능

- 🛡️ **L3-L7 네트워크 정책:** 보안 정책을 IP, 포트 뿐 아니라 HTTP 경로, DNS 등으로 확장
- 🧰 **IP 관리:** Pod IP 할당, NAT, 라우팅 최적화
- 🔒 **투명한 암호화:** WireGuard 기반 네트워크 트래픽 암호화

kube-proxy 대체

- ⚡ **eBPF 기반 서비스 로드밸런싱:** 더 적은 오버헤드와 레이턴시
- 🔄 **Direct Server Return (DSR):** 반환 트래픽 경로 최적화
- 🔧 **호환성:** 기존 Service/Endpoints 리소스와 완벽 호환

Hubble: 실시간 네트워크 플로우 관측

- 📈 **가시성:** L3-L7 트래픽 플로우 모니터링
- 🔍 **디버깅:** 드롭된 패킷 원인 분석
- 🌐 **API/UI:** 그래픽 서비스 맵, 메트릭 대시 보드

Cilium + Istio Ambient = 커널 레벨 eBPF와 메시 기능의 완벽한 시너지

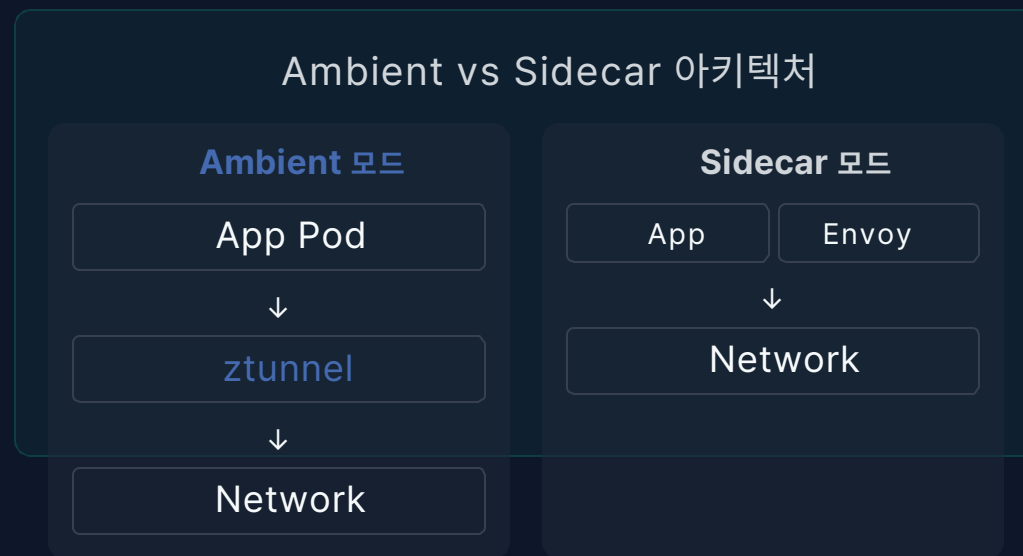
Istio Ambient Mesh

사이드카 없는 새로운 아키텍처

-  **Ztunnel** : 노드별 투명 프록시, mTLS 처리 담당
-  **Waypoint** : 선택적 L7 프록시 (필요시에만 적용)
-  **점진적 적용** : 네임스페이스 라벨 기반 온보딩

장점

-  **리소스 효율성**: Pod마다 사이드카를 배포할 필요 없음
-  **투명한 통합**: 기존 애플리케이션 변경 불필요
-  **운영 단순화**: 중앙집중식 관리 및 업그레이드



점진적 도입 방법

```
# 네임스페이스에 Ambient 모드 활성화
kubectl label namespace production \ istio.io/dataplane-mode=ambient
```

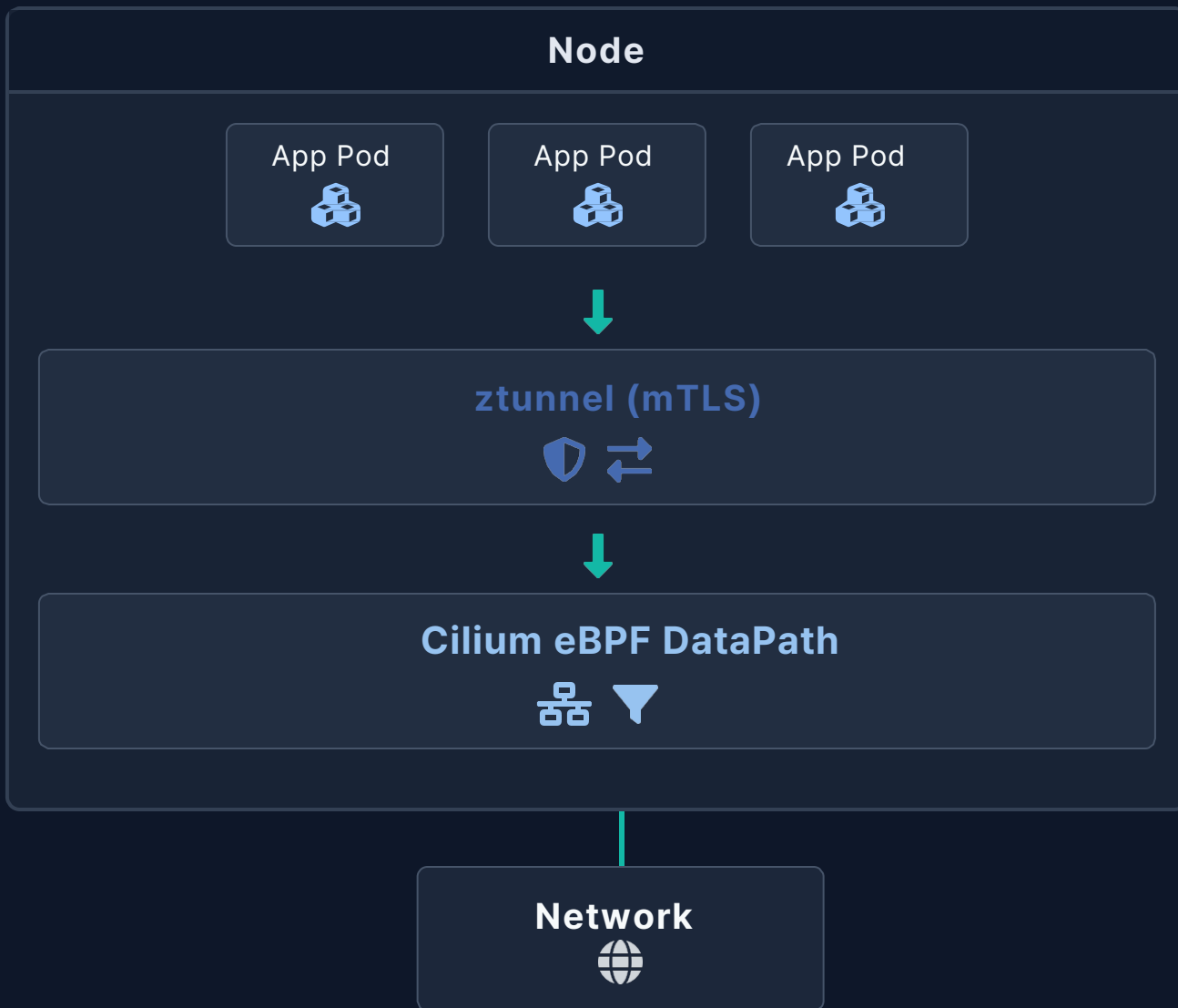
```
# 필요시 L7 처리를 위한 Waypoint 추가
kubectl apply -f gateway.yaml
```

