

2025 널카말카 2차 발표 주제

- GitOps — CI/CD with ArgoCD and Github Actions Workflows (10분)
- AWS CloudFront Image Resizing을 통한 LCP 최적화 (5분)
- Kubernetes Gateway API에 대해서 (시간 관계 상 아쉽지만 생략)

주제가 변경되어 K8s Gateway API에 대한 내용은 발표하지 않으나,
이미 3달 전부터 준비를 해뒀기 때문에 자료만 첨부하였음.

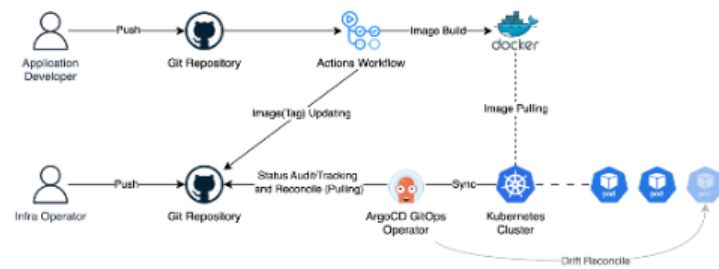
```
+ + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + +
+ +
+ +
```

+

+ +

+ +





@yulmaru (Kim Jun Young)

[Kubernetes CI/CD] GitOps with ArgoCD (Kustomize Demo)

2025-10-03

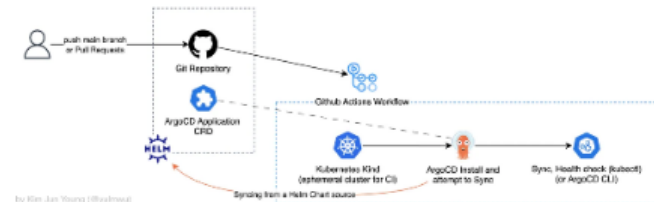
ArgoCD를 통한 쿠버네티스 GitOps 구성 및 Kustomize 배포

CI/CD

argocd

kubernetes

🔗 게시글 보기



by Kim Jun Young (@yulmaru)

[Kubernetes CI/CD] ArgoCD + CI with Github Actions and Kind

2025-12-06

Github Actions 및 Kind를 통한 쿠버네티스 ArgoCD CI(Continuous integration) 구성하기

CI/CD

argocd

kubernetes

🔗 게시글 보기



Sessions

Session 00 Presenter Introduction

Session 01 **GitOps** — Kubernetes CD with ArgoCD

 by KJY

Session 02 **Github Actions, KinD** — ArgoCD CI Pipeline

 by KJY

Session 00. Presenter Introduction



Kim Jun Young *@yulmwu*

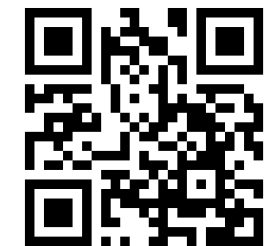
세명컴퓨터고등학교 스마트보안솔루션과 2학년

교내 클라우드 컴퓨팅/DevOps 전공동아리 Null4U 부장

- github.com/yulmwu
- Discord @rlawnsdud
- 010-2980-1336
- normal8781@gmail.com

DevOps/Backend 기술 블로그 운영

<https://velog.io/@yulmwu>



Session 01

DevOps는 알겠는데, GitOps가 뭐길래



kubernetes 에서 강조하는 걸까?

#CD #GitOps #ArgoCD #Kustomize



새로운 기능 만들었는데 배포하고 출시해주셈

FE/BE 개발팀

ㄴㄴ 지금 안됨; 테스트 제대로 해보심? 안정성이 최고임



인프라 운영팀



유저

신기능 나온다면서 언제나와요??????

