DEVIEW 2021

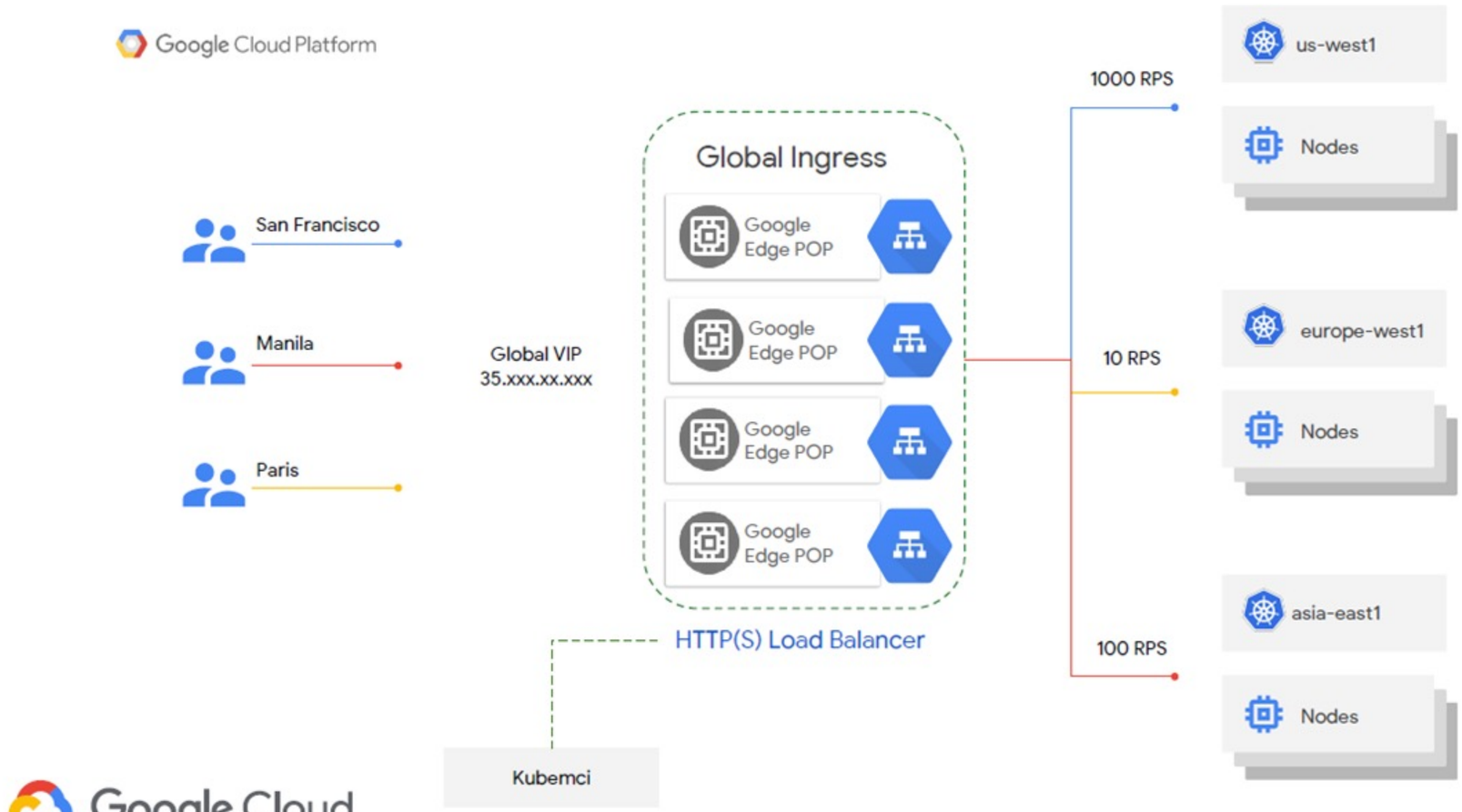
당신의 대문을 책임집니다. Istio/Envoy로 Multi-IDC L7 로드밸런서 만들기

김동경 NAVER

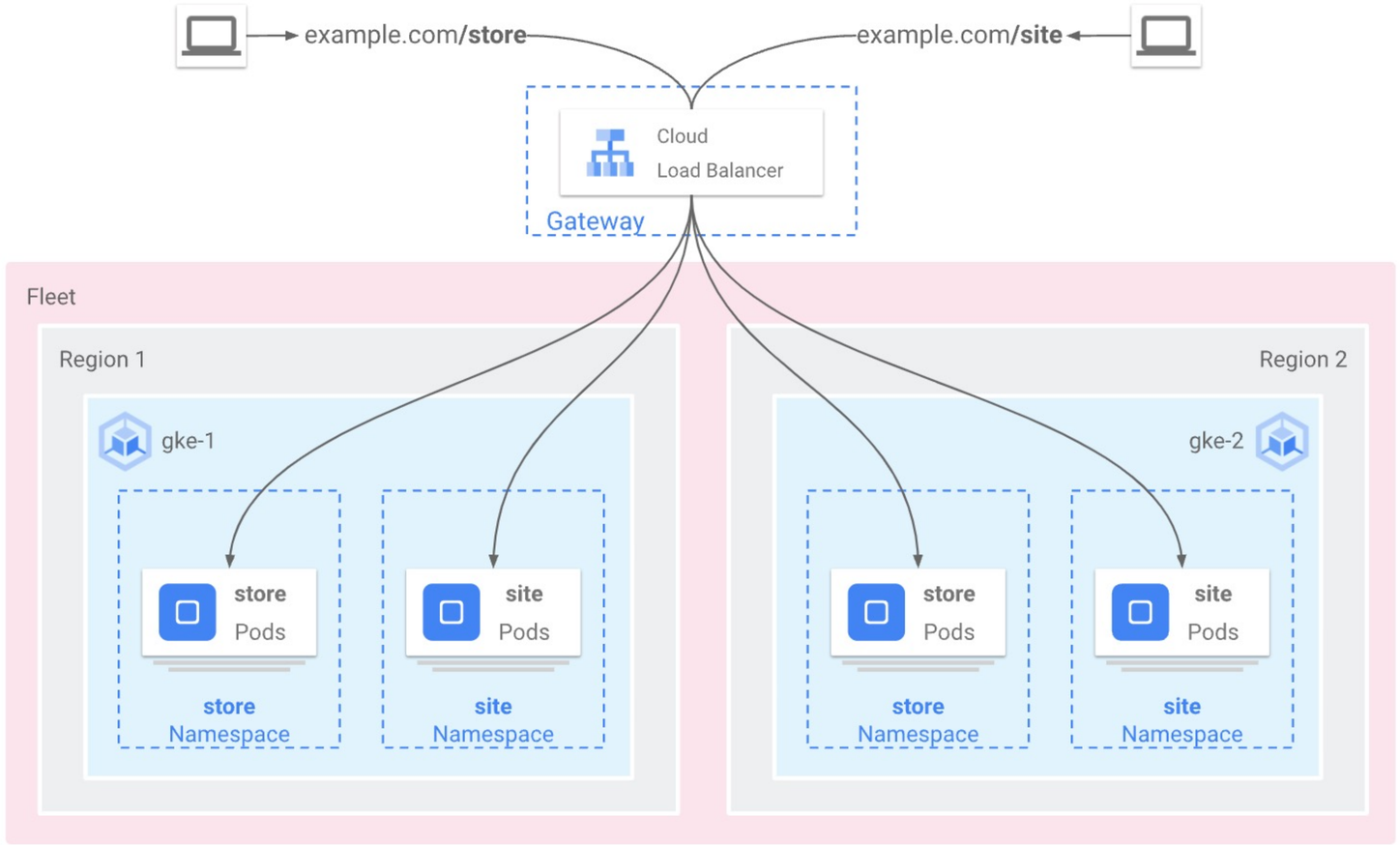
CONTENTS

1. Multi IDC 환경을 위한 로드 밸런서의 필요성

1.1 Cloud-Native 환경의 Multi-Clusters

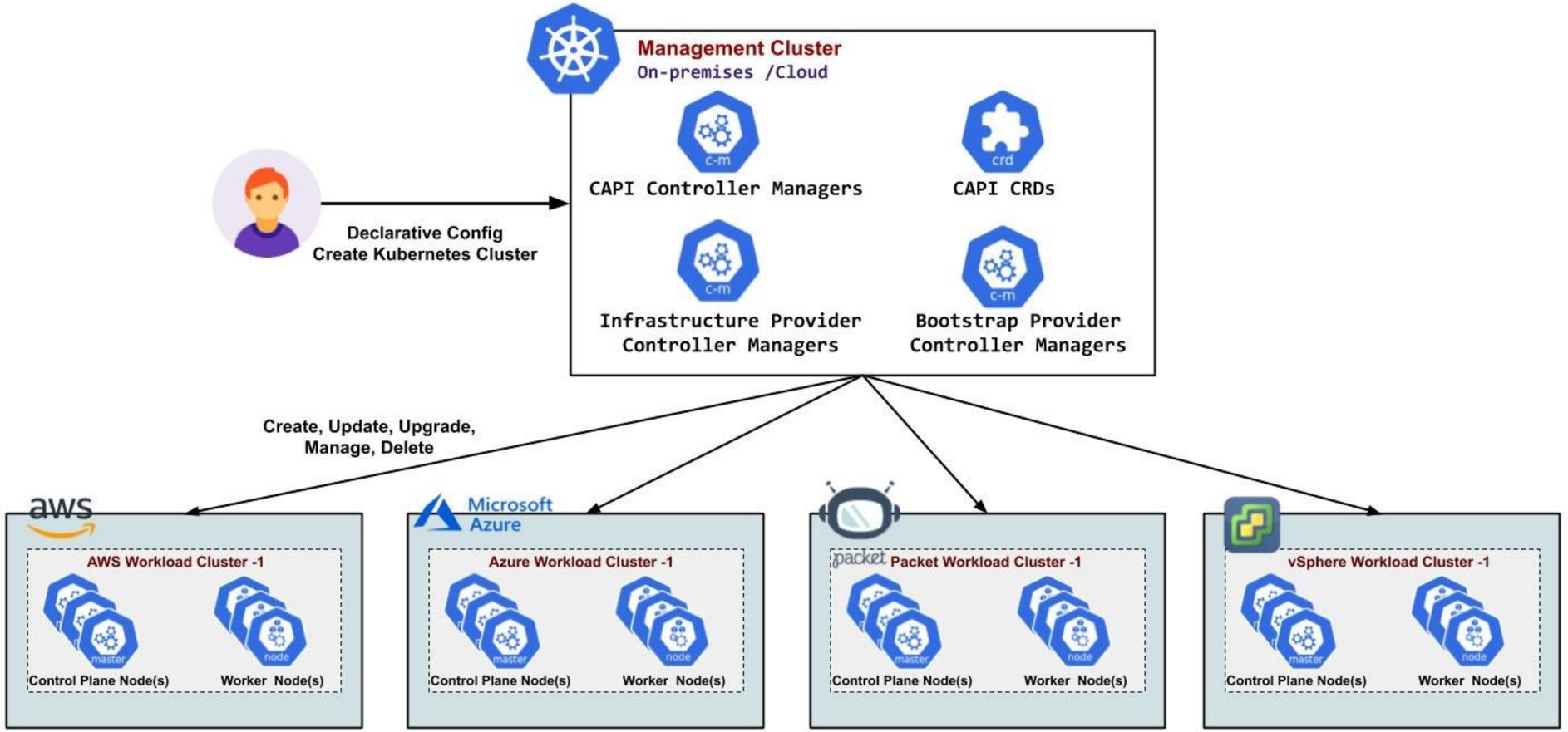


1.1 Cloud-Native 환경의 Multi-Clusters

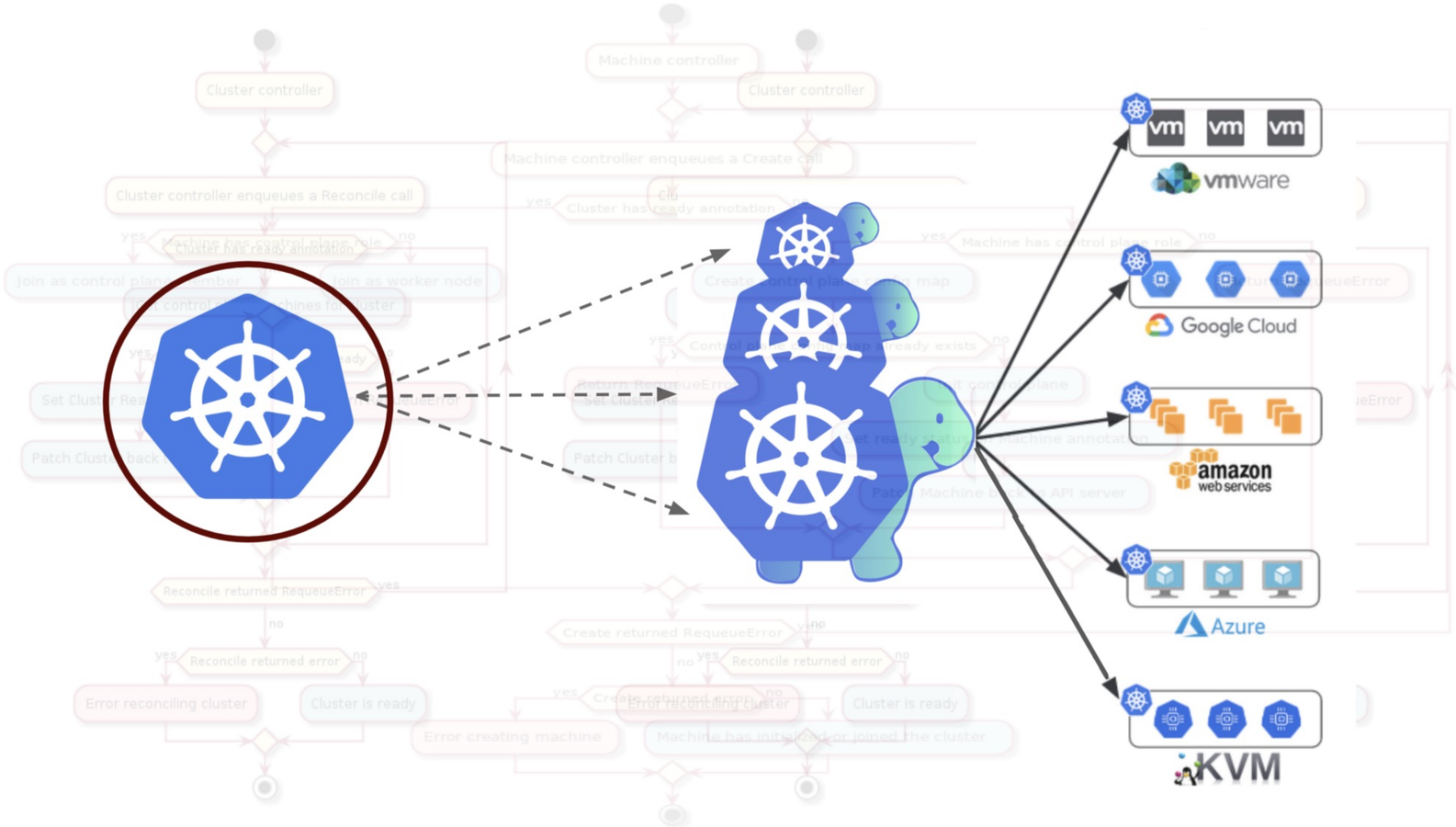


1.2 Cluster API to Create Clusters

DEVIEW 2021



1.2 Cluster API to Create Clusters



https://itnext.io/multi-cloud-and-multi-cluster-declarative-kubernetes-cluster-creation-and-management-with-cluster-6df8efdc2a89

1.3 NAVER의 컨테이너 클러스터 환경

평촌
(14개)

춘천
(6개)

가산
(2개)

일본 도쿄

싱가포르

미국 서부

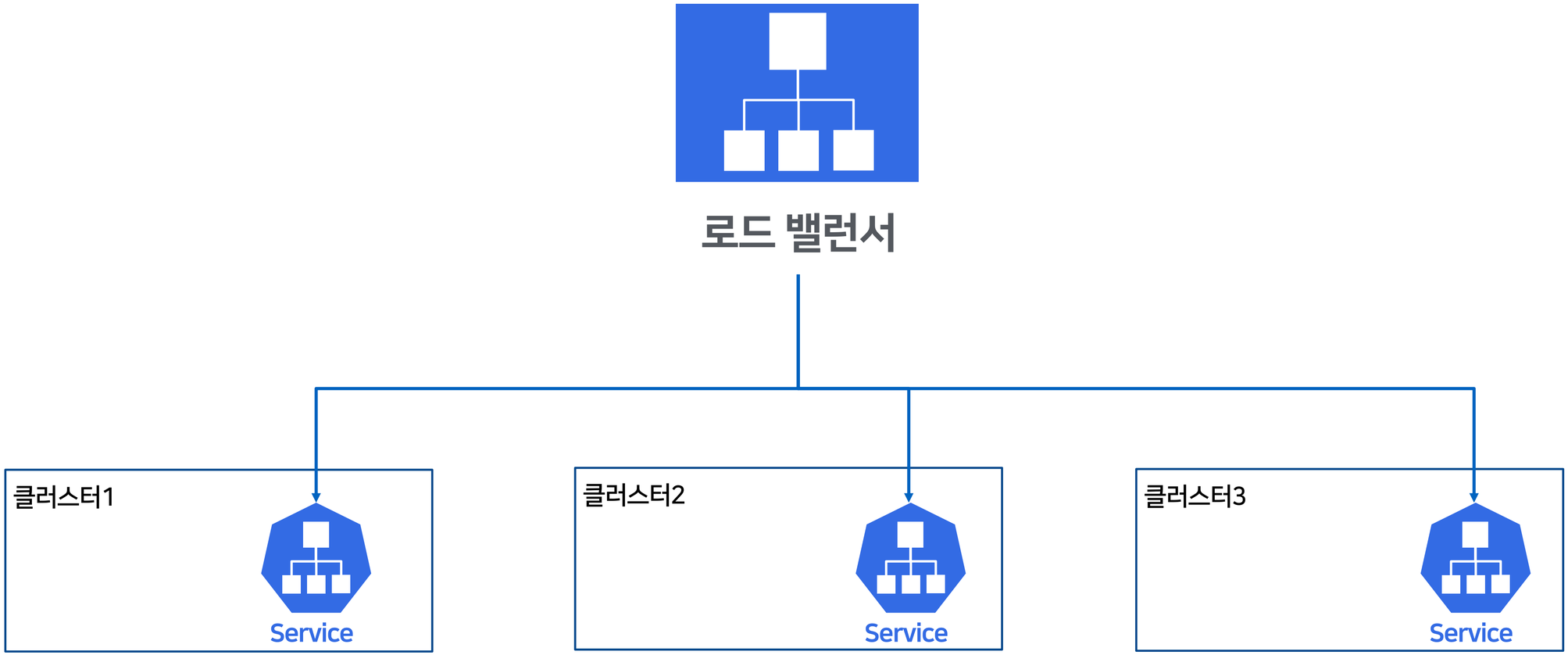
미국 동부

7개 지역

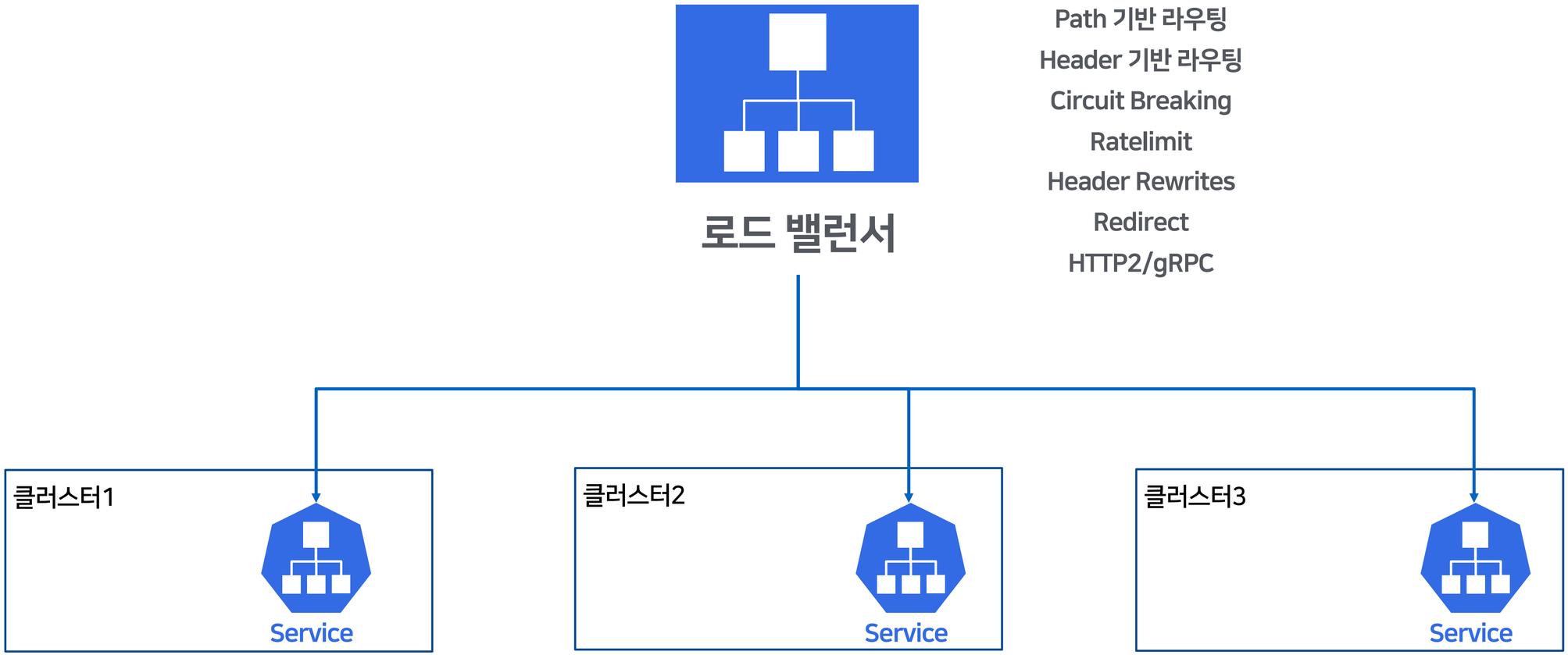
26개 클러스터

800개 노드(각각 최대)

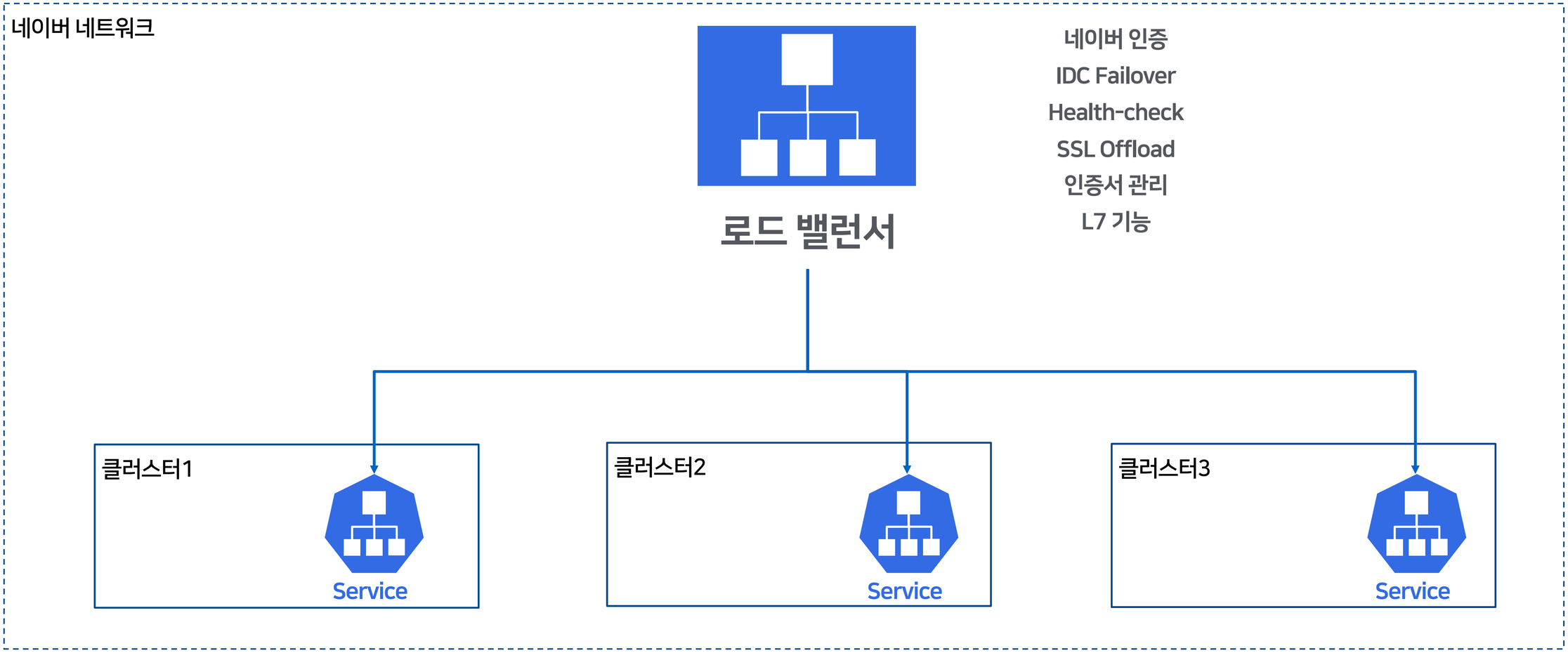
1.4 다양한 클러스터를 아우르는 로드 밸런서의 필요성



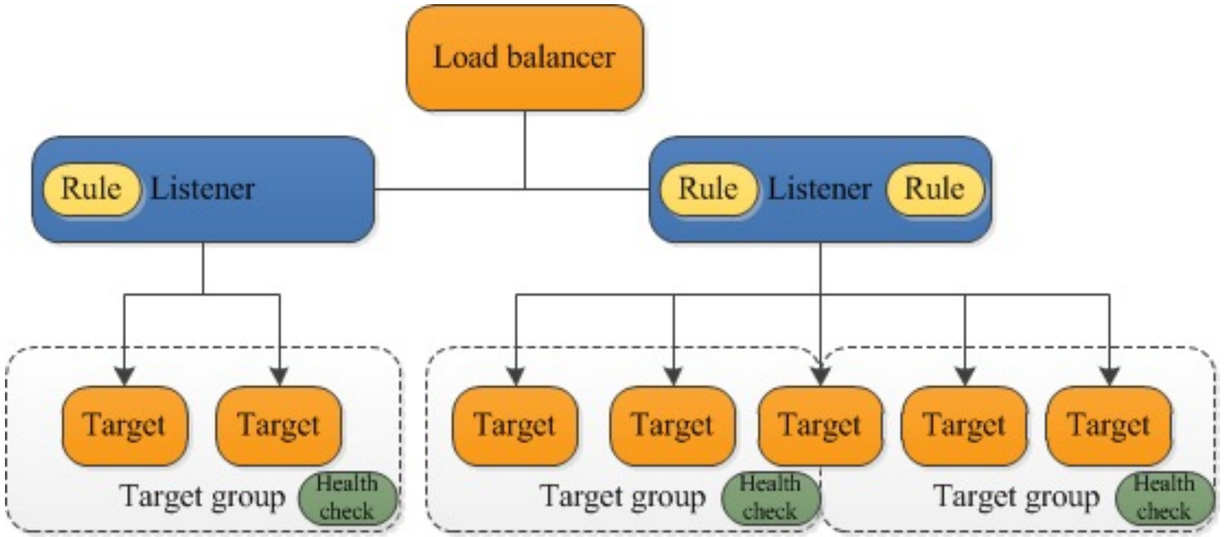
1.5 L7 로드 밸런서의 필요성



1.6 다양한 공용 기능의 처리



1.6 다양한 공용 기능의 처리

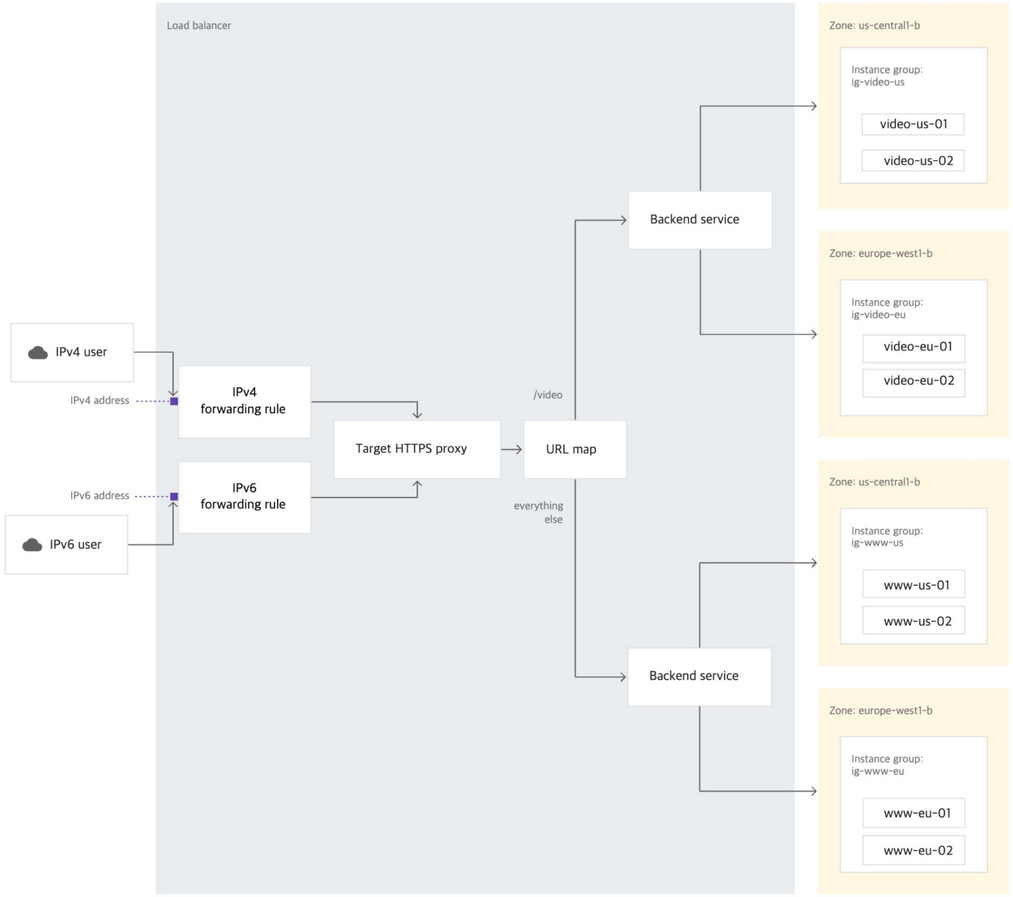


AWS – Application Load Balancer(L7)

- SSL offload
- 고정 응답
- 헬스 체크
- 인증
- 로드밸런싱
- 라우팅
- Failover

https://aws.amazon.com/elasticloadbalancing/features/?nc1=h_ls

1.6 다양한 공용 기능의 처리



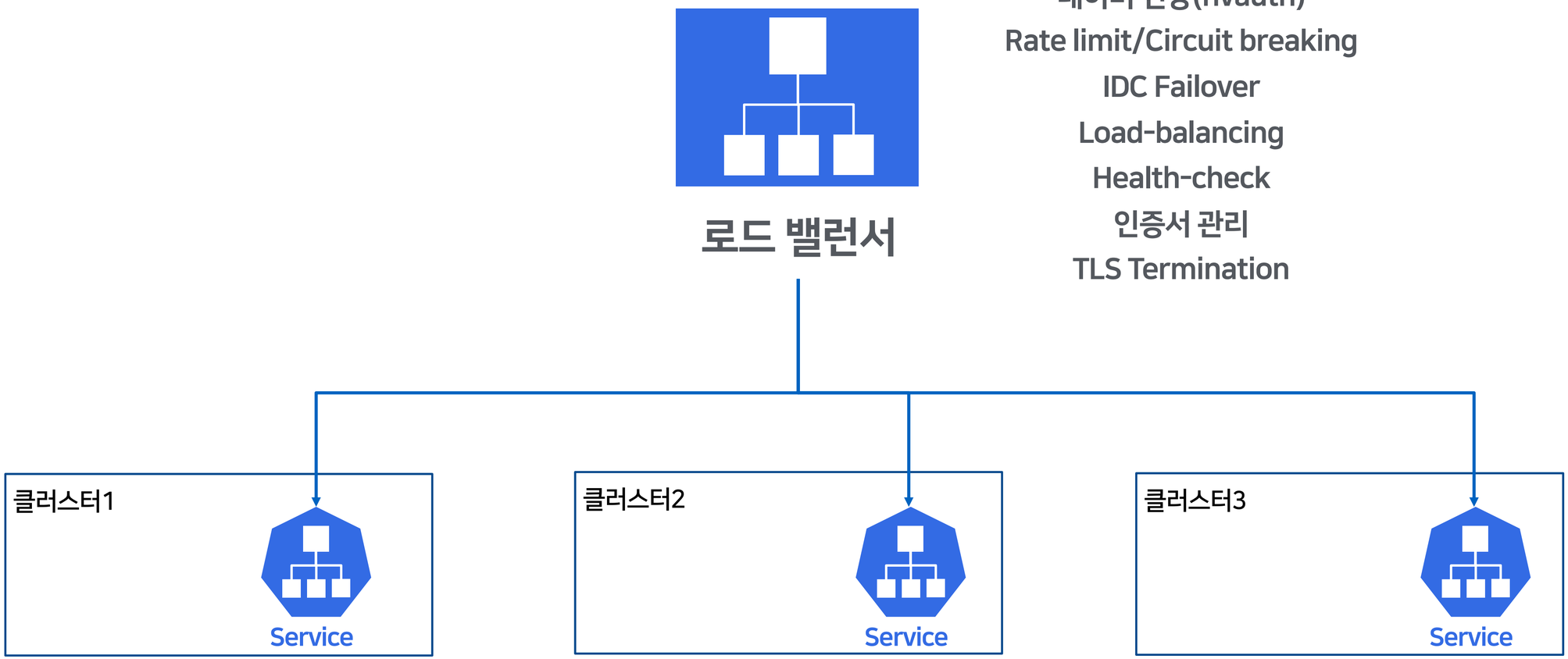
GCP – External Load Balancer

- SSL offload
- 고정 응답
- 헬스 체크
- 인증
- 로드밸런싱
- 라우팅
- Failover

https://cloud.google.com/load-balancing/docs/features

2. 동적으로 설정을 변경할 수 있는 L7의 필요성

2.1 L7 로드밸랜서


로드 밸런서
네이버 인증(nvauth)
Rate limit/Circuit breaking
IDC Failover
Load-balancing
Health-check
인증서 관리
TLS Termination
클러스터1
Service
클러스터2
Service
클러스터3
Service

2.1 L7 로드밸런서

Cloud-Native

On-Demand

Distributed

Scalable

Programmable

2.2 L7 로드 밸런서로 사용하기 위한 대안들

Programmable 하지 않음

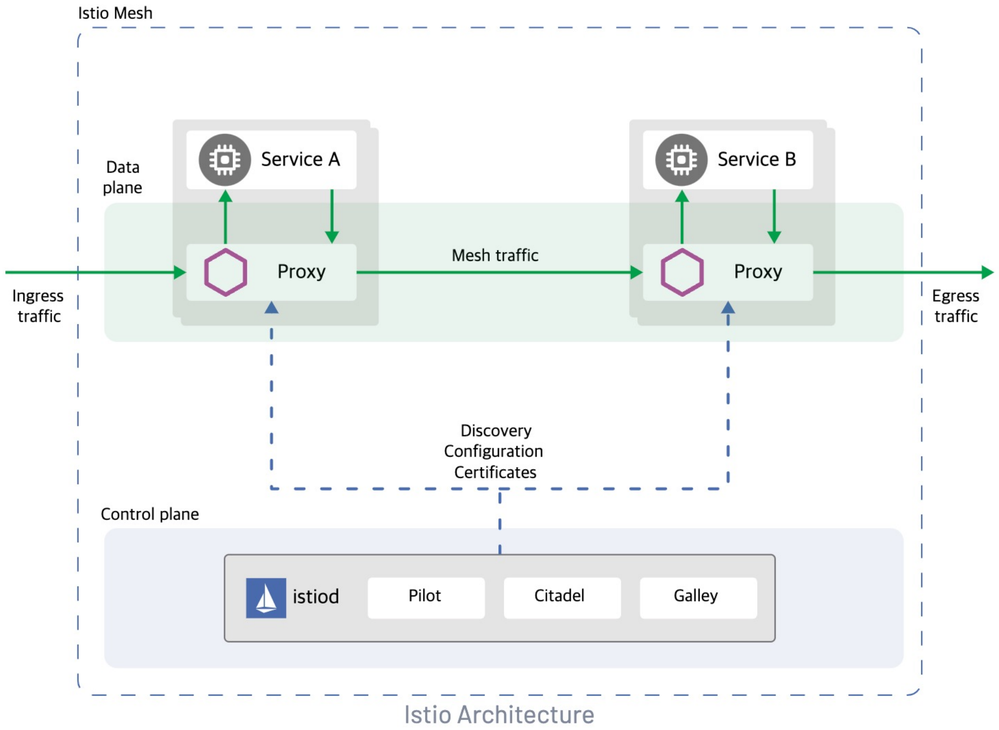
Rich한 L7 기능을 제공하지 않음

3. Istio/Envoy를 Load balancer로 사용하기

3.1 Istio/Envoy의 특징 및 L7 기능

Service Mesh

3.1 Istio/Envoy의 특징 및 L7 기능

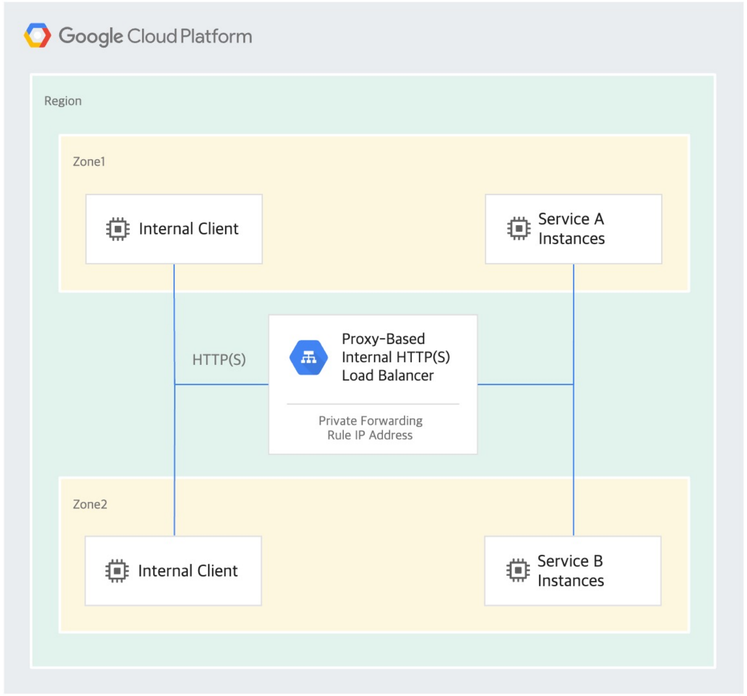


Istio Architecture

- Dynamic service discovery
- Load balancing
- TLS termination
- HTTP/2 and gRPC proxies
- Circuit breakers
- Health checks
- Staged rollouts with %-based traffic split
- Fault injection
- Rich metrics

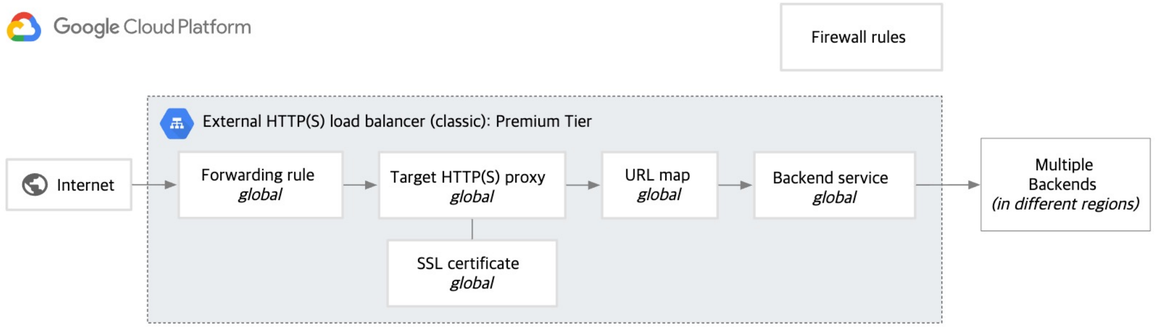
https://istio.io/latest/about/service-mesh/