3. 아키텍처 통합 시나리오

목표 아키텍처와 데이터 경로 협업, 정책 계층화 소개



목표 아키텍처 다이어그램



주요 구성요소

 **App Pod**
사이드카 없는 애플리케이션 컨테이너, 투명한 메시 연결

 **ztunnel**
노드별 프록시, mTLS 처리 및 L4 보안 정책 적용

 **Cilium eBPF**
커널 레벨 네트워킹, L3/L4 처리 및 관측성 제공

데이터 경로 협업 & 정책 계층화

데이터 경로 협업

-  **L3-L4 Cilium eBPF**가 고성능 패킷 처리
-  **mTLS ztunnel**이 투명하게 암호화/복호화
-  **L7 Waypoint**가 필요한 서비스만 처리 (선택적)
-  **관측성 Hubble + Istio 텔레메트리** 결합

통합 시나리오 이점

-  **고성능:** eBPF 기반 패킷 처리로 지연 시간 최소화
-  **계층화된 방어:** 네트워크 레벨부터 애플리케이션 레벨까지
-  **통합 관측성:** 네트워크와 서비스 수준의 완전한 가시성

정책 계층화 전략

-  **Cilium 정책:** 네트워크 경계, IP/포트 기반 제어
네임스페이스 간 통신, 포드 레벨 네트워크 격리, TCP/UDP 포트 제한
-  **Istio 정책:** 서비스 ID 기반, L7 라우팅/인증
서비스 간 인증/인가, 경로 기반 라우팅, 헤더/페이로드 검증

4. 테스트 환경 구성

인프라 사항, 핵심 설치 순서, Pod Security 고려사항



테스트 클러스터 및 네트워킹 구성

클러스터 인프라 사양

- 컨트롤 플레인: 3노드 (HA 구성)
- 워커 노드: 3노드
- OS: Ubuntu 24.04 LTS
- 컨테이너 런타임: Cilium v1.17.6
- K8s 배포: RKE2 v1.32.7-rc2+rke2r1

네트워킹 아키텍처

- CNI: Cilium (kube-proxy 대체)
- Service Mesh: Istio Ambient
- Gateway: Gateway API + Istio
- 관측성: Prometheus, Grafana

핵심 설치 순서



1. 기본 클러스터
RKE2 + CIS 프로파일



2. Cilium 설치
kube-proxy 대체, Hubble
활성화



3. Gateway API
표준 CRD 배포



4. Istio Ambient
istiod, CNI, ztunnel



5. 관측성
Prometheus, Grafana

설치 프로세스

VM 설치

1 S/W 설치 및 VM 생성(rke2 설치)

shell 명령어

Shell

```
# Install Vagrant
brew install vagrant
vagrant plugin install vagrant-vbguest

# Install Virtualbox
brew install --caks virtualbox

# VM install
vagrant up
```

2 Istio 설치

VM(master1)에 shell 명령어

Shell

```
# VM(master1) 접속
vagrant ssh master1

# Install istio
vagrant@master01:~$ /vagrant/scripts/install_istio_ambient.sh
install
```

각종 컴포넌트 설치

1 MetalLB

Helm 명령어

Shell

```
# create namespace
kubectl create ns metallb-system

# pod security setting
kubectl label --overwrite ns metallb-system \
  pod-security.kubernetes.io/enforce=privileged \
  pod-security.kubernetes.io/audit=privileged \
  pod-security.kubernetes.io/warn=privileged

# helm repo
helm repo add metallb https://metallb.github.io/metallb
helm repo update metallb

# helm install
helm upgrade --install metallb metallb/metallb --version 0.15.2 -f ./scripts/helm/metallb-0.15.2.yaml -n metallb-system --wait

# expose ip setting
kubectl apply -f ./scripts/helm/metallb/IPAddressPool.yaml
kubectl apply -f ./scripts/helm/metallb/L2Advertisement.yaml
```

설치 프로세스

각종 컴포넌트 설치

2 NFS-Server 4.0.2

Helm 명령어

Shell

```
# helm repo
helm repo add nfs-subdir-external-provisioner https://kubernetes-
sigs.github.io/nfs-subdir-external-provisioner
helm repo update nfs-subdir-external-provisioner

# helm install
helm upgrade --install nfs-provisioner nfs-subdir-external-
provisioner/nfs-subdir-external-provisioner --version 4.0.18 -
f ./scripts/helm/nfs-subdir-external-provisioner-4.0.2.yaml -n kube-
system
```