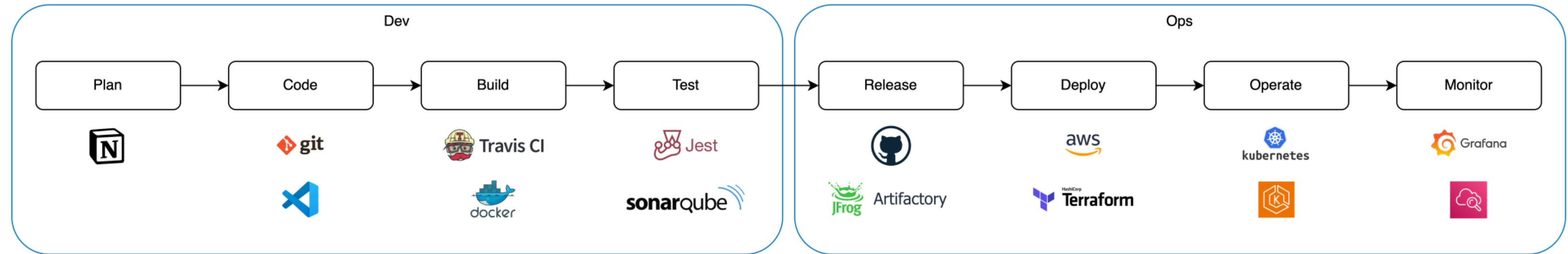


유저

이거 버그나는데요???

(서비스 종료)

DevOps = Development + Operations + α



단순 개발 + 인프라 운영 뿐만 아니라 CI/CD, 테스트, 모니터링, 보안 등 통합

장점: 빠른 개발부터 배포, 커뮤니케이션/협업 쉬워짐, 자동화 등

단점: 러닝 커브, 자동화 의존 등



DevOps 개발자 (노예)

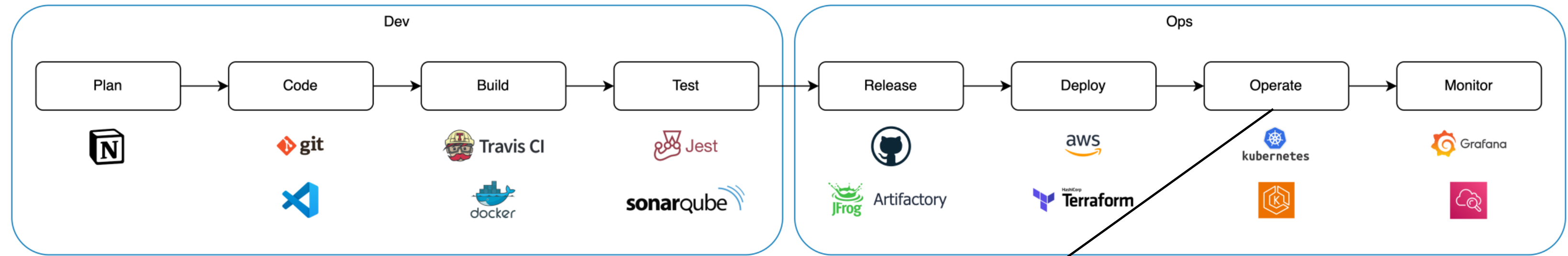
CI/CD = Continuous Integration, Continuous Deployment

개발, 테스트 부터 프로덕션까지의 배포 과정을 자동화하는 DevOps 방법론

CI

CD

Session 1. What is GitOps?