3.1 Istio/Envoy의 특징 및 L7 기능



GCP의 Envoy 기반 내부 로드 밸런서

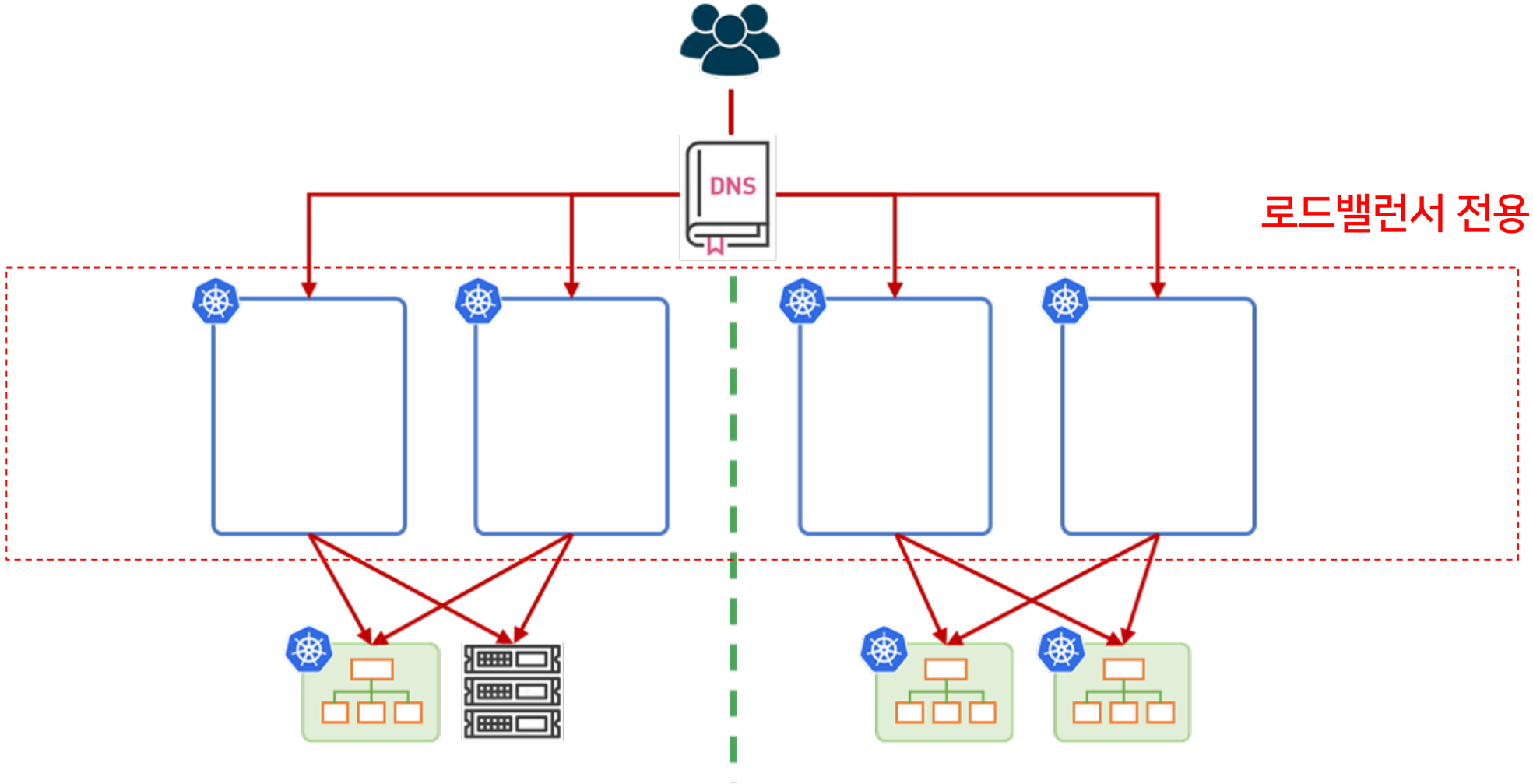
https://cloud.google.com/load-balancing/docs/l7-internal



GCP의 Envoy 기반 외부 로드 밸런서

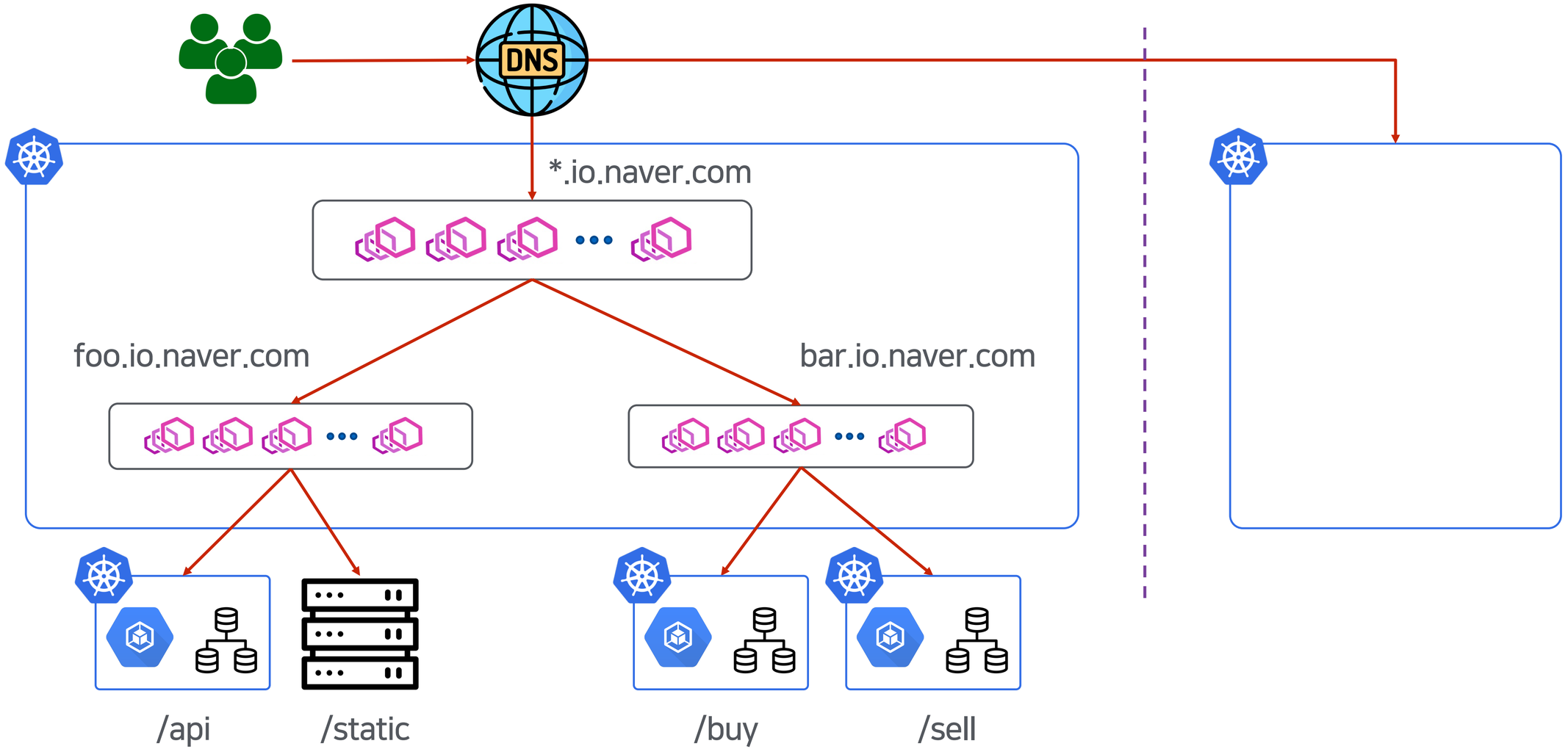
https://cloud.google.com/load-balancing/docs/https

3.2 Multi-IDC에 배포

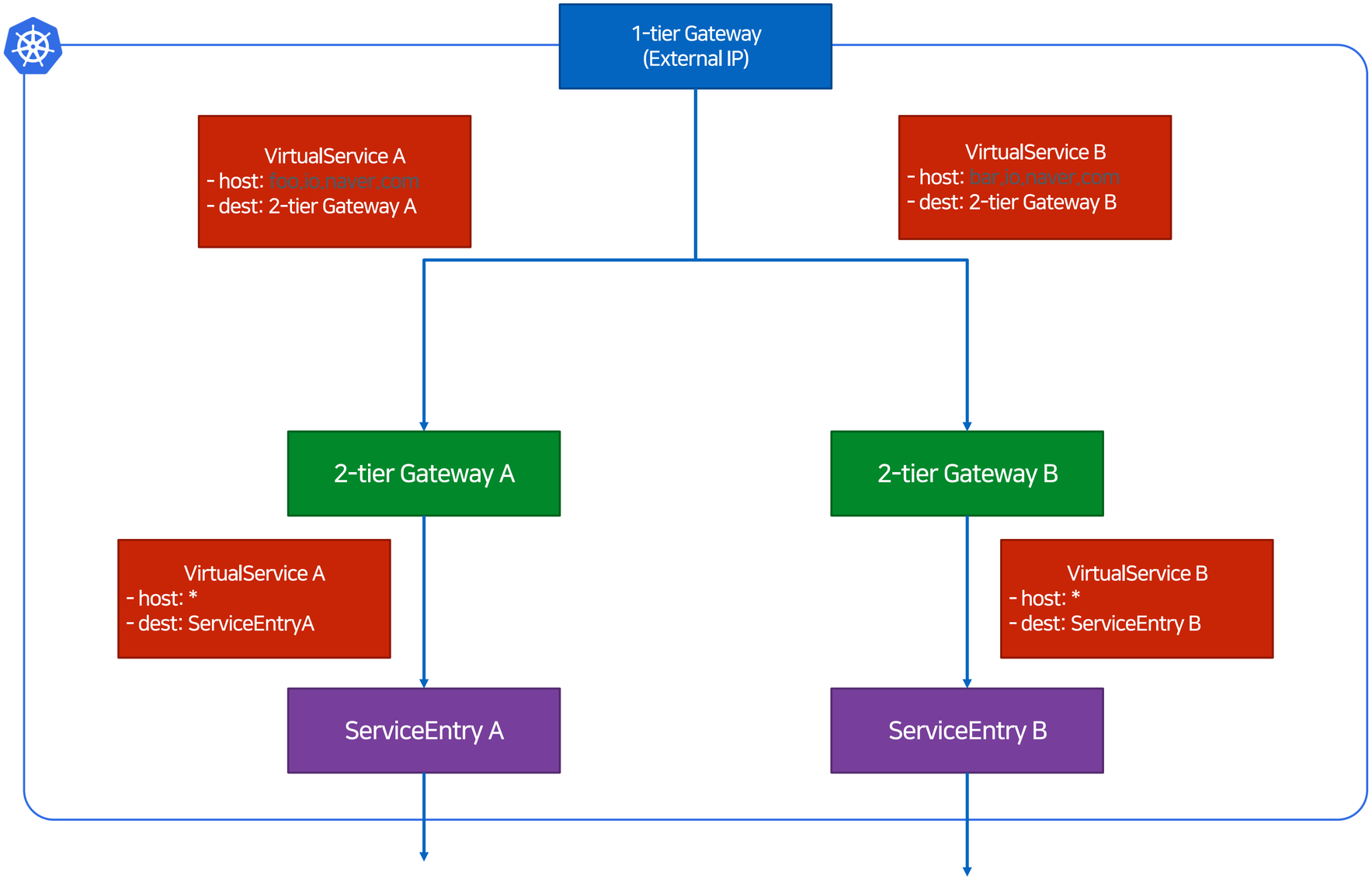


로드밸런서 전용 클러스터 구축

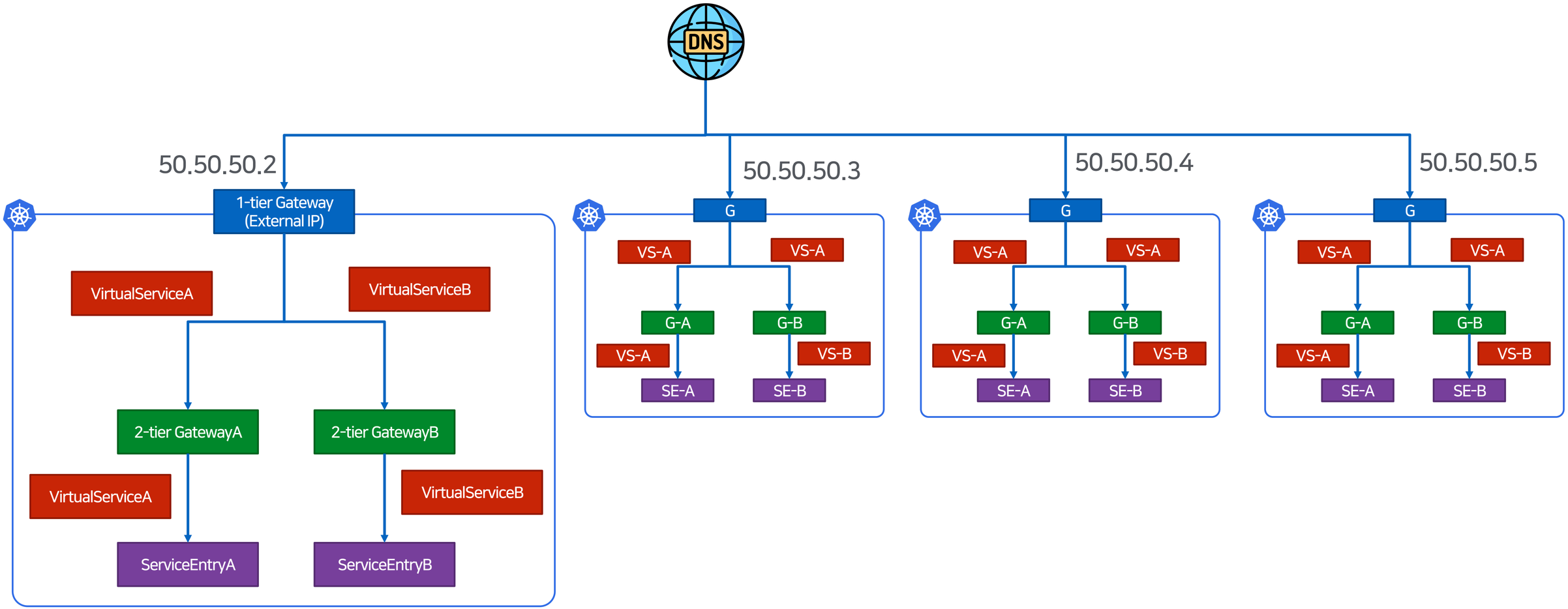
3.3 2-Tier 아키텍처의 도입



3.4 Istio/Envoy 기반의 구조

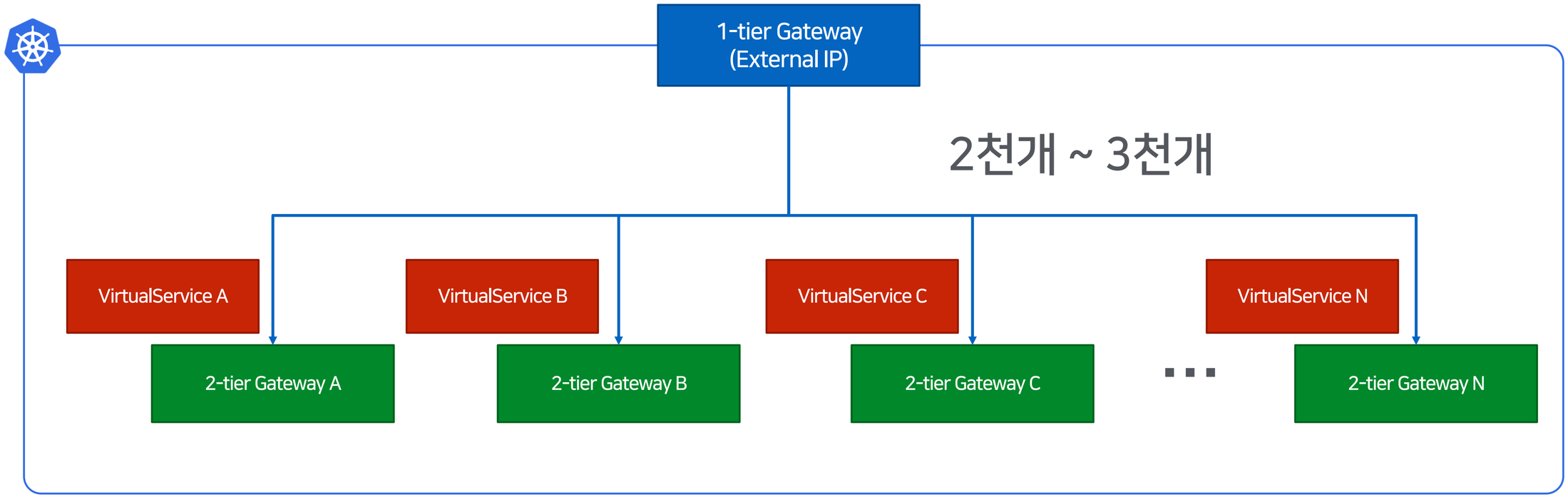


3.4 Istio/Envoy 기반의 2-tier 아키텍처의 도입

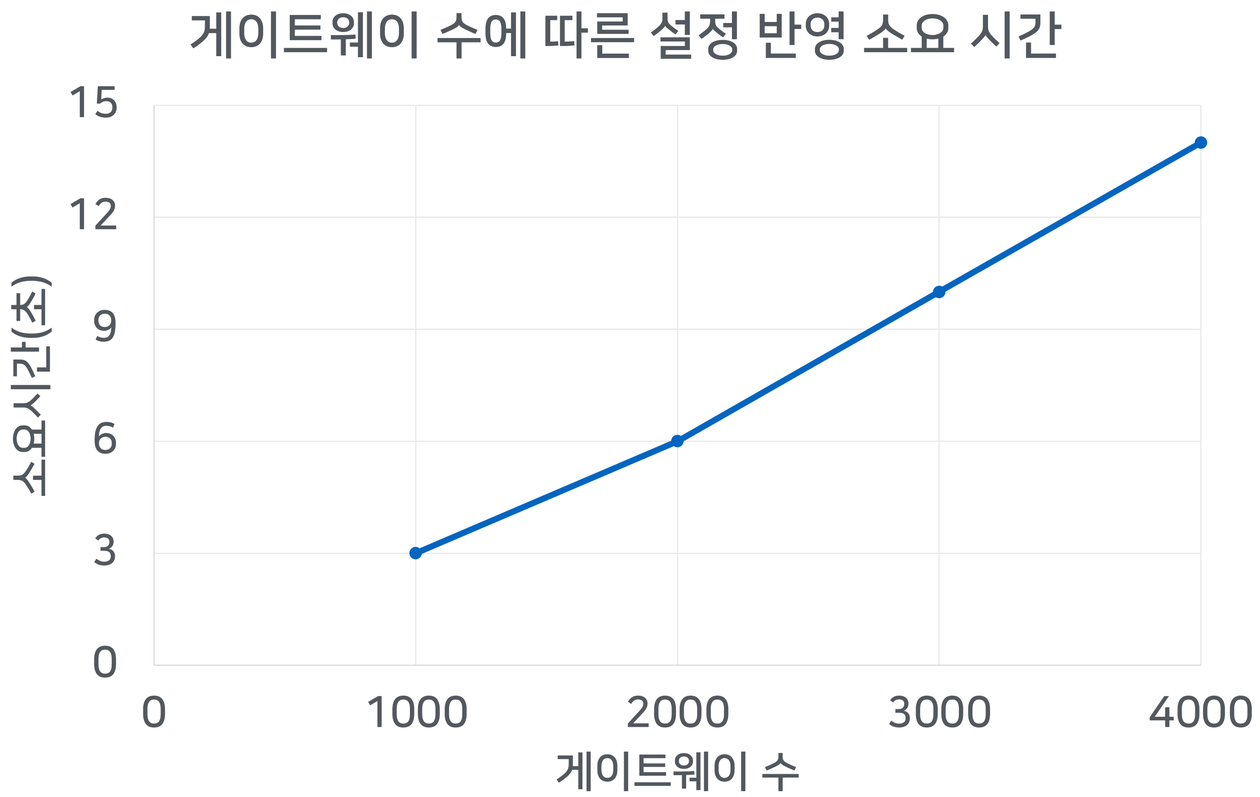


Public IP를 아끼기 위한 2 tier 구조

3.5 게이트웨이 성능 이슈



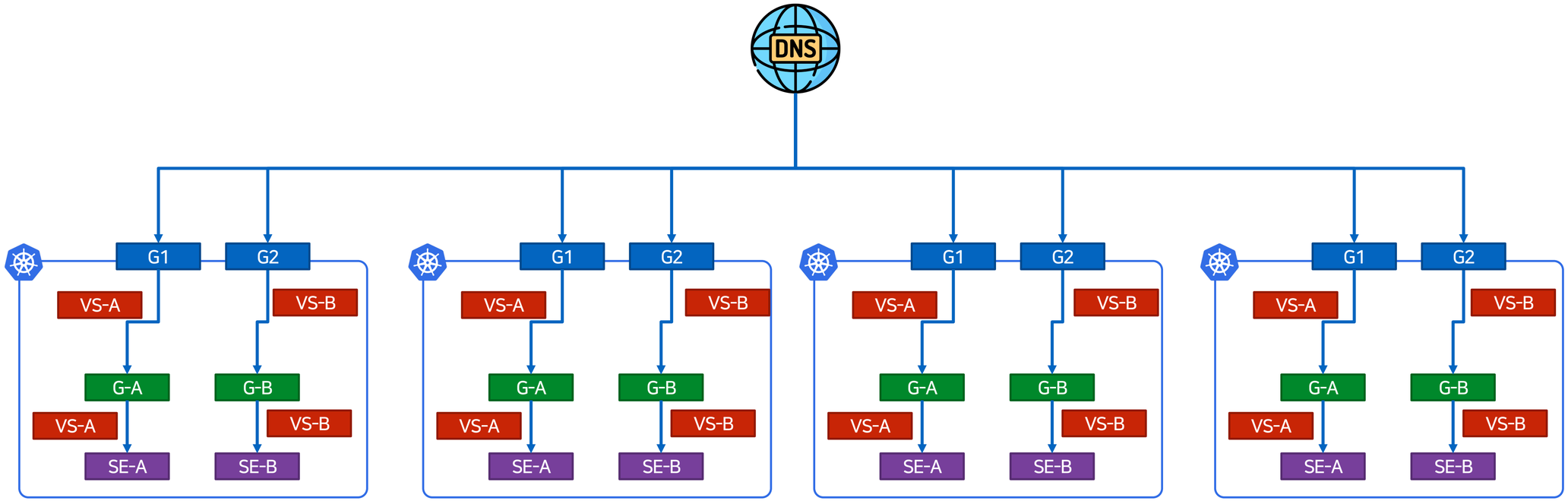
3.5 게이트웨이 성능 이슈



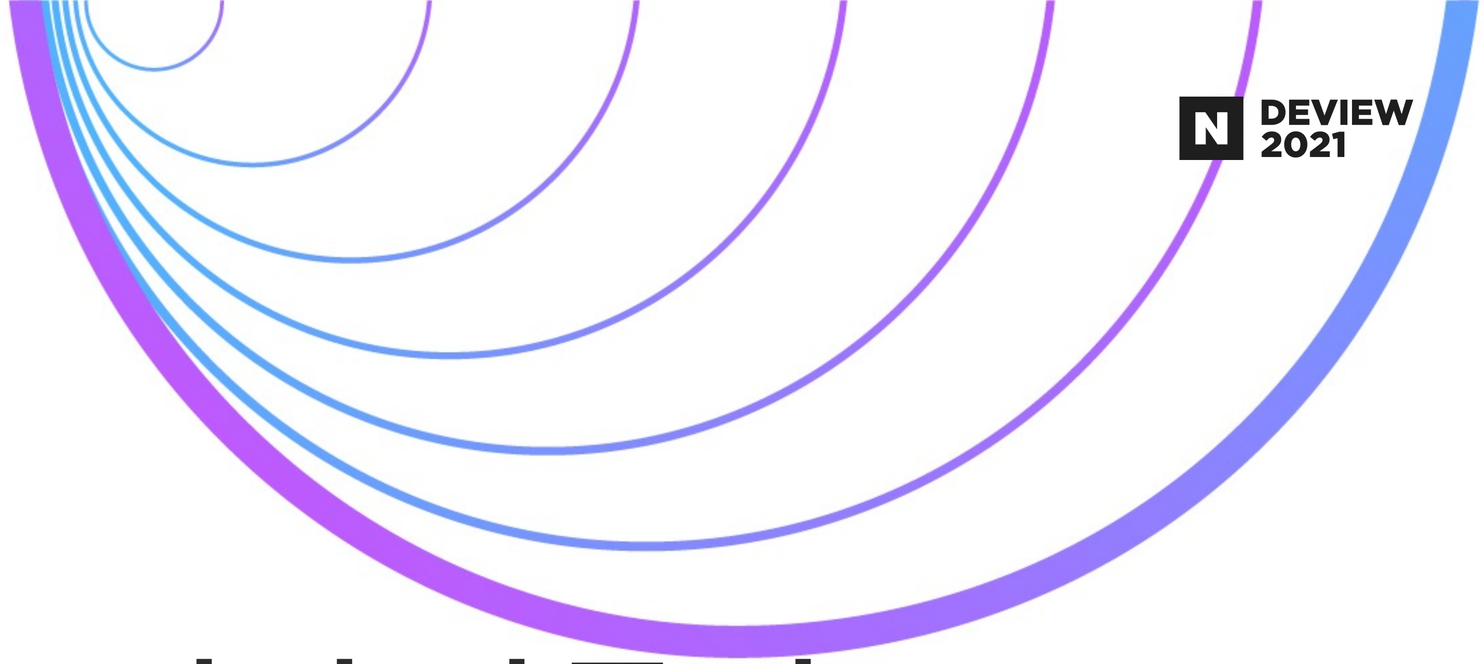
2500개 이상에서 OOM 발생

(디폴트 설정 기준)

3.5 게이트웨이 성능 이슈

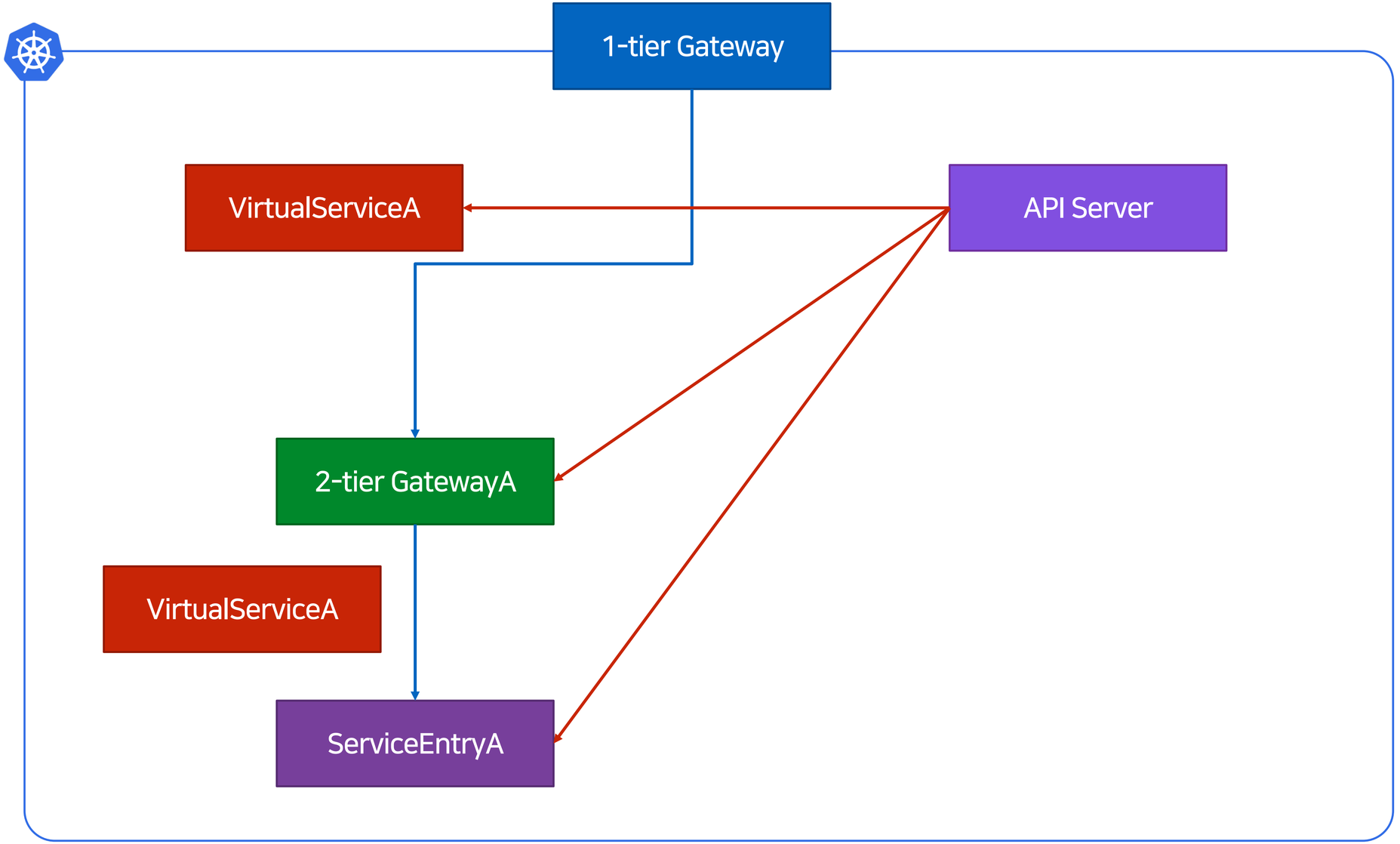


게이트웨이에 연결된 서비스 수에 따라 Migration 필요



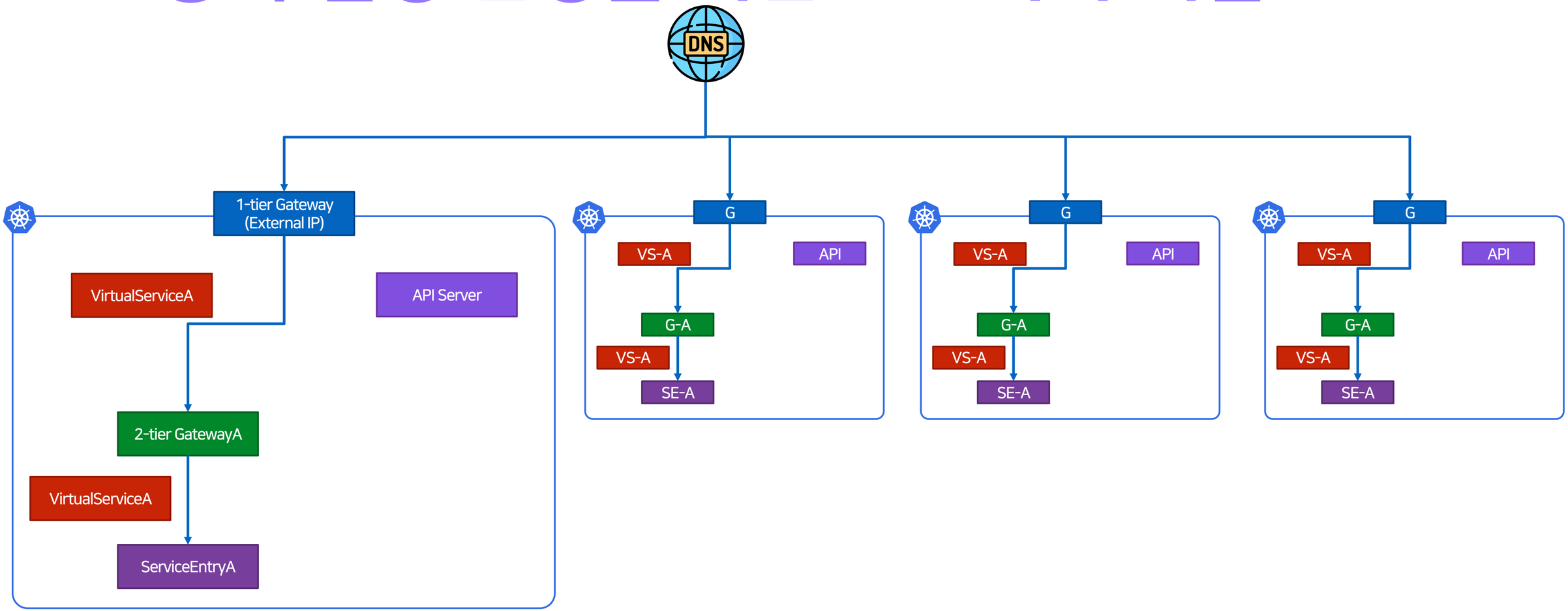
4. API 서버 만들기

4.1 동적 설정 변경을 위한 API 서버 개발

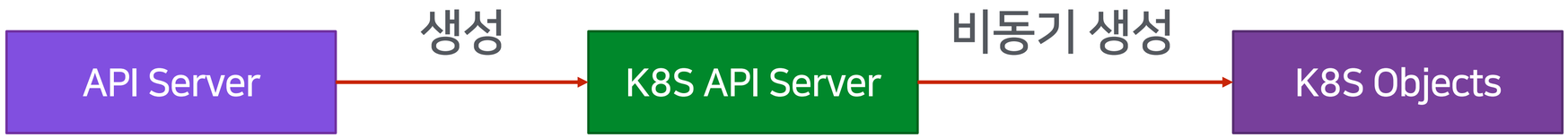


4.1 동적 설정 변경을 위한 API 서버 개발

DEVIEW 2021

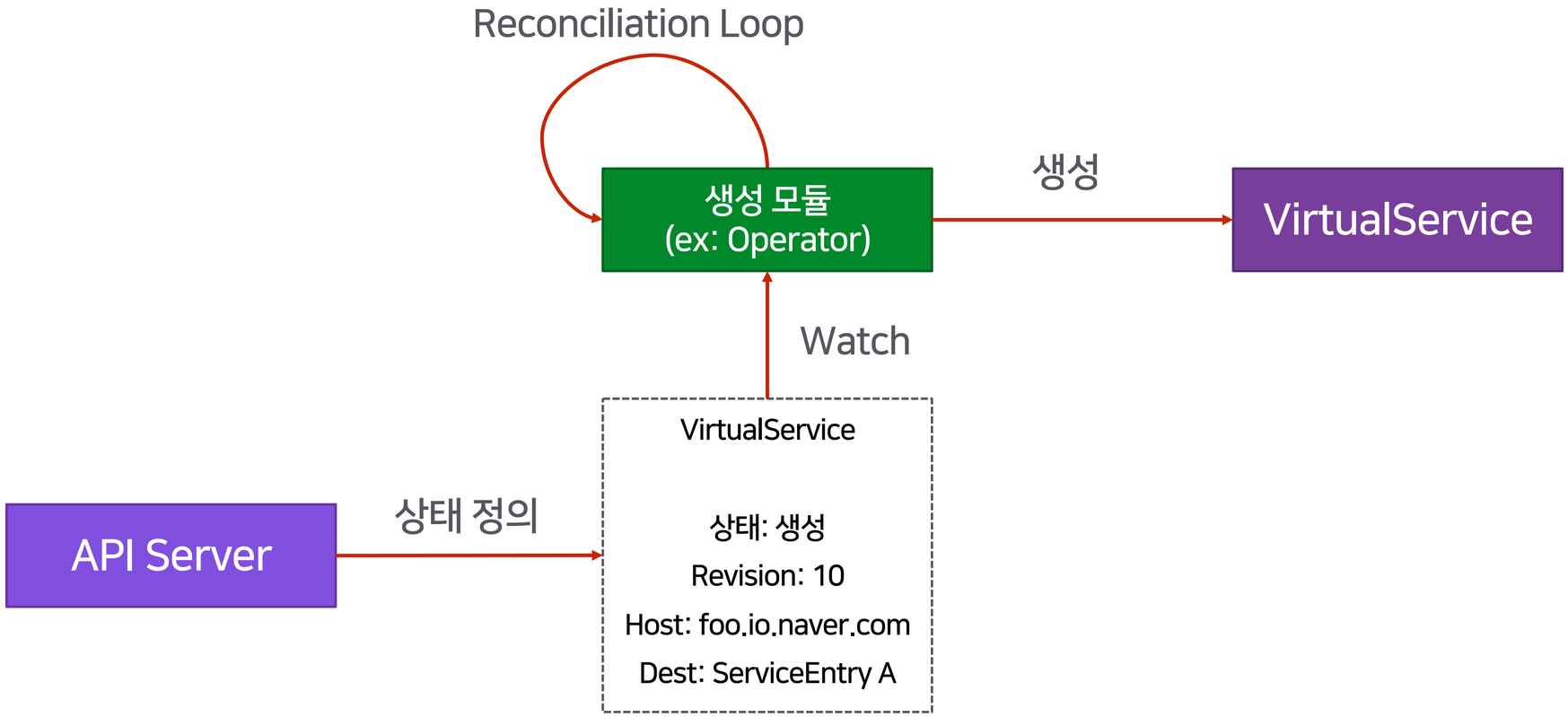


4.1 동적 설정 변경을 위한 API 서버 개발



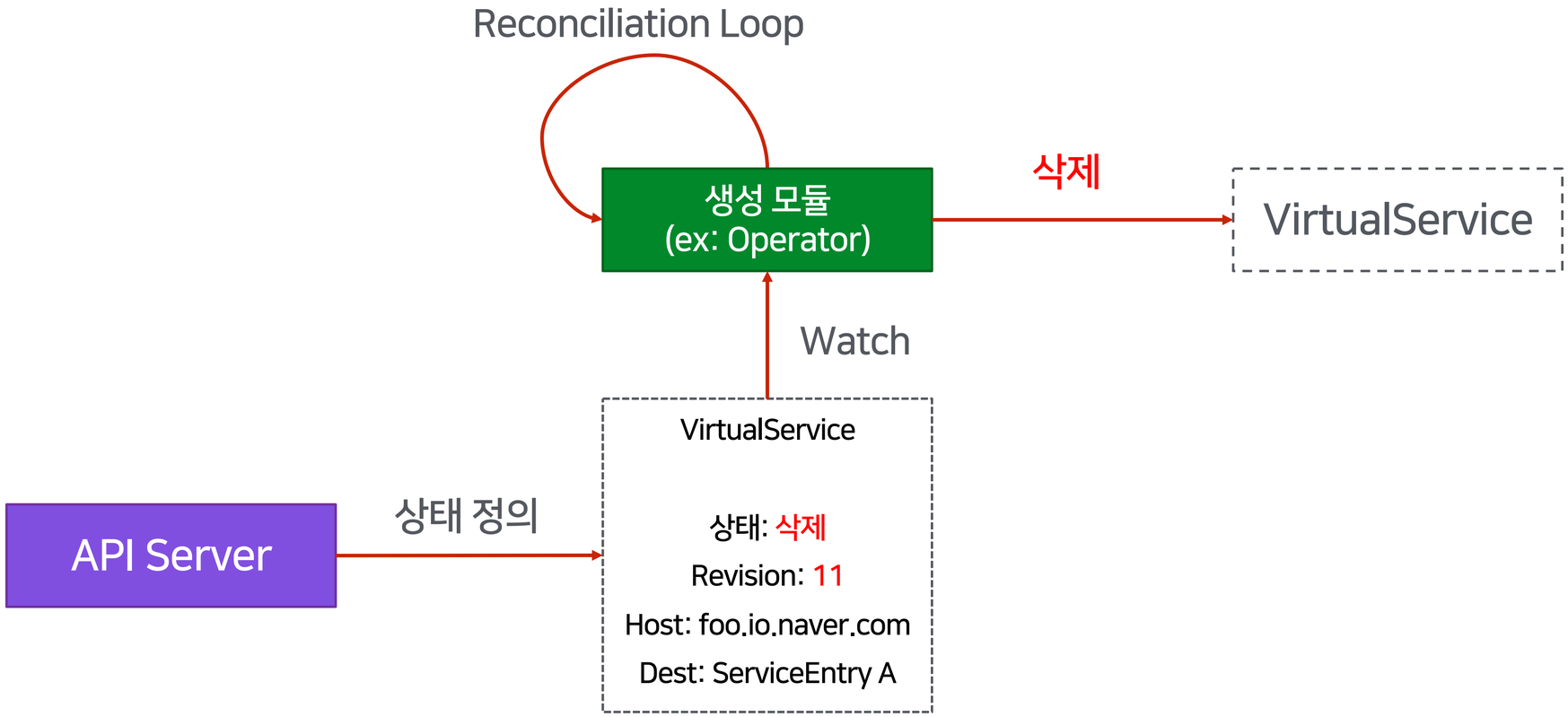
선언적(Declarative) API

4.1 동적 설정 변경을 위한 API 서버 개발



선언적(Declarative) API

4.1 동적 설정 변경을 위한 API 서버 개발



선언적(Declarative) API

4.1 동적 설정 변경을 위한 API 서버 개발

DEVIEW 2021

API Server

Declarative API
Producer/Consumer
Reconciliation Loop

4.2 모델 디자인

DEVIEW 2021

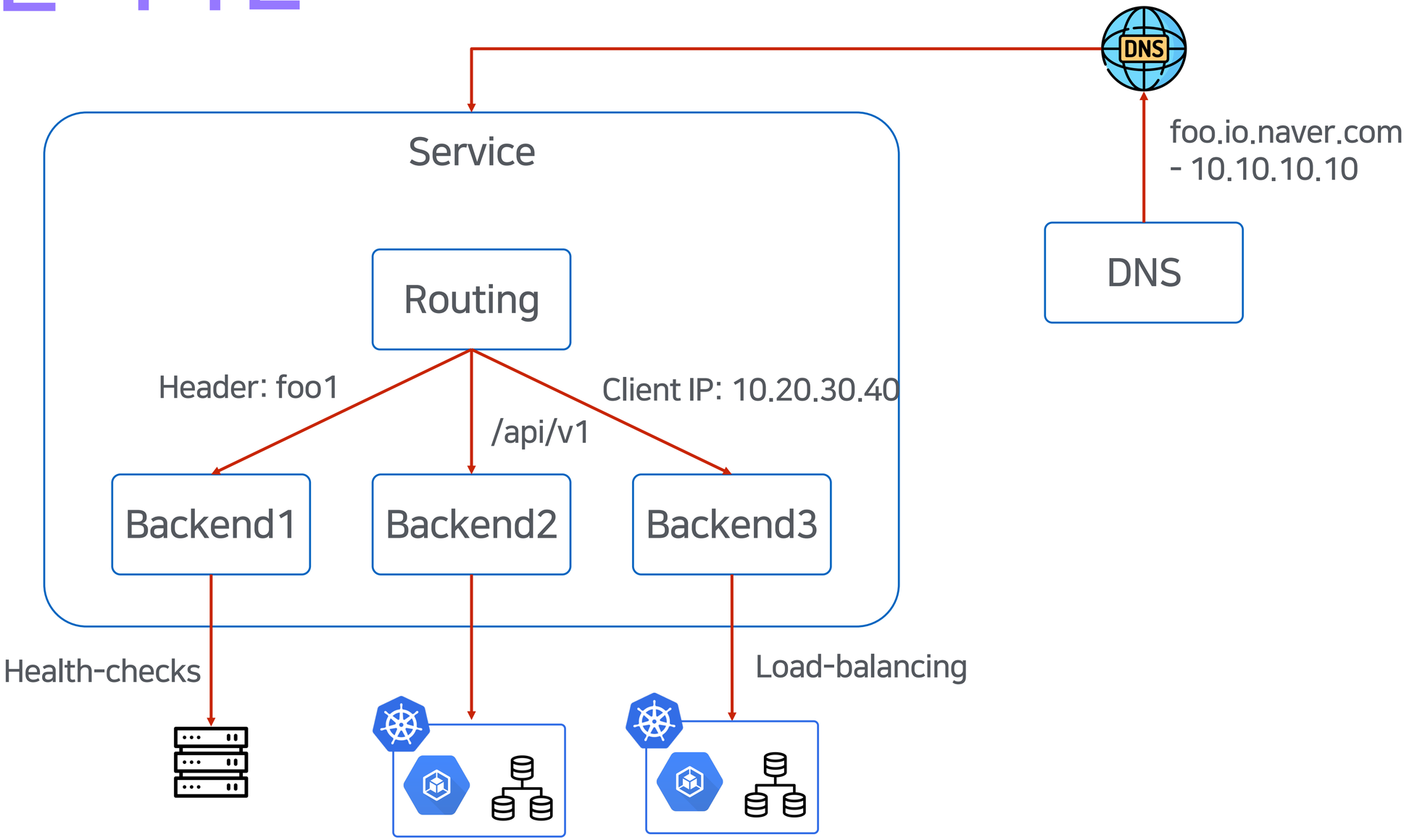
Service

Routing

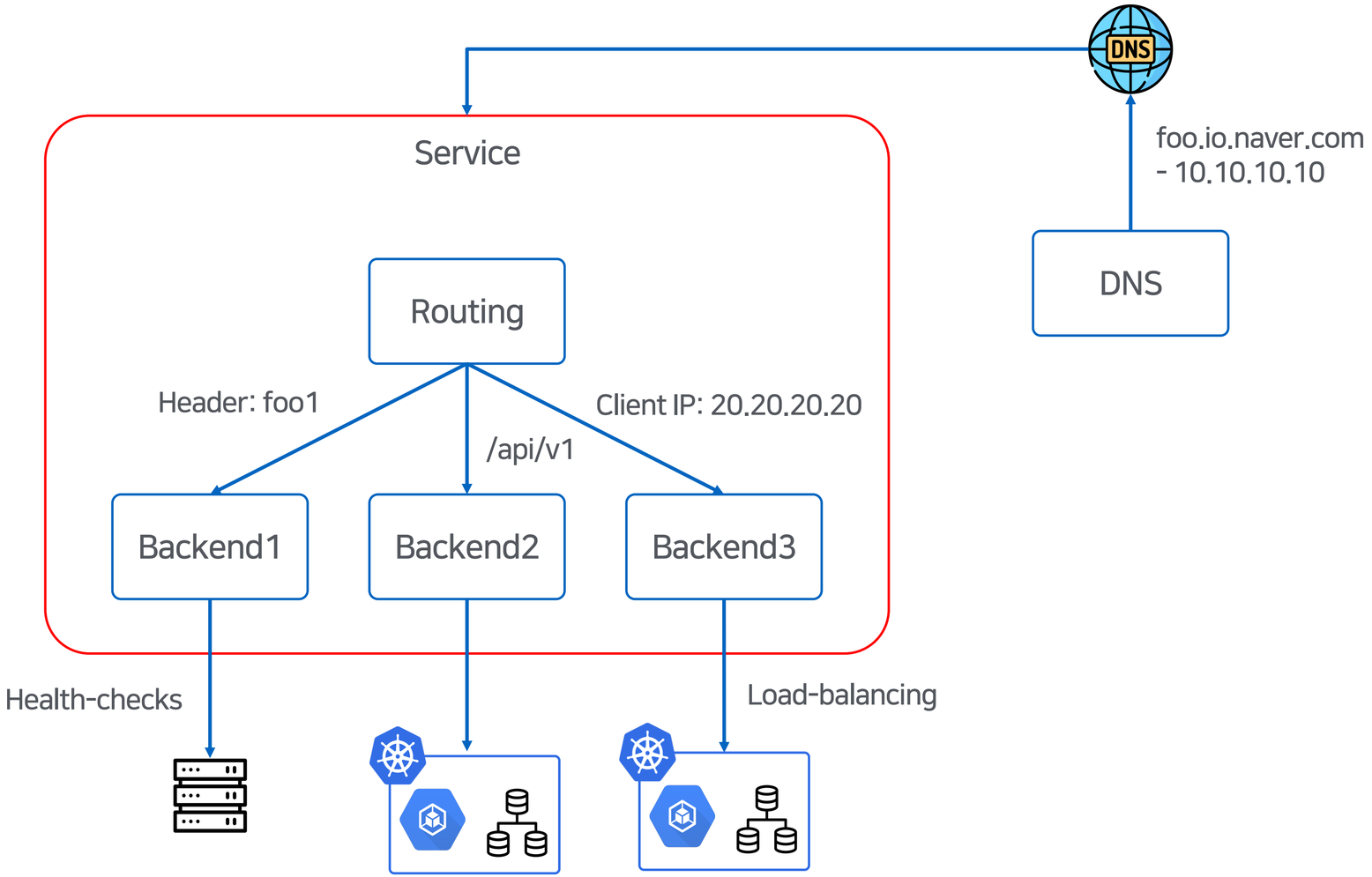
Backend

DNS

4.2 모델 디자인


DNS
foo.io.naver.com
- 10.10.10.10
DNS
Service
Routing
Header: foo1
/api/v1
Client IP: 10.20.30.40
Backend1
Backend2
Backend3
Health-checks
Load-balancing