3 Istio Gateway 1.27.0

Helm 명령어

Shell

```
# helm repo
helm repo add istio https://istio-
release.storage.googleapis.com/charts
helm repo update istio

# helm install
helm upgrade --install istio-ingress istio/gateway --version 1.27.0 -
f ./scripts/helm/istio-gateway-1.27.0.yaml -n istio-system --wait --
skip-schema-validation

# hubble expose
kubectl apply -f ./network/hubble-expose.yaml
```

설치 프로세스

각종 컴포넌트 설치

4 Kiali 2.14.0

Helm 명령어

Shell

```
# helm repo
helm repo add kiali https://kiali.org/helm-charts
helm repo update kiali

# helm install
helm upgrade --install kiali-server kiali/kiali-server --version 2.14.0 -f ./scripts/helm/kiali-server-2.14.0.yaml -n istio-system --wait

# kiali expose
kubectl apply -f ./network/kiali-expose.yaml
```

5 Grafana 12.1.0

Helm 명령어

Shell

```
# helm repo
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update grafana

# helm install
helm upgrade --install grafana grafana/grafana --version 9.3.2 -f ./scripts/helm/grafana-12.1.0.yaml -n monitoring --create-namespace

# grafana expose
kubectl apply -f ./network/grafana-expose.yaml
```

6 Prometheus 3.5.0

Helm 명령어

Shell

```
# create namespace
kubectl create ns node-metrics

# pod security setting
kubectl label --overwrite ns node-metrics \
  pod-security.kubernetes.io/enforce=privileged \
  pod-security.kubernetes.io/audit=privileged \
  pod-security.kubernetes.io/warn=privileged

# helm repo
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update prometheus-community

# helm install
helm upgrade --install prometheus prometheus-community/prometheus --version 27.31.0 -n monitoring --create-namespace -f ./scripts/helm/prometheus-3.5.0.yaml

# expose ip setting
kubectl apply -f ./network/prometheus-network-policy.yaml
```

Pod Security

네임스페이스별 PodSecurity

1 시스템 네임스페이스 설정

kubectl 명령어

YAML

```
# 시스템 네임스페이스
kubectl label ns kube-system \
  pod-security.kubernetes.io/enforce=privileged

kubectl label ns istio-system \
  pod-security.kubernetes.io/enforce=privileged
```

2 일반 애플리케이션 네임스페이스

kubectl 명령어

YAML

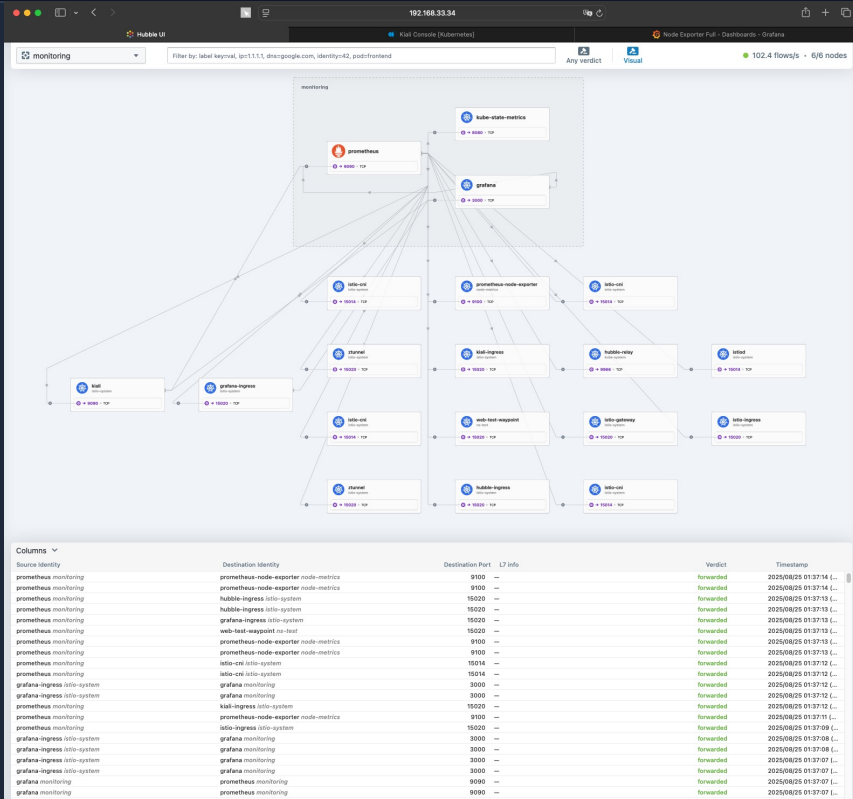
```
# 일반 애플리케이션 네임스페이스
kubectl label ns ns-test \
  pod-security.kubernetes.io/enforce=restricted
```

레벨	목적/성격	주요 허용 사항	주요 금지-제한	대표 보안 설정 예
Privileged	가장 완화됨(거의 무제한). 시스템/인프라 파드 등 특수 목적	privileged 컨테이너, hostNetwork/hostPID/hostIPC, hostPath 등 고위험 옵션 사용 가능	제한 거의 없음(운영 정책으로만 통제)	필요 시에만 사용, 별도 네임스페이스로 격리 권장
Baseline	일반 앱용 최소 보안. 알려진 권한 상승 방지	일반적인 컨테이너 실행, 최소 Capabilities	privileged, hostNetwork/hostPID/hostIPC, hostPath, NET_RAW 등 위험 Capabilities	allowPrivilegeEscalation: false 권장, Capabilities 최소화, 불가피한 예외만 허용
Restricted (기본 적용)	가장 엄격. 강한 격리와 하드닝	비루트 실행, 읽기 전용 루트FS, 최소 Capabilities	권한 상승, 특권 모드, 호스트 네임스페이스-hostPath 등 대부분 금지	runAsNonRoot: true, allowPrivilegeEscalation: false, readOnlyRootFilesystem: true, capabilities: drop=ALL, seccompProfile: RuntimeDefault