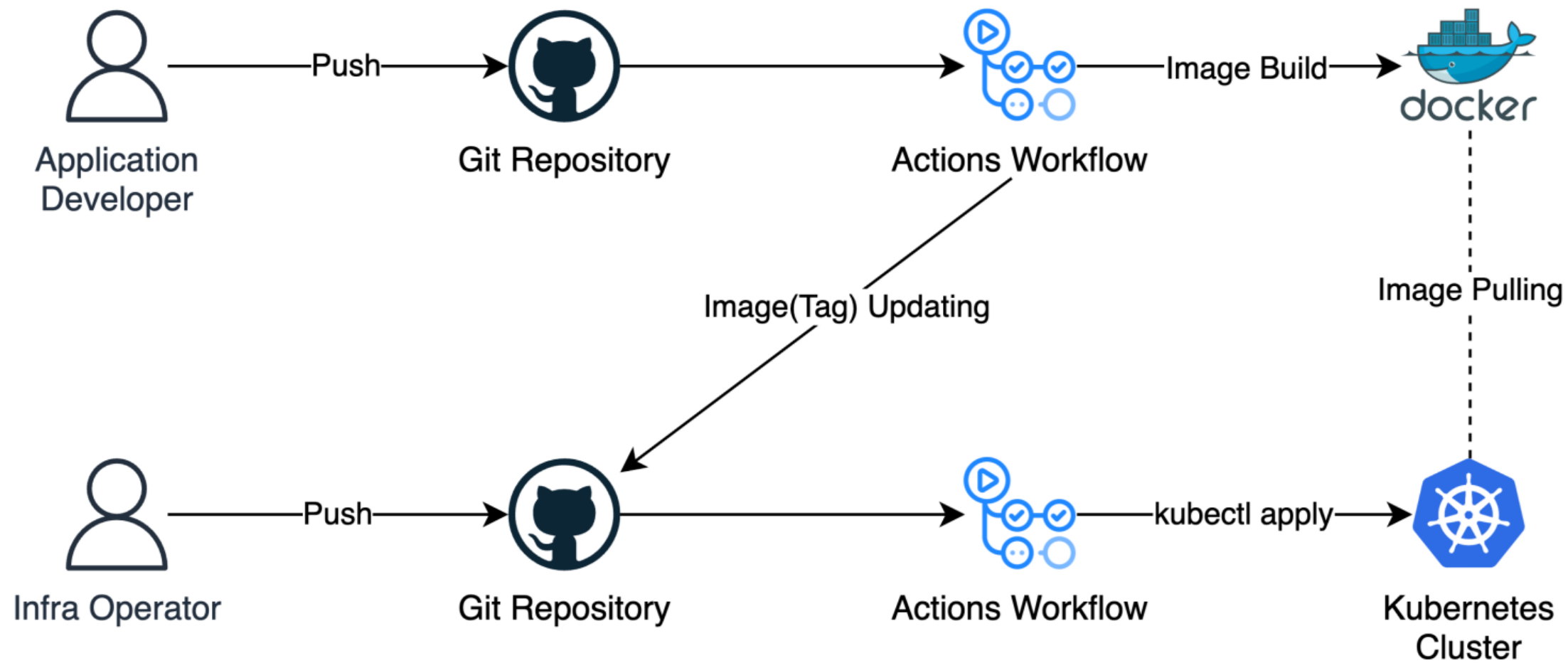
우리가 CI/CD에서 쿠버네티스를 어떻게 다룰까? 🤔

```
> kubectl apply -f deployment.yaml ...  
> kubectl -n app get service/app-service ...
```

(첫번째 시도) 이걸 CI/CD 파이프라인에 포함한다?

Session 1. What is GitOps?

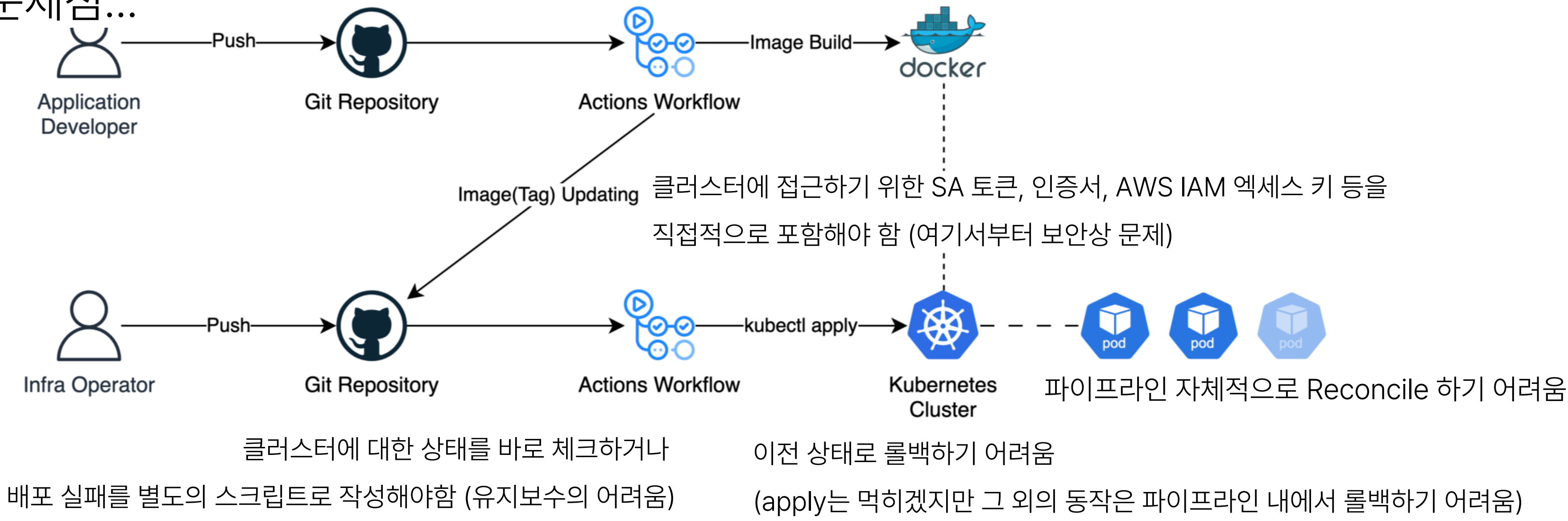
1) 애플리케이션 개발자가 소스코드를 올리고, CI/CD 파이프라인(워크플로우)에 따라 컨테이너 이미지화됨