4.2 모델 디자인 – 생성되는 K8S 오브젝트



Service

Default Domain

Namespace

Service(K8S)

Ingress-gateway(2-tier)

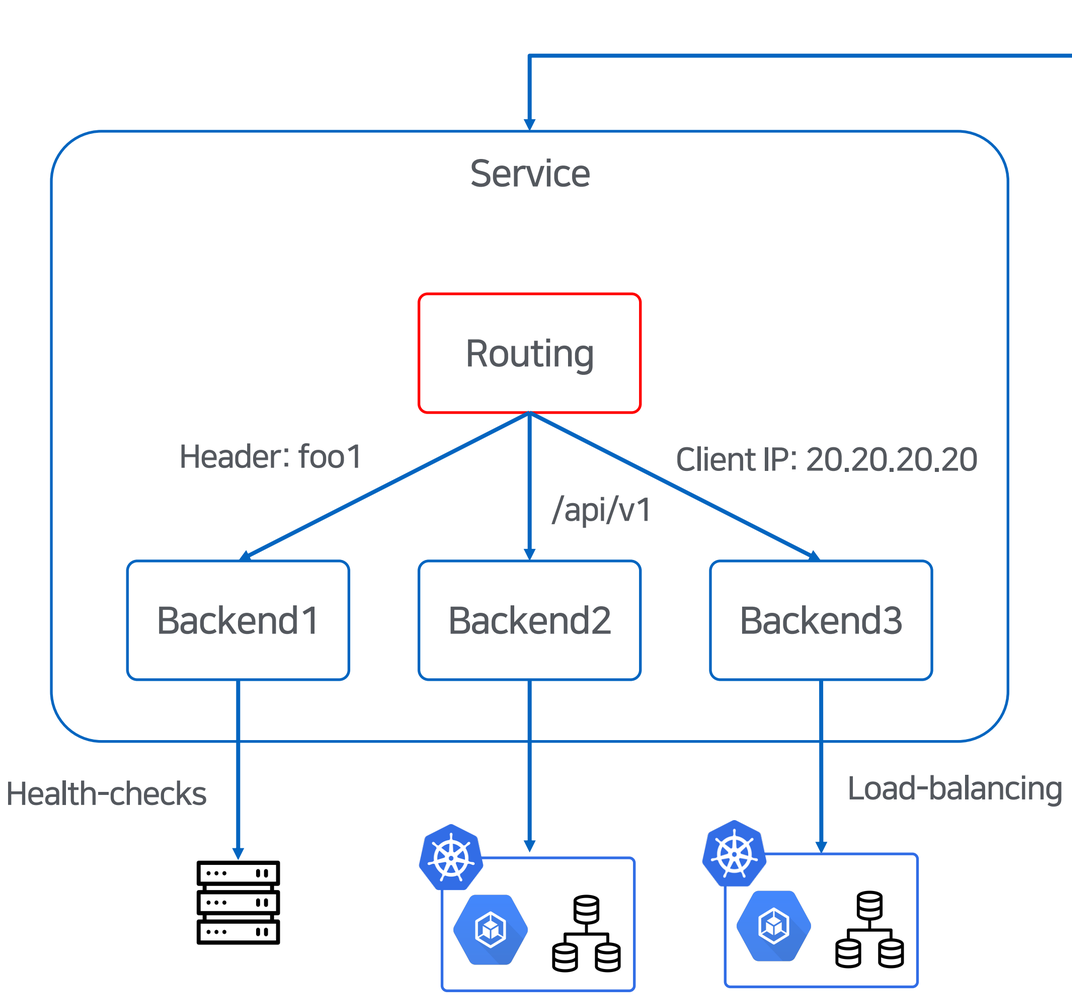
HPA

PDB

Deployment

ServiceAccount

4.2 모델 디자인 – 생성되는 K8S 오브젝트



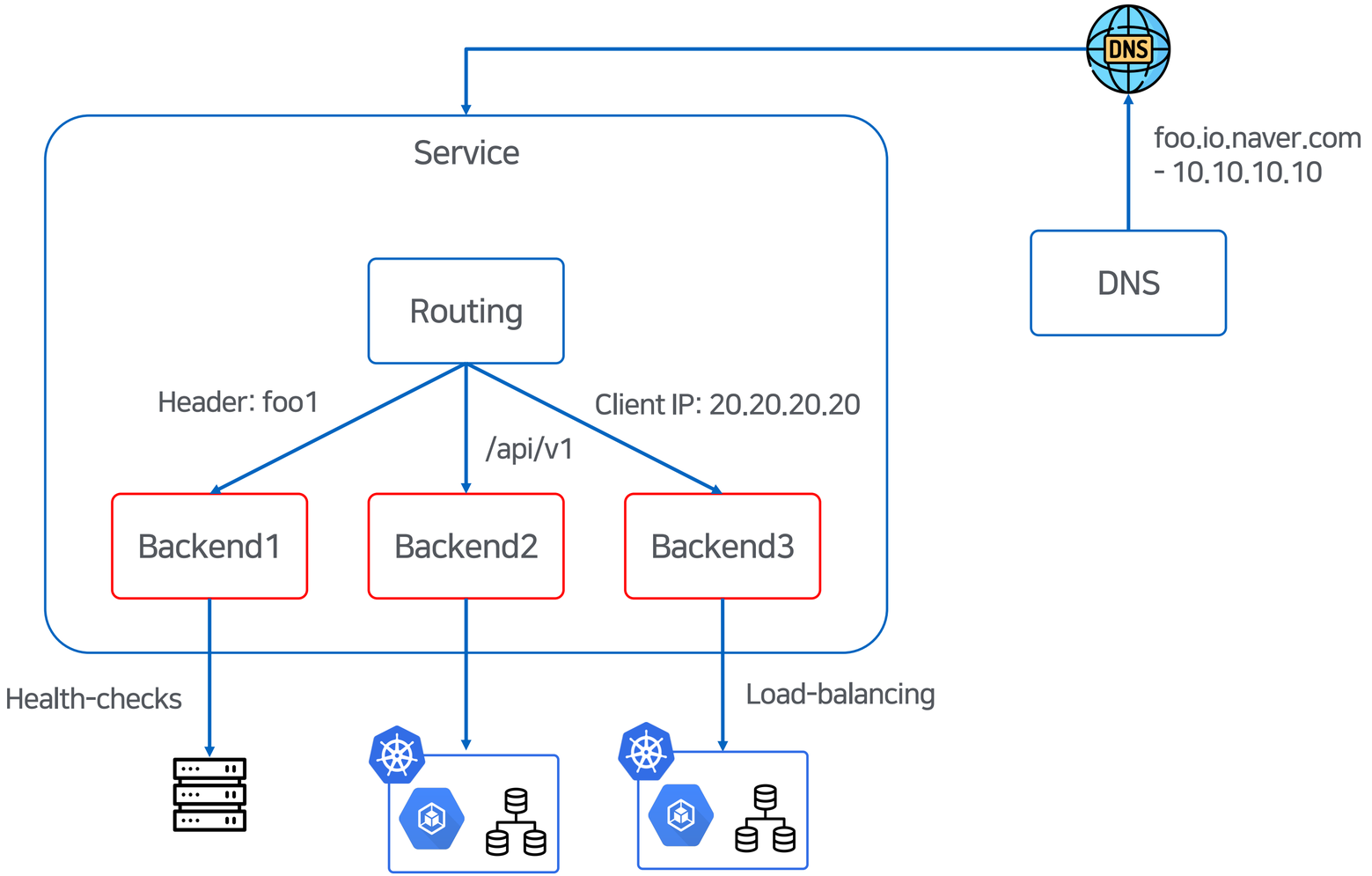
foo.io.naver.com
- 10.10.10.10

DNS

Routing

VirtualService

4.2 모델 디자인 – 생성되는 K8S 오브젝트



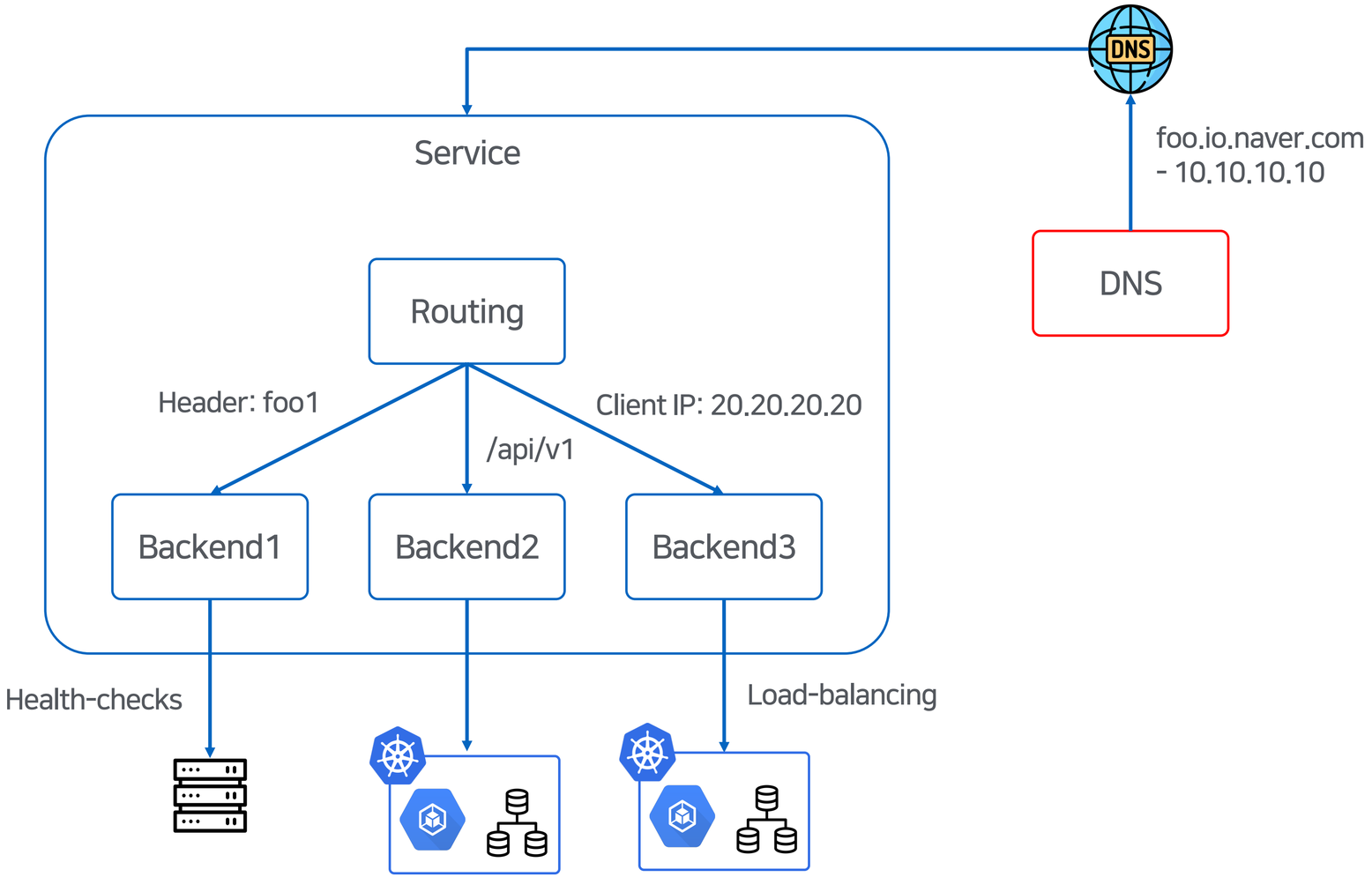
Backend

ServiceEntry

DestinationRule

EnvoyFilter
(Active Health Check)

4.2 모델 디자인 – 생성되는 K8S 오브젝트



DNS

DNS Record
(ETCD 데이터)

4.2 모델 디자인 – 구조체

Service

```
type Service struct {
  Name      string                 `json:"name" binding:"entityName"`
  Project   string                 `json:"project"`
  Revision  string                 `json:"revision"`
  Domains   []*domain.Domain       `json:"domains,omitempty"`
  CreatedAt  time.Time             `json:"createdAt"`
  UpdatedAt  time.Time             `json:"updatedAt"`
  DeletedAt  time.Time             `json:"deletedAt"`
  IsDeleting bool                  `json:"isDeleting"`
}
```

4.2 모델 디자인 – 구조체

Routing

```
type Routing struct {
  Service    string             `json:"service" binding:"entityName"`
  Revision   string             `json:"revision"`
  CreatedAt  time.Time          `json:"createdAt"`
  UpdatedAt  time.Time          `json:"updatedAt"`
  DeletedAt  time.Time          `json:"deletedAt"`
  IsDeleting bool               `json:"isDeleting"`
  HttpRoutes []*HttpRoute       `json:"httpRoutes"`
}
```

```
type HttpRoute struct {
  Name      *string             `json:"name,omitempty"`
  RouteRules []*RouteRule       `json:"routeRules"`
  Redirect   *Redirect          `json:"redirect,omitempty"`
  Rewrite    *Rewrite           `json:"rewrite,omitempty"`
  Backends   []*Backend         `json:"backends"`
  Retry      *Retry              `json:"retry,omitempty"`
  Timeout    *common.Duration    `json:"timeout,omitempty"`
  Headers    *Headers            `json:"headers,omitempty"`
}
```

4.2 모델 디자인 – 구조체

Routing

```
type RouteRule struct {
  Uri    *StringMatch `json:"uri,omitempty"`
  Method *StringMatch `json:"method,omitempty"`

  Headers map[string]StringMatch `json:"headers,omitempty"`
  WithoutHeaders map[string]StringMatch `json:"withoutHeaders,omitempty"`
  QueryParams map[string]StringMatch `json:"queryParams,omitempty"`

  IgnoreUriCase bool `json:"ignoreUriCase"`
}
```

4.2 모델 디자인 – 구조체

Backend

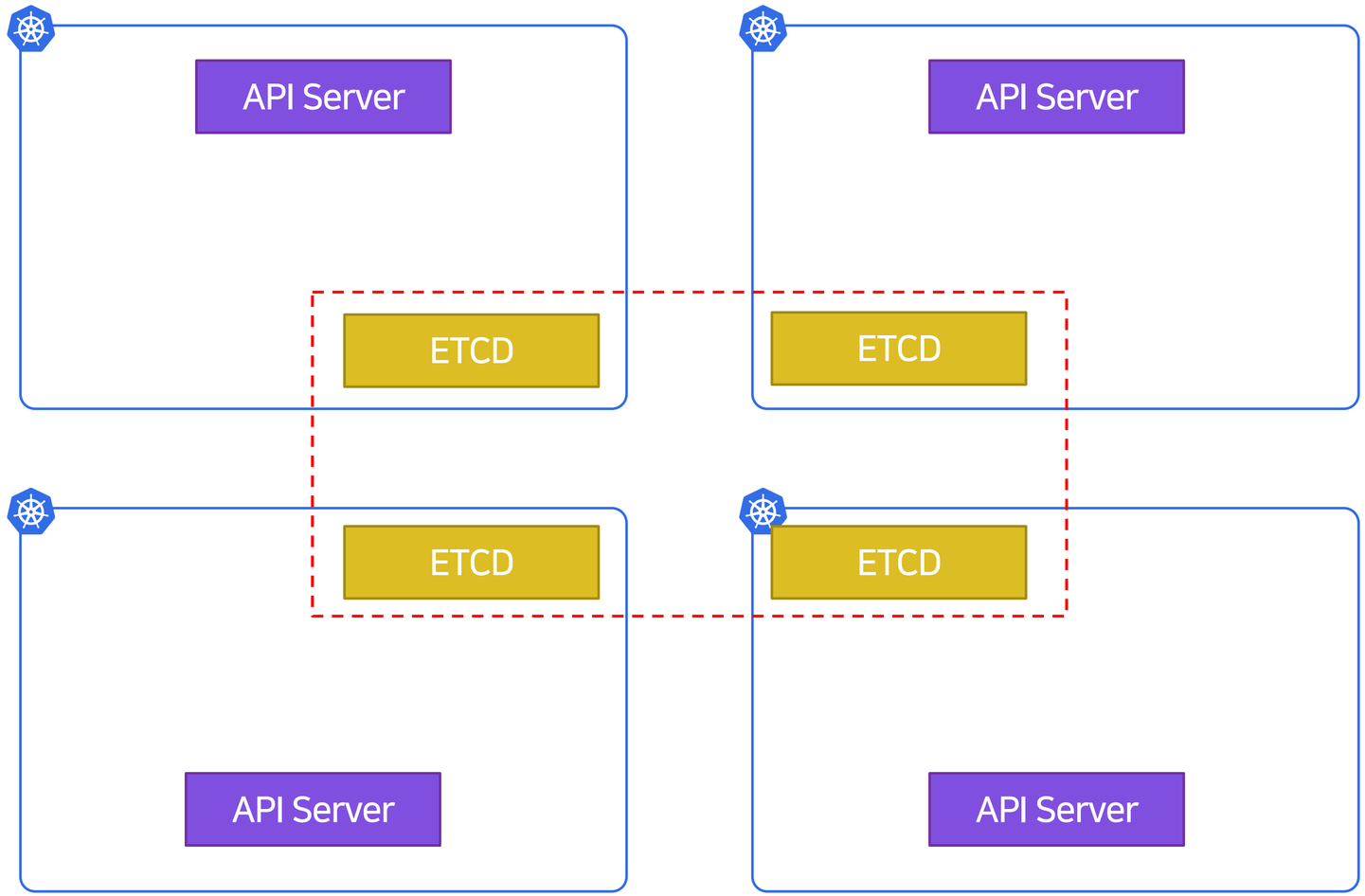
```
type Backend struct {
  Name        string                 `json:"name" binding:"entityName"`
  Service     string                 `json:"service" binding:"entityName"`
  Endpoints    []Endpoints           `json:"endpoints"`
  Resolution   string                `json:"resolution"`
  Revision    string                 `json:"revision"`
  CreatedAt    time.Time             `json:"createdAt"`
  UpdatedAt    time.Time             `json:"updatedAt"`
  DeletedAt    time.Time             `json:"deletedAt"`
  IsDeleting   bool                  `json:"isDeleting"`
  BackendOptions *BackendOptions `json:"backendOptions,omitempty"`
}
```

4.2 모델 디자인 – 구조체

Backend

```
type BackendOptions struct {
  Name               string                `json:"name"`
  Service            string                `json:"service"`
  Backend            string                `json:"backend"`
  ConnectionSettings *ConnectionSettings   `json:"connectionSettings,omitempty"`
  LoadBalancer       *LoadBalancer         `json:"loadbalancer,omitempty"`
  ActiveHealthChecks []*ActiveHealthCheck  `json:"activeHealthChecks,omitempty"`
}
```