Hubble UI

Cilium CNI가 수집한 eBPF 기반 네트워크/보안 플로우를 시각화하는 웹 대시보드

Cilium(eBPF) 기반으로 "사이드카 없이" 커널 수준 플로우·정책 판정을 직접 관찰하는 것이 강점



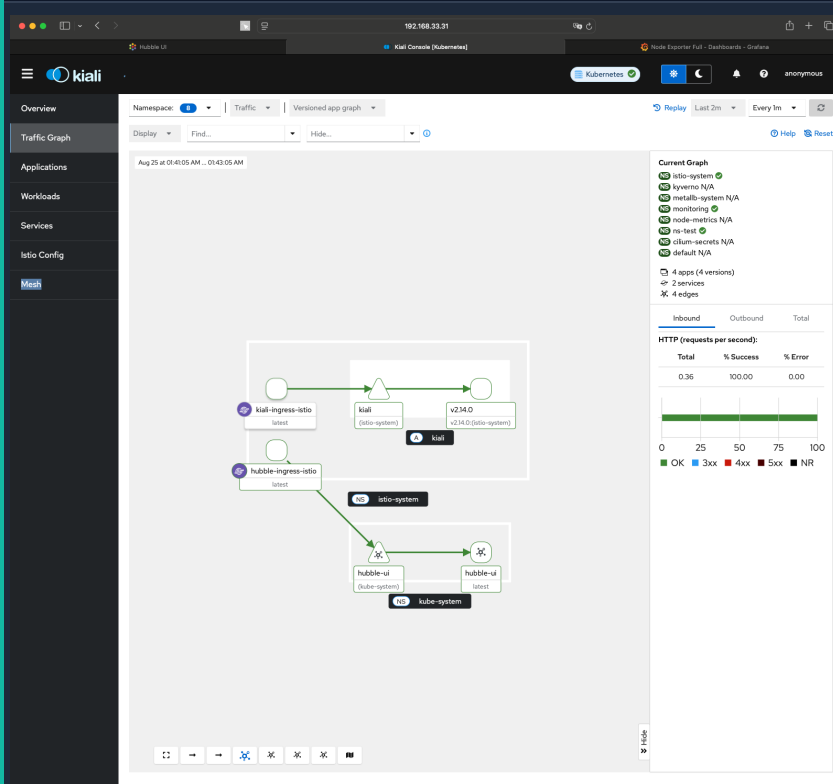
주요 기능

- 서비스 토폴로지 맵: 네임스페이스/서비스/파드 간 연결 관계와 트래픽 방향을 시각화
- 플로우 탐색기: 소스·대상, 포트/프로토콜, 정책 판정(allow/drop), 이유, RTT, 바이트/패킷 수 등 상세 이벤트를 실시간/과거 범위로 조회
- L7 가시성(옵션): HTTP/gRPC/DNS 등 선택적 L7 정보(메서드, 상태코드, 쿼리 등) 표시
- 정책 디버깅: CiliumNetworkPolicy 적용 결과와 드롭 사유(정책 미일치, 룰 충돌, DNS 실패 등) 파악
- 필터링/검색: 라벨, 네임스페이스, IP/포트, Verdict(allow/drop), 주제어(HTTP path 등)로 빠르게 좁혀보기
- 멀티클러스터(Cluster Mesh) 지원: 클러스터 간 흐름을 단일 화면에서 관찰
- 내보내기: 선택한 플로우를 JSON 등으로 export하여 외부 분석 도구와 연계

Kiali

Istio 서비스 메시를 시각화·모니터링·운영하기 위한 오픈소스 웹 대시보드

메시 토폴로지, 트래픽, 에러율, 정책 상태를 한눈에 보여 주고 설정 검증과 디버깅 도움



핵심 기능

- 서비스 그래프: 네임스페이스/서비스/워크로드 간 호출 관계, 트래픽 양, 지연, 오류율을 방향 그래프로 시각화
- 헬스/알람: 서비스·워크로드 상태와 4xx/5xx 변화에 따른 색상 표시, 오류 지점 식별
- 트래픽 분석: 요청 비율, 지연(P90/P99), 실패율, 리트라이/미러링/카나리 등 정책 적용 효과 확인
- 설정 검증: VirtualService, DestinationRule 등 Istio 리소스의 유효성 검사와 충돌/누락 감지
- 분산 추적 연계: Jaeger/Tempo 등과 연결해 호출 단위 트레이스 확인
- 멀티클러스터/멀티네임스페이스 보기, 사이드카 주입 상태·버전 매트릭 뷰, 네임스페이스별 권한 제어

5. 보안 설계

mTLS 단계적 적용 및 정책 계층화 전략, 인증서 관리



mTLS 및 정책 계층화

mTLS 단계적 적용

1 Phase 1: Permissive Mode

PeerAuthentication

YAML

```
# Phase 1: Permissive Mode
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: production
spec:
  mtls:
    mode: PERMISSIVE
```

2 Phase 2: Strict Mode (점진적 전환)

PeerAuthentication

YAML

```
# Phase 2: Strict Mode
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: production
spec:
  mtls:
    mode: STRICT
```

mTLS 및 정책 계층화

정책 계층화 전략

1 Cilium: 네트워크 경계 제어

CiliumNetworkPolicy

YAML

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: deny-cross-namespace
spec:
  endpointSelector: {}
  ingress:
    - fromEndpoints:
    - matchLabels:
        k8s:io.kubernetes.pod.namespace: production
```

2 Istio: 서비스 레벨 인가

AuthorizationPolicy

YAML