2) 인프라 운영자는 쿠버네티스 매니페스트(클러스터 상태를 선언적으로 나타낸 구성 파일)를 깃 레포지토리에 올리고
이에 포함된 CI/CD 파이프라인에 따라 쿠버네티스 클러스터에 API(명령어)를 보냄

Session 1. What is GitOps?

문제점...

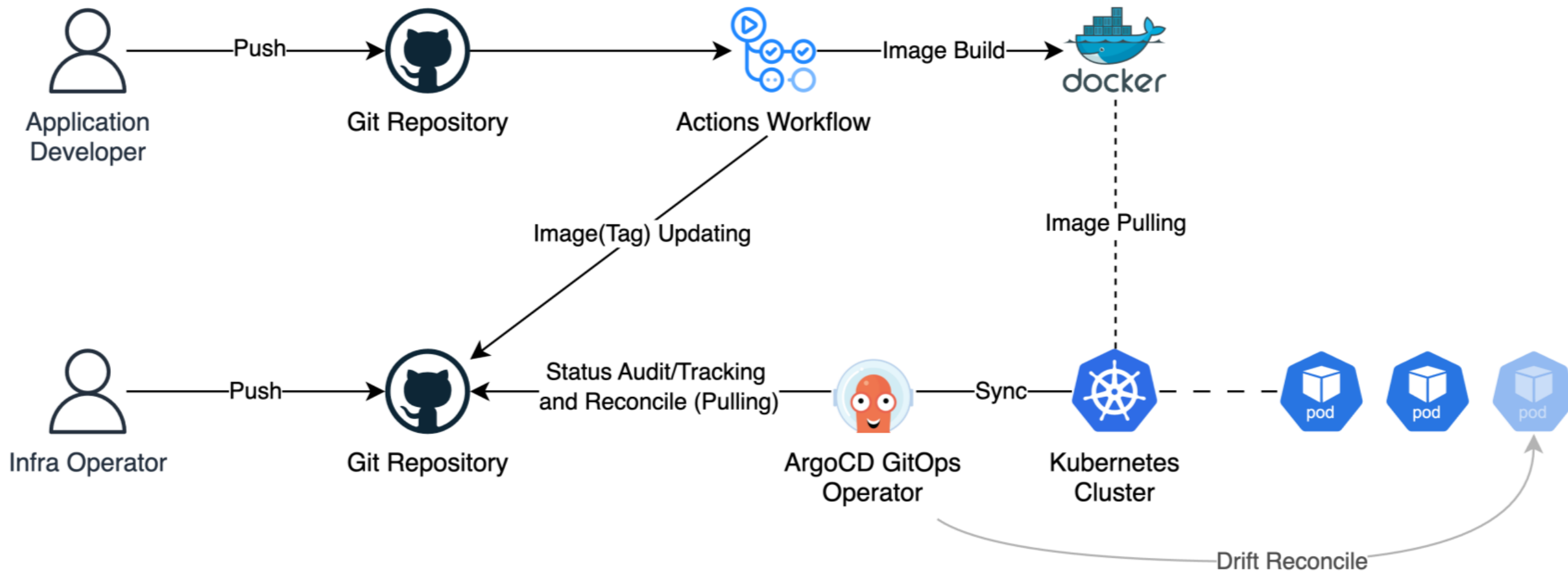


***Drift:** 원하고자 하는 상태(매니페스트에 적힌 상태)와 실제 상태가 다를 경우.
쿠버네티스에선 이런 경우 자동으로 조정(Reconcil)되는 것이 철학임

이건 좀..?
장점: 클러스터 내 구성할게 크게 없음

Session 1. What is GitOps?