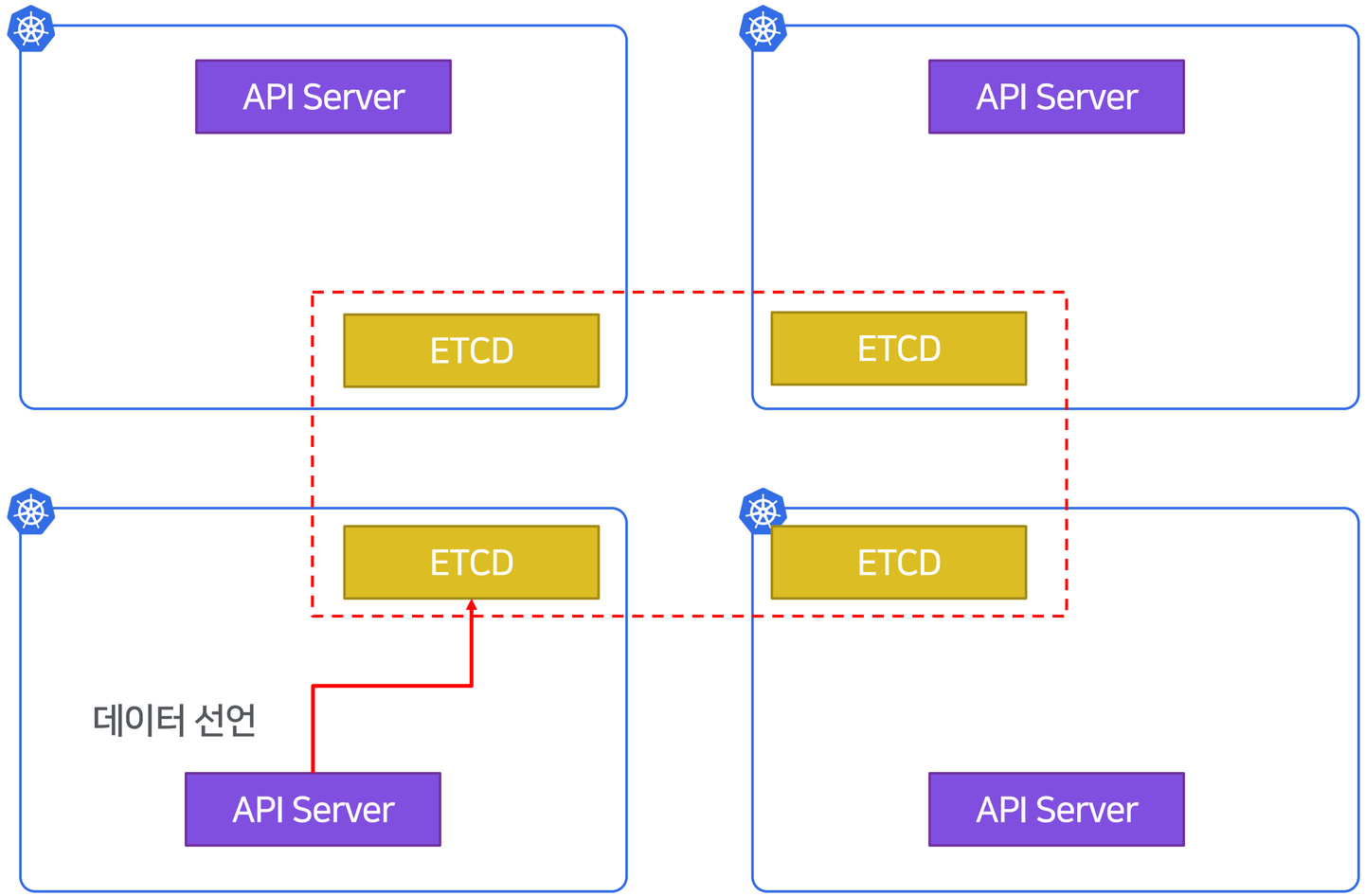
4.3 ETCD를 활용한 Producer/Consumer 구조



ETCD Global clustering

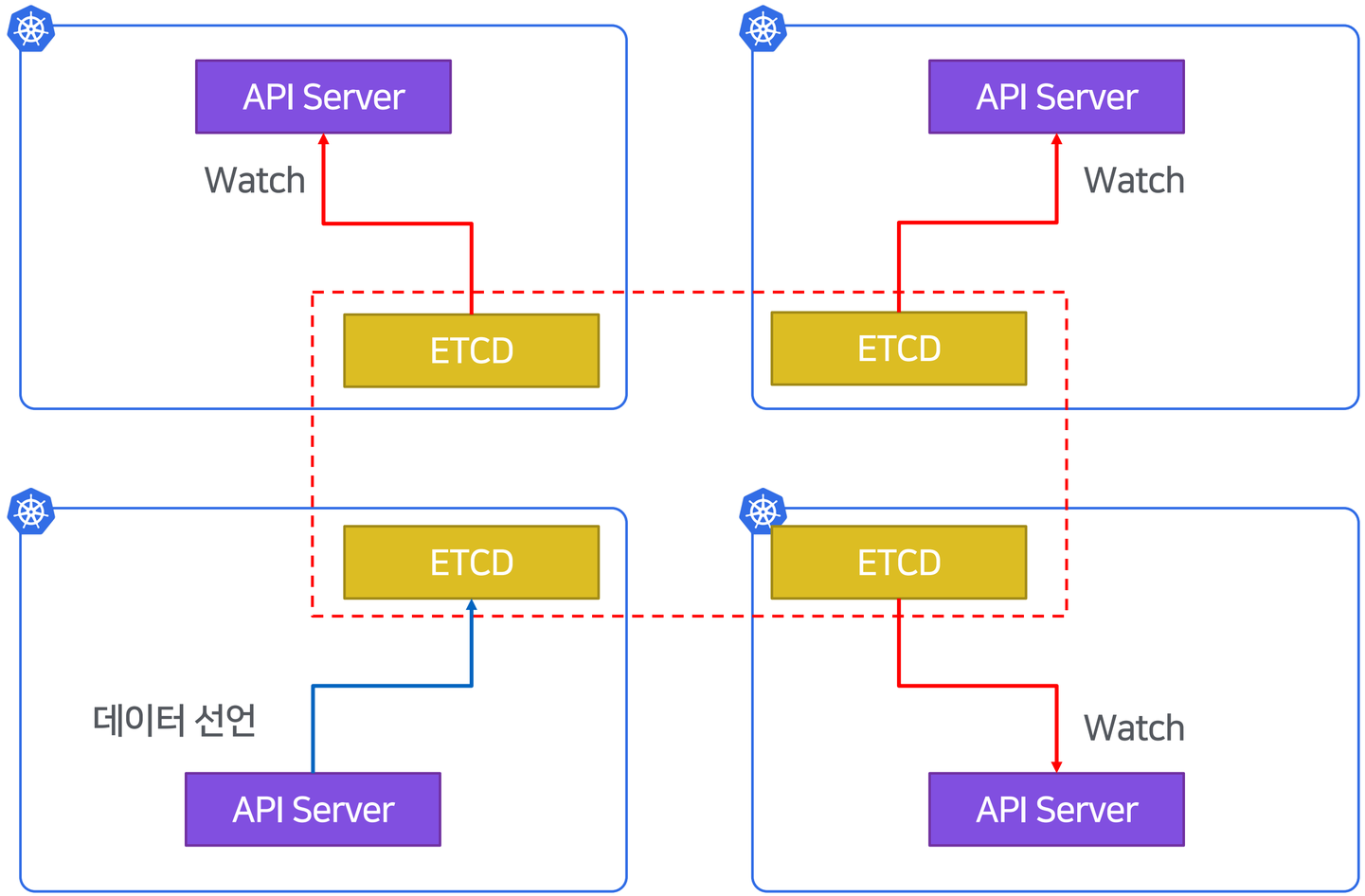
4.3 ETCD를 활용한 Producer/Consumer 구조

DEVIEW 2021



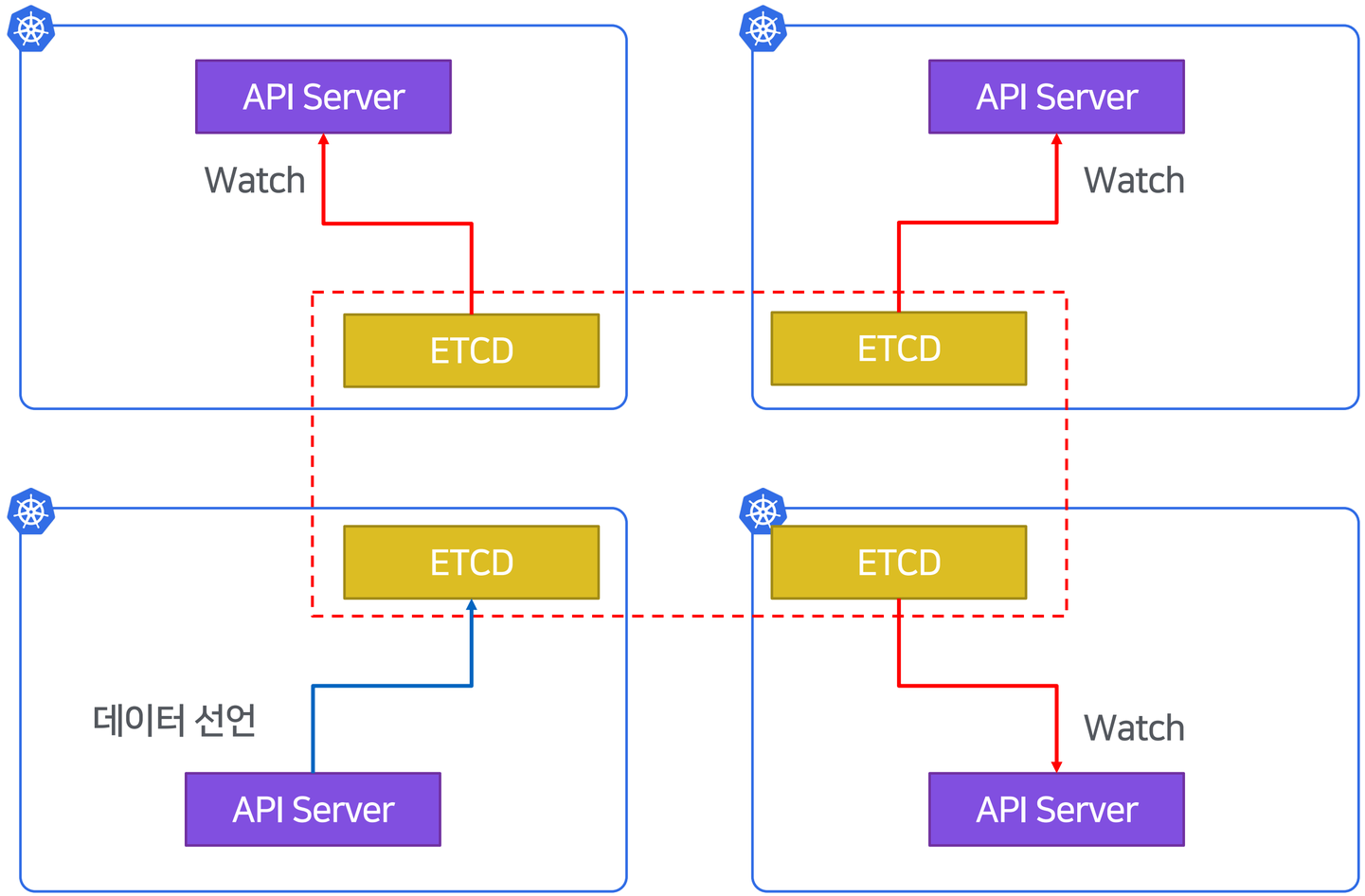
4.3 ETCD를 활용한 Producer/Consumer 구조

DEVIEW 2021



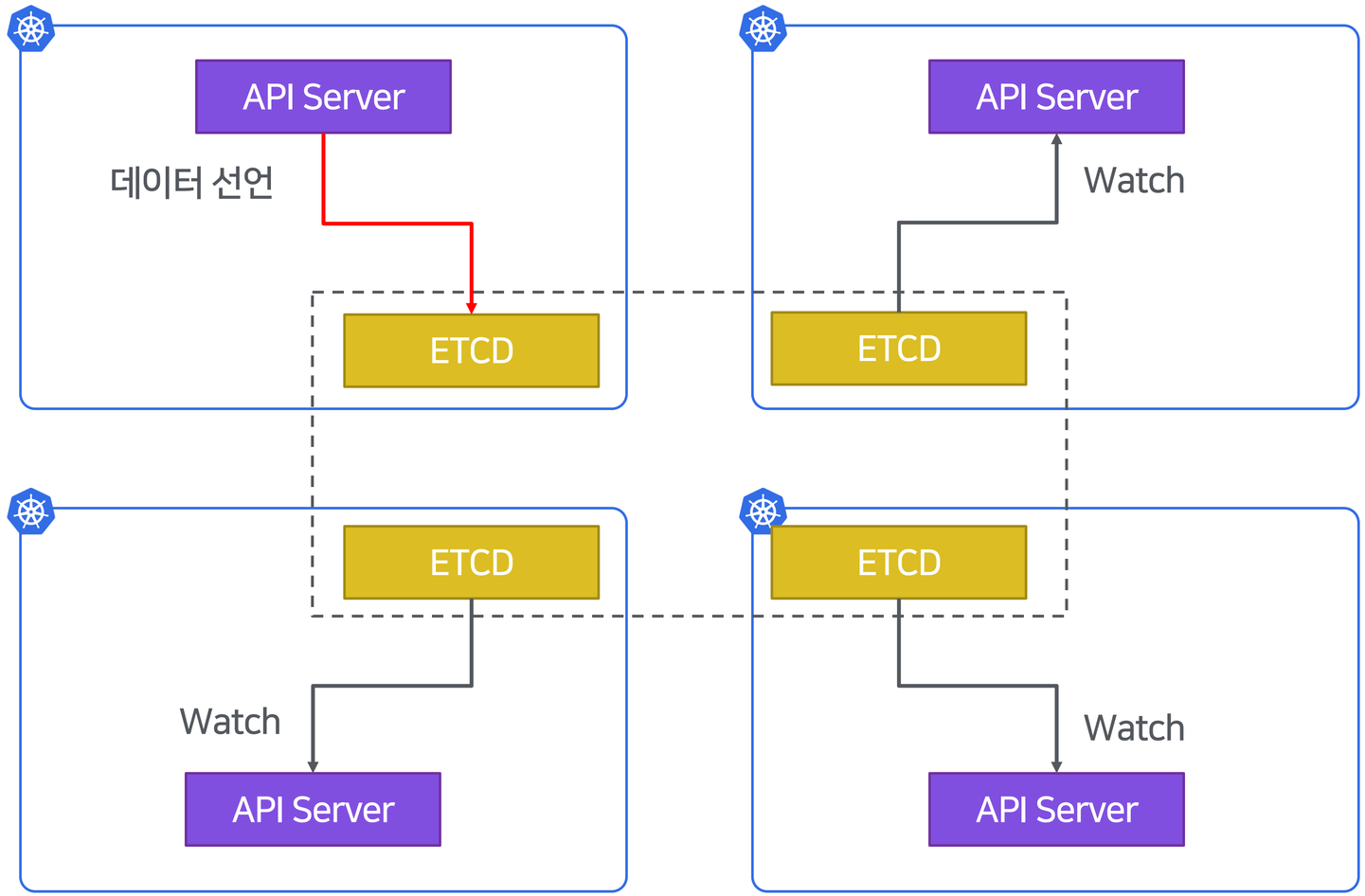
4.3 ETCD를 활용한 Producer/Consumer 구조

DEVIEW 2021



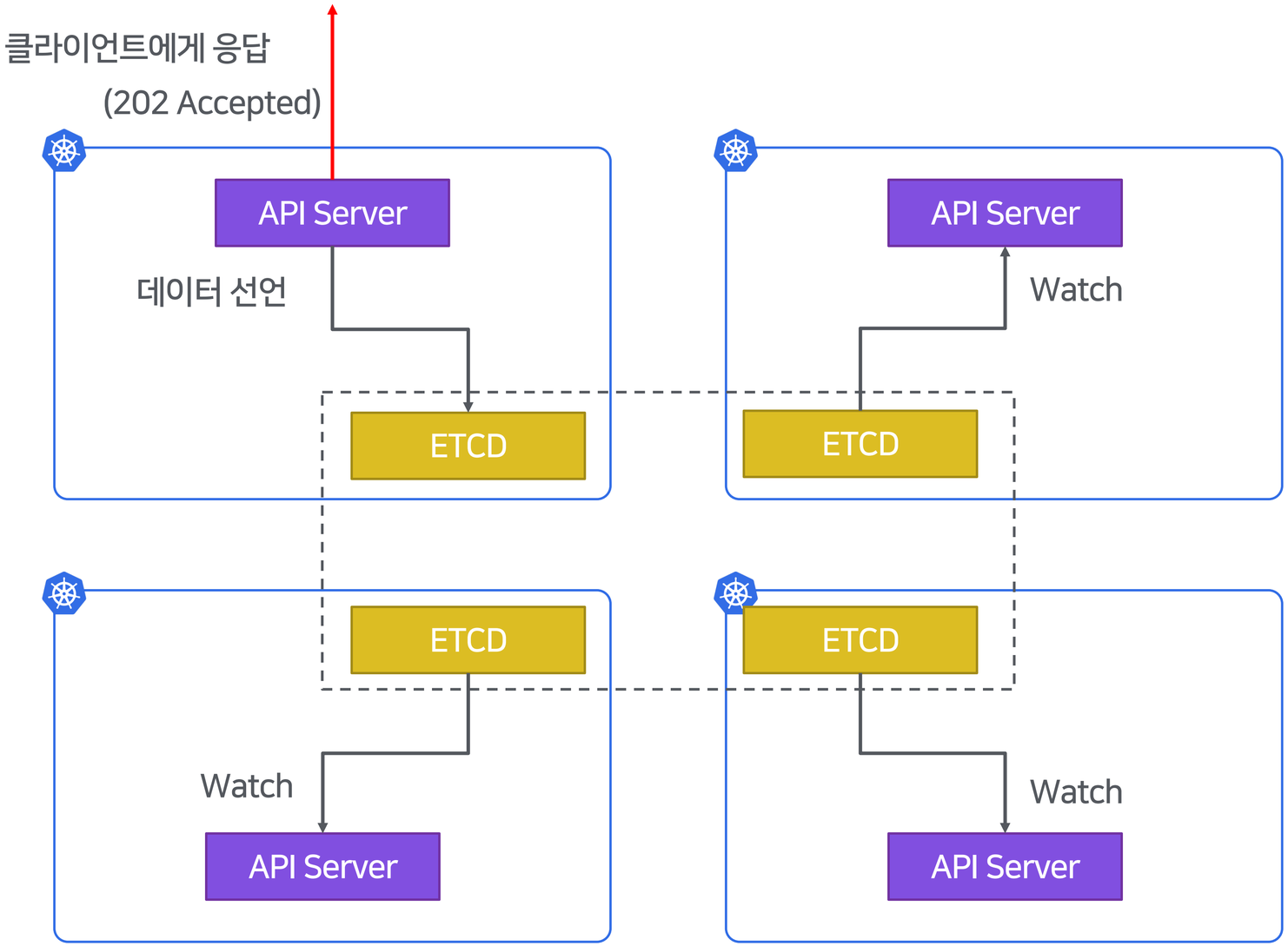
Producer / Consumer

4.3 ETCD를 활용한 Producer/Consumer 구조



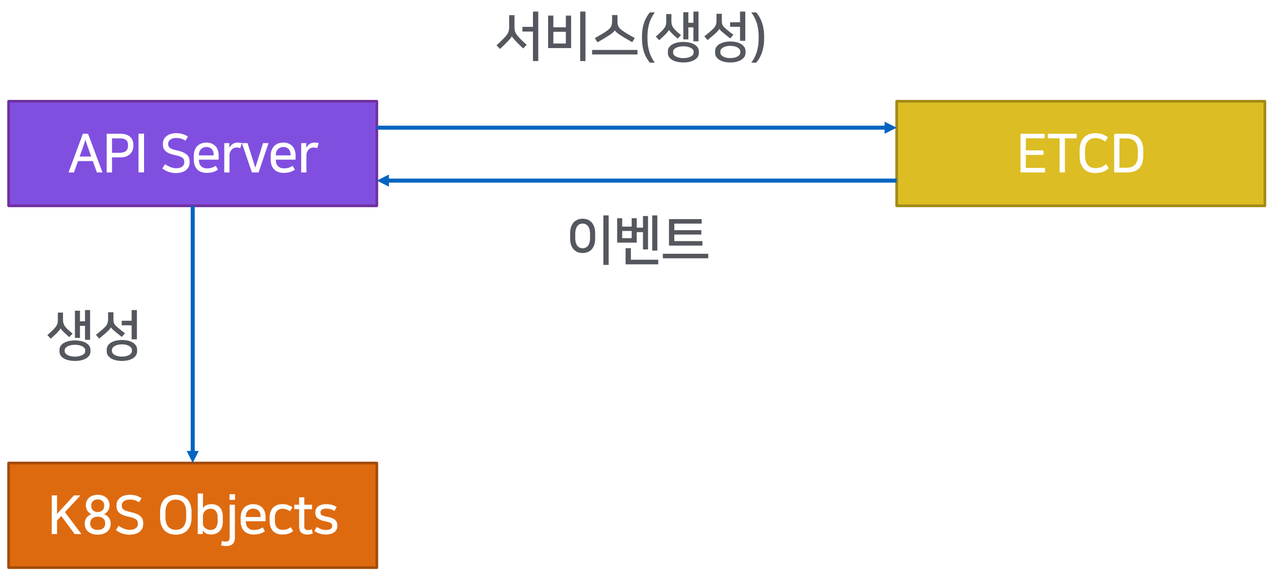
Producer / Consumer

4.3 ETCD를 활용한 Producer/Consumer 구조

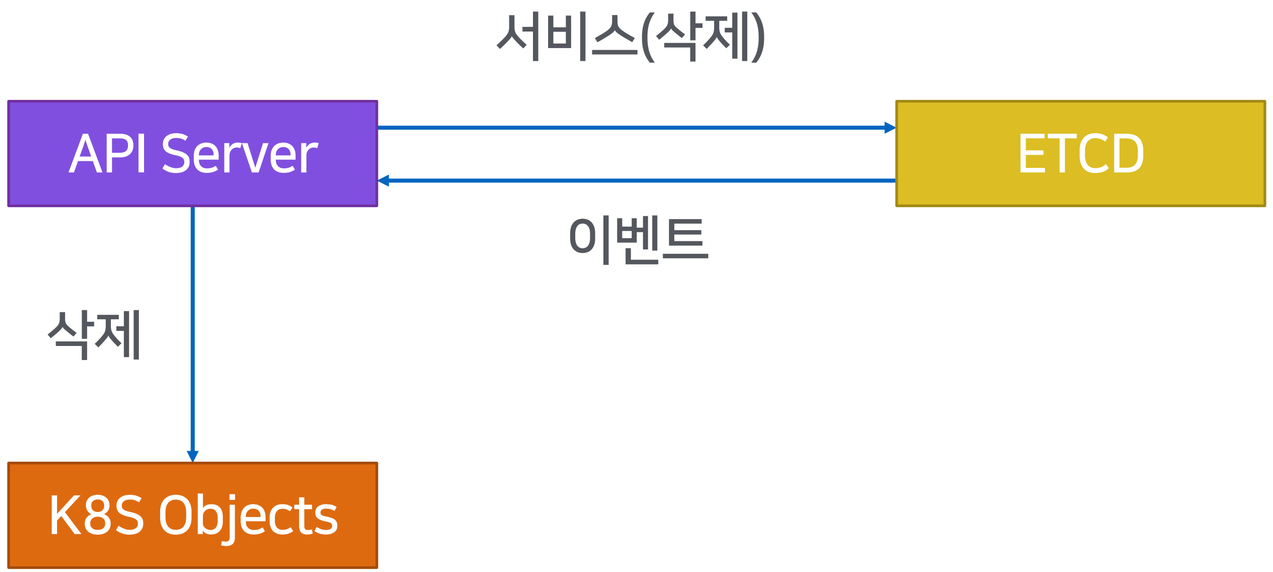


비동기 처리

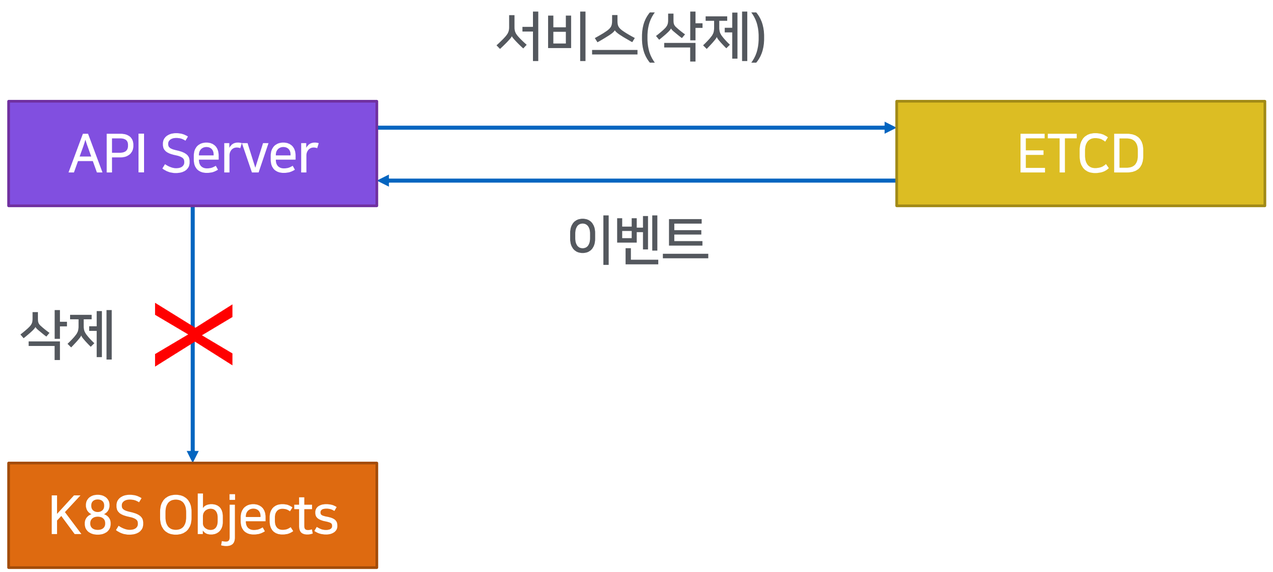
4.4 장애 고려 사항



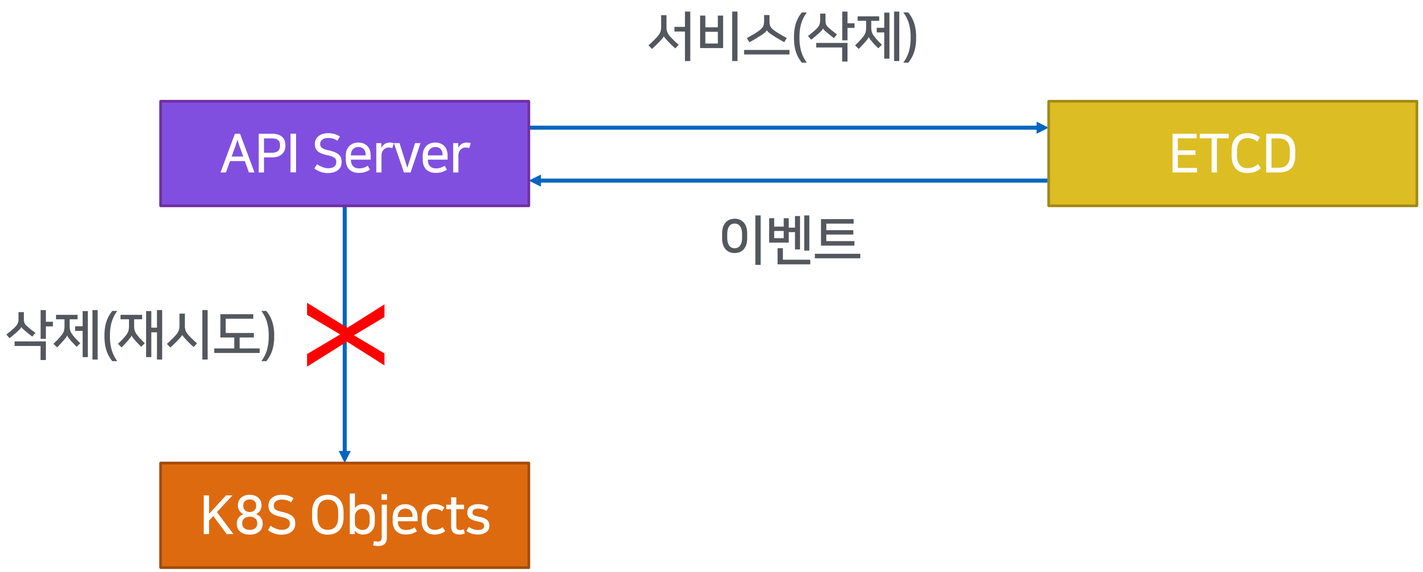
4.4 장애 고려 사항



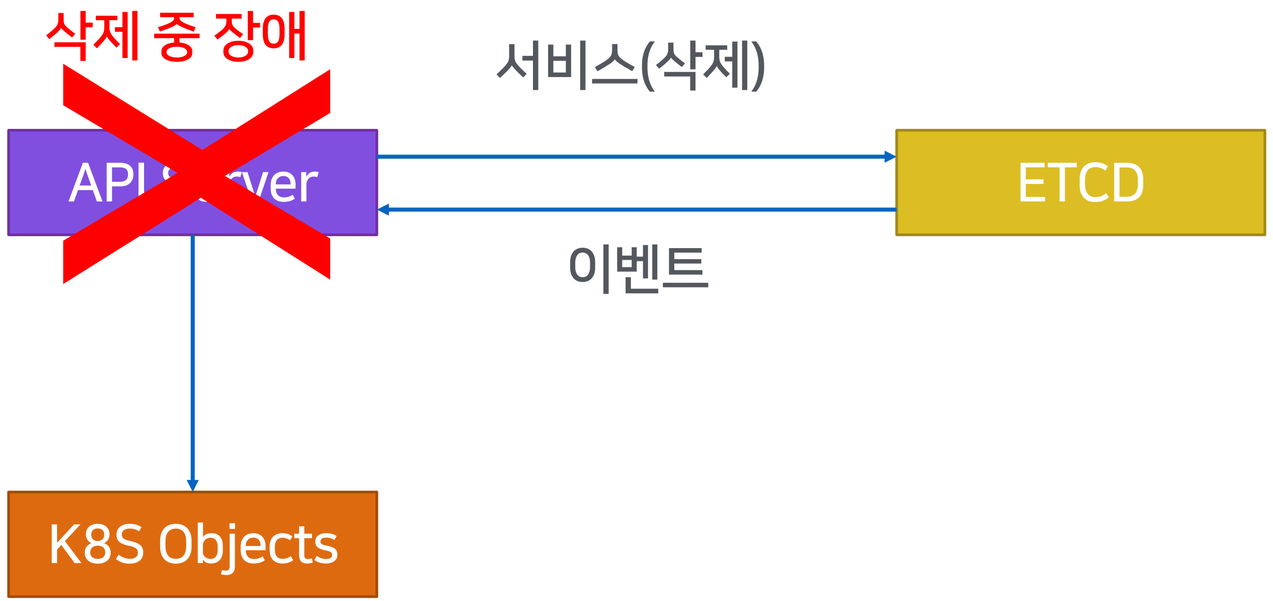
4.4 장애 고려 사항



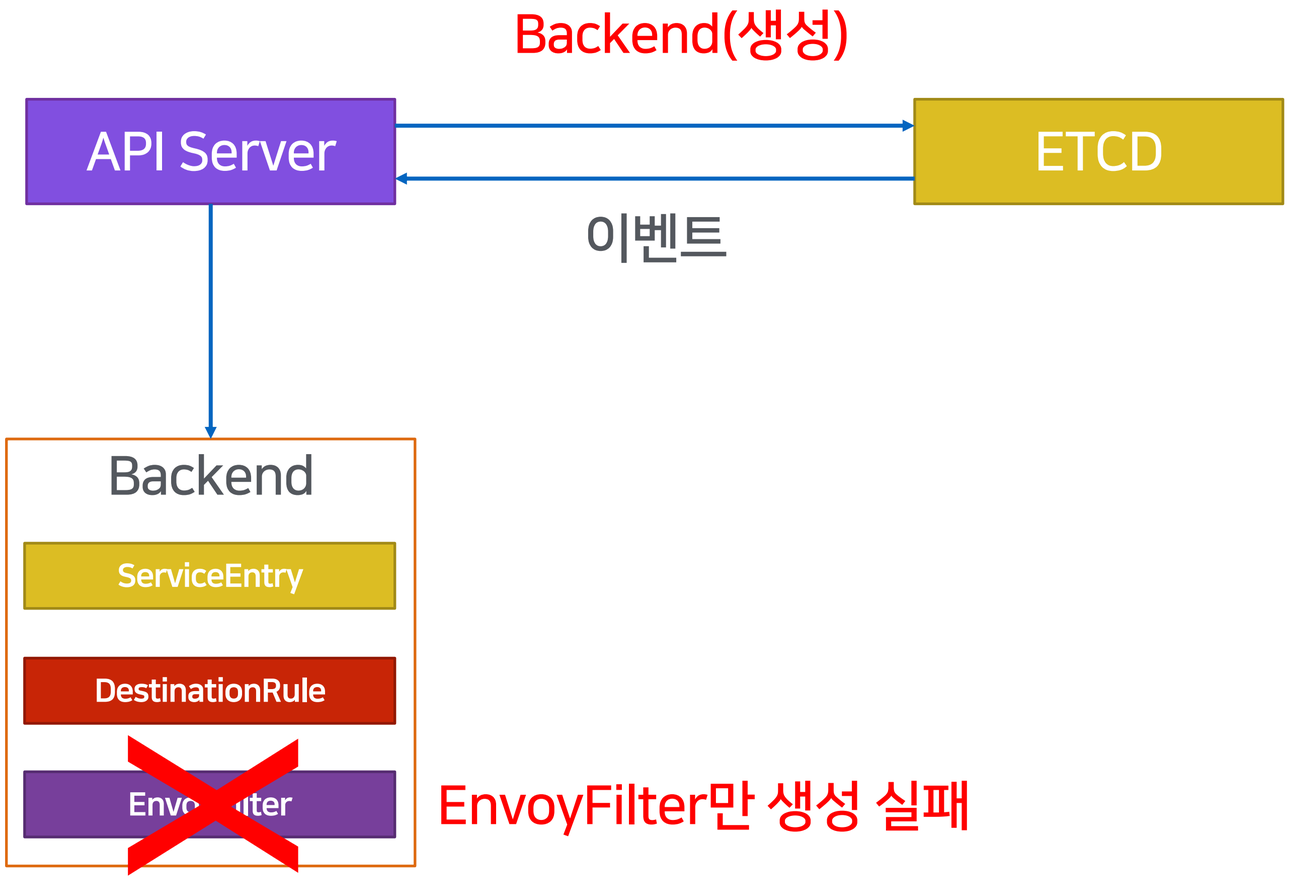
4.4 장애 고려 사항



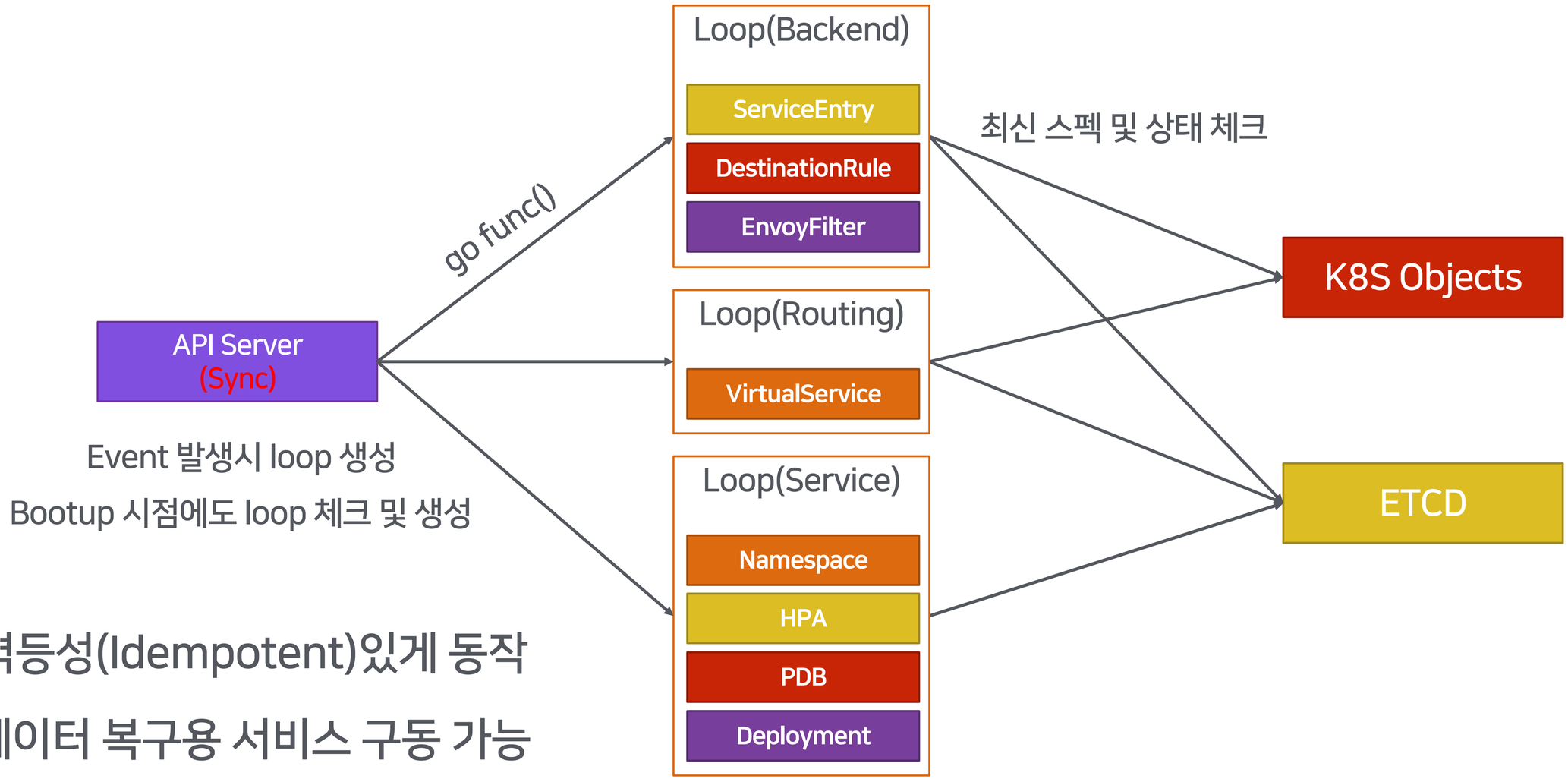
4.4 장애 고려 사항



4.4 장애 고려 사항


Backend(생성)
API Server
ETCD
이벤트
Backend
ServiceEntry
DestinationRule
EnvoyFilter만 생성 실패

4.4 장애 고려 사항



멱등성(Idempotent)있게 동작

데이터 복구용 서비스 구동 가능

4.5 Reconciliation Loop - 구현

DEVIEW 2021

```
func (loop *Loop) StartLoop(ctx context.Context, loopKey string, dataKey string, initResources func() ([]LoopOp, string, TerminationOp, error))

  for {
  ...
    loop.keepLoopKey(ctx, loopKey)
```

Loop 키 획득(transaction)

```
    operations, revision, terminationOp, err := initResources()
```

연산 획득

```
    for _, op := range operations {  }
```

연산 수행

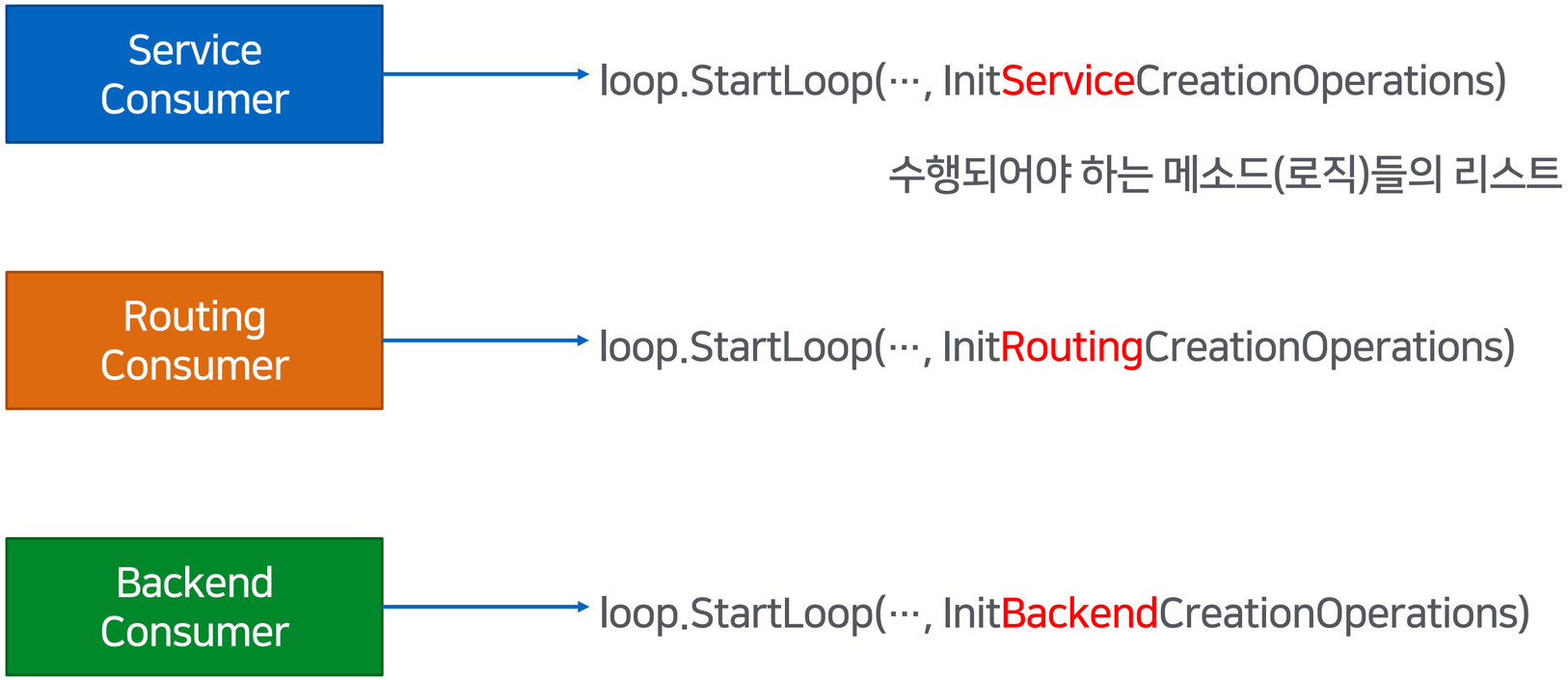
```
    if terminationOp(ctx, eventKey, revision, opErrs) { }
```

종료 여부 판단

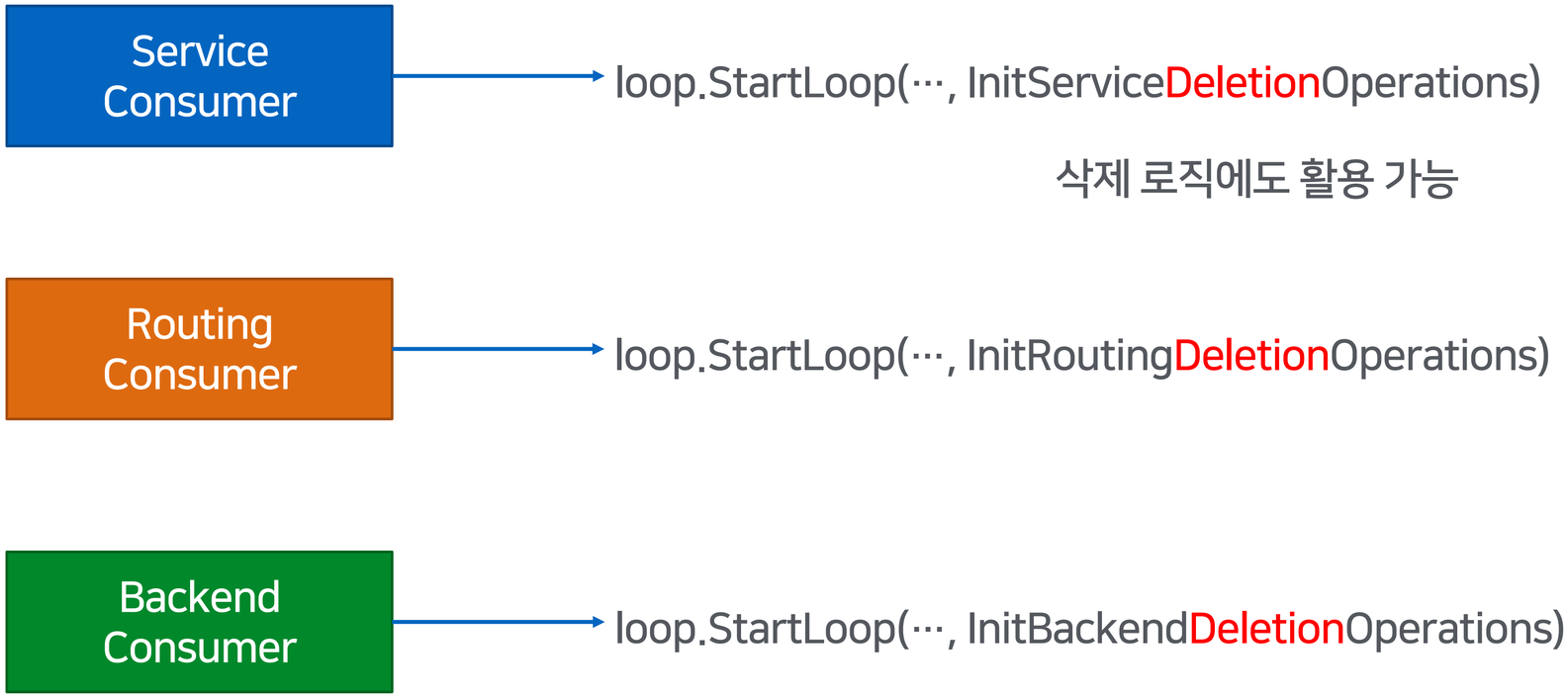
```
    if loop.isTimedOut(start) {
      go loop.cleanUpLoopKey(ctx, loopKey)
      return
    }
  ...
  }
```

Timeout 처리

4.5 Reconciliation Loop - 활용



4.5 Reconciliation Loop - 활용

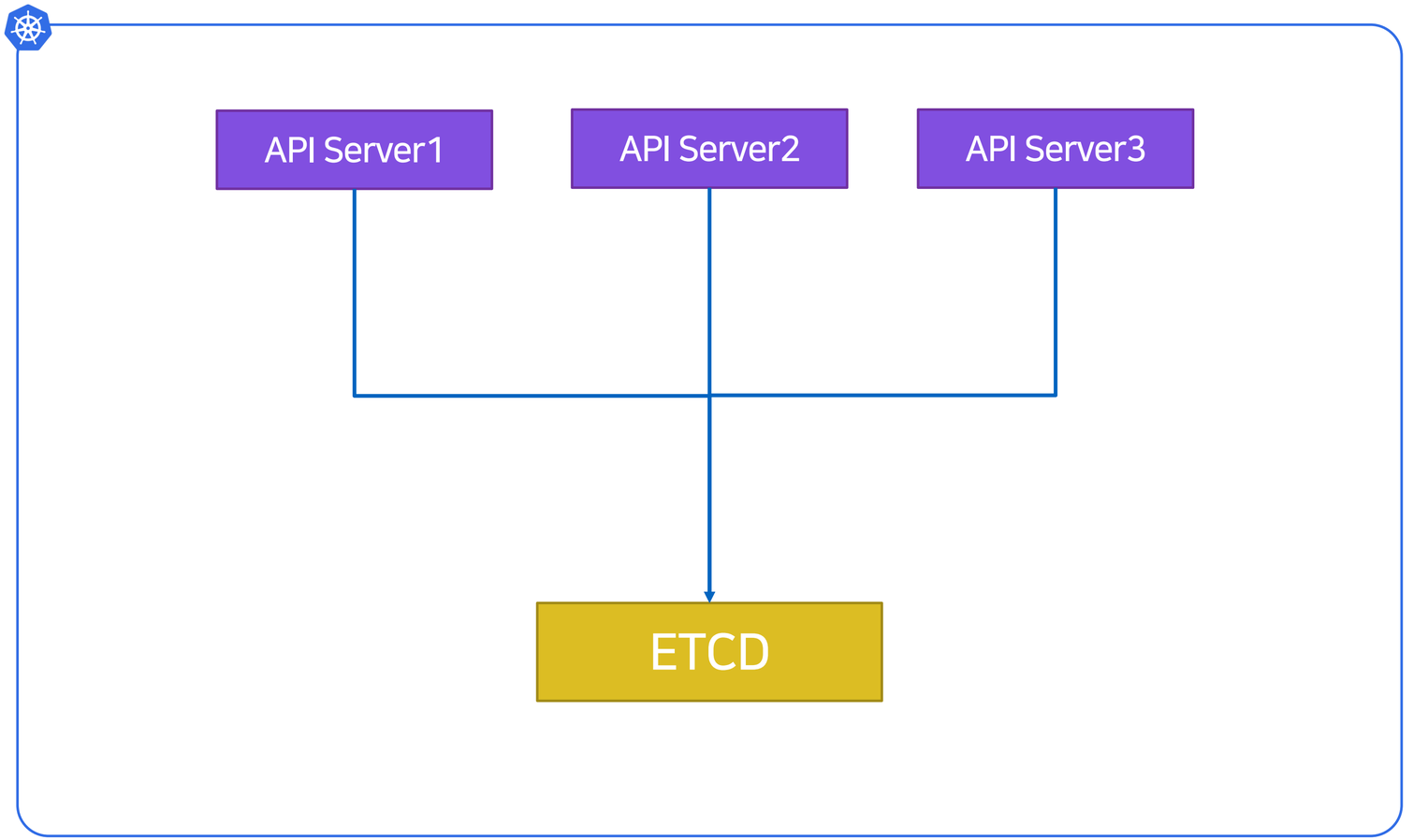


4.5 Reconciliation Loop - 종료

루프의 종료 조건

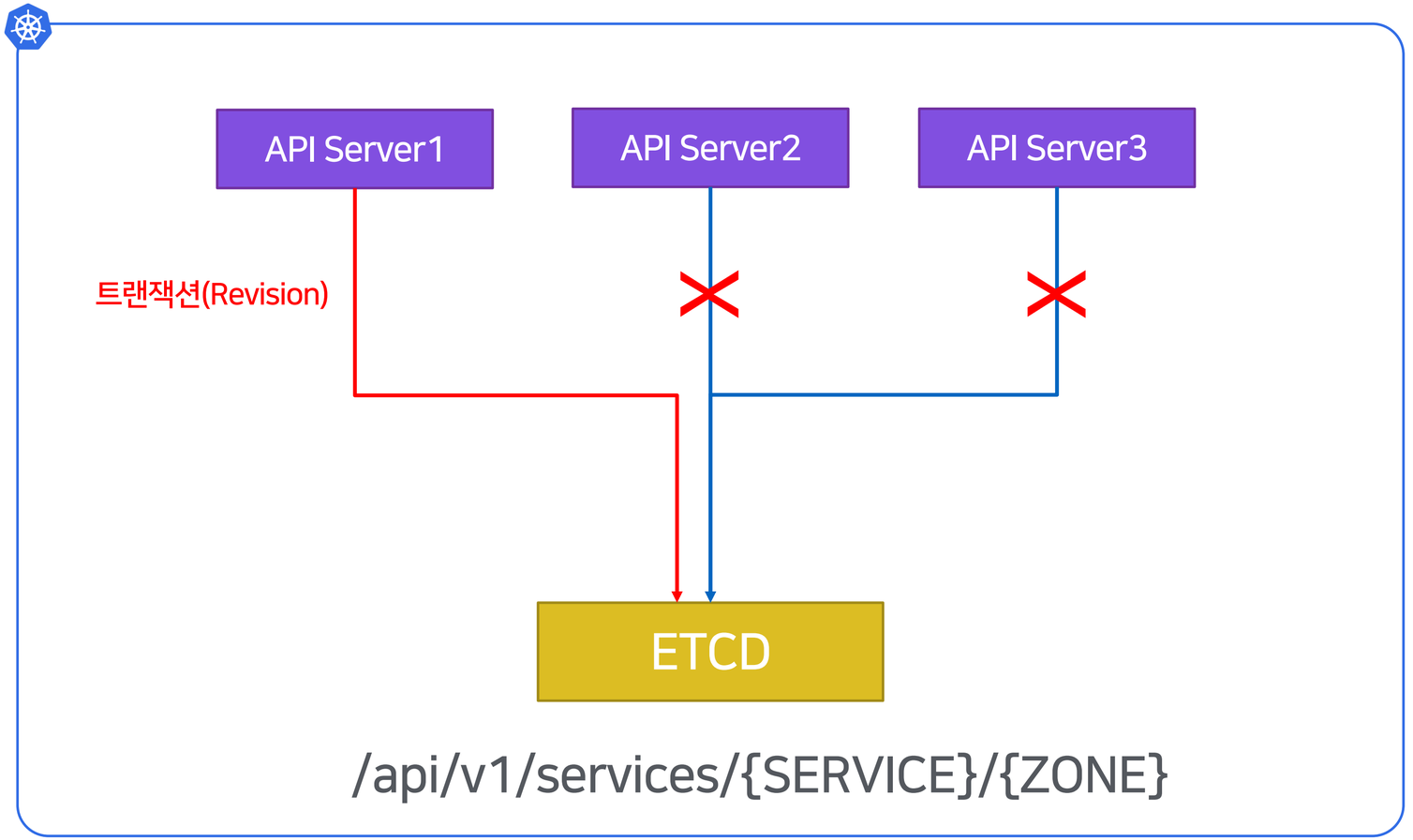
1. 연산 수행 시 에러 발생 안함
2. 최신 데이터 반영 완료
3. 타임아웃 발생 시 종료