```
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: frontend-policy
spec:
  selector:
    matchLabels:
      app: frontend
  rules:
    - from:
      - source:
          principals: ["cluster.local/ns/production/sa/gateway"]
```

정책 적용 워크플로우

1. Cilium L3/L4

네트워크 경계 및 IP/포트 기반 정책



2. mTLS (ztunnel)

통신 채널 암호화/인증



3. Istio L7

서비스 ID 기반 라우팅/인증

인증서 및 신뢰 체계 관리

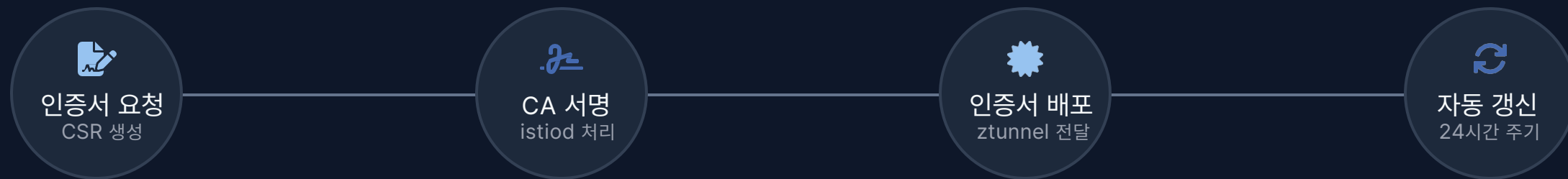
Root CA 관리 옵션

- 🔒 자체 CA 구성: Istio 내장 CA 또는 Vault 연동 가능
- ↔ 외부 CA 연동: 기업 PKI 인프라와 통합 구성
- 🛡 키 보호: Root 키는 HSM으로 보호 권장

SAN 검증 원칙

- ✅ FQDN 기반 검증: 서비스 FQDN 포함 필수
- 🏠 node-ip 설정: 모든 노드 IP 주소 SAN에 포함
- 🔗 Cross-cluster: 클러스터 간 신뢰 체인 구성

Workload 인증서 발급 및 갱신 프로세스



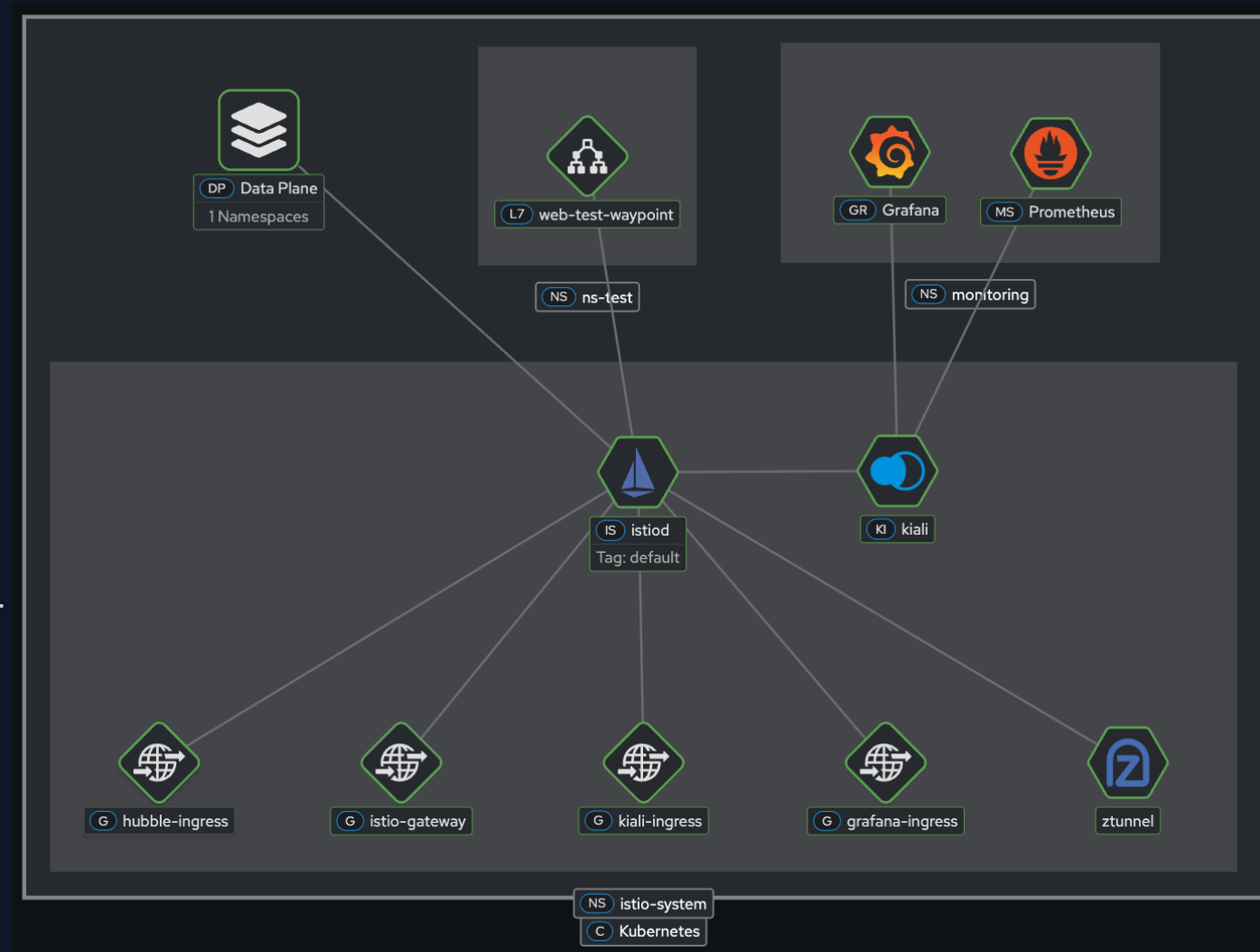
신뢰 모델 고려사항

- 🔑 키 순환: Root CA 키 주기적 교체
- 🏢 다중 클러스터: 공통 Root 또는 상호 신뢰
- 🏠 Spiffe 통합: 워크로드 ID 표준화



6. 테스트 App 배포

Istio Ambient(ztunnel + Waypoint)와
Kubernetes Gateway API
Cilium L7/FQDN 네트워크 정책을 함께
사용해 L4/L7 보안과 트래픽 제어를 구현



테스트 app 파일 구성 설명

네임스페이스/Waypoint/하드닝 규칙

네임스페이스/컨트롤면 리소스

- namespace.yaml
ns-test 네임스페이스 생성
- waypoint.yaml
네임스페이스 라벨과 합쳐서 ns-test 내 워크로드의 L7 정책 집행 지점
- cluster-policy.yaml (Kyverno)
Waypoint(관리형 Gateway) 파드에 보안 하드닝 강제

인그레스(Gateway/HTTPRoute)

데이터면 리소스

- gateway.yaml
istio-system 네임스페이스에 Istio Ingress Gateway(Gateway API) 리스너 설정(HTTP/80, hostname: test.test)
- http-route.yaml
hostnames: test.test, PathPrefix "/"를 web-test-service:80으로 라우팅
외부 → Ingress Gateway → HTTPRoute → Service → Ambient(HBONE/mTLS) → Waypoint(L7 인가) → 파드

테스트 app 파일 구성 설명

보안 정책(Istio/Cilium/K8s)

정책

- **peer-authentication.yaml**
Ambient 구간에서 mTLS(STRICT) 강제. Waypoint/HBONE/ztunnel 경로만 허용되며, 평문 직접 통신은 차단
- **network-policy.yaml**
외부에서 NodePort 등으로 직접 접근은 차단되고, Istio Ingress/ztunnel 경유 트래픽만 통과
- **cilium-network-policy.yaml**
Waypoint(관리형 Gateway) 파드에 보안 하드닝 강제
- **waypoint-authz.yaml**
Waypoint에서 서비스 단위 L7 인가: HTTP 메서드 "GET"만 허용

앱 배포(Service/Deployment)

애플리케이션

- **service.yaml**
web-test-service(Service, NodePort 30000 → targetPort 8080)
- **deployment.yaml**
ns-test에 nginxinc/nginx-unprivileged:latest 컨테이너(비루트)로 web-test 배포

테스트 결과

외부 엔드포인트 설정·접근 검증

LoadBalancer IP 추출 및 GET / 200 확인