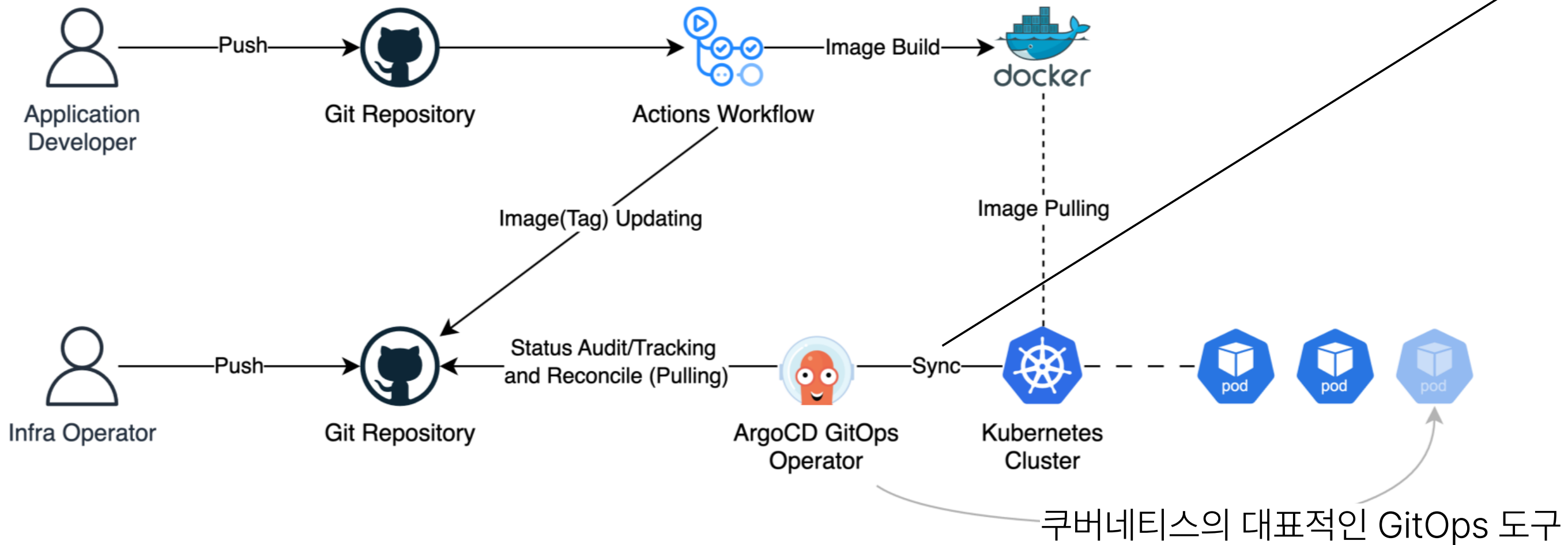
그럼 매니페스트 깃 레포지토리를 소스로 사용하면서 상태를 조정하는(Apply) 입장이
CI/CD 파이프라인이 아닌 쿠버네티스 클러스터 내부라면..?



GitOps = 깃 레포지토리를 소스(Single Source of Truth)로 사용하면서
애플리케이션의 배포와 관리를 자동화하는 DevOps 방법론

GitOps가 쿠버네티스에선...