4.6 Concurrency Control

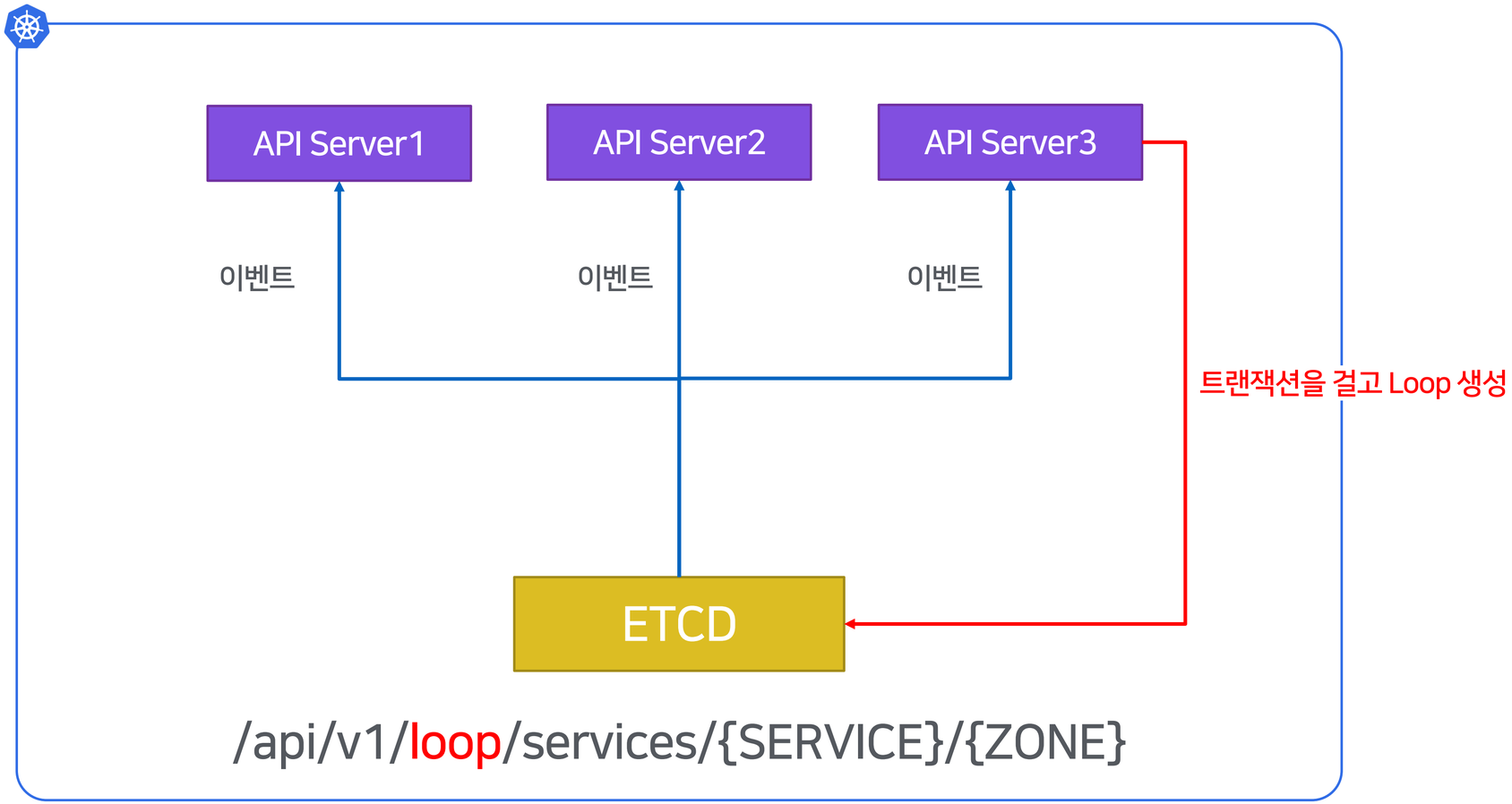


4.6 Concurrency Control

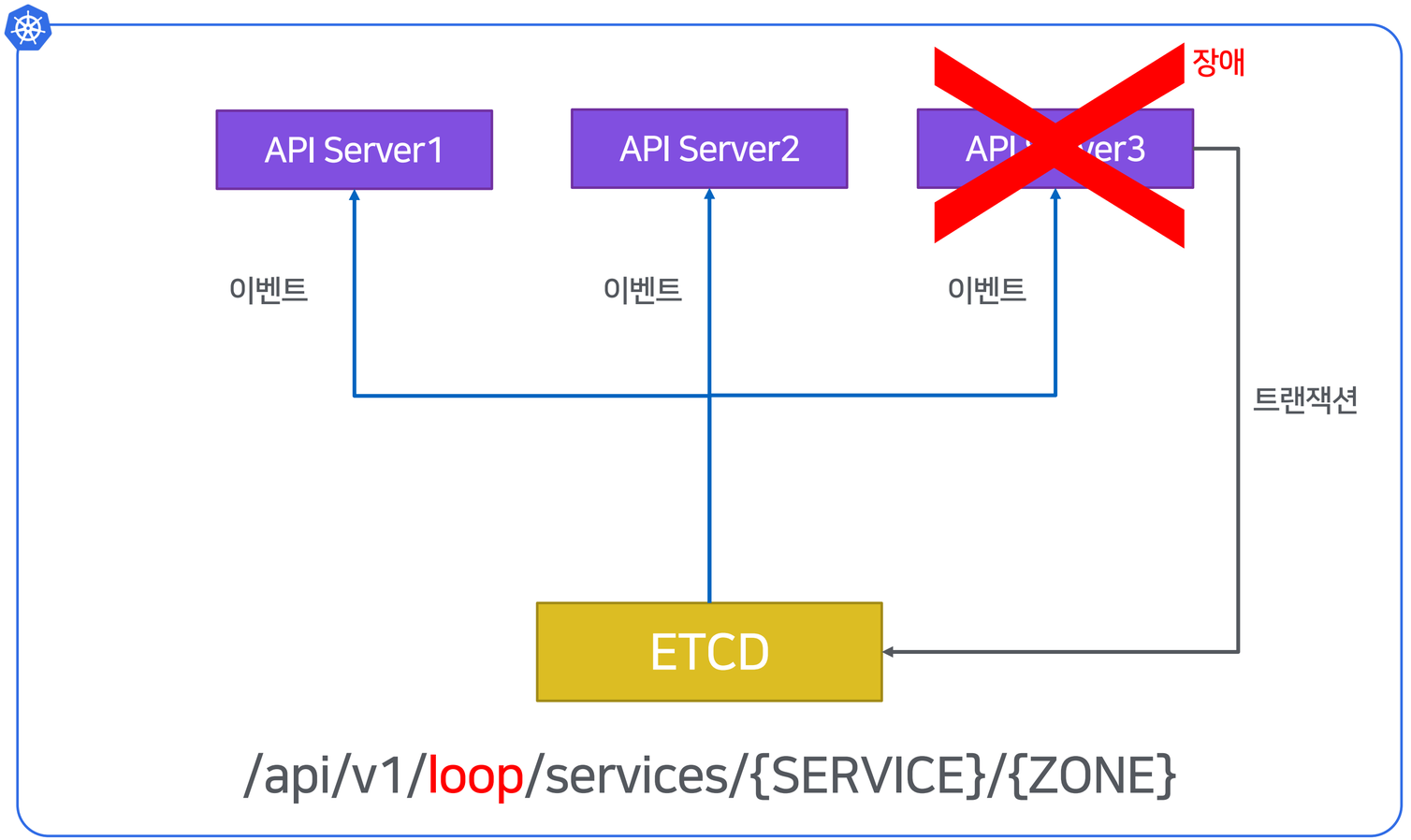
작업 이후 클라이언트에게 응답



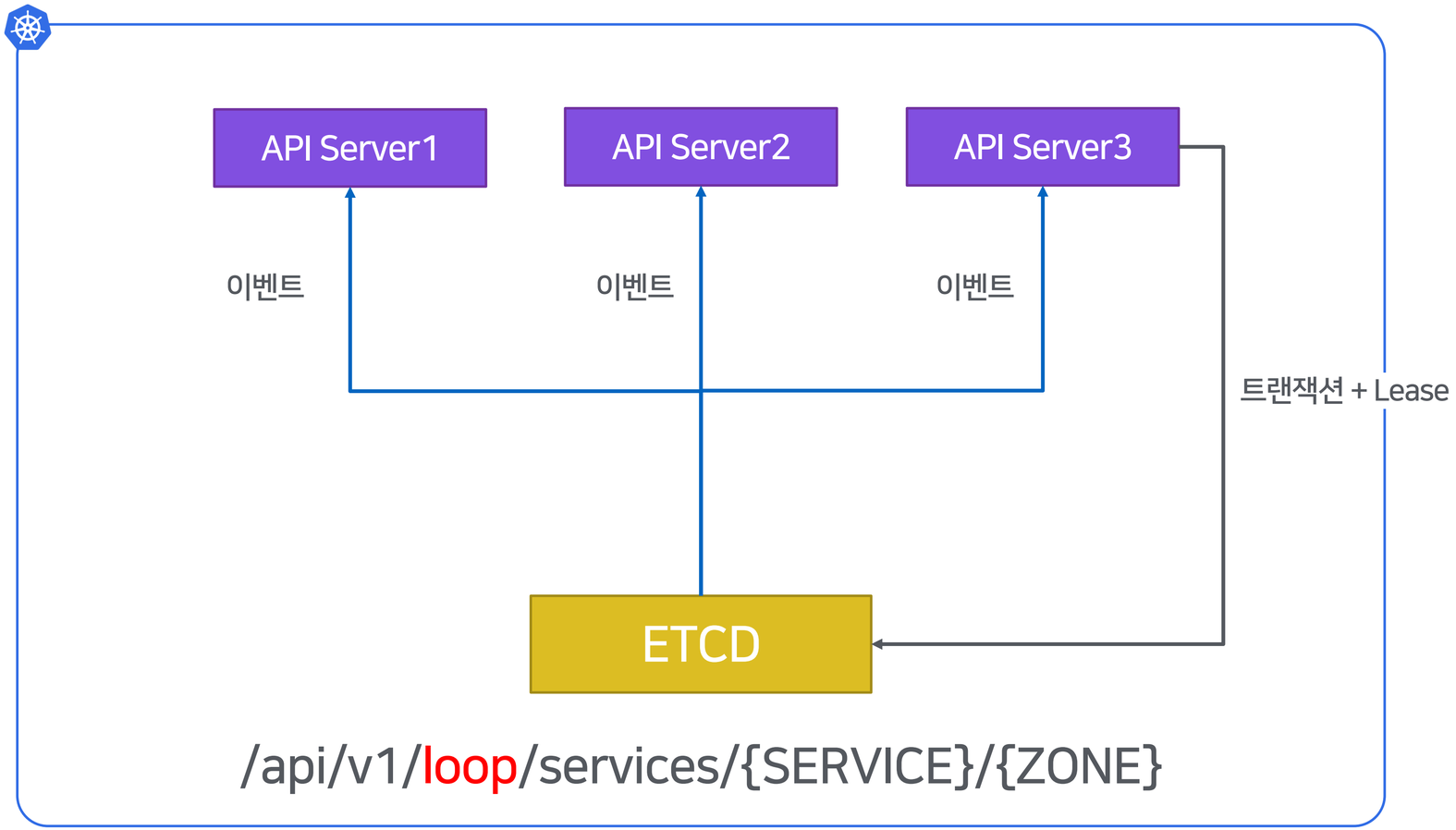
4.6 Concurrency Control



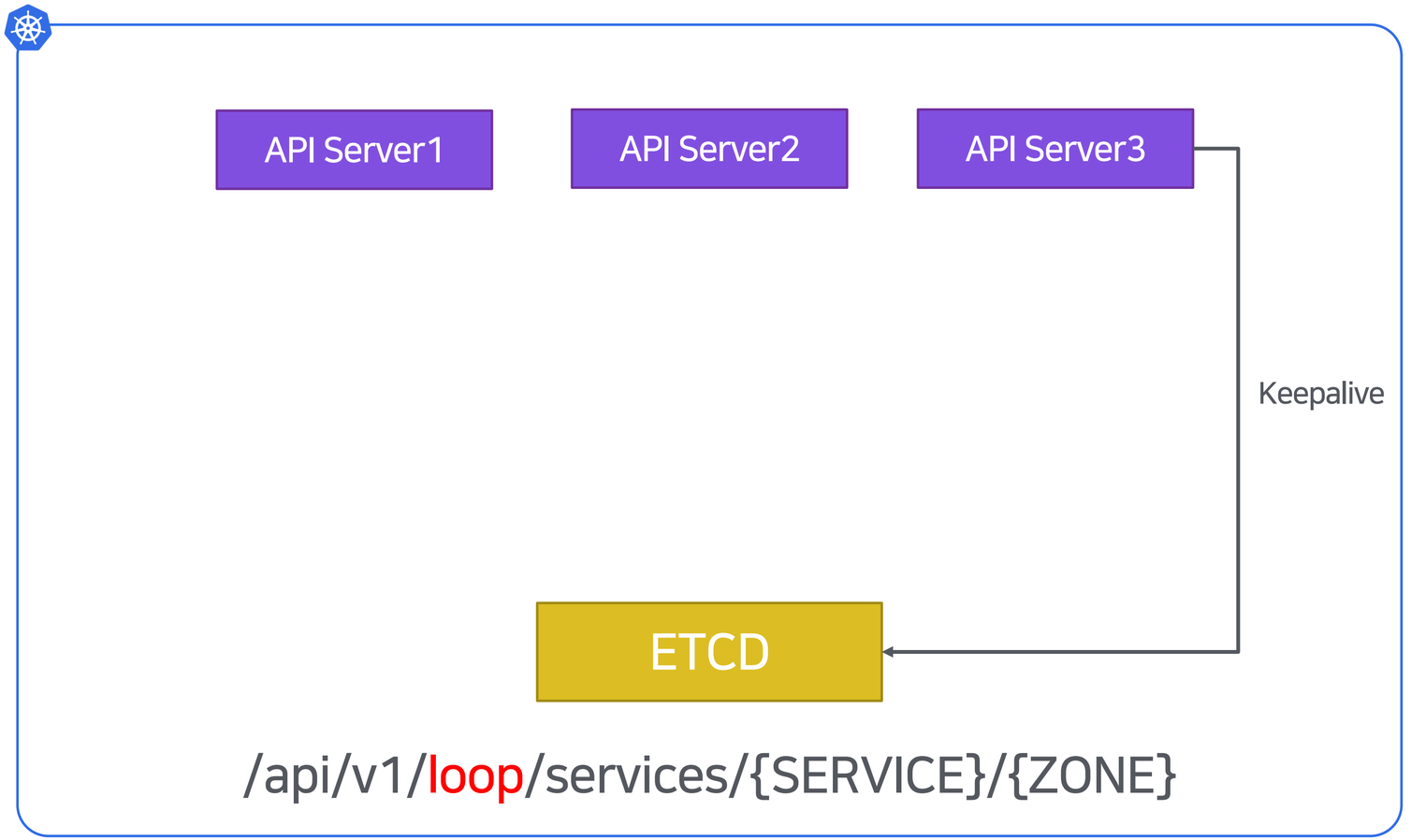
4.6 Concurrency Control



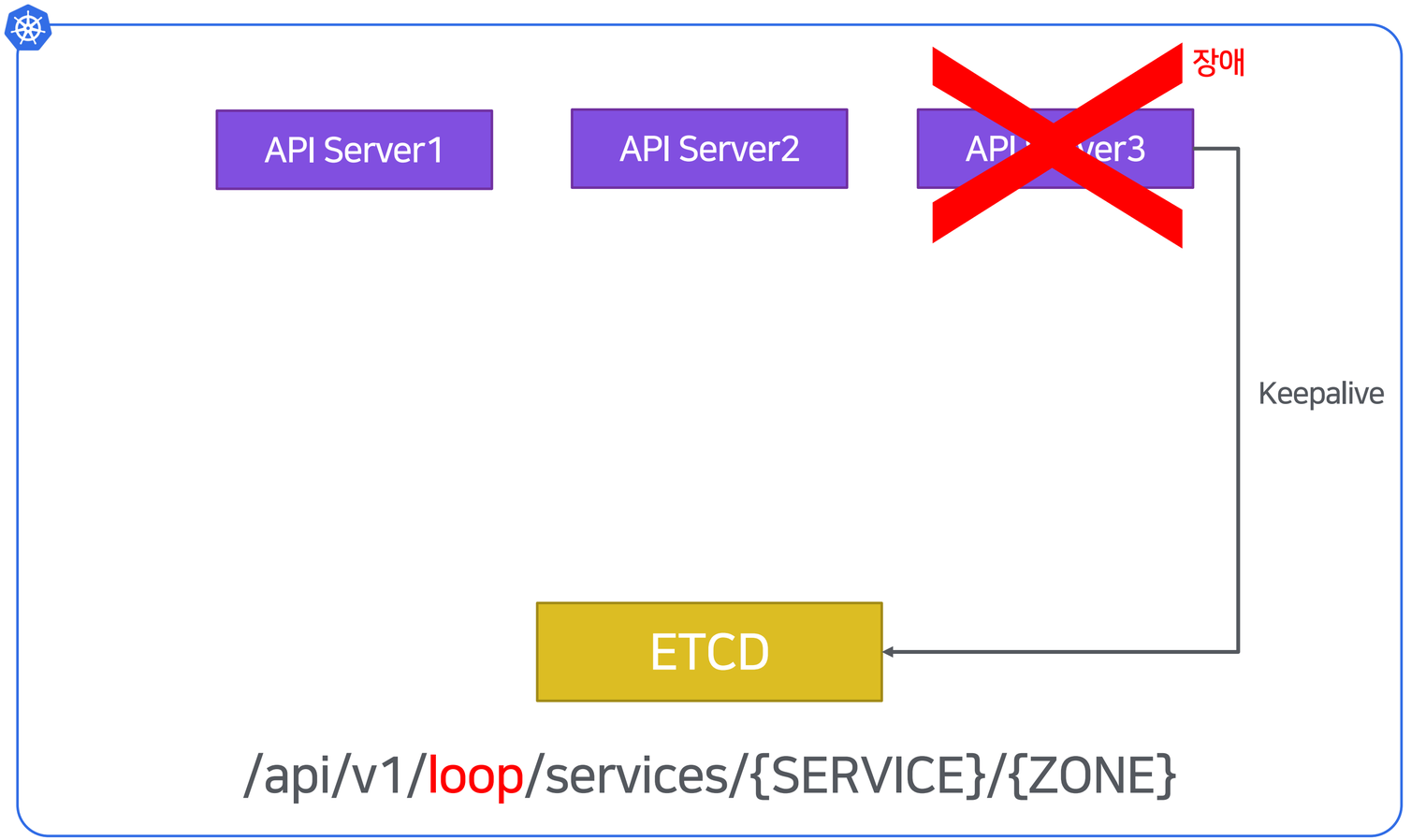
4.6 Concurrency Control



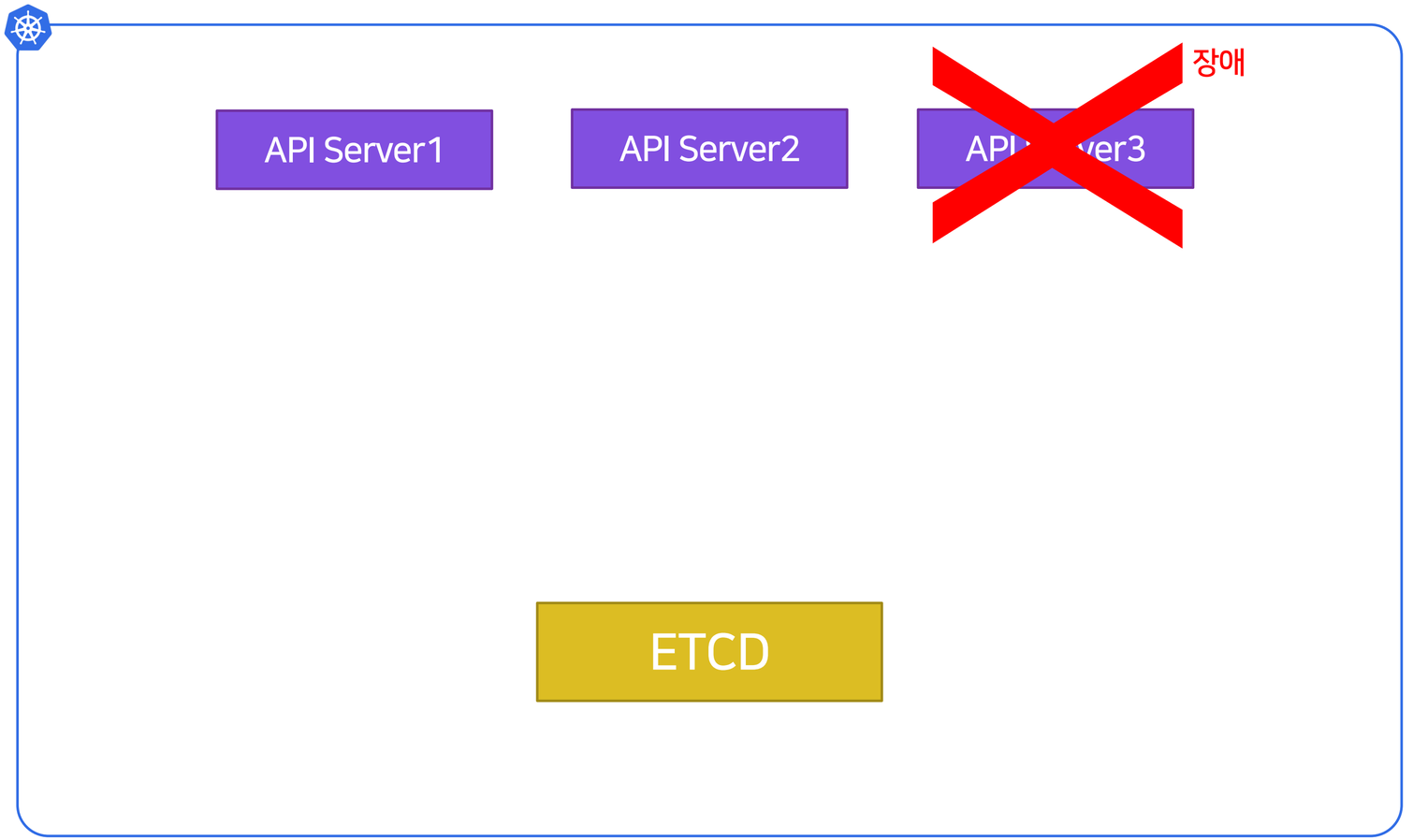
4.6 Concurrency Control



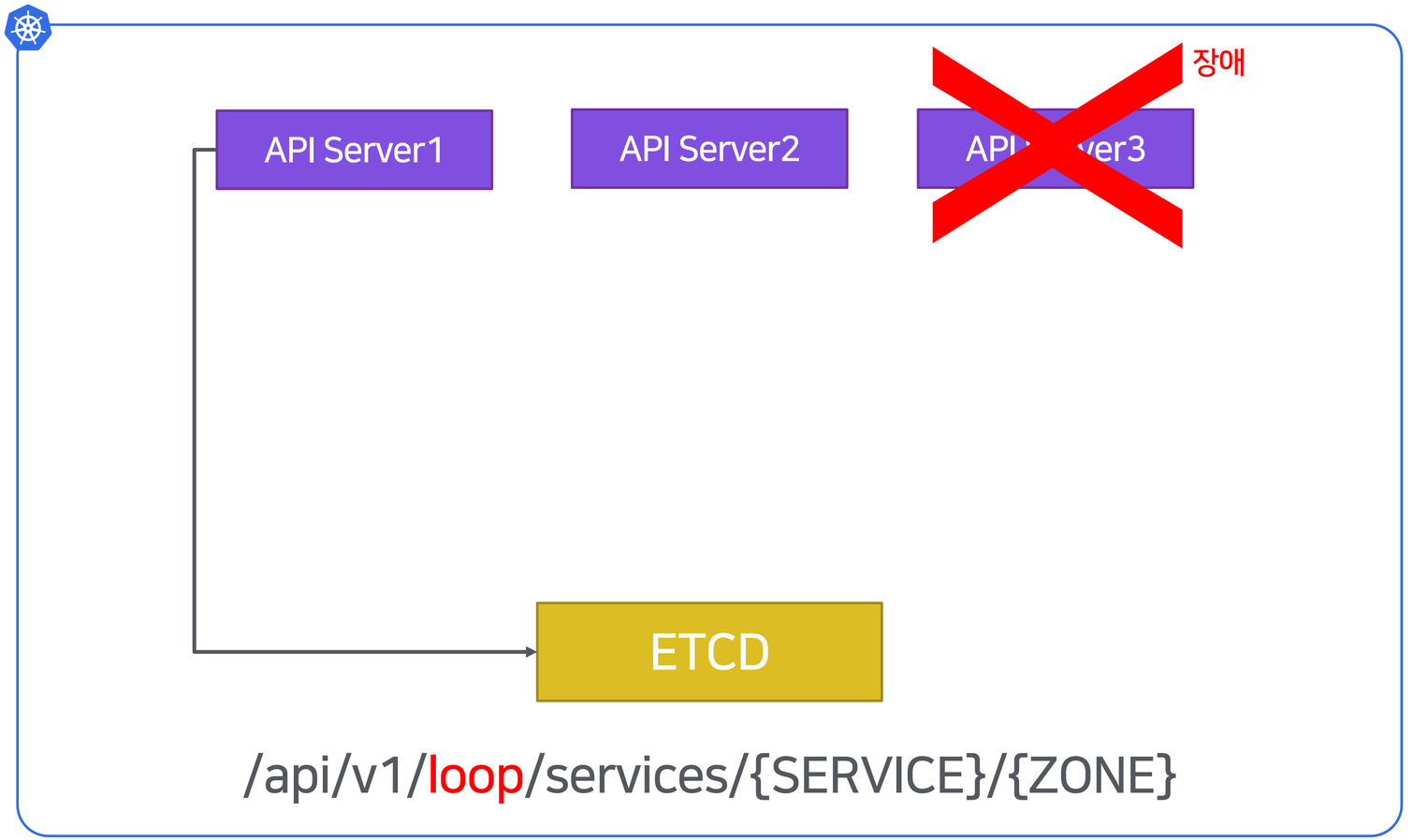
4.6 Concurrency Control



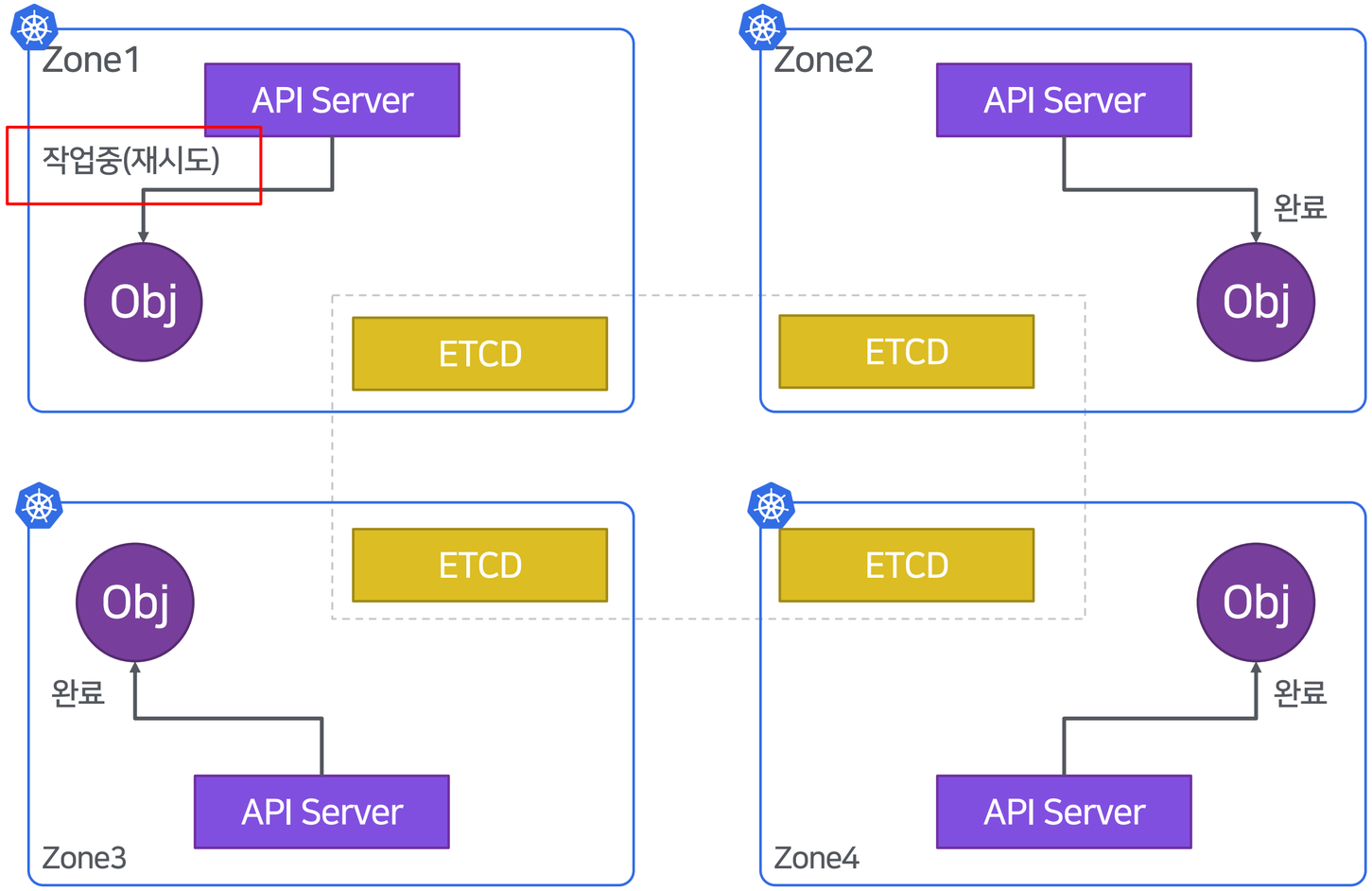
4.6 Concurrency Control



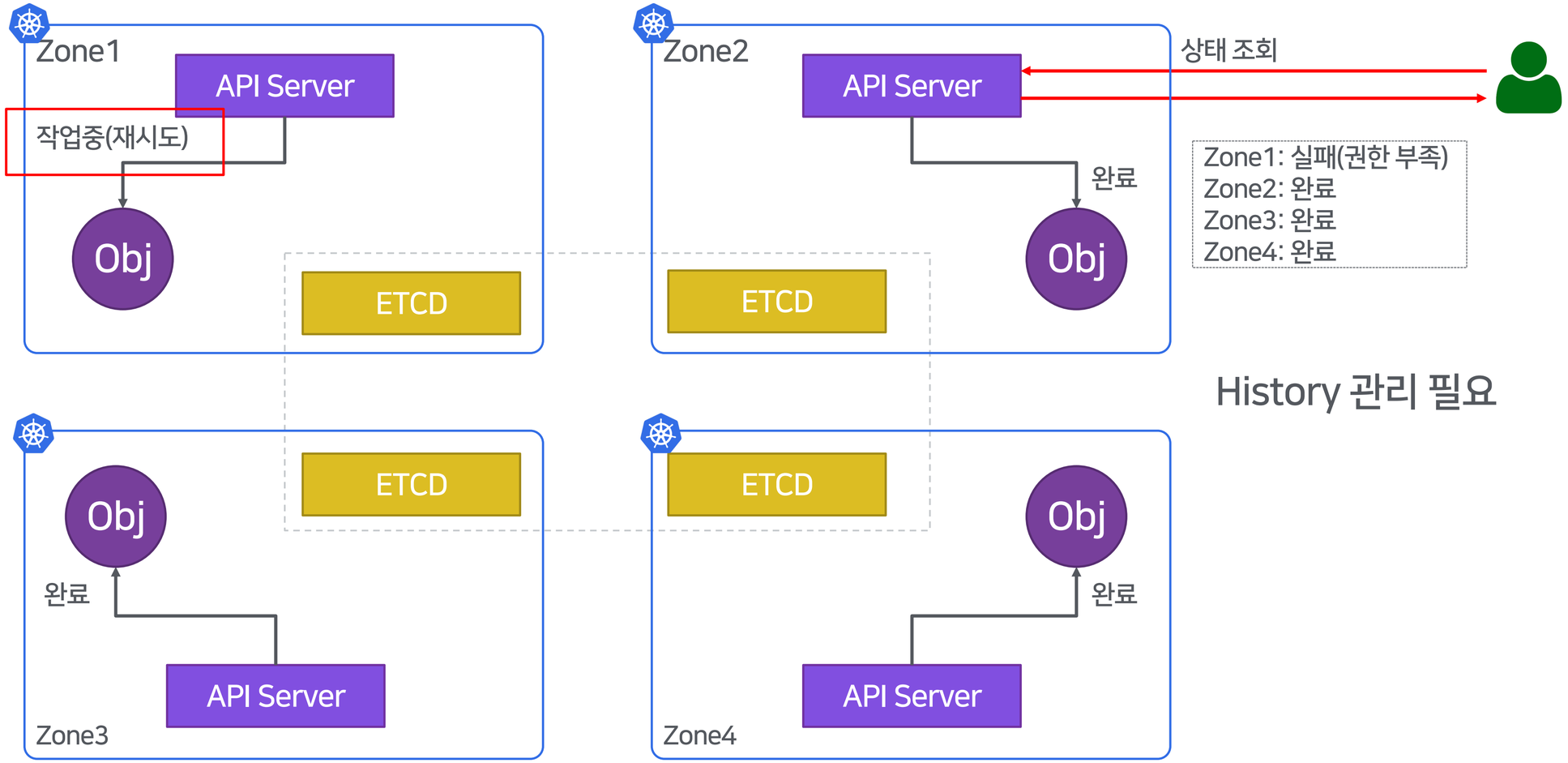
4.6 Concurrency Control



4.7 상태 관리


Zone1
API Server
작업중(재시도)
Obj
ETCD
Zone2
API Server
완료
Obj
ETCD
Obj
완료
API Server
Zone3
ETCD
ETCD
Obj
완료
API Server
Zone4

4.7 상태 관리


Zone1
API Server
작업중(재시도)
Obj
ETCD
Zone2
API Server
완료
Obj
ETCD
상태 조회
Zone1: 실패(권한 부족)
Zone2: 완료
Zone3: 완료
Zone4: 완료
History 관리 필요
ETCD
Obj
완료
API Server
Zone3
ETCD
Obj
완료
API Server
Zone4

4.8 ETCD 키 모델링

DEVIEW 2021

Data: /api/v1/{type}/{ID}

Result: /api/v1/{type}/{ID}/{zone}/{timestamp}

Search(prefix)

- /api/v1/{type}: 이벤트 Watch
- /api/v1/{type}/{ID}: 모든 존의 상태
- /api/v1/{type}/{ID}/{zone}: 특정 존의 상태

Loop: /api/v1/loop/{type}/{ID}/{zone}

Service

Data: /api/v1/services/{service}

Result: /api/v1/services/{service}/{zone}/timestamp

Search:

- /api/v1/services
- /api/v1/services/{service}
- /api/v1/services/{service}/{zone}

Loop: /api/v1/loop/services/{service}/{zone}

Backend

Data: /api/v1/backends/{backend}

Result: /api/v1/backends/{backend}/{zone}/timestamp

Search:

- /api/v1/backends
- /api/v1/backends/{backend}
- /api/v1/backends/{backend}/{zone}

Loop: /api/v1/loop/backends/{backend}/{zone}

Domain

Data: /api/v1/domains/{domain}

Result: /api/v1/domains/{domain}/{zone}/timestamp

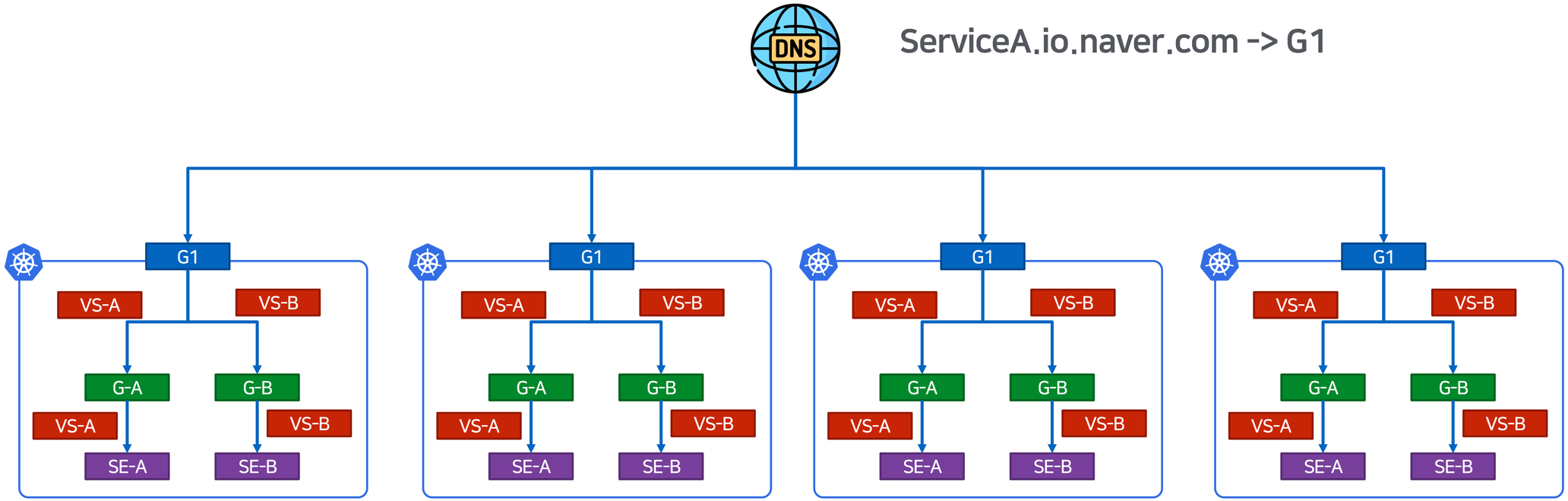
Search:

- /api/v1/domains
- /api/v1/domains/{domain}
- /api/v1/domains/{domain}/{zone}

Loop: /api/v1/loop/domains/{domain}/{zone}

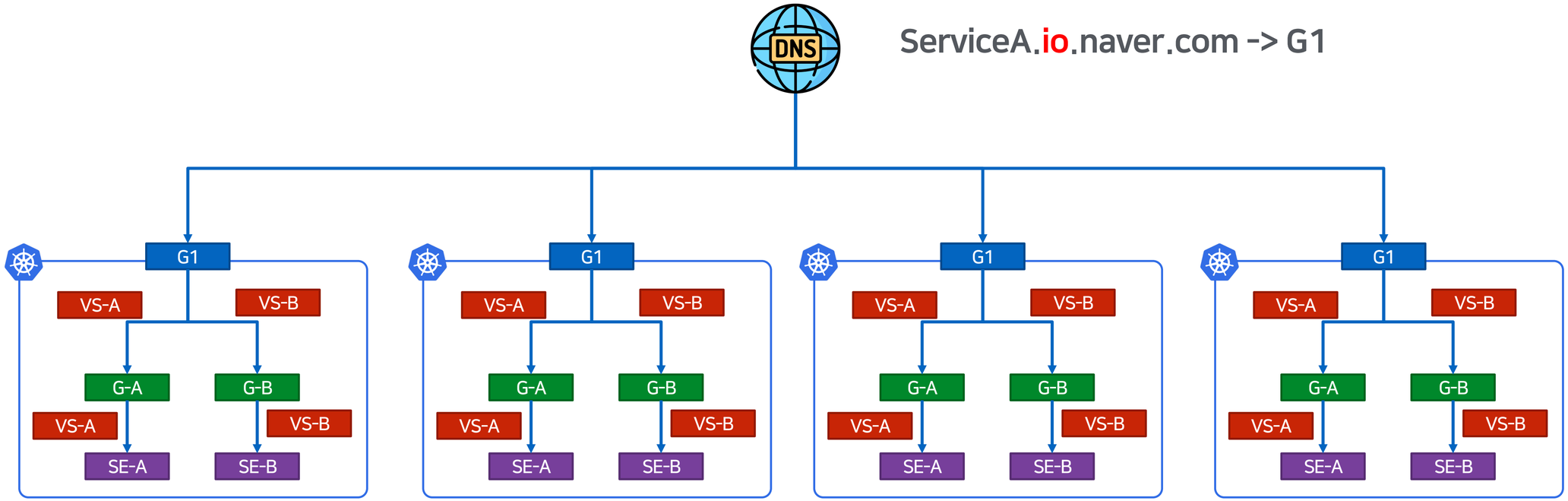
5. Multi-Domain과 인증서 기능

5.1 Multi-Domain과 인증서 관리 기능

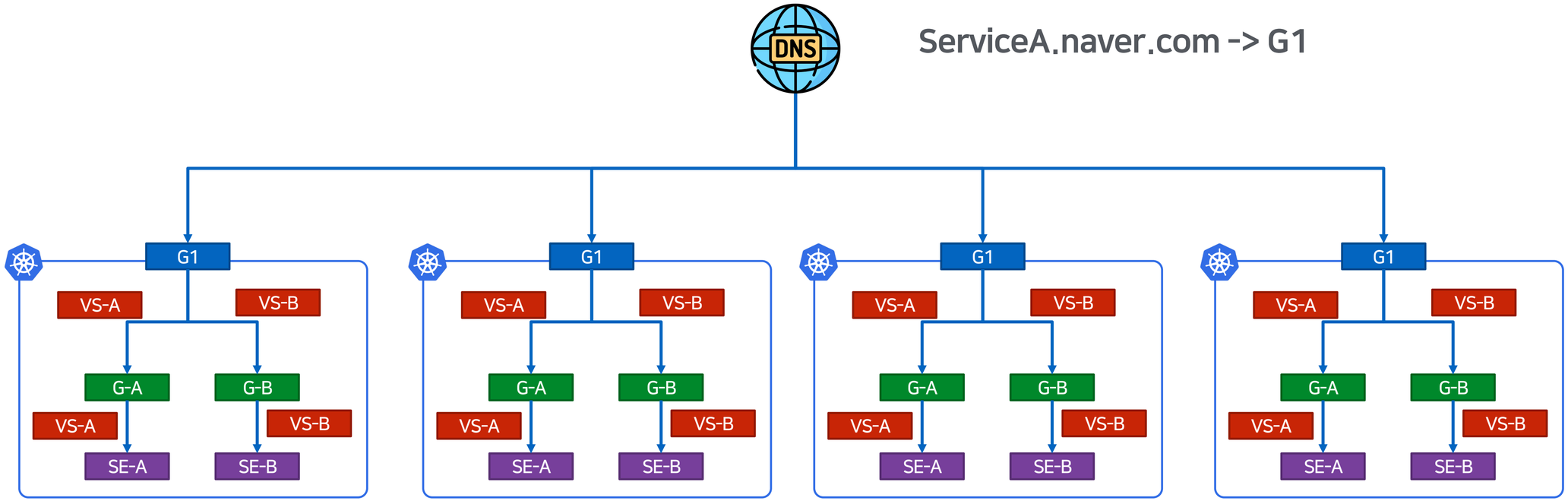


기본 도메인 제공

5.1 Multi-Domain과 인증서 관리 기능

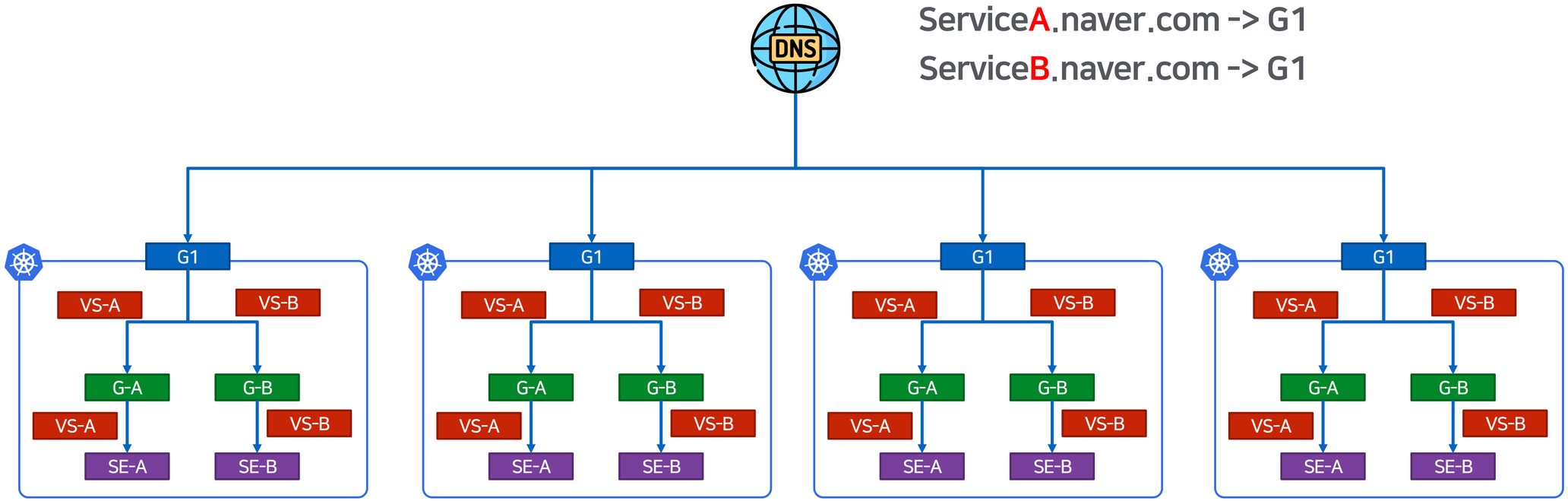


5.1 Multi-Domain과 인증서 관리 기능



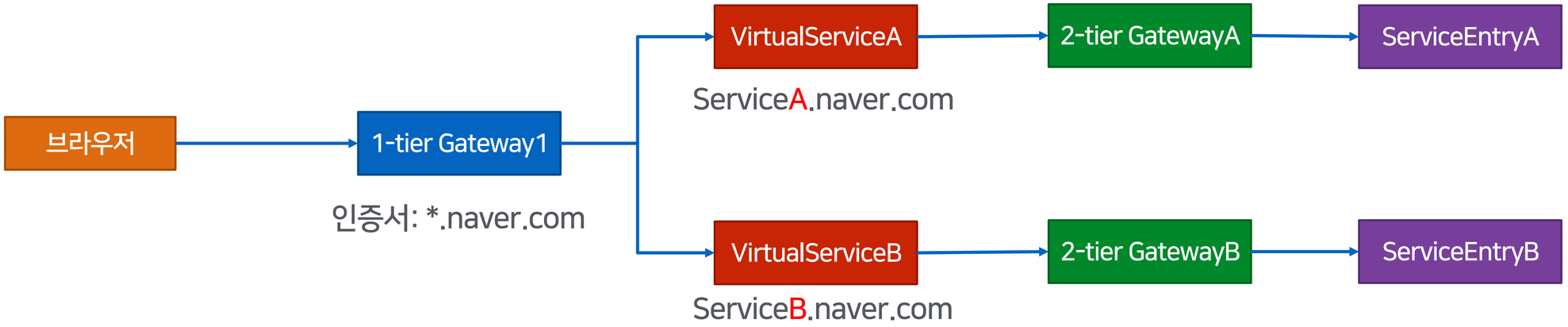
실제 Public 서비스용 도메인

5.1 Multi-Domain과 인증서 관리 기능



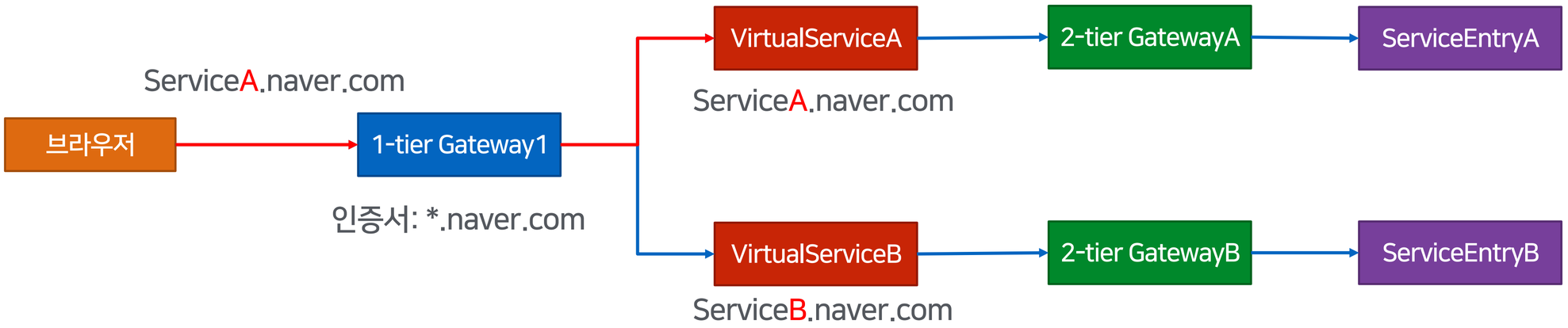
Suffix가 비슷한 도메인을 사용

5.2 도메인 Conflict 이슈



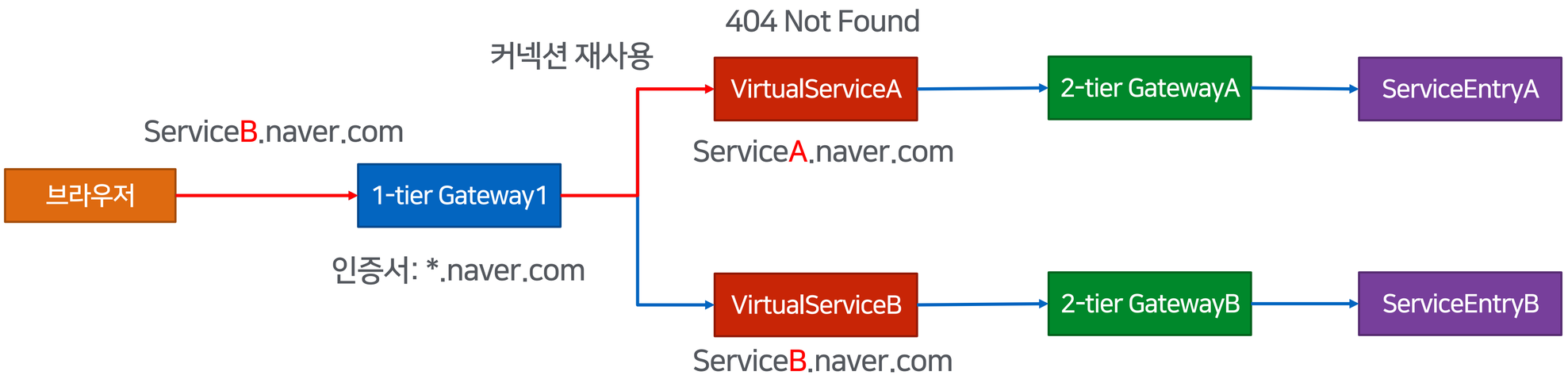
같은 wildcard를 가진 인증서를 사용

5.2 도메인 Conflict 이슈



같은 wildcard를 가진 인증서를 사용

5.2 도메인 Conflict 이슈



같은 wildcard를 가진 인증서를 사용

5.2 도메인 Conflict 이슈

DEVIEW 2021

HTTP 2.0 스펙

9.1.1. Connection Reuse

Connections that are made to an origin server, either directly or through a tunnel created using the CONNECT method (Section 8.3), MAY be reused for requests with multiple different URI authority components. A connection can be reused as long as the origin server is authoritative (Section 10.1). For TCP connections without TLS, this depends on the host having resolved to the same IP address.