```
INGRESS_IP=$(kubectl -n istio-system get svc -l gateway.networking.k8s.io/gateway-name=istio-gateway -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')  
# 만약 NodePort 사용 시: NODE_IP=<노드IP>; INGRESS_PORT=<ingress svc의 NodePort>; URL은 http://$NODE_IP:$INGRESS_PORT
```

```
# 1) 허용 케이스: GET /  
curl -i -H "Host: test.test" http://$INGRESS_IP/
```

#실행결과(기대: 200 OK (nginx 기본 페이지))

```
m@Macmini source % INGRESS_IP=$(kubectl -n istio-system get svc -l gateway.networking.k8s.io/gateway-name=istio-gateway -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')
```

```
m@Macmini source % curl -i -H "Host: test.test" http://$INGRESS_IP/
```

```
HTTP/1.1 200 OK  
server: istio-envoy  
date: Sun, 24 Aug 2025 17:33:09 GMT  
content-type: text/html  
content-length: 615  
last-modified: Wed, 13 Aug 2025 14:33:41 GMT  
etag: "689ca245-267"  
accept-ranges: bytes  
x-envoy-upstream-service-time: 14  
  
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome to nginx</title>  
<style>  
html { color-scheme: light dark; }  
body { width: 35em; margin: 0 auto; font-family: Tahoma, Verdana, Arial, sans-serif; }  
</style>  
</head>  
<body>  
<h1>Welcome to nginx!</h1>  
<p>If you see this page, the nginx web server is successfully installed and working. Further configuration is required.</p>  
  
<p>For online documentation and support please refer to  
<a href="http://nginx.org/">http://nginx.org/</a>.<br/>  
Commercial support is available at  
<a href="http://nginx.com/">http://nginx.com/</a>.</p>  
  
<p>em>Thank you for using nginx.</em></p>  
</body>  
</html>
```

```
m@Macmini source % INGRESS_IP=$(kubectl -n istio-system get svc -l gateway.networking.k8s.io/gateway-name=istio-gateway -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')
```

```
m@Macmini source % curl -i -H "Host: test.test" http://$INGRESS_IP/
```

```
HTTP/1.1 200 OK  
server: istio-envoy  
date: Sun, 24 Aug 2025 17:33:09 GMT  
content-type: text/html  
content-length: 615  
last-modified: Wed, 13 Aug 2025 14:33:41 GMT  
etag: "689ca245-267"  
accept-ranges: bytes  
x-envoy-upstream-service-time: 14
```

테스트 결과

메서드 차단 검증

POST / 요청 거부 확인(정책 기반)

2) 거부 케이스: POST / (기대: 403/차단 (Waypoint AuthorizationPolicy가 GET만 허용, Cilium도 POST 차단))

```
curl -i -X POST -H "Host: test.test" http://$INGRESS_IP/
```

```
m@Macmini source % curl -i -X POST -H "Host: test.test" http://$INGRESS_IP/
```

```
HTTP/1.1 405 Method Not Allowed
```

```
server: istio-envoy
```

```
date: Sun, 24 Aug 2025 17:36:16 GMT
```

```
content-type: text/html
```

```
content-length: 157
```

```
x-envoy-upstream-service-time: 4
```

```
<html>
```

```
<head><title>405 Not Allowed</title></head>
```

```
<body>
```

```
<center><h1>405 Not Allowed</h1></center>
```

```
<hr><center>nginx/1.29.1</center>
```

```
</body>
```

```
</html>
```

테스트 결과

경로 차단 검증

GET /foo 요청 403 확인(Cilium L7 경로 정책)

3) 거부 케이스: GET /foo (기대: 403/차단 (Cilium L7 정책이 path "^/\$" 외에는 거부))
curl -i -H "Host: test.test" http://\$INGRESS_IP/foo

```
m@Macmini source % curl -i -H "Host: test.test" http://$INGRESS_IP/foo
HTTP/1.1 403 Forbidden
content-length: 19
content-type: text/plain
date: Sun, 24 Aug 2025 20:15:11 GMT
server: istio-envoy
x-envoy-upstream-service-time: 2

RBAC: access denied%
```

테스트 결과

Egress 제어 검증

FQDN 기반 허용/차단(httpbin.org만 허용)

파드 내에서:

1) 허용 케이스: https://httpbin.org/get(기대: 200/3xx 응답 정상)

curl -I https://httpbin.org/get

2) 거부 케이스: https://example.com(# 기대: 연결 실패 또는 차단 (Cilium FQDN 정책))

curl -I https://example.com