클러스터에서 GitOps 도구가 깃 레포지토리(소스)를 Pull 하여 상태를 가져와 동기화함



Git의 버전 관리와 협업, Diff 및 롤백 등의 기능을 그대로 사용한 채로
클러스터를 자동으로 동기화하기 때문에 간편한 유지보수가 가능

ArgoCD, FluxCD

프로덕션 배포(CD)를 자동화하기 때문에 ~CD라는 네이밍이 쓰임

Declarative —

클러스터의 원하는 상태를 깃 레포지토리에 선언적 매니페스트로 저장

≈ 쿠버네티스 철학

Versioned and Immutable —

깃(**Git**)을 통한 버전 관리 및 버전 별 불변성,
Pull Request 등을 통한 상호 검토로 Human Error 방지

≈ 깃과 깃 레포지토리(Github 등), 협업 효율성 등 극대화

Pulled Automatically —

ArgoCD 등의 컨트롤러(에이전트)가 자동으로 깃 레포지토리를 **Pull**하여 상태 적용

≈ 깃 레포지토리에 대한 CI/CD 파이프라인에서
해당 클러스터로 Push 하는 모델(ex. kubectl)이 필요 없음

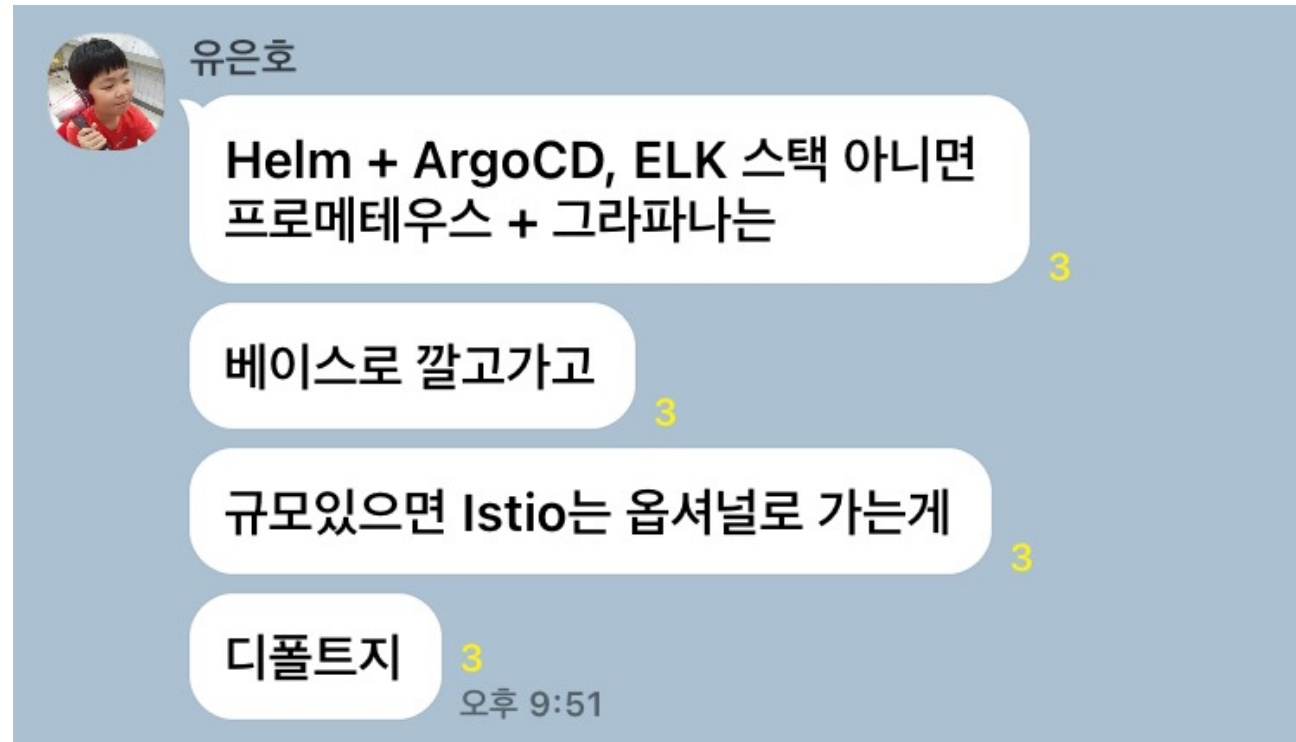
Continuously Reconciled —

깃 레포지토리의 선언된 상태에 맞도록 자동 조정(**Reconcile**)

≈ ArgoCD 등의 GitOps 도구가 Operator(Custom Controller) 형태로
클러스터에서 동작하기 때문에 Operator 자체적으로 Reconcile 할 수 있음

ArgoCD — 쿠버네티스의 GitOps 도구 중 하나, FluxCD와 함께 가장 많이 쓰임

- ArgoCD는 중앙 집중식으로 애플리케이션 단위로 동작한다면
- FluxCD는 비교적 가볍고 실시간 동기화, 모니터링에 집중된 도구



FluxCD 대신 ArgoCD를 선택한 이유?

⇒ Application CRD 단위로 구성할 수 있는게 구성
+ 유지보수 하기가 편하다고 판단함

TMI: 필자가 쿠버네티스 운영하면서 Helm, Prometheus, Grafana와 함께 항상 쓰던 도구 중 하나임

Session 1. What is ArgoCD?