For `https` resources, connection reuse additionally depends on having a certificate that is valid for the host in the URI. The certificate presented by the server MUST satisfy any checks that the client would perform when forming a new TLS connection for the host in the URI.

An origin server might offer a certificate with multiple `subjectAltName` attributes or names with wildcards, one of which is valid for the authority in the URI. For example, a certificate with a `subjectAltName` of `*.example.com` might permit the use of the same connection for requests to URIs starting with `https://a.example.com/` and `https://b.example.com/`.

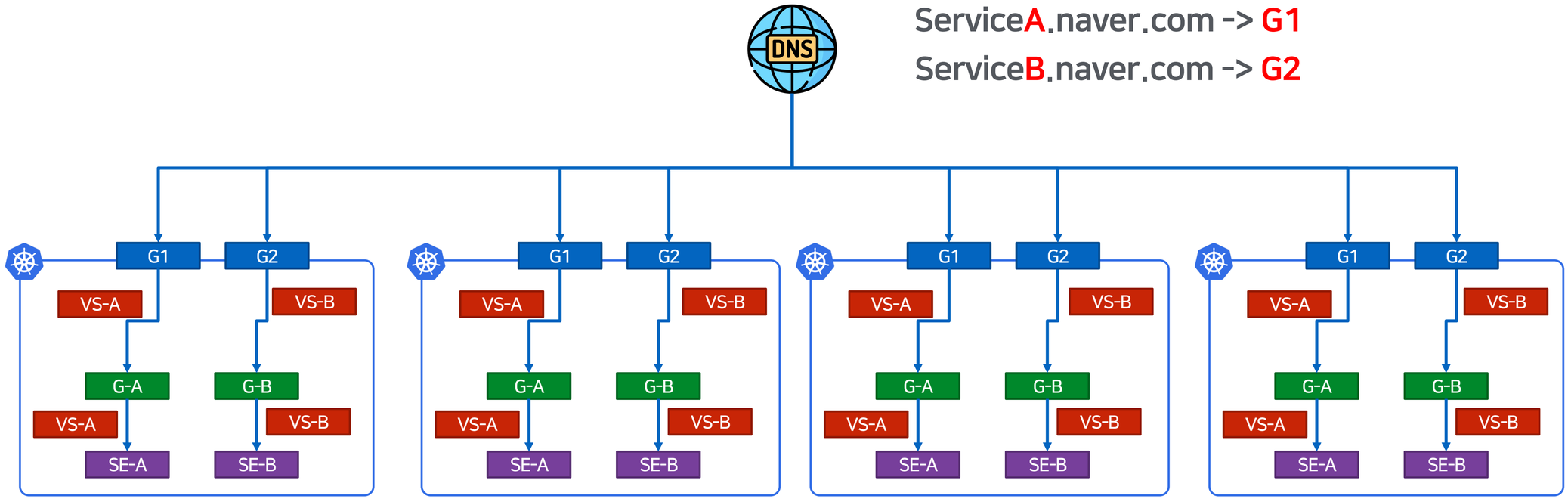
In some deployments, reusing a connection for multiple origins can result in requests being directed to the wrong origin server. For example, TLS termination might be performed by a middlebox that uses the TLS Server Name Indication (SNI) *[TLS-EXT]* extension to select an origin server. This means that it is possible for clients to send confidential information to servers that might not be the intended target for the request, even though the server is otherwise authoritative.

A server that does not wish clients to reuse connections can indicate that it is not authoritative for a request by sending a 421 (Misdirected Request) status code in response to the request (see Section 9.1.2).

A client that is configured to use a proxy over HTTP/2 directs requests to that proxy through a single connection. That is, all requests sent via a proxy reuse the connection to the proxy.

https://httpwg.org/specs/rfc7540.html#reuse

5.2 도메인 Conflict 이슈



인증서 등록 시, 도메인 충돌이 나지 않도록 관리

5.3 모델 디자인 – 모델 추가

Service

Routing

Backend

DNS

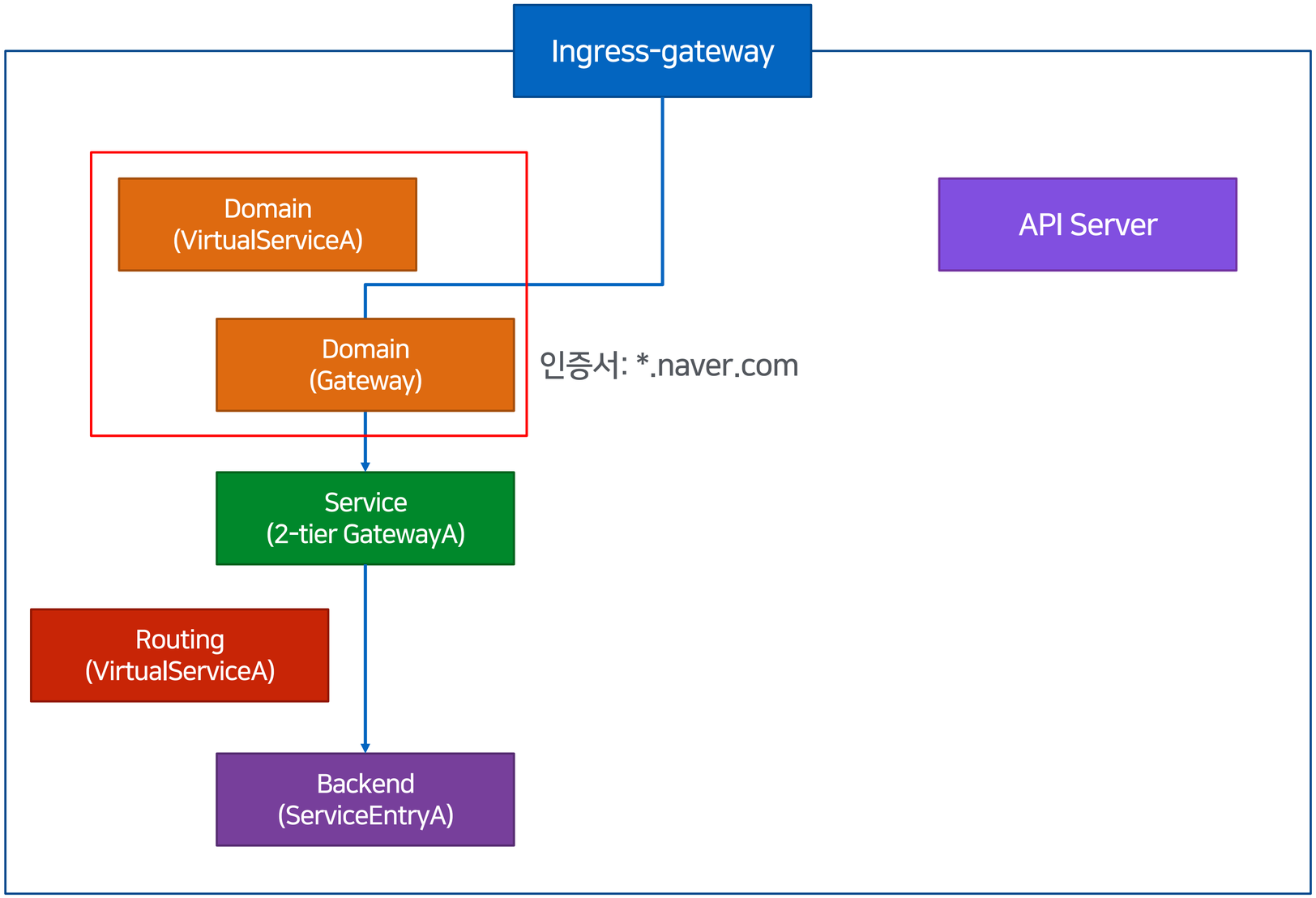
Domain

1-tier VirtualService
1-tier Gateway

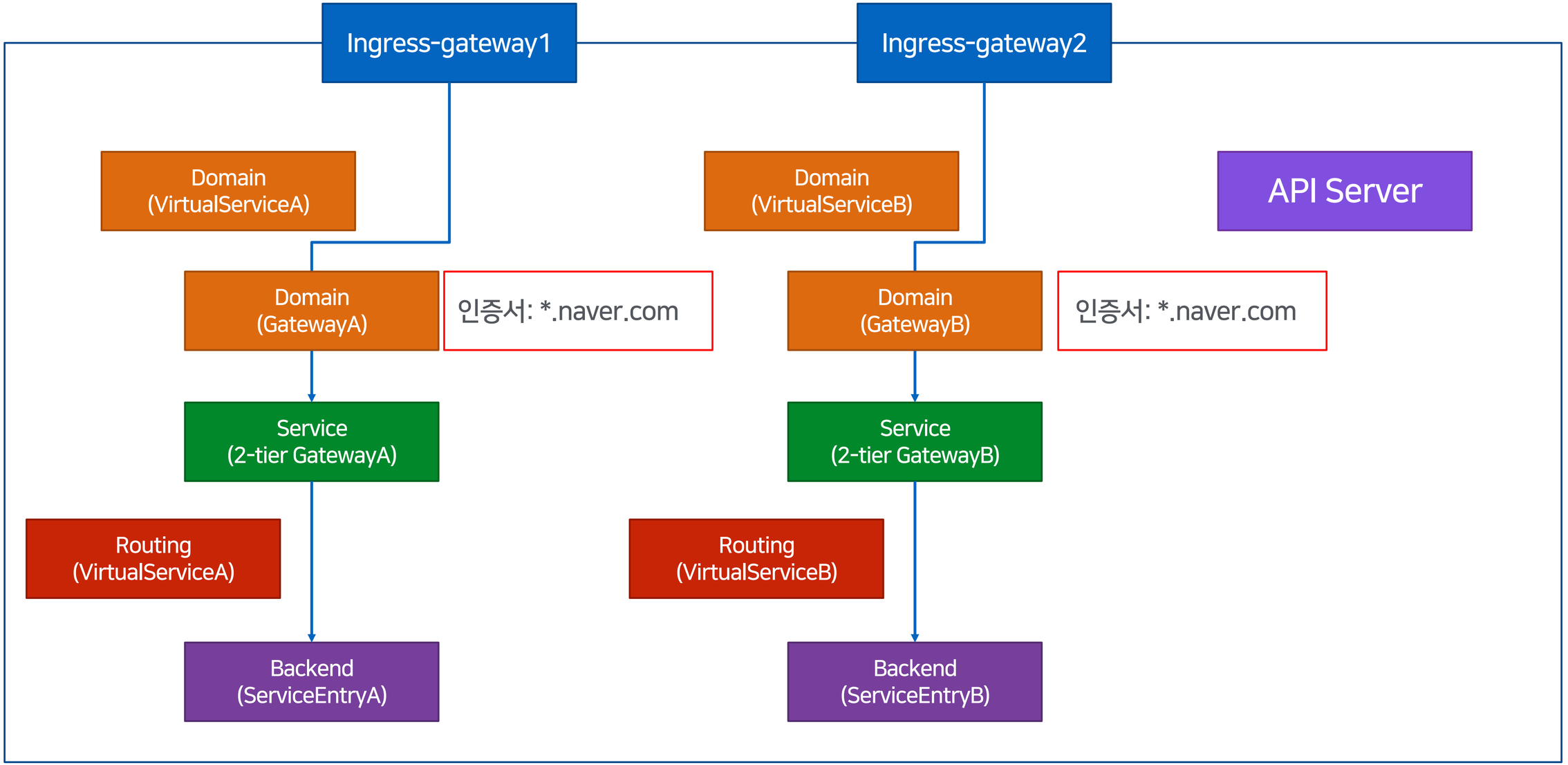
Cert

인증서

5.3 모델 디자인 - 적용


Ingress-gateway
Domain
(VirtualServiceA)
API Server
Domain
(Gateway)
인증서: *.naver.com
Service
(2-tier GatewayA)
Routing
(VirtualServiceA)
Backend
(ServiceEntryA)

5.3 모델 디자인 - 적용


Ingress-gateway1
Ingress-gateway2
Domain
(VirtualServiceA)
Domain
(VirtualServiceB)
API Server
Domain
(GatewayA)
인증서: *.naver.com
Domain
(GatewayB)
인증서: *.naver.com
Service
(2-tier GatewayA)
Service
(2-tier GatewayB)
Routing
(VirtualServiceA)
Routing
(VirtualServiceB)
Backend
(ServiceEntryA)
Backend
(ServiceEntryB)

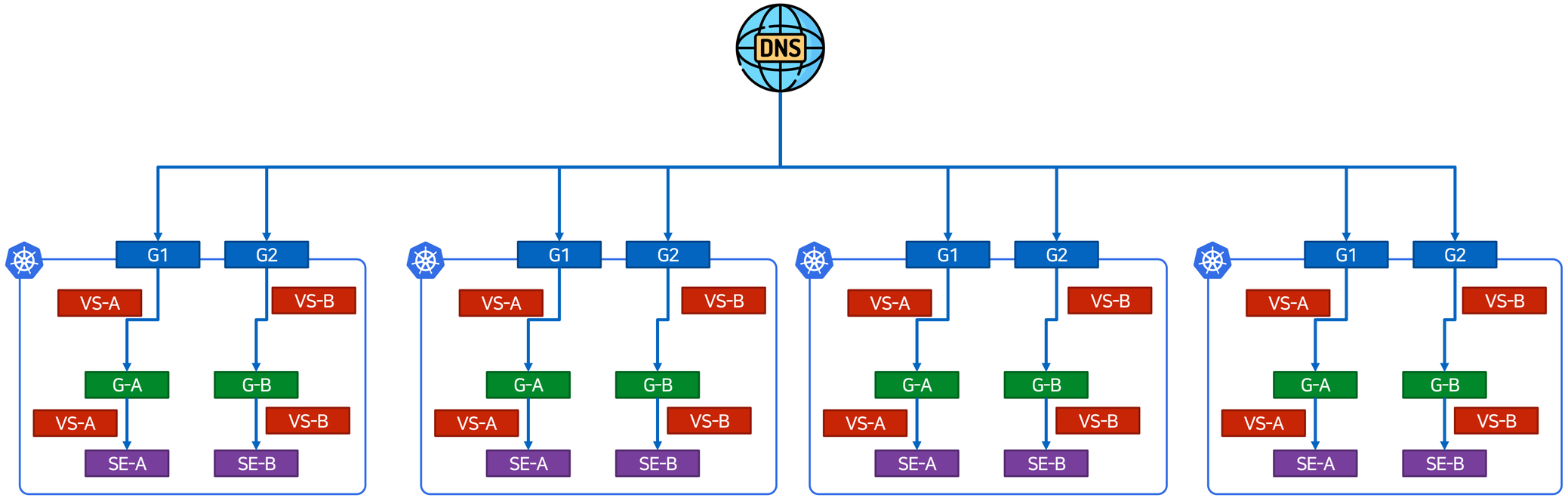
5.3 모델 디자인 - 적용

Domain

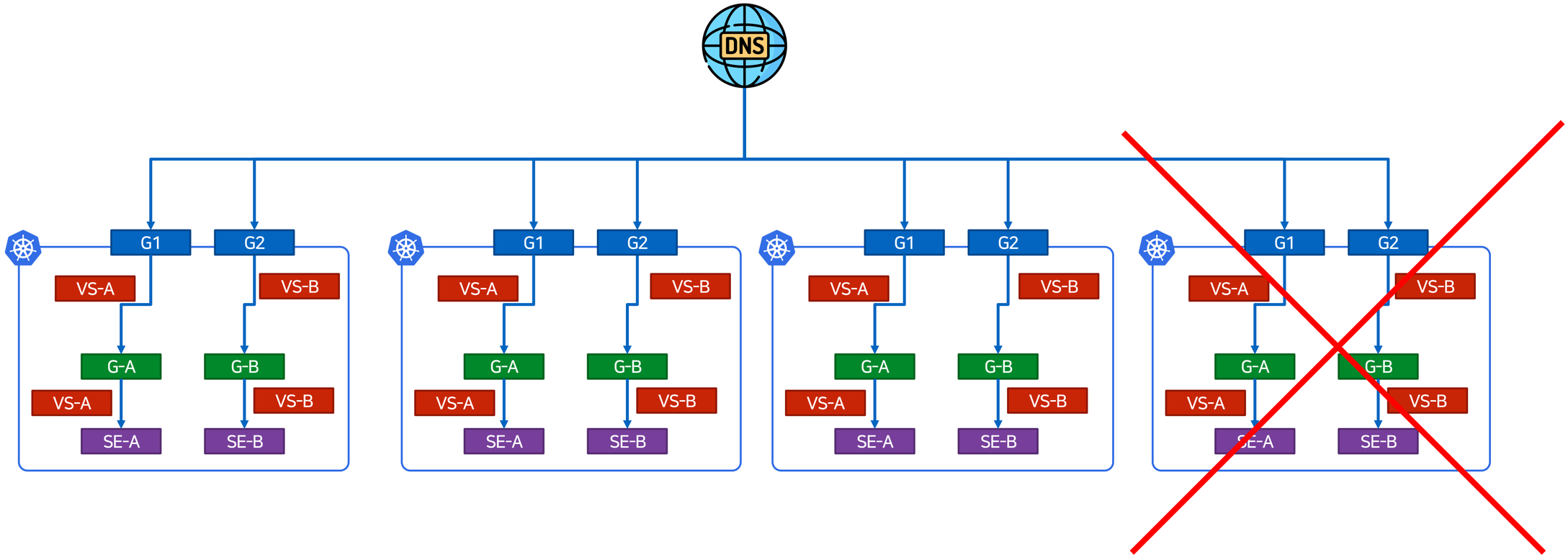
```
type Domain struct {
  Name         string          `json:"name" binding:"entityName"`
  Service      string          `json:"service" binding:"entityName"`
  Host         string          `json:"host" binding:"hostname_rfc1123,lowercase"`
  Gateway      string          `json:"gateway"` // gateway ID (gateway0, gateway1, gateway2 ...)
  HttpsRedirect bool           `json:"httpsRedirect"`
  Revision     string          `json:"revision"`
  CreatedAt    time.Time       `json:"createdAt"`
  UpdatedAt    time.Time       `json:"updatedAt"`
  DeletedAt    time.Time       `json:"deletedAt"`
  IsDeleting   bool            `json:"isDeleting"`
}
```

6. Health Checks

6.1 클러스터 제외

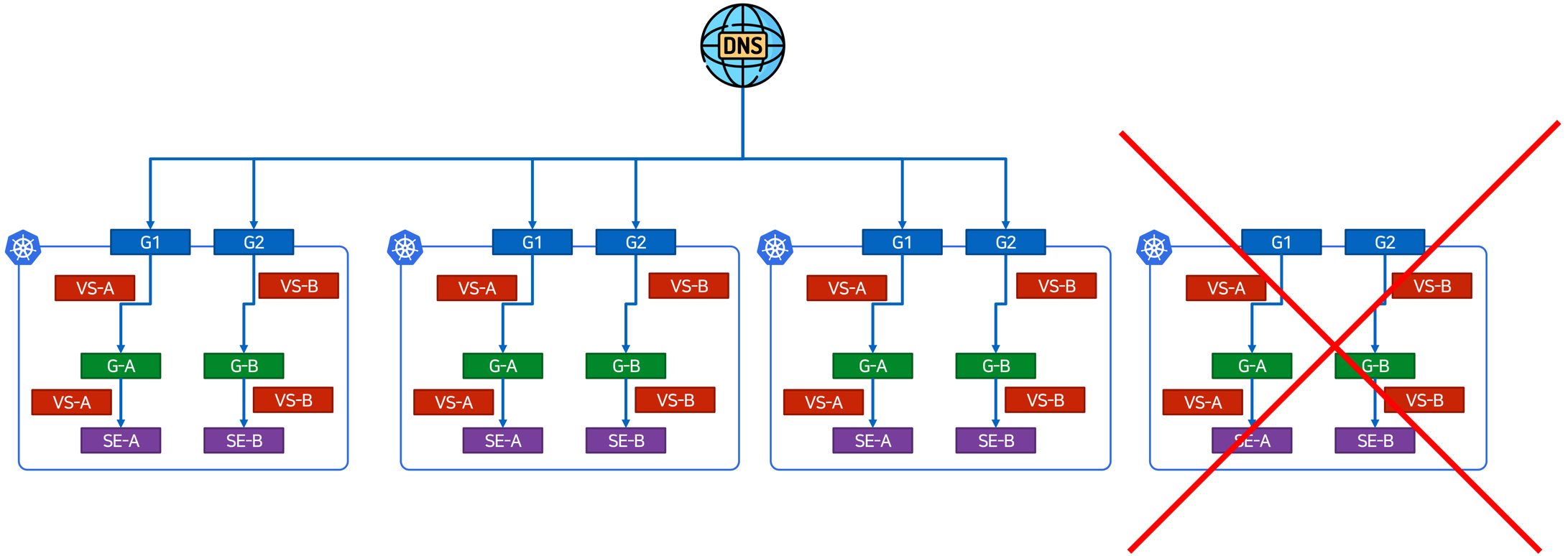


6.1 클러스터 제외



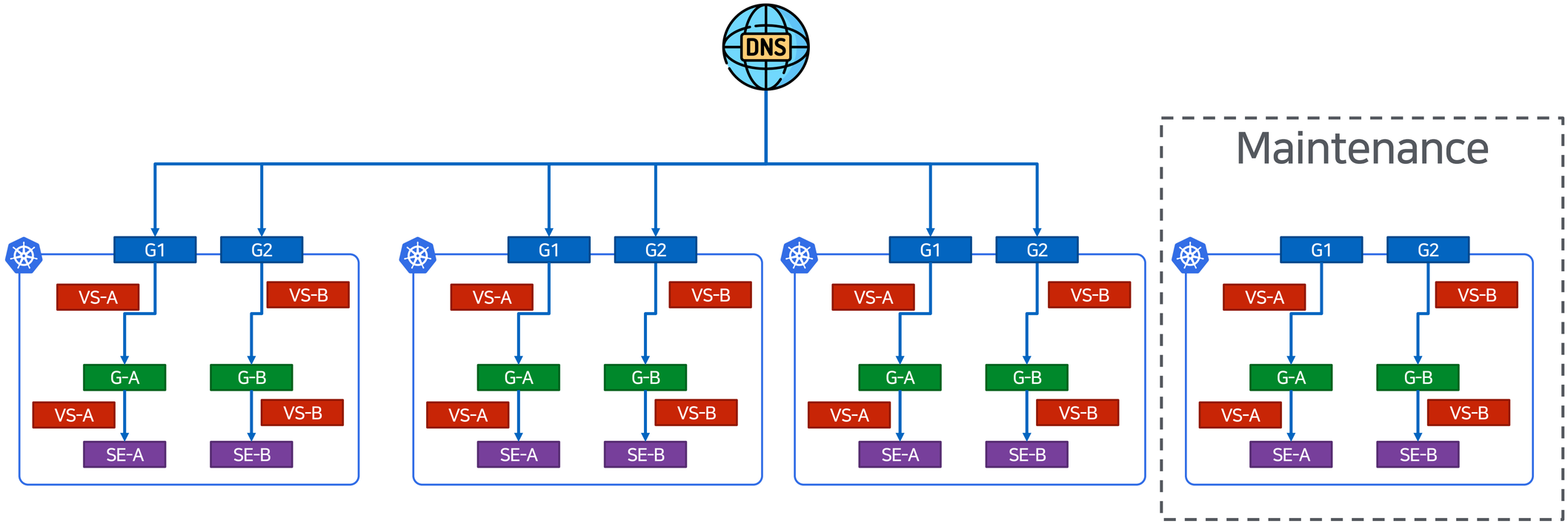
특정 클러스터의 장애

6.1 클러스터 제외



특정 클러스터의 장애

6.1 클러스터 제외



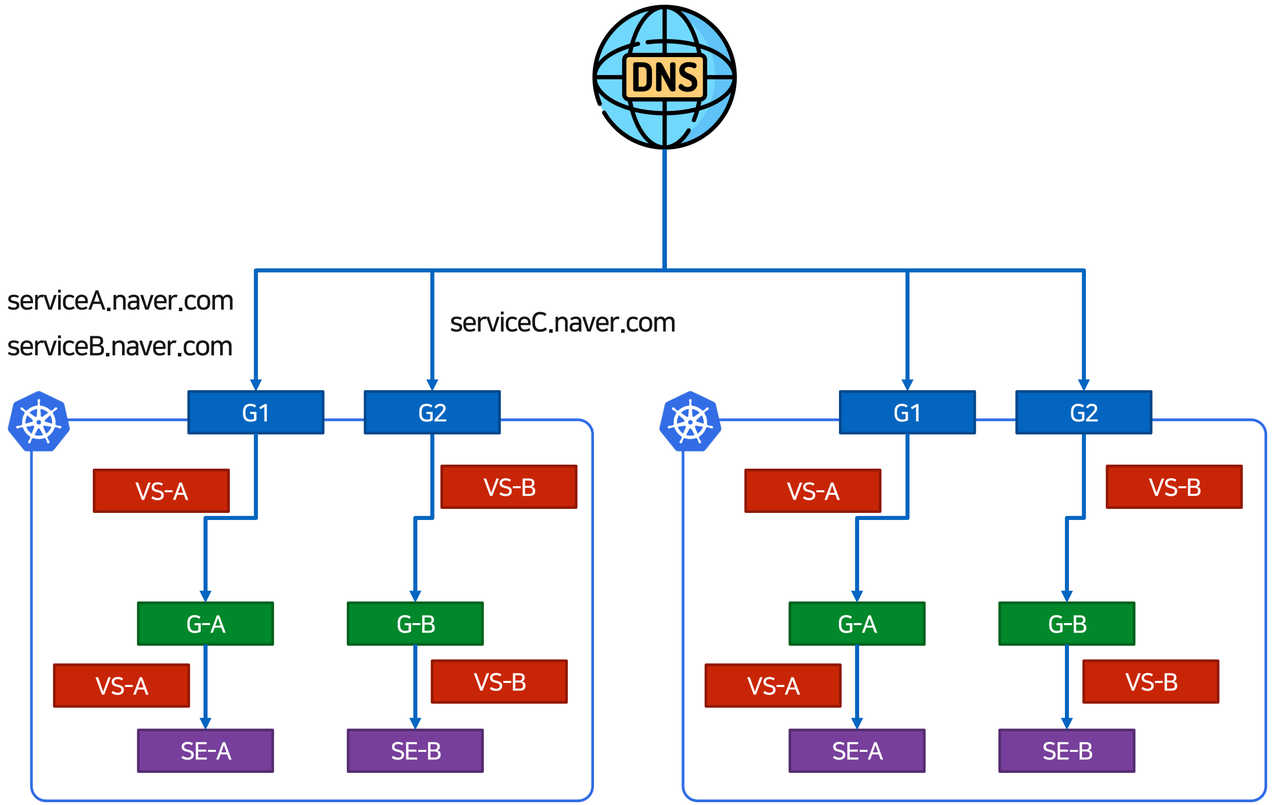
Maintenance 모드로 전환

6.2 Maintenance Mode

serviceA.naver.com
- 10.10.10.10 (Cluster1-GW1)
- 10.10.10.11 (Cluster2-GW1)
- 10.10.10.12 (Cluster3-GW1)
- 10.10.10.13 (Cluster4-GW1)

serviceB.naver.com
- 10.10.10.10 (Cluster1-GW1)
- 10.10.10.11 (Cluster2-GW1)
- 10.10.10.12 (Cluster3-GW1)
- 10.10.10.13 (Cluster4-GW1)

serviceC.naver.com
- 10.10.10.14 (Cluster1-GW2)
- 10.10.10.15 (Cluster2-GW2)
- 10.10.10.16 (Cluster3-GW2)
- 10.10.10.17 (Cluster4-GW2)

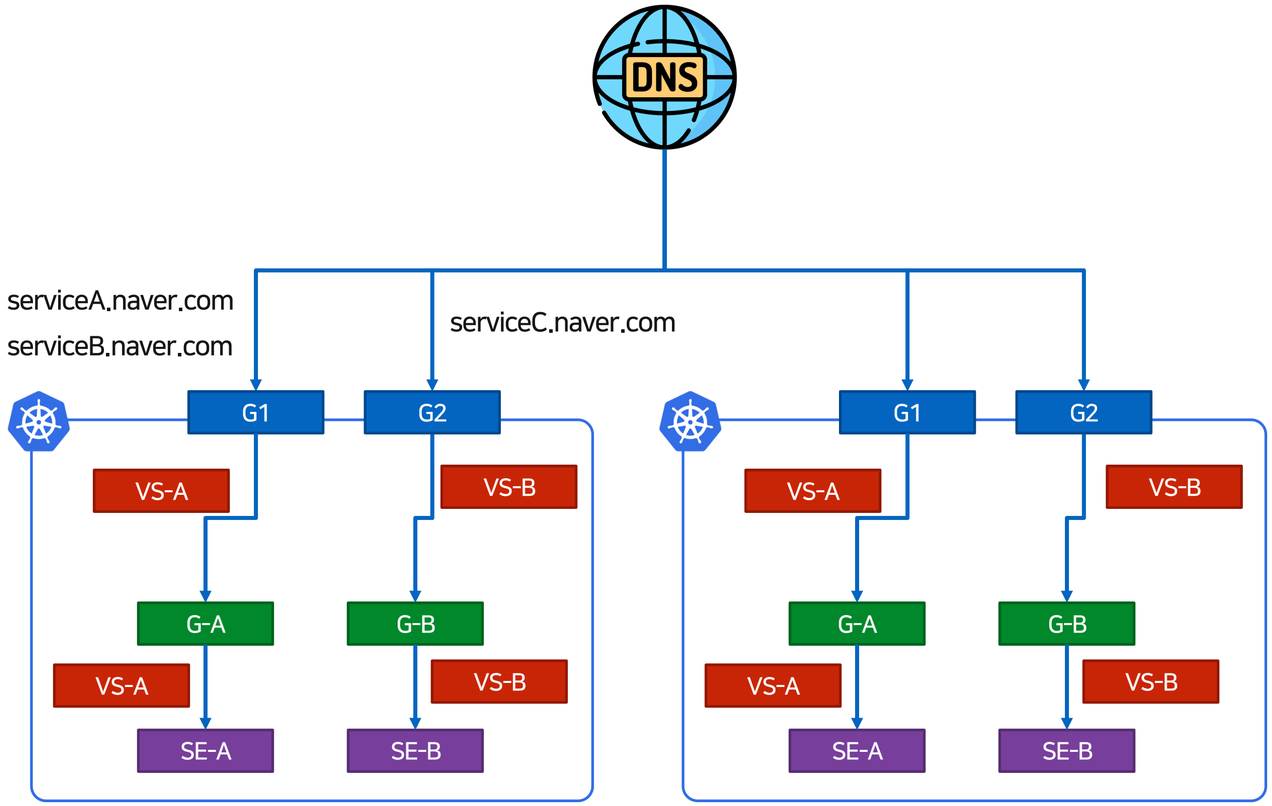


6.2 Maintenance Mode

serviceA.naver.com
- 10.10.10.10 (Cluster1-GW1)
- 10.10.10.11 (Cluster2-GW1)
- 10.10.10.12 (Cluster3-GW1)
- ~~10.10.10.13 (Cluster4-GW1)~~

serviceB.naver.com
- 10.10.10.10 (Cluster1-GW1)
- 10.10.10.11 (Cluster2-GW1)
- 10.10.10.12 (Cluster3-GW1)
- ~~10.10.10.13 (Cluster4-GW1)~~

serviceC.naver.com
- 10.10.10.14 (Cluster1-GW2)
- 10.10.10.15 (Cluster2-GW2)
- 10.10.10.16 (Cluster3-GW2)
- ~~10.10.10.17 (Cluster4-GW2)~~

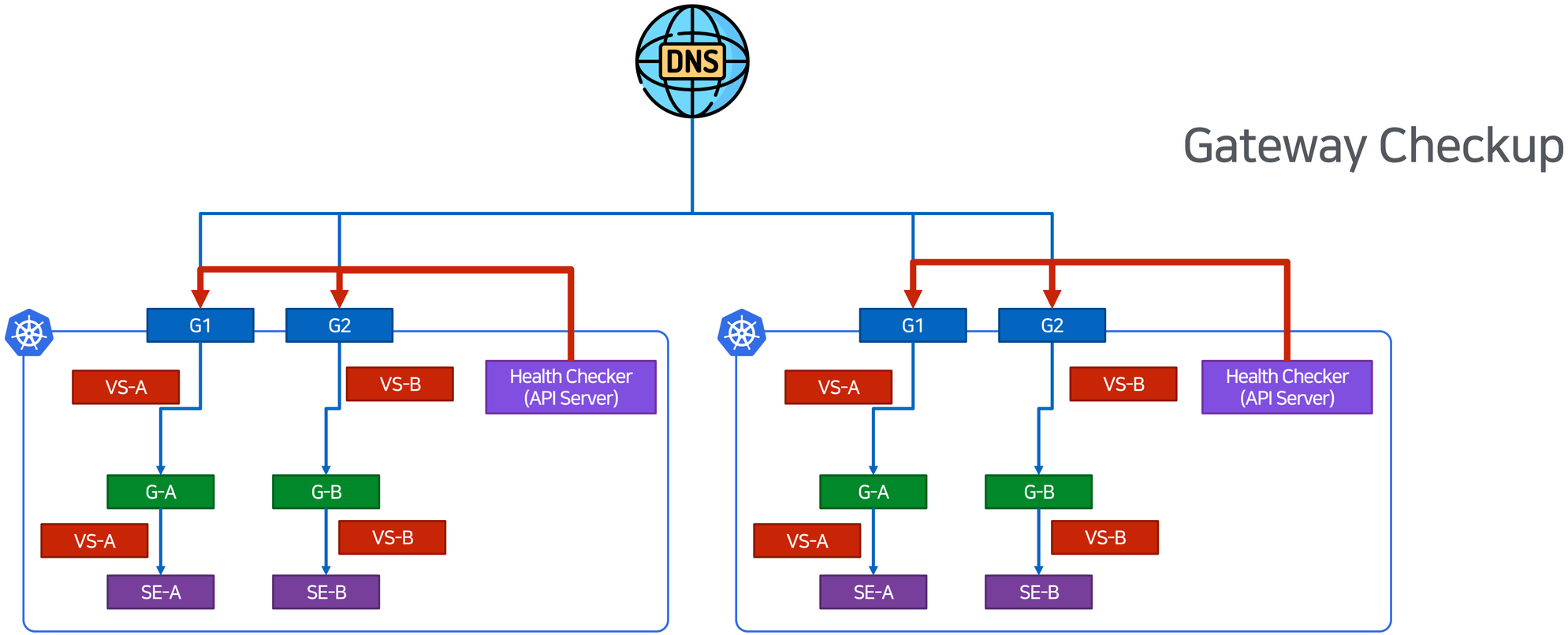


6.2 Maintenance Mode

DNS

```
type DNSRecord struct {
    Host          string    `json:"host"`
    Gateway       string    `json:"gateway"`
    TTL           int32     `json:"ttl"`
    ZonesToExclude []string `json:"zonesToExclude"`
    Revision      string    `json:"revision"`
    CreatedAt     time.Time `json:"createdAt"`
    UpdatedAt     time.Time `json:"updatedAt"`
    DeletedAt     time.Time `json:"deletedAt"`
    IsDeleting    bool      `json:"isDeleting"`
}
```