```
/home/curl_user $ curl -I https://httpbin.org/get
HTTP/2 200
date: Sun, 24 Aug 2025 21:25:41 GMT
content-type: application/json
content-length: 256
server: gunicorn/19.9.0
access-control-allow-origin: *
access-control-allow-credentials: true

/home/curl_user $ curl -I https://example.com
curl: (35) Recv failure: Connection reset by peer
```

성능 및 운영비용 비교

7. 성능 및 운영비용 비교

사이드카 패턴 대비 Ambient 패턴 성능, 리소스, 운영 비교



리소스 사용량 절감



운영 복잡성 감소 성능



벤치마크 개선

리소스 및 성능 비교

리소스 사용량 비교

조건: 1KB HTTP/1.1, 1,000 rps, 2 worker

프록시	CPU(vCPU)	메모리(MB)
Sidecar(Envoy)	0.20	60
Waypoint(Envoy)	0.25	60
ztunnel	0.06	12

출처: <https://istio.io/latest/docs/ops/deployment/performance-and-scalability/>

ztunnel vs sidecar →

CPU 약 -70%(0.06/0.20), 메모리 약 -80%(12/60)

성능: p99 지연(ms) 비교(mTLS 사용)

CNI	Sidecar	Ambient(ztunnel)	Ambient 개선율
Kube CNI	4.72	3.60	23.7%↓ ((4.72-3.60)/4.72)
Cilium CNI	8.80	6.80	22.7%↓ ((8.80-6.80)/8.80)

출처: <https://dzone.com/articles/istio-ambient-mesh-performance-test-and-benchmarking>

TIP: 대규모 클러스터일수록 Ambient 방식의 리소스 절감과 운영 단순화 효과가 더욱 커집니다

성능: mTLS 오버헤드(지연 증가, baseline=no mesh)

항목	Sidecar(Envoy)	Ambient(ztunnel)
지연 증가율	+166%	+8%

출처: <https://arxiv.org/html/2411.02267v1>

운영 복잡성 감소

항목	운영 이점
배포	사이드카 주입 설정 불필요, 워크로드 재시작 없음
업그레이드	노드별 ztunnel만 업데이트, 서비스 무중단
디버깅	중앙집중식 로깅 및 메트릭, 일관된 관측성
확장성	Pod 수에 무관한 프록시 리소스 사용

8. 단계별 도입 로드맵

도입 단계별 주요 Task 및 검증 포인트/일정

도입 단계별 로드맵

Phase 1

2주

기반 구축

기초 네트워킹 기반

- RKE2 설치
K8s 클러스터 초기화 및 CIS 보안 프로파일 적용
- Cilium 배포
kube-proxy 대체 모드로 설치, Hubble 활성화
- 관측성 검증
Hubble UI에서 플로우 가시성 확인, 모든 노드 정상 동작

Phase 2

1주

Ambient 도입

비프로덕션 검증

- Gateway API 배포
표준 GatewayClass/HTTPRoute CRD 설치
- Istio 설치
테스트 네임스페이스에 ambient 라벨링, PERMISSIVE mTLS 설정
- 무중단 검증
기존 애플리케이션 무중단 동작, mTLS 암호화 플로우 확인

도입 단계별 로드맵

Phase 3

4주

점진적 확장

프로덕션 온보딩

- 핵심 서비스 전환
주요 서비스부터 Ambient 라벨링, 단계적 적용
- Waypoint 및 L7 정책
필요한 서비스에 Waypoint 배포, 고급 라우팅 정책 적용
- mTLS STRICT 전환
99.9% 가용성 유지하며 전체 서비스 Strict 모드 적용

Phase 4

2주

전면 적용

운영 메트릭 목표 달성

- 전체 클러스터 적용
남은 모든 네임스페이스 Ambient 전환 완료
- 통합 관측성 및 정책
전사 표준 정책 적용, 중앙 관측성 대시보드 구축
- 운영 메트릭 검증
리소스 사용량, 성능, 가용성 목표치 달성 확인

9~11. 데모, 운영, 리스크

데모 시나리오

시나리오 1: Ambient 적용 전후 비교

```
kubectl label ns demo
istio.io/dataplane-mode=ambient
```

Hubble을 통한 mTLS 암호화 플로우 확인

시나리오 2: L7 정책 적용

Waypoint Gateway 생성 및 HTTPRoute 규칙 구성

백엔드 v2 서비스로 특정 경로 라우팅

시나리오 3: 보안 정책

서비스 계정 기반 AuthorizationPolicy 적용
HTTP 메소드 및 경로별 접근 제어

운영 가이드 체크리스트

설치 전 점검사항

- 모든 노드 시간 동기화 (NTP)
- 클러스터 DNS 해상도 검증
- 이미지 레지스트리 접근성
- 필수 포트 오픈 (6443, 15010-15014)

구성 일관성 점검

- 노드 IP 설정 일관성 확인
- TLS SAN 필드 검증 정책
- 충돌 사전 감지

변경 관리 및 복구

단계적 라벨링 및 롤백 계획 준비 etcd
및 설정 백업 자동화

위험 완화 전략

기술적 위험 요소

- 트래픽 경로 변경 제한된 NS부터 적용
- 인증서 신뢰 문제 사전 SAN 검증
- PSA 정책 충돌 사전 라벨 설정

운영 위험 완화