Application CRD

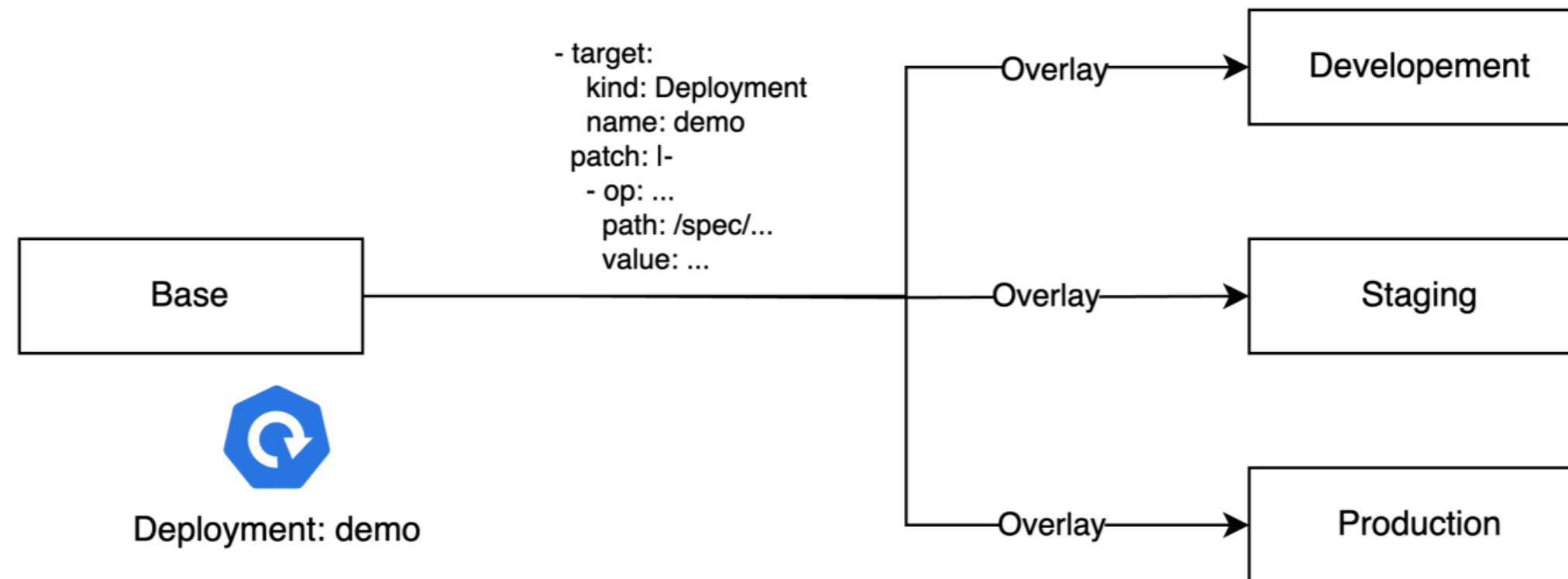
```
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: demo-prod
5    namespace: argocd
6  spec:
7    project: demo-proj
8    source:
9      repoURL: https://github.com/eocndp/argocd-kustomize-demo    깃 레포지토리, Branch, 경로 등 지정
10     targetRevision: main    소스로는 바닐라 매니페스트, Kustomize, Helm 차트를 지정할 수 있음
11     path: overlays/prod
12   destination:
13     server: https://kubernetes.default.svc
14     namespace: prod
15   syncPolicy:
16     automated:
17       prune: true
18       selfHeal: true
19   syncOptions:
20     - CreateNamespace=true
21     - timeout.reconcile=30s
```

Kustomize? Helm?

Session 1. What is Kustomize?

Kustomize = 쿠버네티스 매니페스트를 커스터마이징하는 도구

Base 매니페스트를 바탕으로 여러 환경(예: Development, Production) 별로 오버레이를 선언하여 쉽게 리소스를 생성할 수 있음



kubectl 기본 내장, Helm과는 다르게 패키지 형태가 아닌, 베이스 매니페스트 위를 덮어쓰는 오버레이 기반임

Session 1. What is Helm?