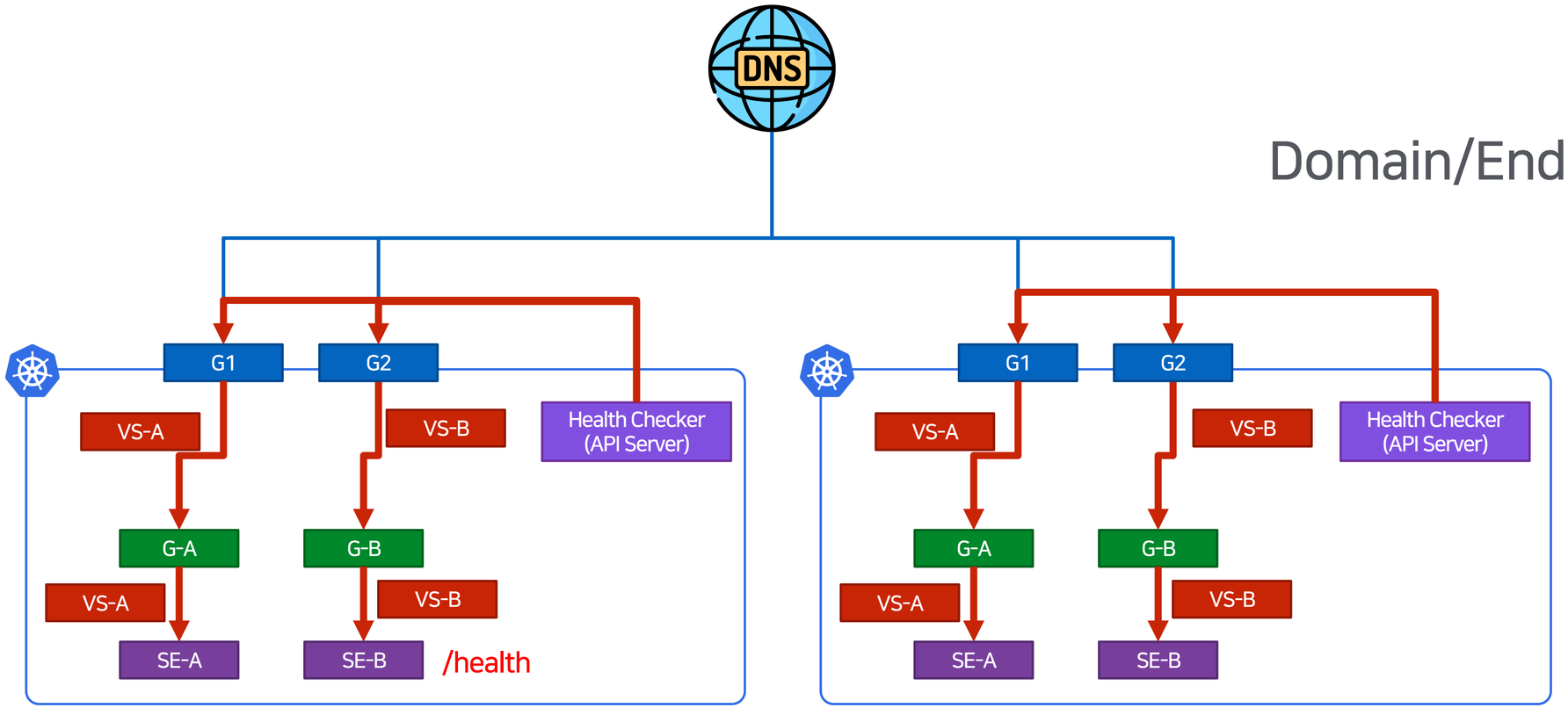
6.3 Health Check - Cluster



최소 2개 이상의 Health Checker가 장애로 인지해야 제외

Threshold(0.5) 보다 많은 비율의 Gateway가 실패해야 제외

6.3 Health Check - Service



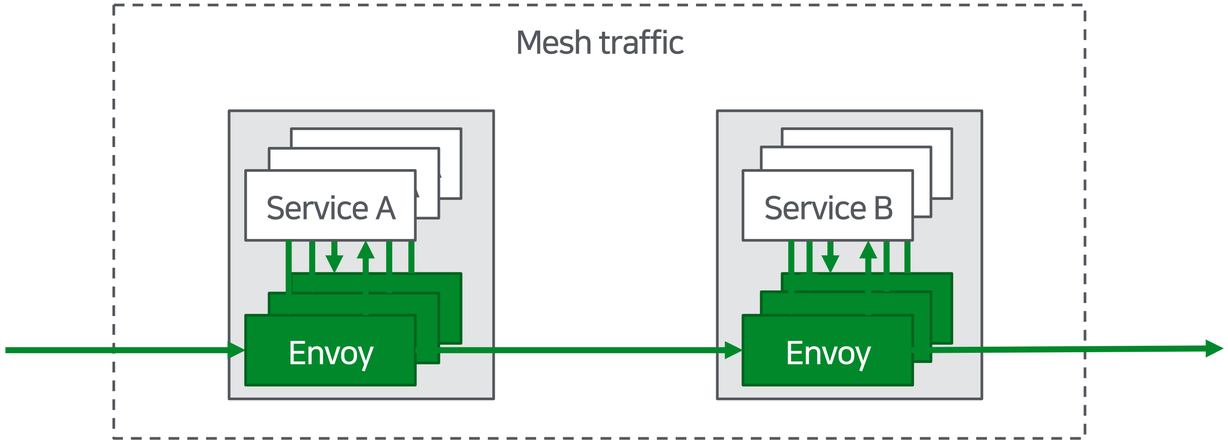
Domain/Endpoints Checkup

사용자의 실제 도메인과 Endpoints를 체크

DEVIEW 2021

7. Global Rate Limit

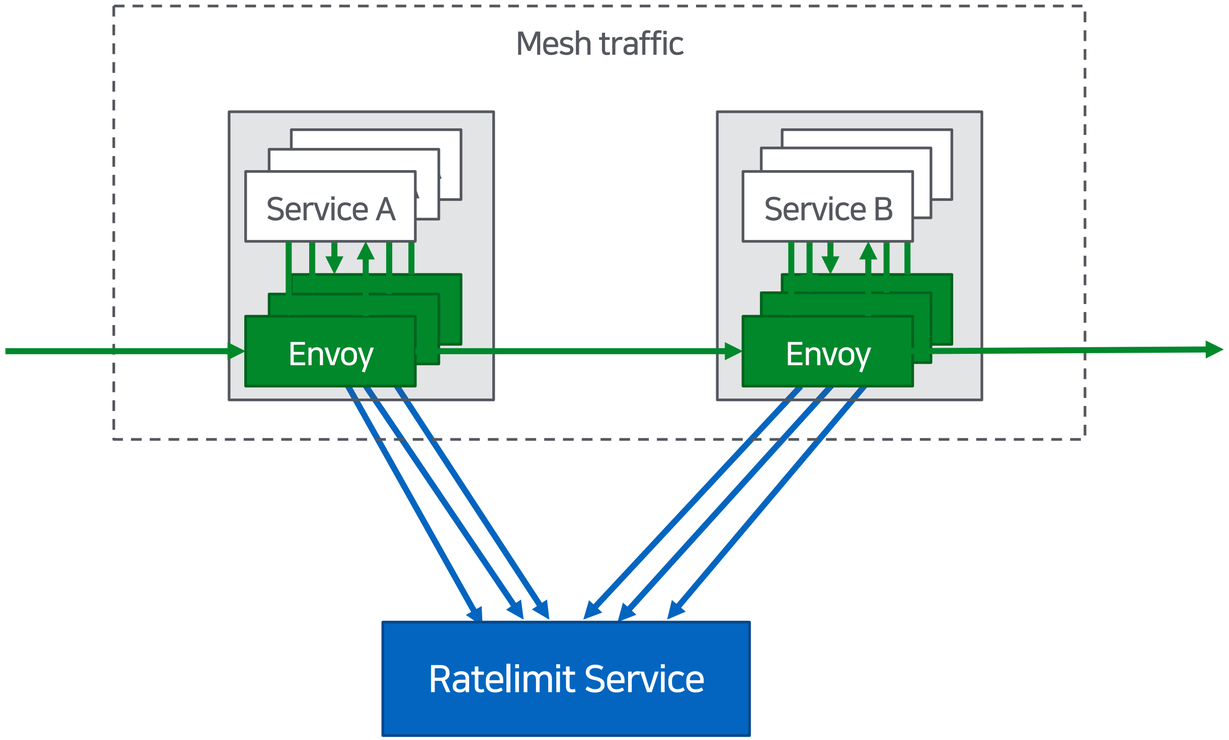
7.1 Global Ratelimit



복수개의 인스턴스가 구동됨

https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/

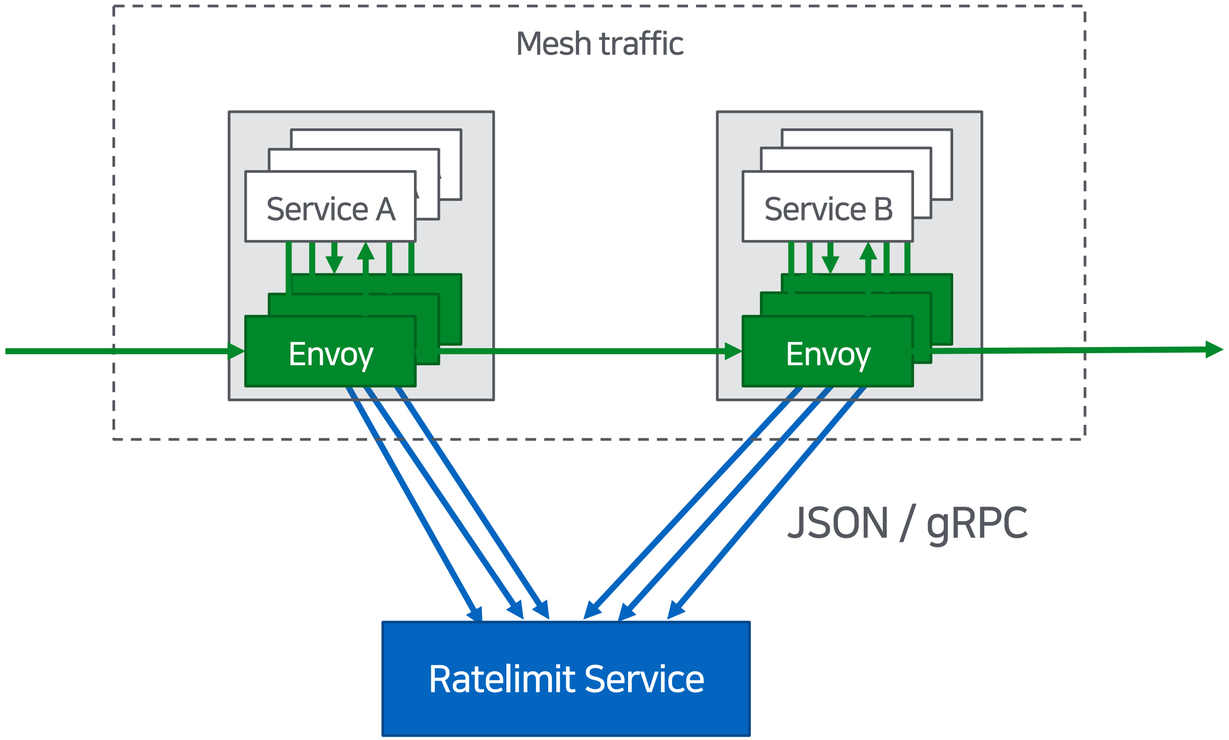
7.1 Global Ratelimit



각 인스턴스들이 Ratelimit 서비스에게 물어보는 방식

https://github.com/envoyproxy/ratelimit

7.1 Global Ratelimit

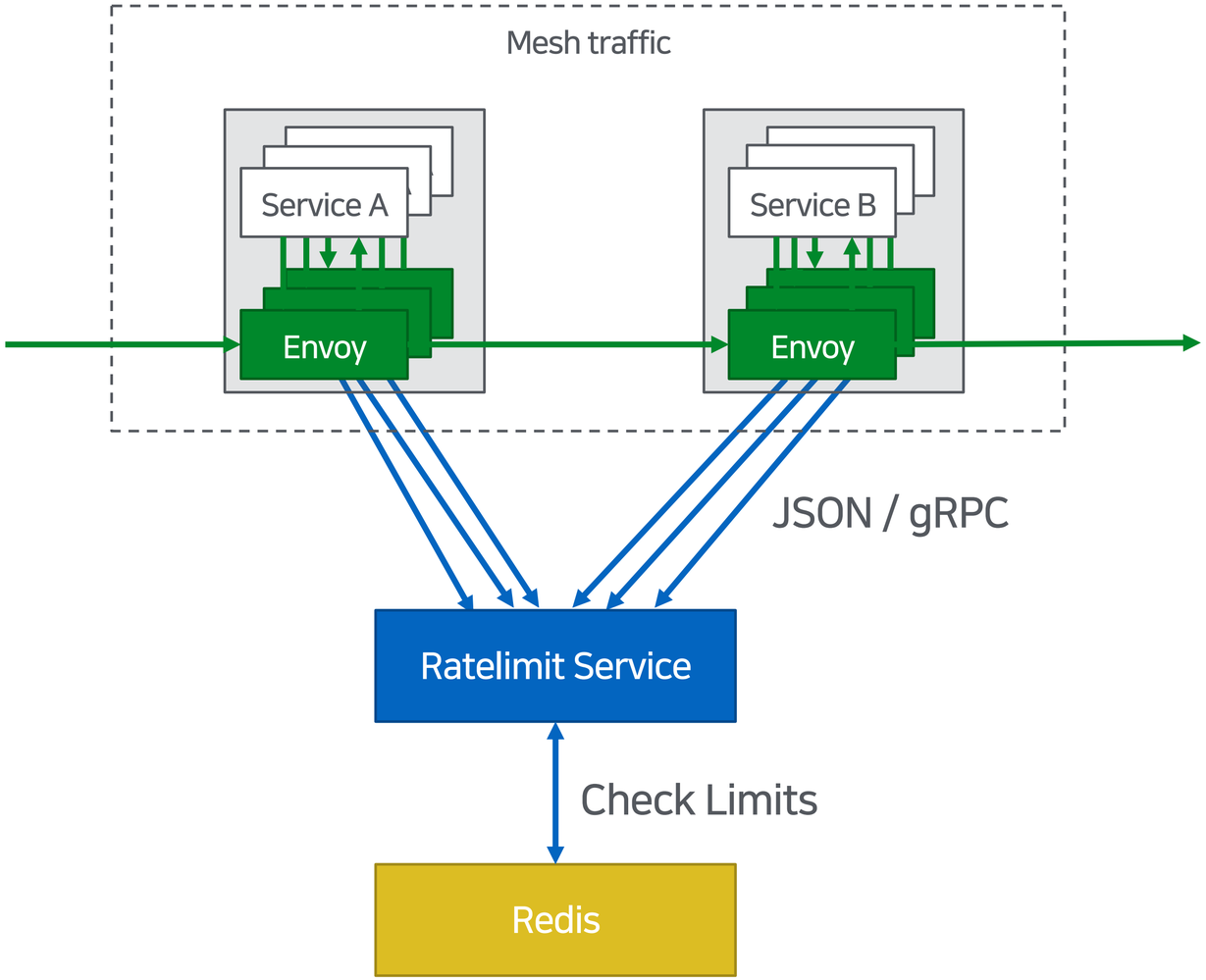



```
service RateLimitService {
  ShouldRateLimit(RateLimitRequest) returns (RateLimitResponse) {
  }
}
```

JSON/gRPC 인터페이스 제공

https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/

7.1 Global Ratelimit

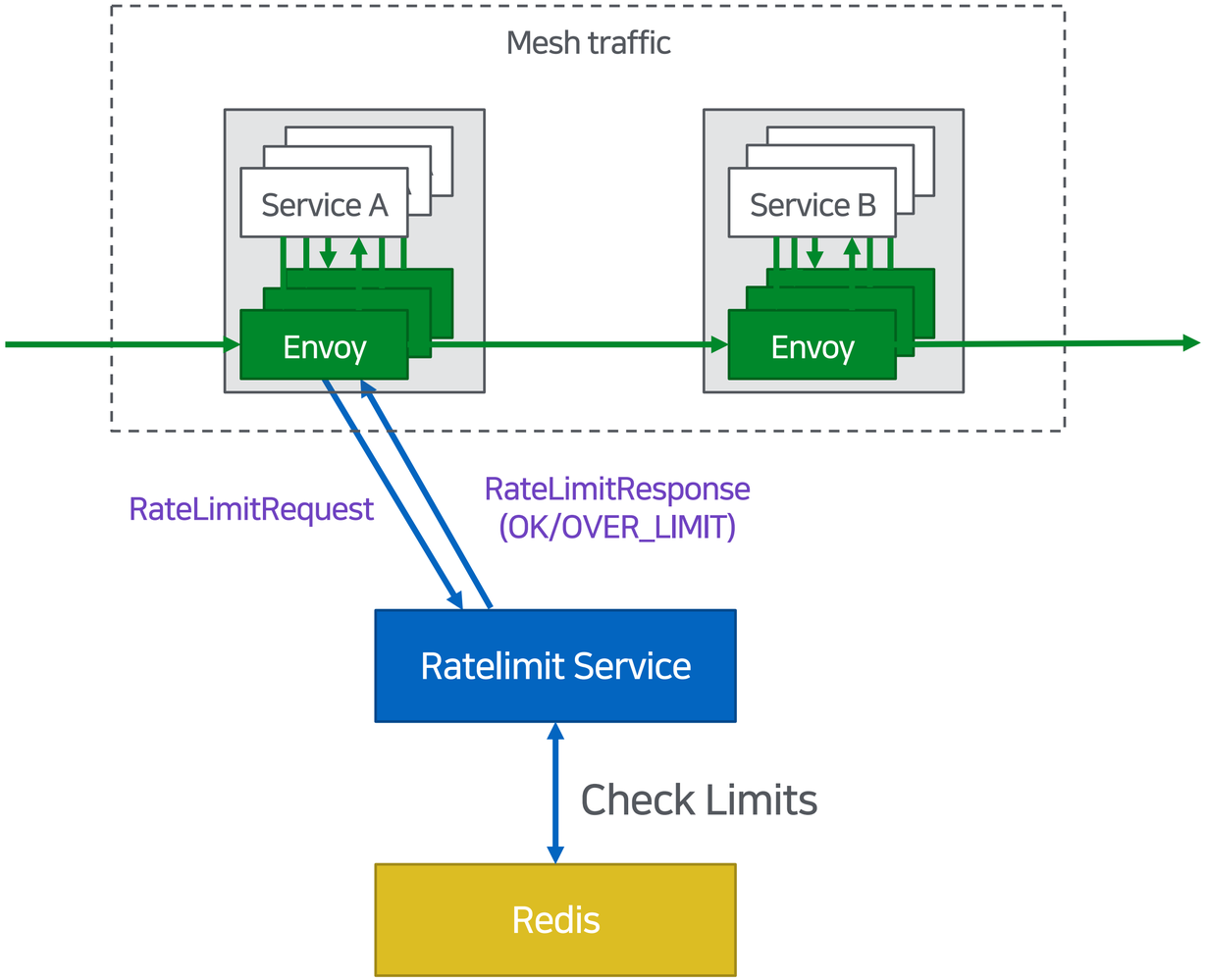



```
message RateLimitRequest {
    string domain = 1;
    repeated RateLimitDescriptor descriptors = 2
    uint32 hits_addend = 3;
}
```

```
message RateLimitResponse {
   enum Code {
      UNKNOWN = 0;
      OK = 1;
      OVER_LIMIT = 2;
   }
}
```

https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/

7.1 Global Ratelimit




```
message RateLimitRequest {
  string domain = 1;
  repeated RateLimitDescriptor descriptors = 2
  uint32 hits_addend = 3;
}
```

```
message RateLimitResponse {
  enum Code {
    UNKNOWN = 0;
    OK = 1;
    OVER_LIMIT = 2;
  }
}
```

https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/

7.2 Global Ratelimit – Descriptor

DEVIEW 2021

```
message RateLimitDescriptor {
    message Entry {
        string key = 1;
        string value = 2;
    }

    message RateLimitOverride {
        uint32 requests_per_unit = 1;
        type.v3.RateLimitUnit unit = 2;
    }

    repeated Entry entries = 1;
    RateLimitOverride limit = 2;
}
```

```
["authenticated": "false"], ["remote_address": "10.0.0.1"]

["authenticated": "false"], ["path": "/foo/bar"]

["authenticated": "false"], ["path": "/foo/bar"], ["remote_address": "10.0.0.1"]
```

```
["authenticated": "true"], ["client_id": "foo"]

["authenticated": "true"], ["client_id": "foo"], ["path": "/foo/bar"]
```

여러 Descriptor로 복잡한 시나리오 정의 가능

https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/

7.2 Global Ratelimit – Descriptor

DEVIEW 2021

```
message RateLimitDescriptor {
    message Entry {
        string key = 1;
        string value = 2;
    }

    message RateLimitOverride {
        uint32 requests_per_unit = 1;
        type.v3.RateLimitUnit unit = 2;
    }

    repeated Entry entries = 1;
    RateLimitOverride limit = 2;
}
```

```
domain: ratelimit.com
descriptors:
 - key: foo
   rate_limit:
     unit: minute
     requests_per_unit: 2
   descriptors:
     - key: bar
       rate_limit:
         unit: minute
         requests_per_unit: 3
     - key: bar
       value: banned
       rate_limit:
         unit: minute
         requests_per_unit: 0
```

여러 Descriptor로 복잡한 시나리오 정의 가능

https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/

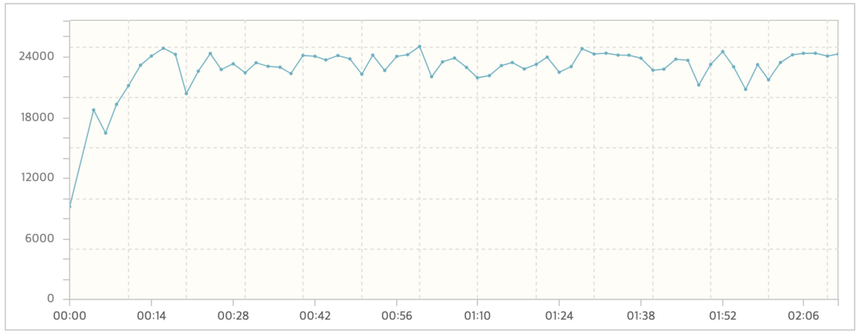
7.3 Global Ratelimit – 벤치마크

DEVIEW 2021

요약

총 Vuser	50
TPS	22,848.6
최고 TPS	24,994.0
평균 테스트시간	2.13 MS
총 실행 테스트	3,016,648
성공한 테스트	3,016,648
에러	0
동작 시간	00:02:16

TPS 그래프 상세 보고서



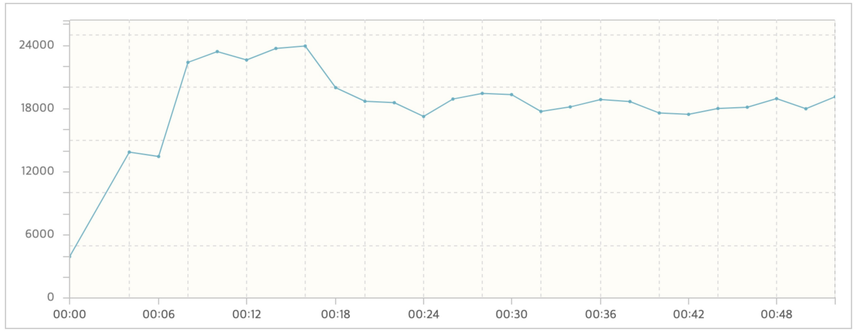
4 ratelimit pods

(JSON)

요약

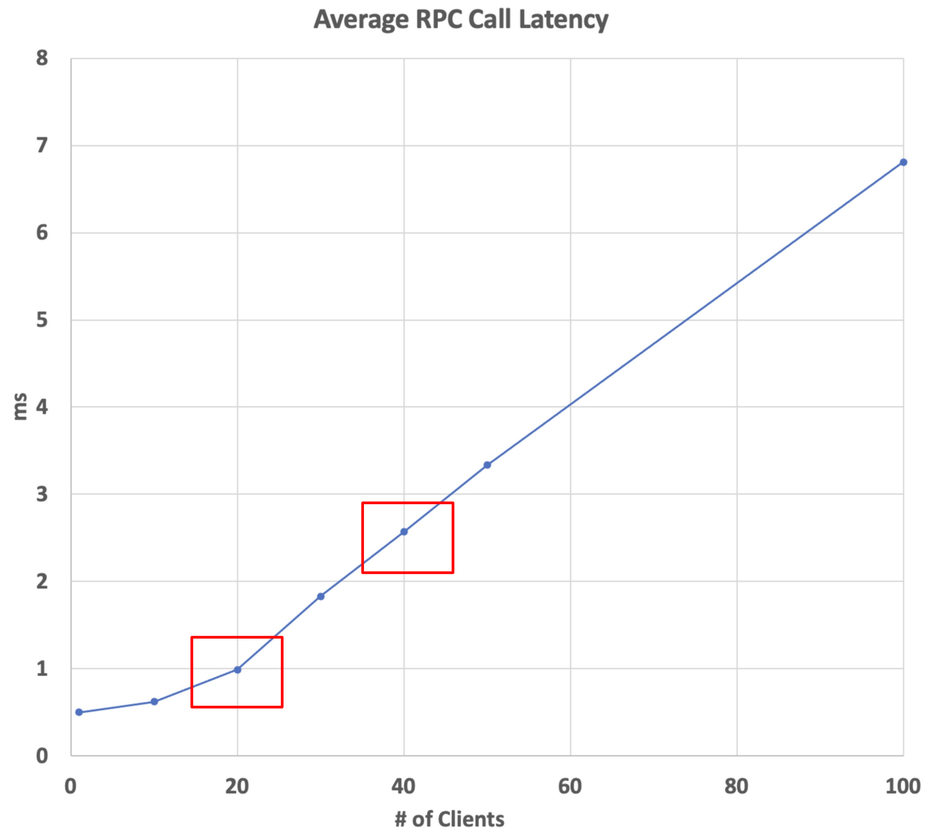
총 Vuser	990
TPS	18,410.9
최고 TPS	23,879.5
평균 테스트시간	46.20 MS
총 실행 테스트	958,535
성공한 테스트	957,751
에러	784
동작 시간	00:01:00

TPS 그래프 상세 보고서



Client 수에 따라 Latency가 많은 차이를 보임

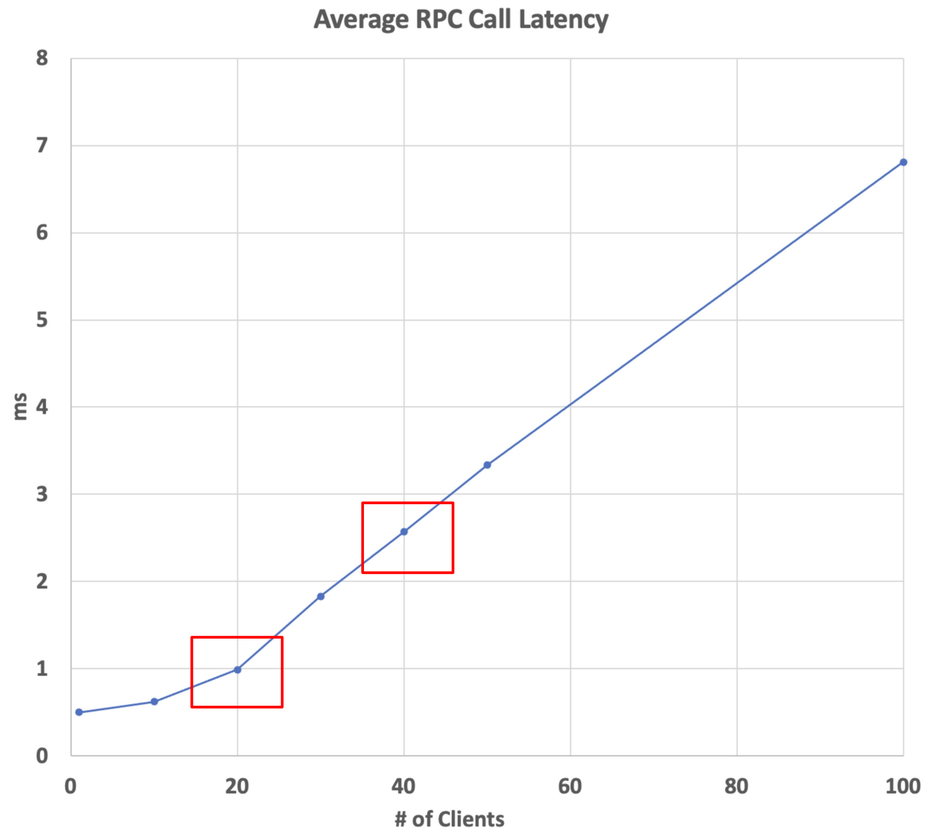
7.3 Global Ratelimit – 벤치마크



1 ratelimit pods

요구되는 수준에 따라, scale-out 수준을 정할 수 있음

7.3 Global Ratelimit – 벤치마크

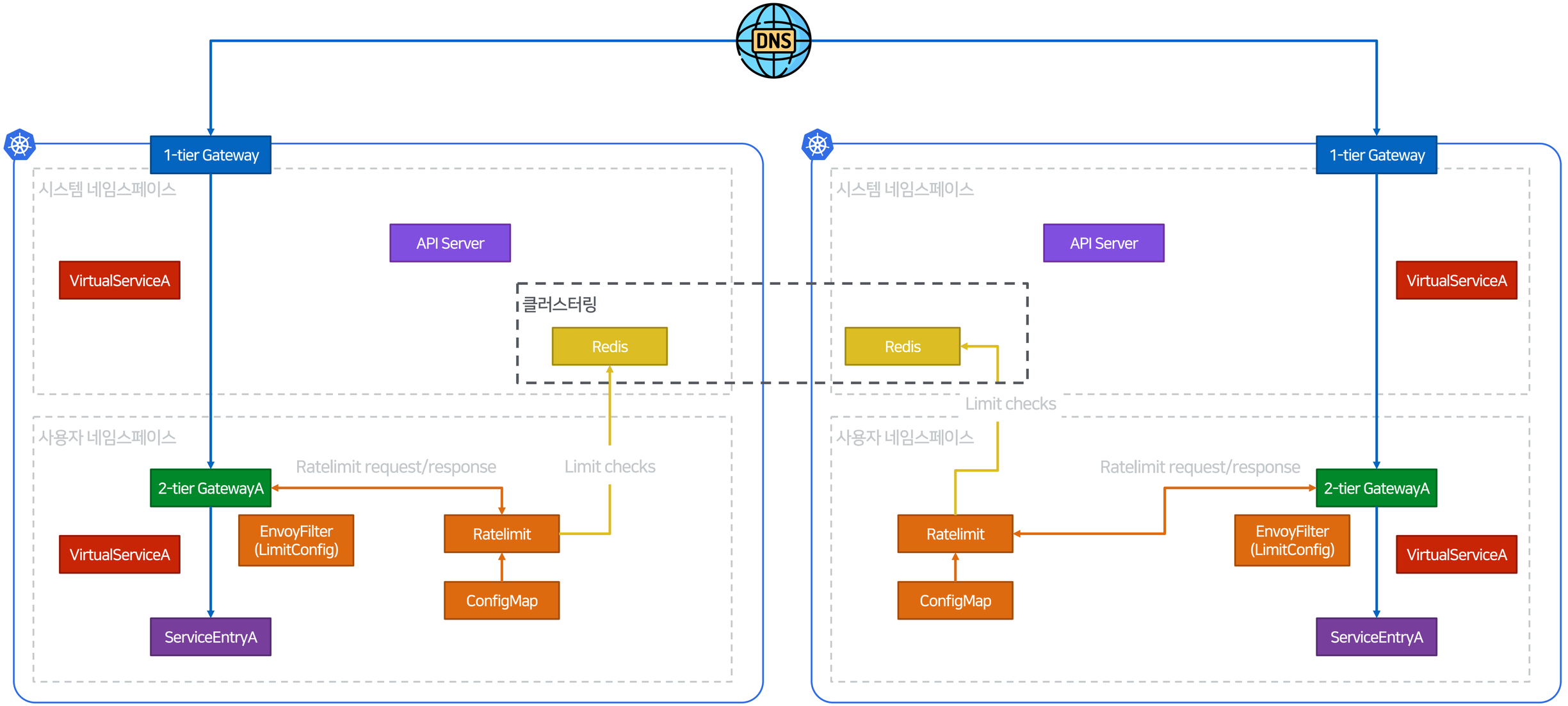


1ms -> 20개 pods : 1 ratelimit pod

3ms -> 40개 pods : 1 ratelimit pod

요구되는 수준에 따라, scale-out 수준을 정할 수 있음

7.4 Global Ratelimit – 아키텍처



7.5 Global Ratelimit – 구조체

DEVIEW 2021

Limit

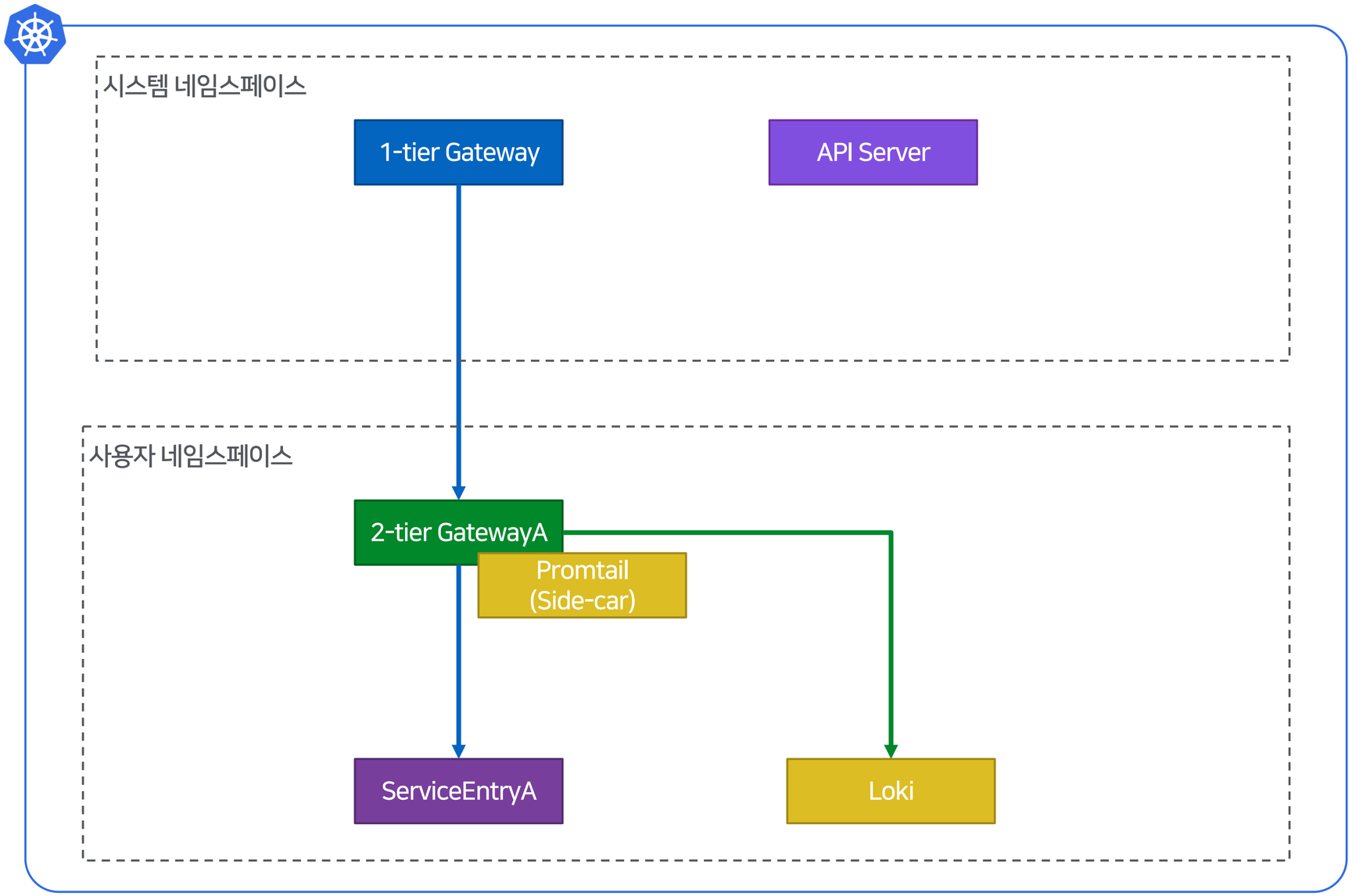
```
type Limit struct {
  Service     string       `json:"service" binding:"entityName"`
  Revision    string       `json:"revision"`
  CreatedAt   time.Time    `json:"createdAt"`
  UpdatedAt   time.Time    `json:"updatedAt"`
  DeletedAt   time.Time    `json:"deletedAt"`
  IsDeleting  bool         `json:"isDeleting"`
  LimitConfigs []*LimitConfig `json:"limitConfigs"`
}
```

```
type LimitConfig struct {
  Domain          string          `json:"domain"`
  LimitDescriptors []*LimitDescriptor `json:"limitDescriptors"`
}

type LimitDescriptor struct {
  LimitActions  []LimitAction  `json:"limitActions"`
  LimitOverride *LimitOverride `json:"limitOverride,omitempty"`
}

type LimitAction struct {
  HeaderValueMatch *HeaderValueMatch `json:"headerValueMatch,omitempty"`
  RequestHeaders   *RequestHeaders   `json:"requestHeaders,omitempty"`
  RemoteAddress    *RemoteAddress    `json:"remoteAddress,omitempty"`
  GenericKey       *GenericKey       `json:"genericKey,omitempty"`
}
```

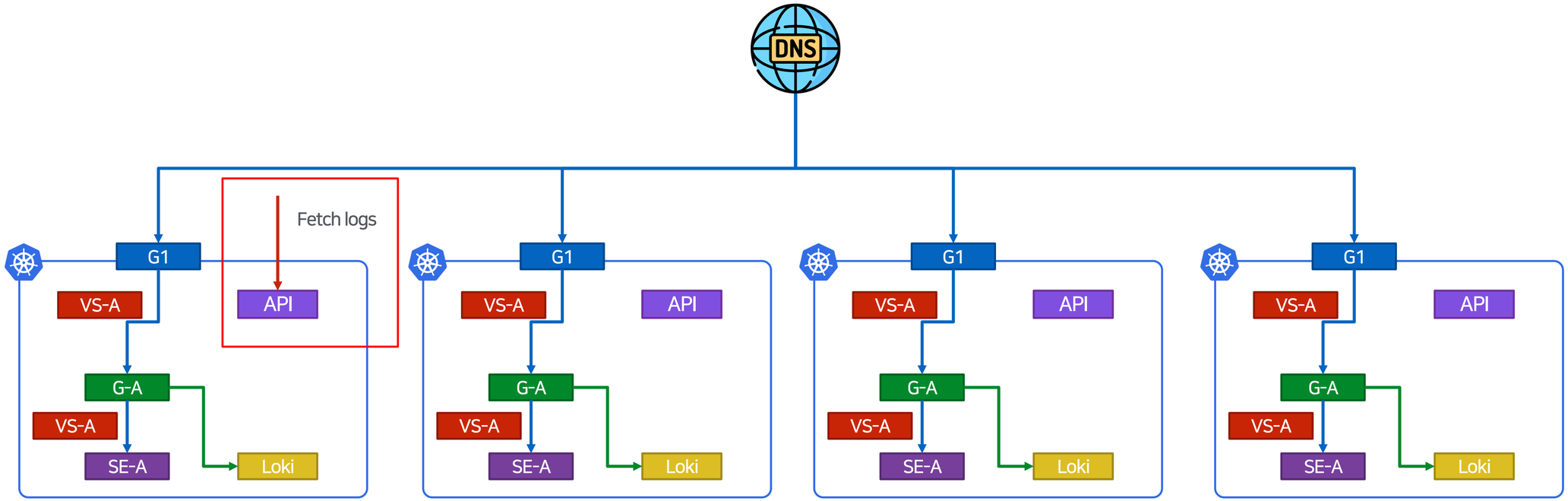
https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/

8. 기타

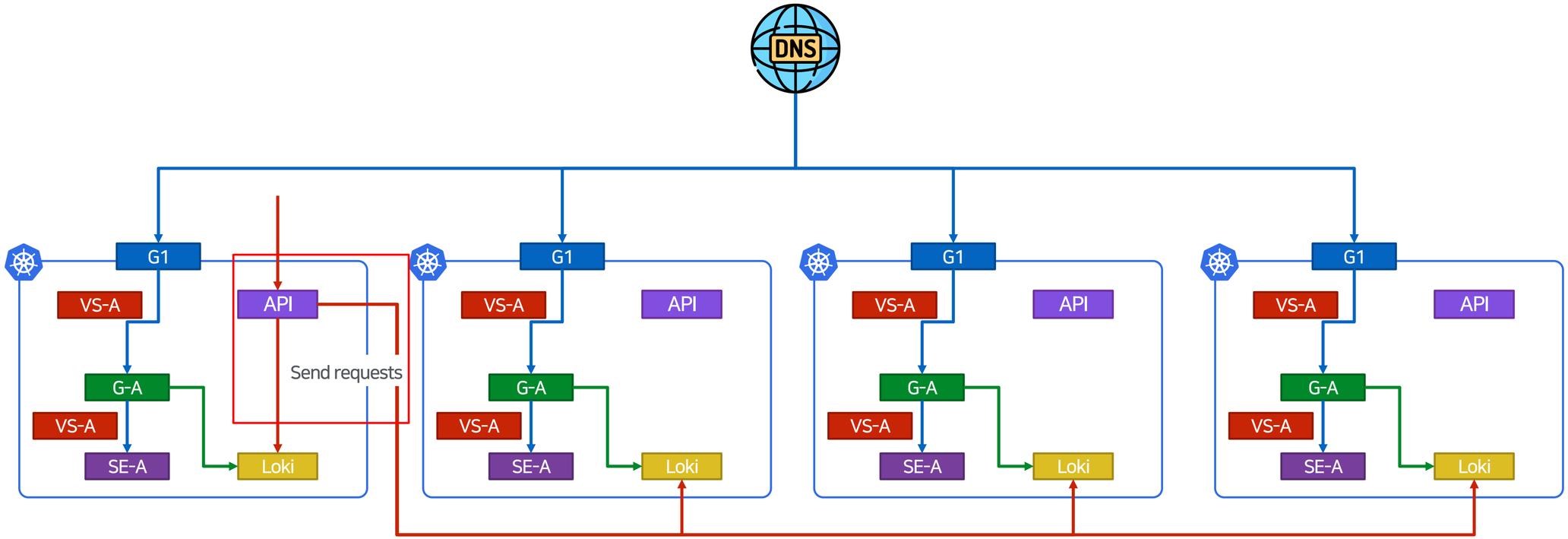
8.1 Logging


시스템 네임스페이스
1-tier Gateway
API Server
사용자 네임스페이스
2-tier GatewayA
Promtail
(Side-car)
ServiceEntryA
Loki