```
#!/bin/bash
# 메시 상태 확인
kubectl get ds -n istio-system ztunnel
istioctl proxy-config cluster ztunnel-xxx
```

비즈니스 연속성

점진적 롤아웃 및 A/B 테스트
5분 내 롤백 가능한 전략 수립

12. 결론 및 Next Steps

핵심 가치 제안

- ⚡ **성능 향상:** eBPF + Ambient로 최대 30% 성능 개선
- ⚙️ **운영 단순화:** 사이드카 제거로 관리 복잡성 감소
- ≡ **점진적 도입:** 기존 워크로드 영향 최소화
- 📍 **미래 지향:** Cloud Native 표준 기술 스택

다음 단계

- 🏗️ **PoC 환경 구축:** 컨테이너 기반 테스트 랩 준비
- 🎓 **팀 교육:** Cilium, Istio Ambient 기술 역량 구축
- 📋 **파일럿 서비스 선정:** 비즈니스 임팩트 최소화
- 📊 **성공 메트릭 정의:** 성능, 가용성, 보안 KPI 설정

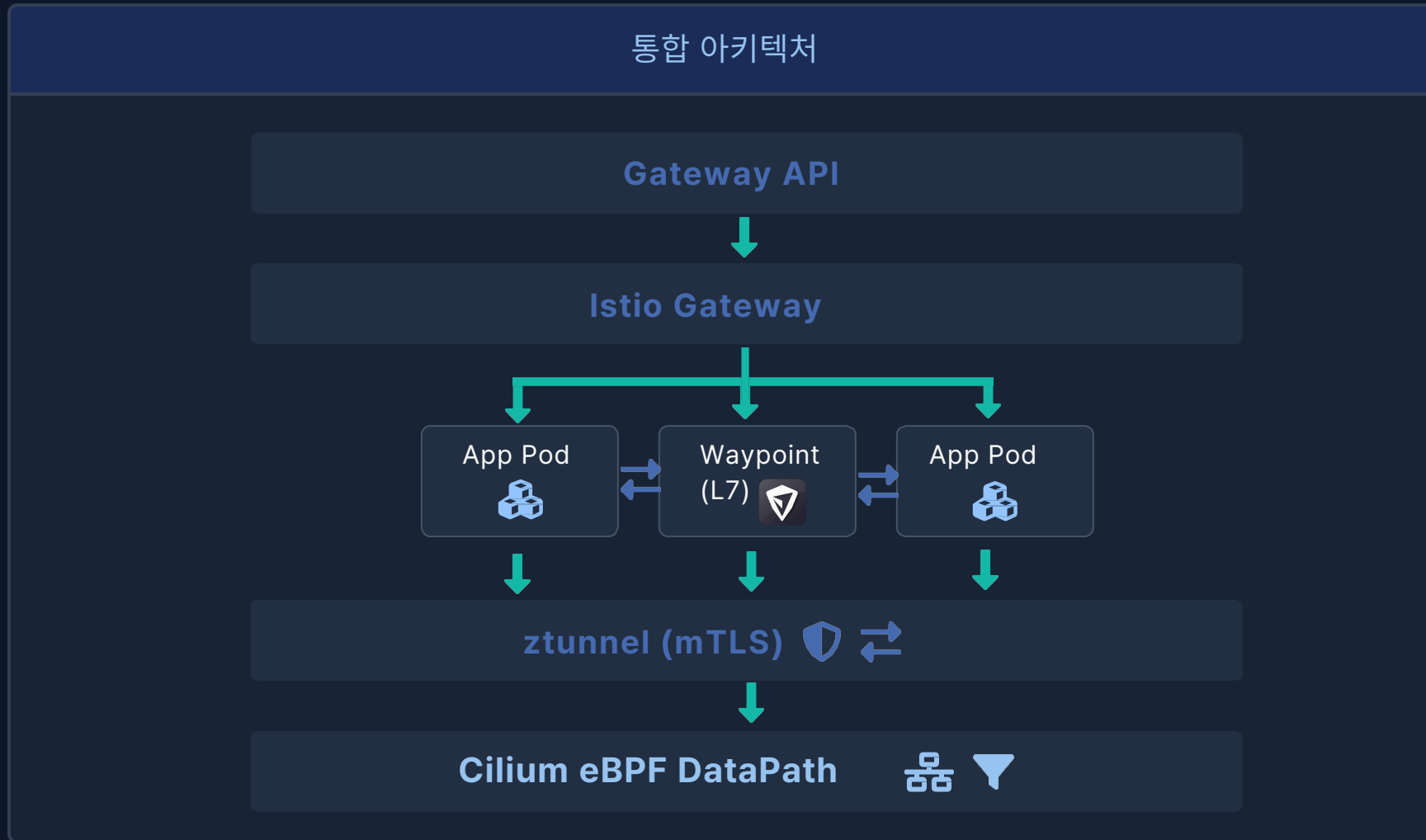
기대 효과

- </> **개발 생산성:** 메시 설정 없이 보안/관측성 자동 제공
- ≡ **운영 효율성:** 중앙집중식 정책 관리
- 📉 **비용 절감:** 리소스 사용량 20-30% 감소
- 🛡️ **보안 강화:** 기본 mTLS, 세밀한 접근 제어

장기 로드맵 (6개월)

- Q1** PoC 완료, 아키텍처 검증
- Q2** 파일럿 서비스 프로덕션 적용
- Q3** 주요 서비스 확대 적용
- Q4** 전사 표준 아키텍처 확정

부록: 참고 자료



부록: Source

가상 머신 이름	CPU 게스트	CPU VMM	RAM 사용량/합계	RAM %
master1	7%	43%	1.25 GB/3.69 GB	34.02%
master2	7%	42%	1.24 GB/3.69 GB	33.65%
master3	11%	54%	1.38 GB/3.69 GB	37.44%
worker1	3%	31%	806.68 MB/5.57 GB	14.13%
worker2	4%	40%	1.19 GB/5.57 GB	21.29%
worker3	3%	33%	1.31 GB/5.57 GB	23.55%

<https://github.com/dasomel/seminar/tree/main/openinfradays/2025>

- source
 - network
 - scripts
 - test
 - check.sh
 - Install.md
 - Test.md
 - Vagrantfile

- network
 - grafana-expose.yaml
 - hubble-expose.yaml
 - kiali-expose.yaml
 - prometheus-network-policy.yaml

- scripts
 - file
 - helm
 - 00.global_variable.sh
 - 01.all_common_setting.sh
 - 02.master_nfs_server.sh
 - 10.prerequisites.sh
 - 11.rke2.sh
 - 12.kube-vip.sh
 - 13.agent.sh
 - install_istio_ambient.sh
 - set-config.sh

- test
 - cilium-network-policy.yaml
 - cluster-policy.yaml
 - deployment.yaml
 - gateway.yaml
 - http-route.yaml
 - kustomization.yaml
 - namespace.yaml
 - network-policy.yaml
 - peer-authentication.yaml
 - service.yaml
 - waypoint.yaml
 - waypoint-authz.yaml