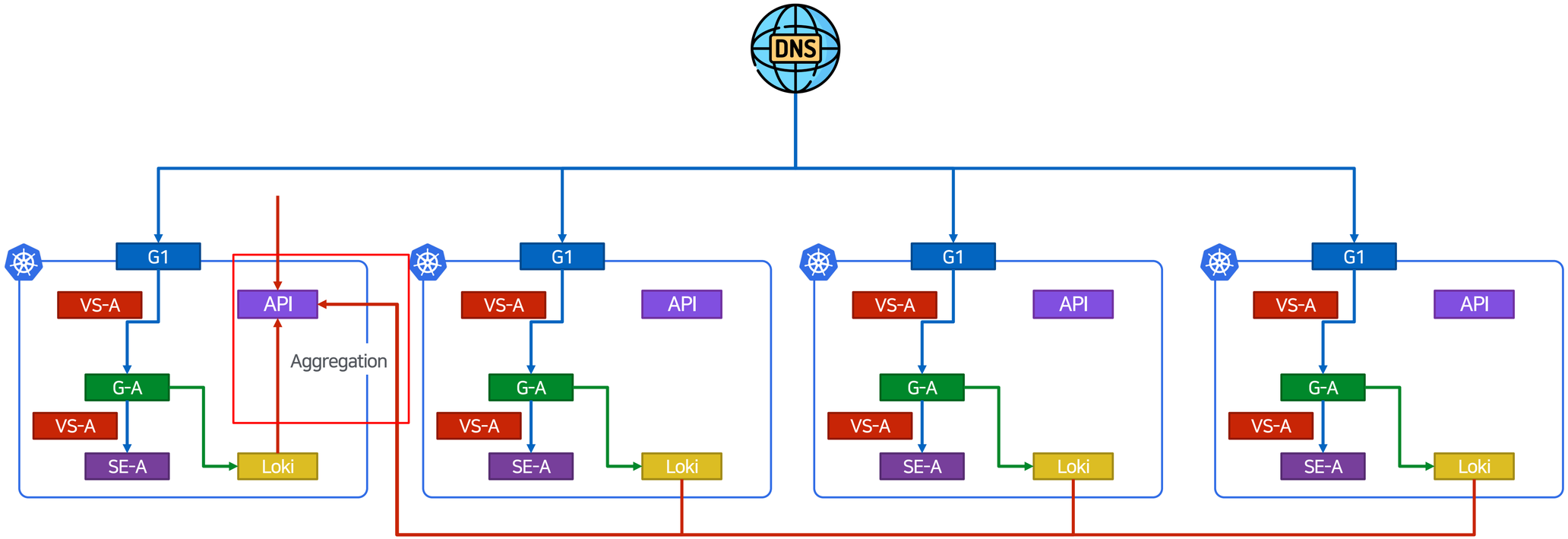
8.1 Logging – 아키텍처



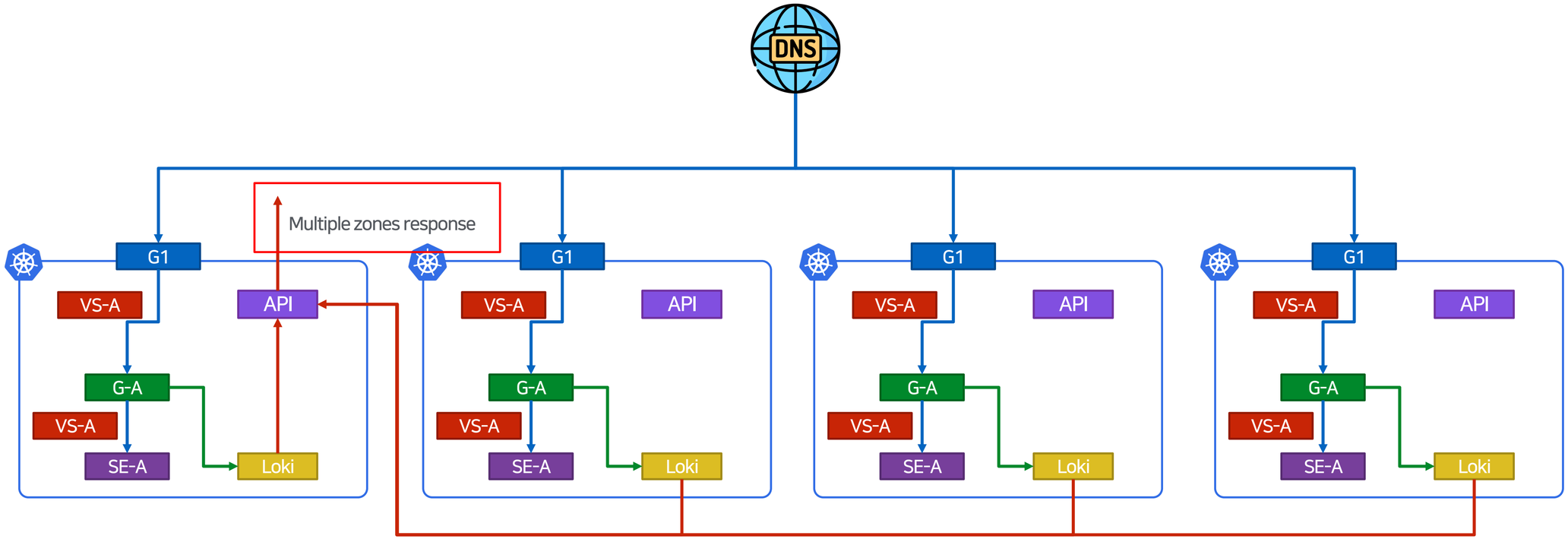
8.1 Logging – 아키텍처



8.1 Logging – 아키텍처

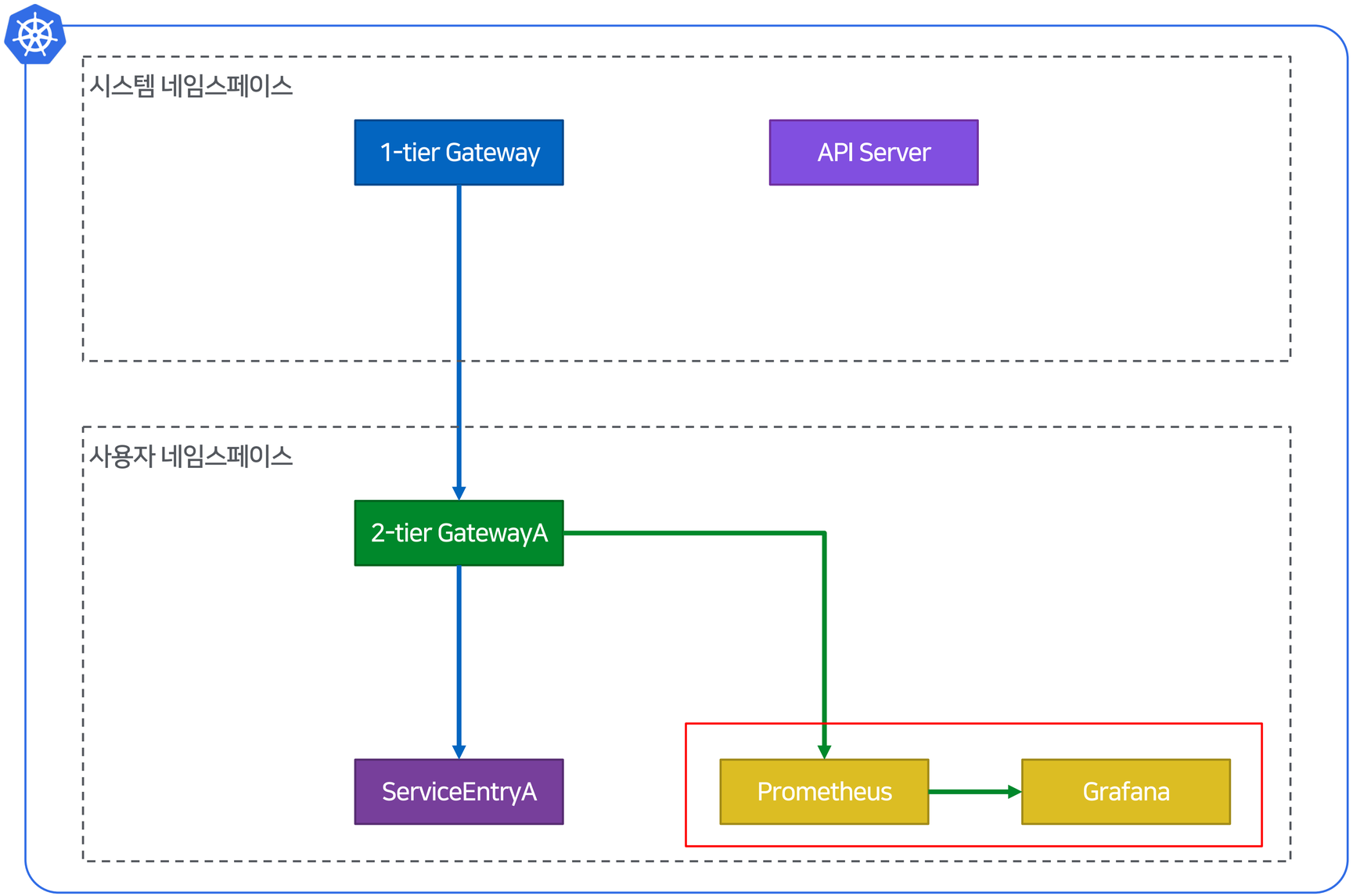


8.1 Logging – 아키텍처

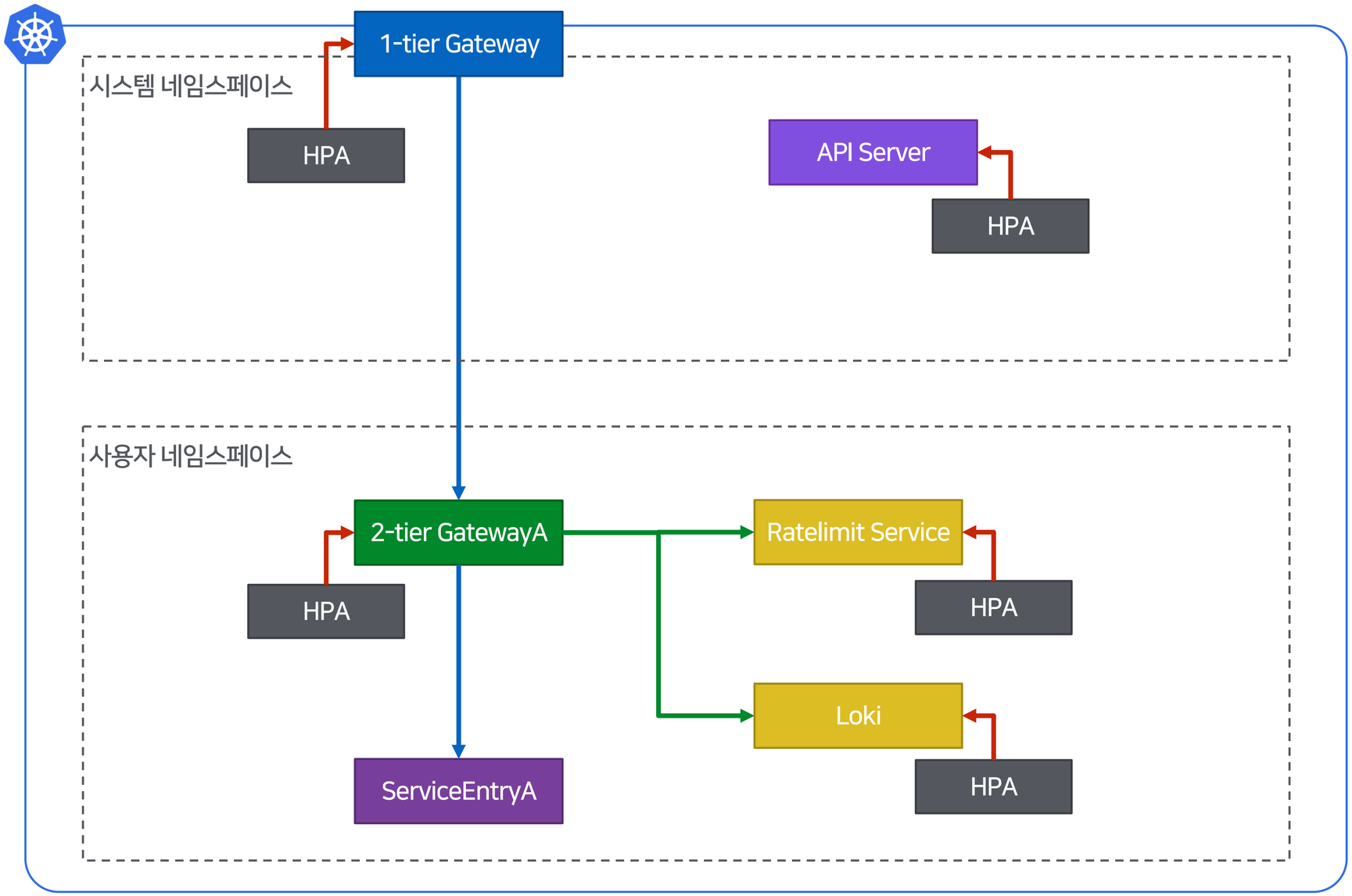


서비스 간 영향을 최소화하기 위한 아키텍처

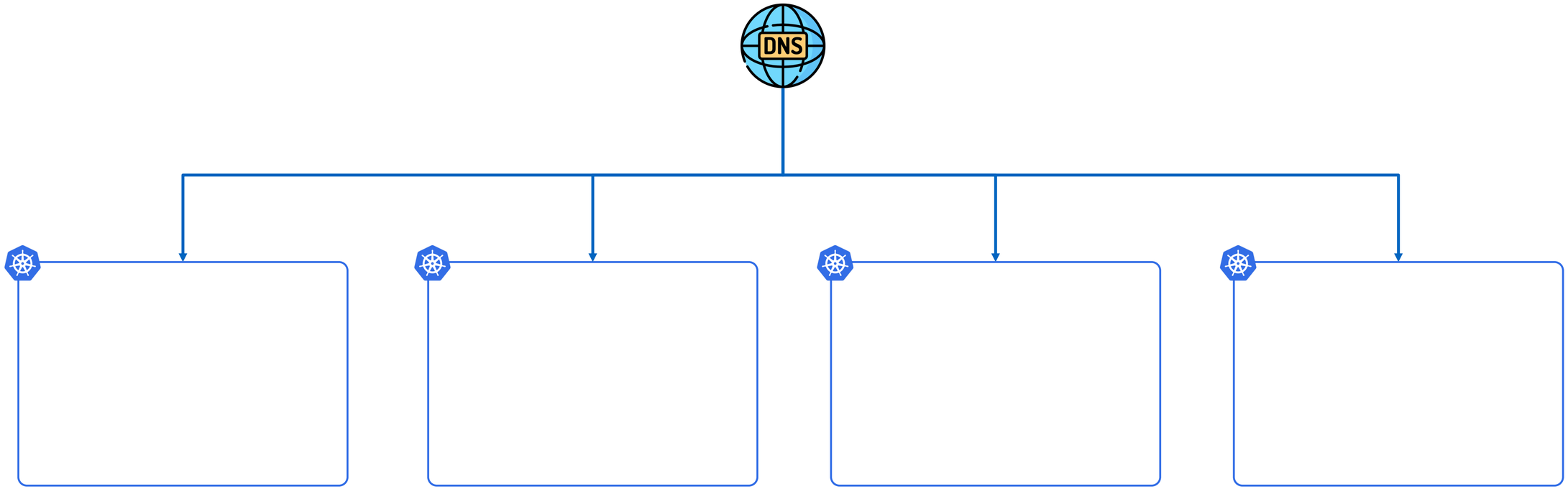
8.2 Metrics


시스템 네임스페이스
1-tier Gateway
API Server
사용자 네임스페이스
2-tier GatewayA
ServiceEntryA
Prometheus
Grafana

8.3 Auto Scaling - HPA

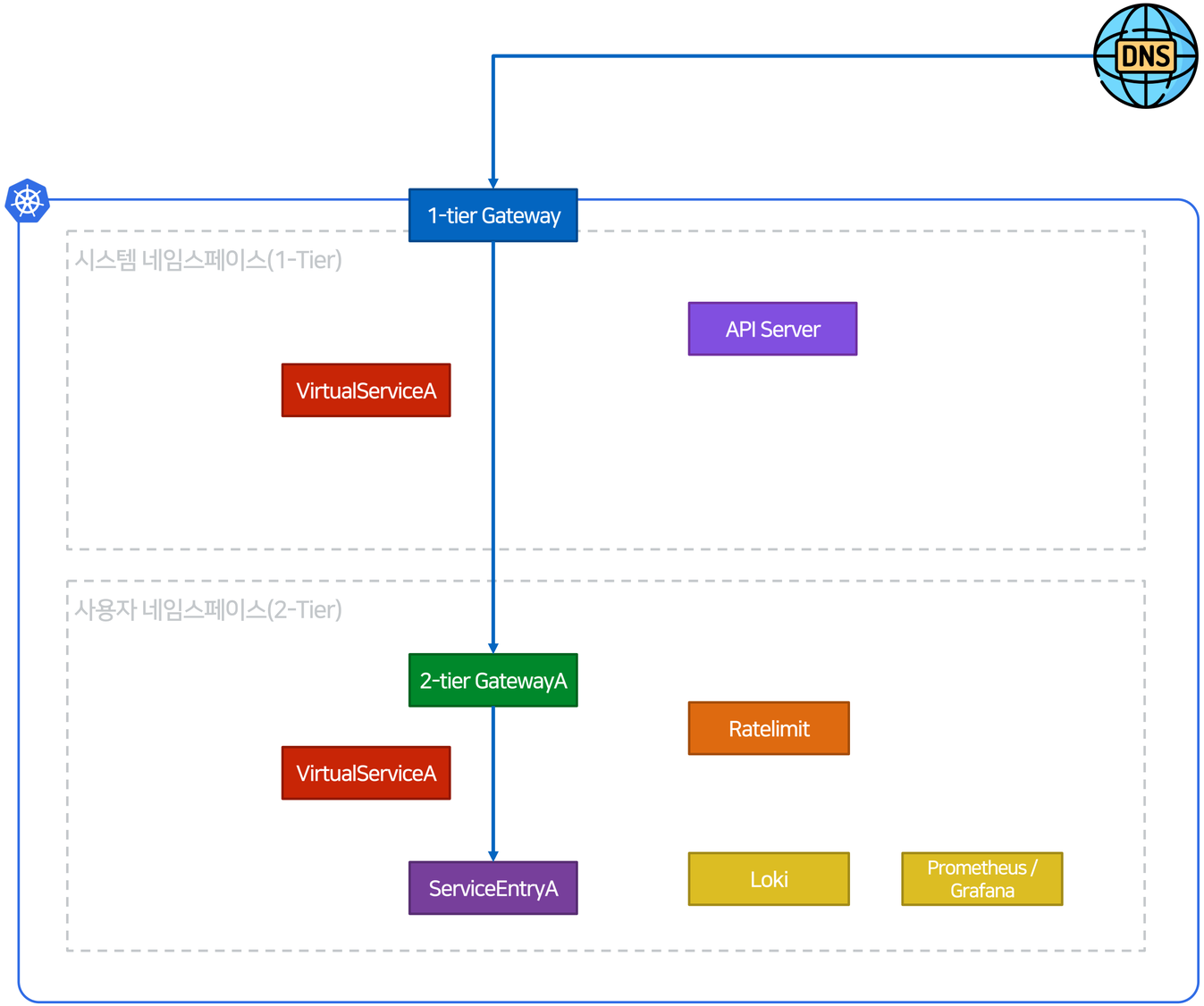


8.4 최종 정리 - IDC Failover를 위한 Multi-clusters 구조

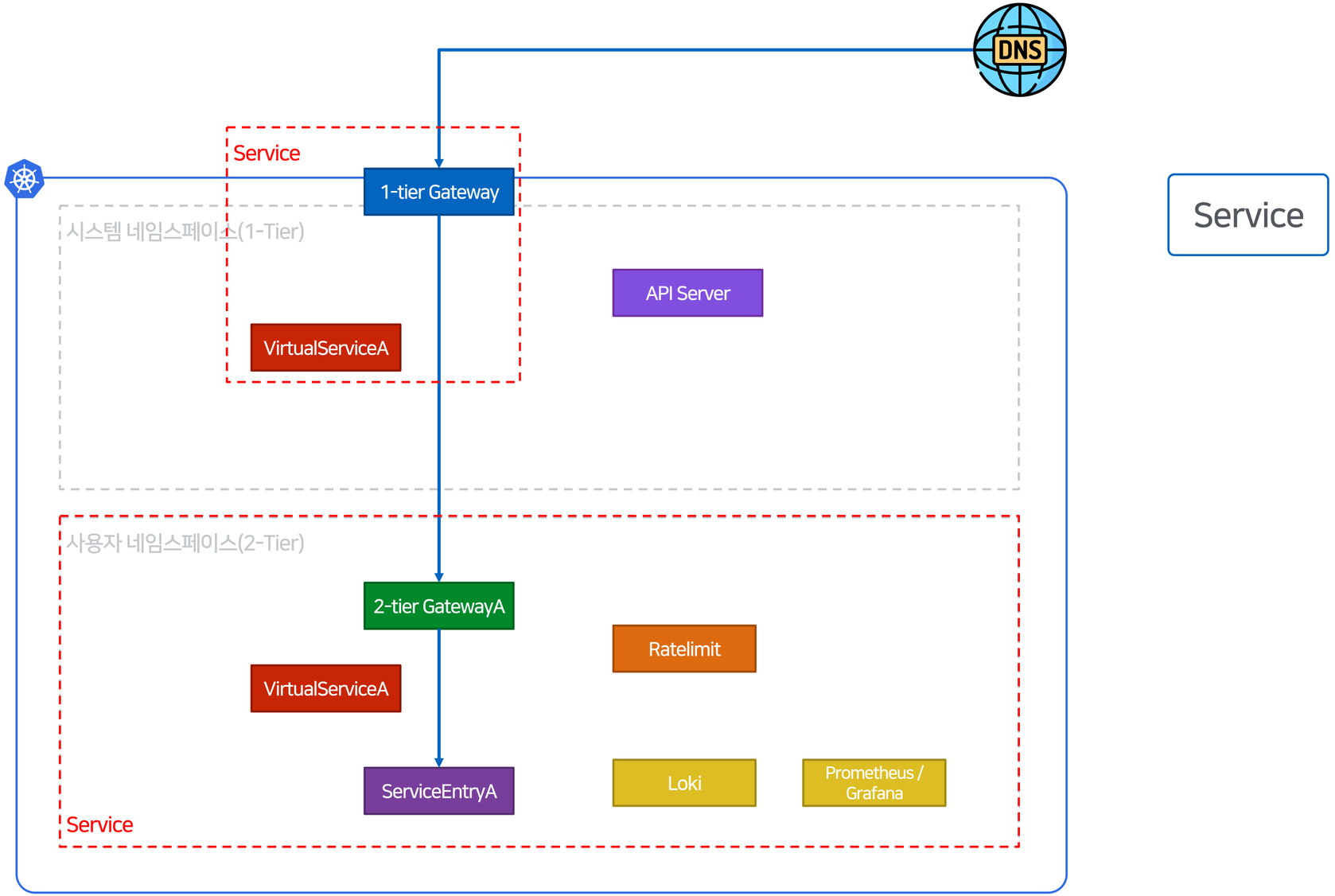


8.4 최종 정리 – Public IP를 아끼기 위한 2-Tier 구조

8.4 최종 정리 – 주요 컴포넌트

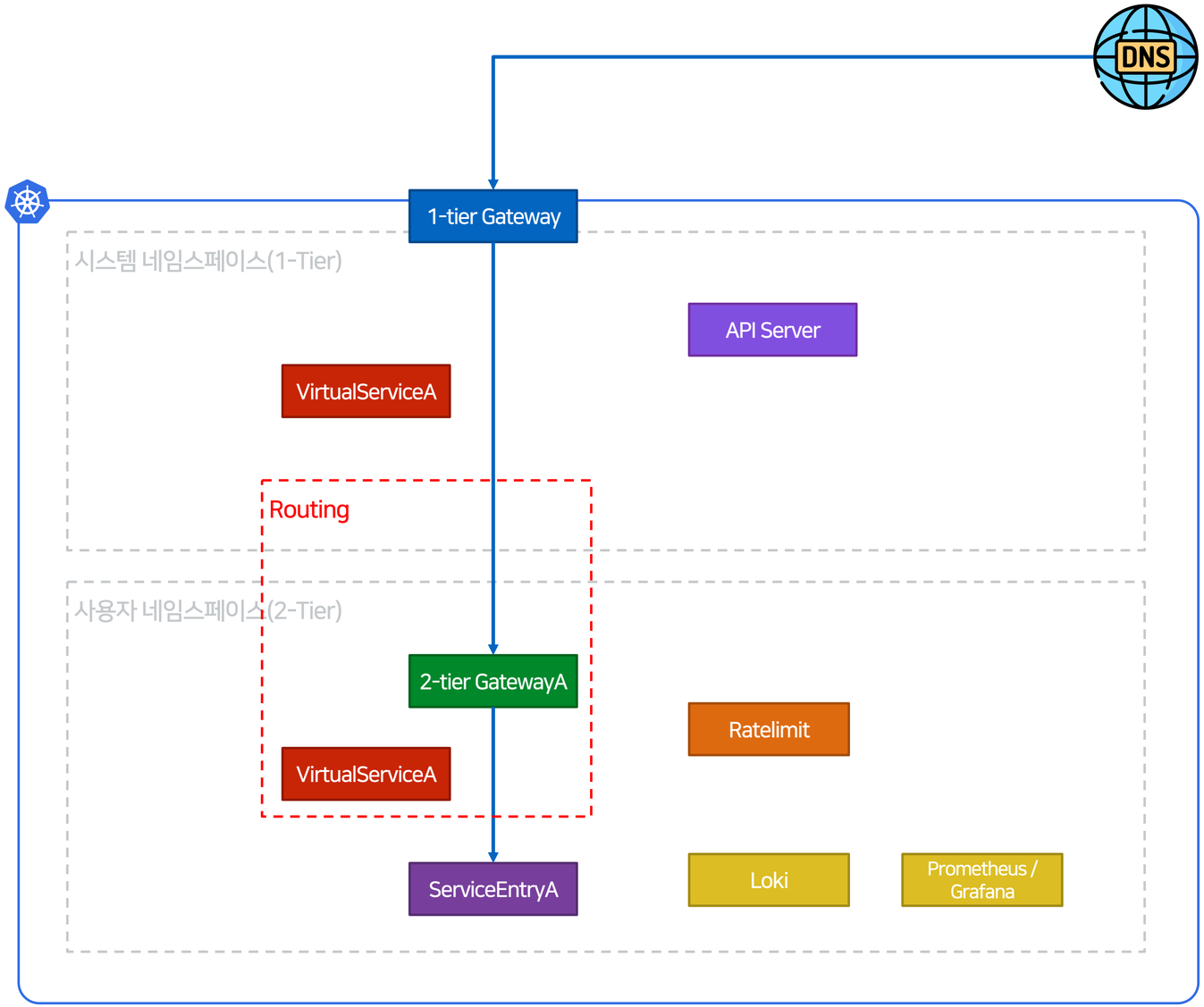


8.4 최종 정리 – 모델별 담당 컴포넌트



Service

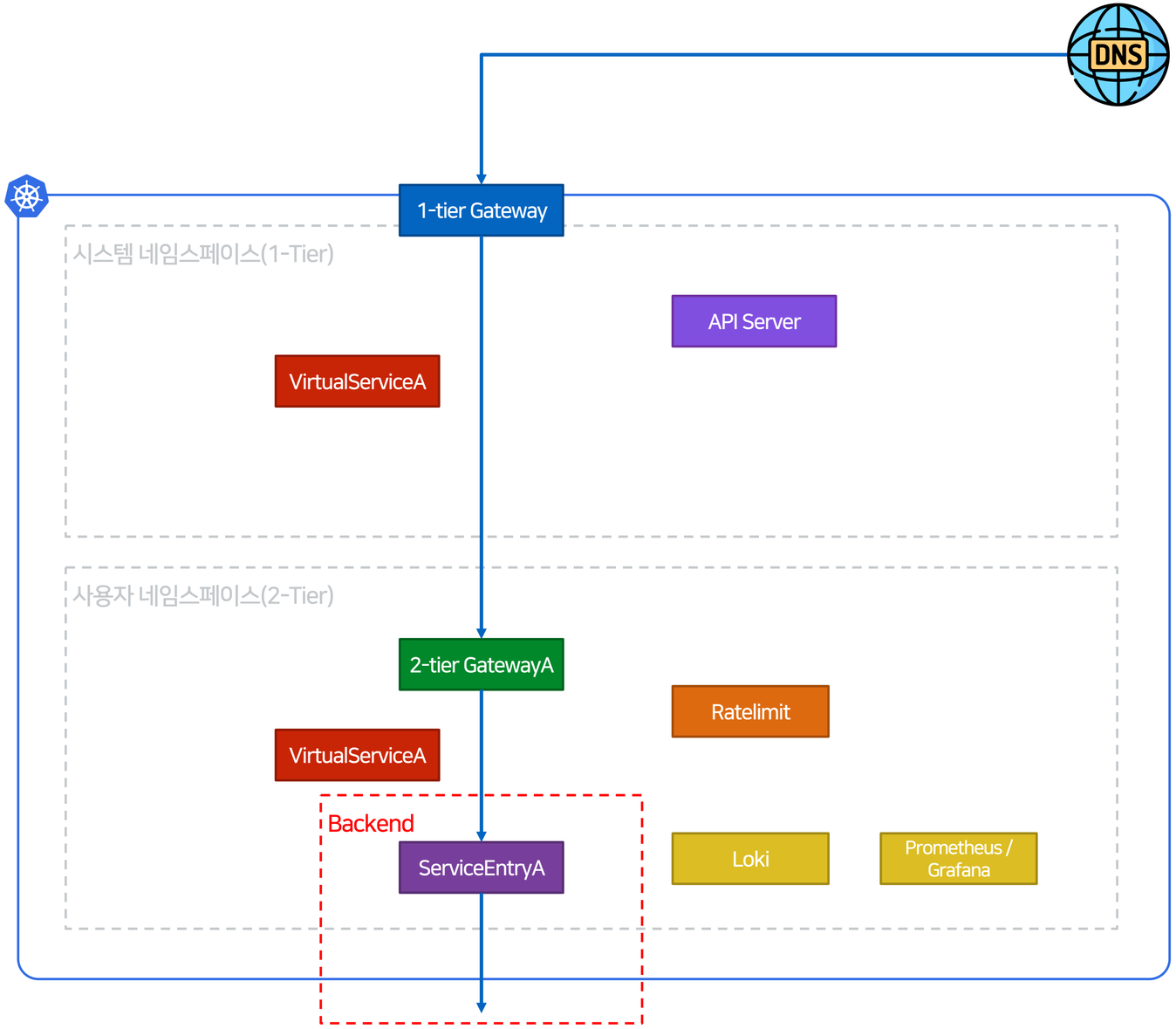
8.4 최종 정리 – 모델별 담당 컴포넌트



Service

Routing

8.4 최종 정리 – 모델별 담당 컴포넌트

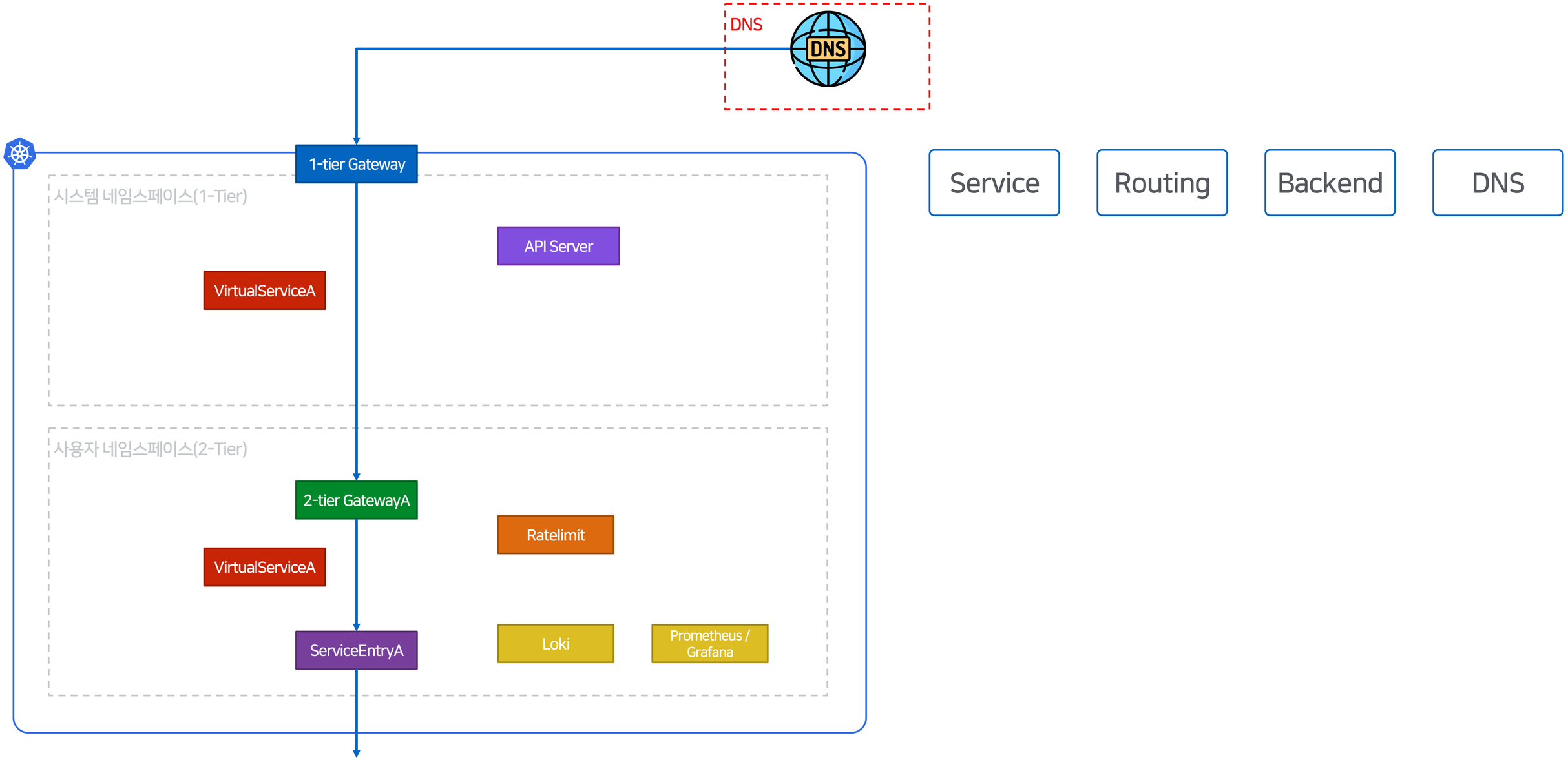


Service

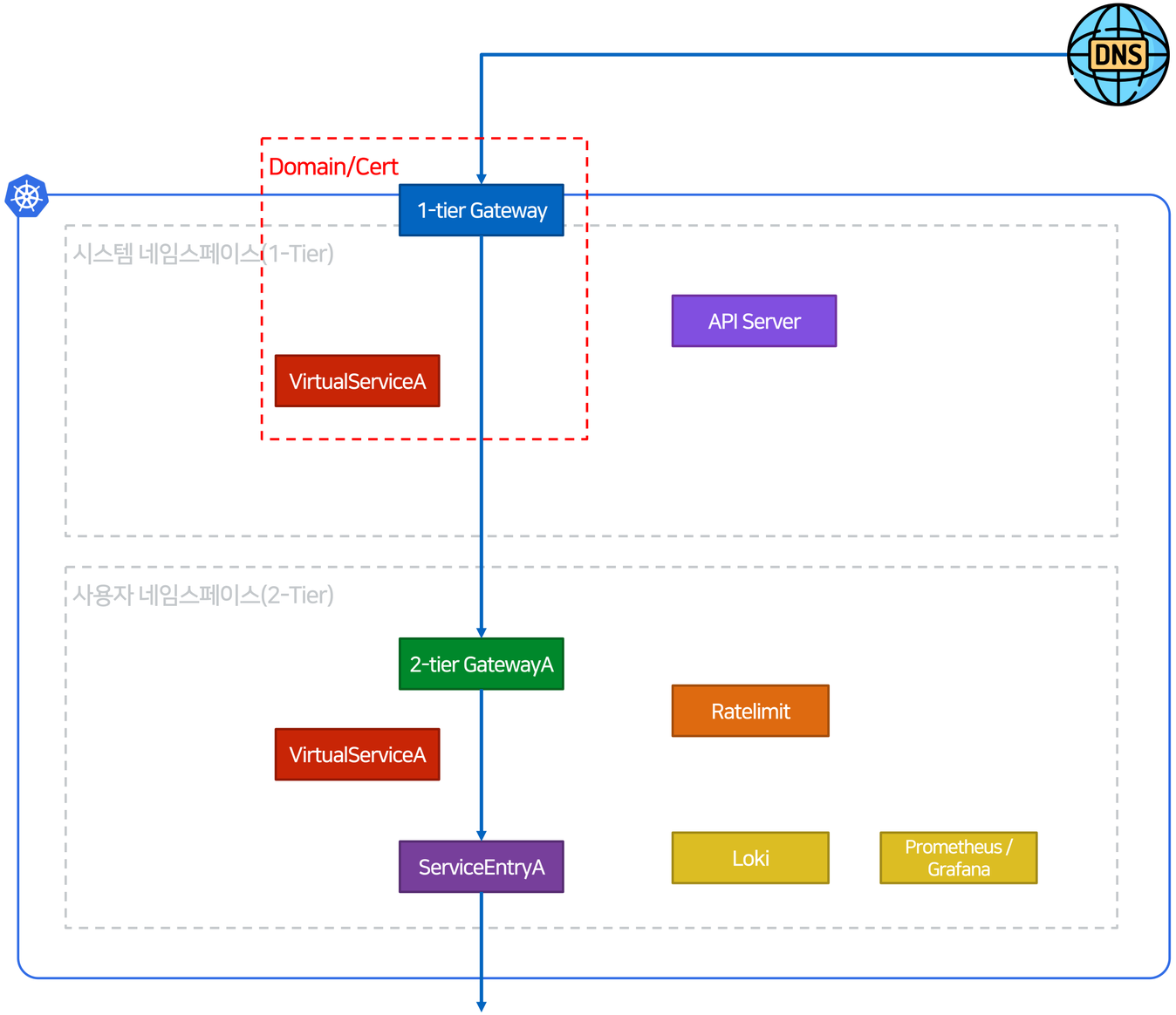
Routing

Backend

8.4 최종 정리 – 모델별 담당 컴포넌트



8.4 최종 정리 – 모델별 담당 컴포넌트



Service

Routing

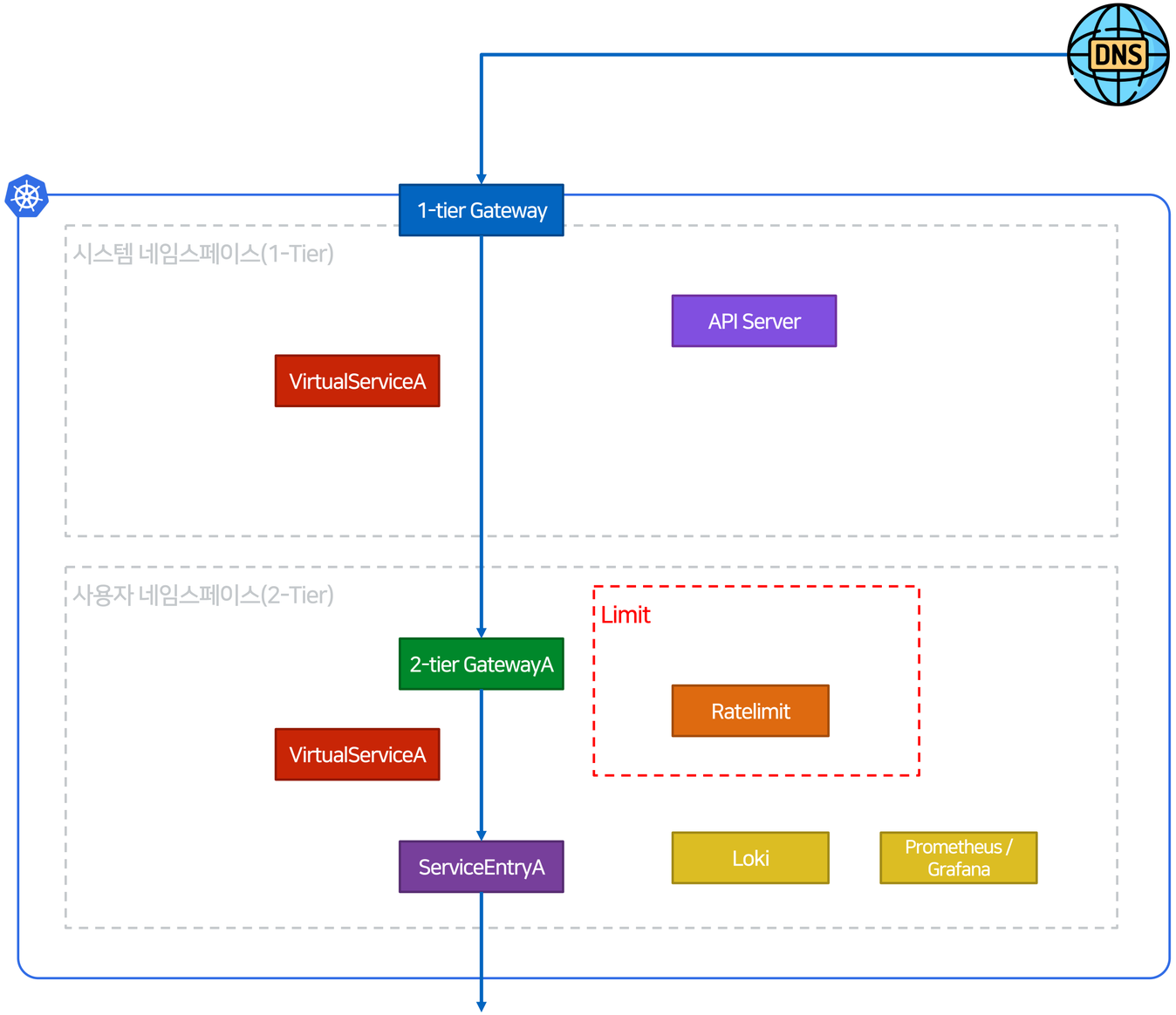
Backend

DNS

Domain

Cert

8.4 최종 정리 – 모델별 담당 컴포넌트



Service

Routing

Backend

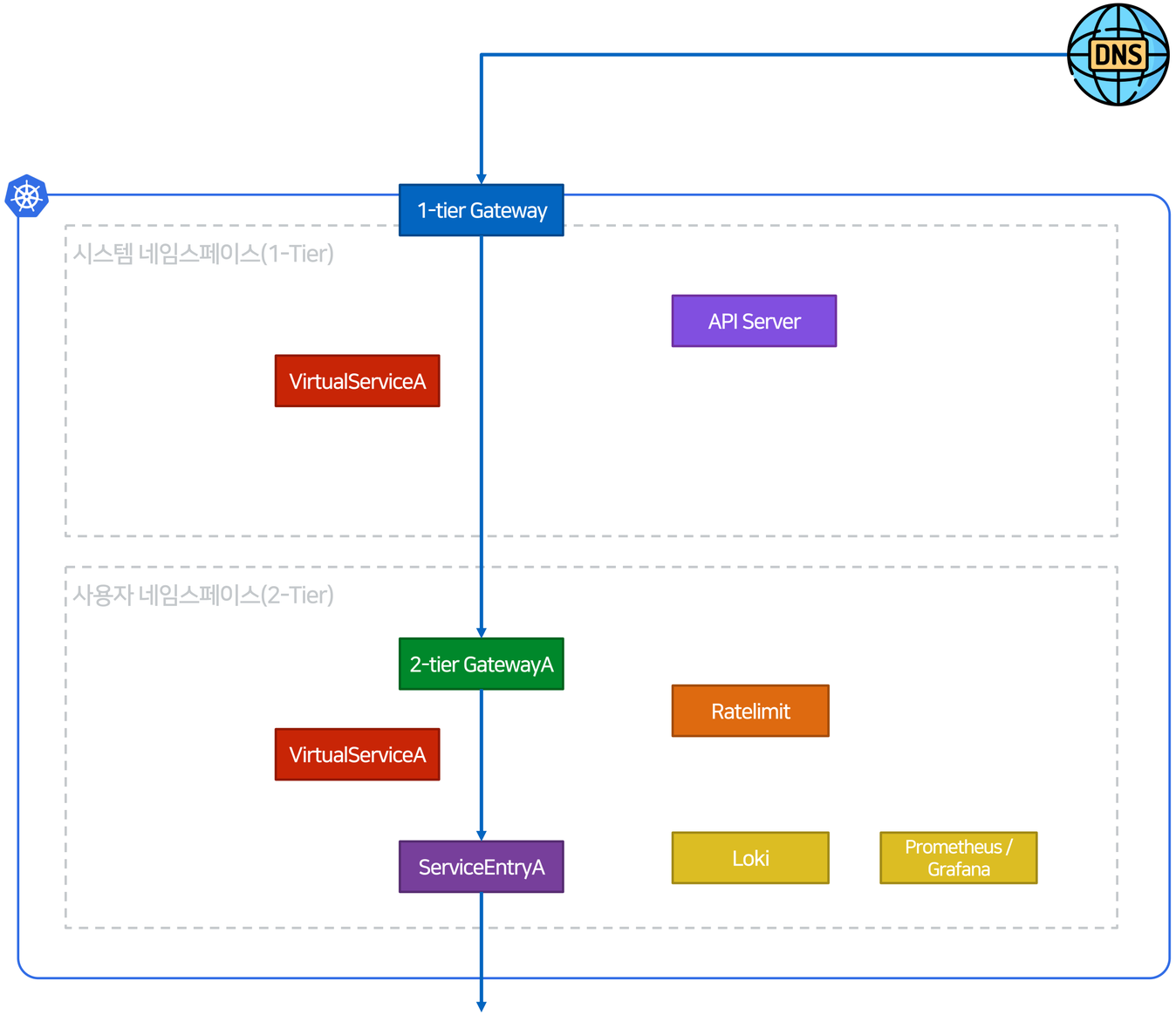
DNS

Domain

Cert

Limit

8.4 최종 정리 – 비동기 처리



Service | Routing | Backend | DNS

Domain | Cert | Limit

Declarative API

Producer/Consumer

Reconciliation Loop

Thank You