Helm = 쿠버네티스 패키지 관리자, 여러 매니페스트를 패키지(차트, Chart)로 묶어 관리함

Kustomize와는 다르게 오버레이 기반이 아닌 템플릿 기반임

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: demo-app
5  spec:
6    selector:
7      app: demo-app
8    ports:
9      - name: http
10      port: {{ .Values.service.port }}
11      targetPort: 80
12    type: {{ .Values.service.type }}
```

(Go Template 기반)

```
1  {{- define "demo-app.name" -}}
2    {{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
3  {{- end -}}
4
5  {{- define "demo-app.fullname" -}}
6    {{- if .Values.fullnameOverride -}}
7      {{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" -}}
8    {{- else -}}
9      {{- $name := include "demo-app.name" . -}}
10     {{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
11   {{- end -}}
12 {{- end -}}
```

템플릿 기반으로 재사용의 효율, 단순화/표준화 등이 쉽기 가능하고 때문에 CI/CD 파이프라인 뿐만 아니라 사실상 쿠버네티스 표준임

Session 1. Kustomize Demo

(Helm Chart를 통한 예제는 CI에서 실습해보겠습니다)

```
.
├── base
│   ├── deployment.yaml
│   └── kustomization.yaml
└── overlays
    ├── dev
    │   └── kustomization.yaml
    └── prod
        └── kustomization.yaml
```

Kustomize 디렉토리 구조

/base엔 오버레이의 베이스 매니페스트를, /overlays 디렉토리엔
Prod, Dev 환경 등의 오버레이 로직을 지정함

(kustomization.yaml은 Kustomize 설정 파일, Base에선 사용할 리소스나 패치 내역 등 선언)

Session 1. Kustomize Demo

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: demo
5    labels: { app: demo }
6  spec:
7    replicas: 1
8    selector:
9      matchLabels: { app: demo }
10   template:
11     metadata:
12       labels: { app: demo }
13     spec:
14       containers:
15       - name: web
16         image: rlawnsdud/demo:arm
17         ports:
18         - containerPort: 3030
19         env:
20         - name: HOST
21           value: "0.0.0.0"
22         - name: PORT
23           value: "3030"
```

```
25  apiVersion: v1
26  kind: Service
27  metadata:
28    name: demo
29    labels: { app: demo }
30  spec:
31    selector: { app: demo }
32    ports:
33    - name: http
34      port: 80
35      targetPort: 3030
```

base/deployment.yaml (Service 포함)
(이를 Patch하여 오버레이함)

Session 1. Kustomize Demo

```
1  resources:
2    - ../../base
3  namePrefix: prod-
4  commonLabels:
5    env: prod
6  patches:
7    - target:
8        kind: Deployment
9        name: demo
10   patch: |-
11     - op: add
12       path: /spec/template/spec/containers/0/env/-
13       value:
14         name: APP_NAME
15         value: "Prod Application"
```

```
1  resources:
2    - ../../base
3  namePrefix: dev-
4  commonLabels:
5    env: dev
6  patches:
7    - target:
8        kind: Deployment
9        name: demo
10   patch: |-
11     - op: add
12       path: /spec/template/spec/containers/0/env/-
13       value:
14         name: APP_NAME
15         value: "Dev Application"
```

(dev도 같은 형태로 선언)

overlays/prod/kustomization.yaml

프로덕션 환경의 오버레이: 예시로 환경 변수의 APP_NAME을 변경하는 예시

이제 ArgoCD Application에서
해당 Kustomize 프로젝트를 소스로 사용하도록 해보자

Session 1. ArgoCD Demo

```
kubectl create namespace argocd

helm repo add argo https://argoproj.github.io/argo-helm
helm repo update
helm install argocd argo/argo-cd \
  --namespace argocd \
  --set server.service.type=NodePort # or LoadBalancer
```

(values.yaml 대신 --set 옵션으로 템플릿에 값 넣기 가능)

Helm Chart로 패키징되어 있는 ArgoCD Operator 설치

설치 후 리소스 확인

•> kubectl get all -n argocd

NAME	READY	STATUS	RESTARTS	AGE
pod/argocd-application-controller-0	1/1	Running	0	166m
pod/argocd-applicationset-controller-5df7d666fb-qs97v	1/1	Running	0	166m
pod/argocd-dex-server-7cd5d4988c-thknm	1/1	Running	0	166m
pod/argocd-notifications-controller-669f79bfd7-jdh8z	1/1	Running	0	166m
pod/argocd-redis-5fcd8bd66d-srcc2	1/1	Running	0	166m
pod/argocd-repo-server-68877c5db9-6cd64	1/1	Running	0	166m
pod/argocd-server-67d79d7d87-864gg	1/1	Running	0	166m

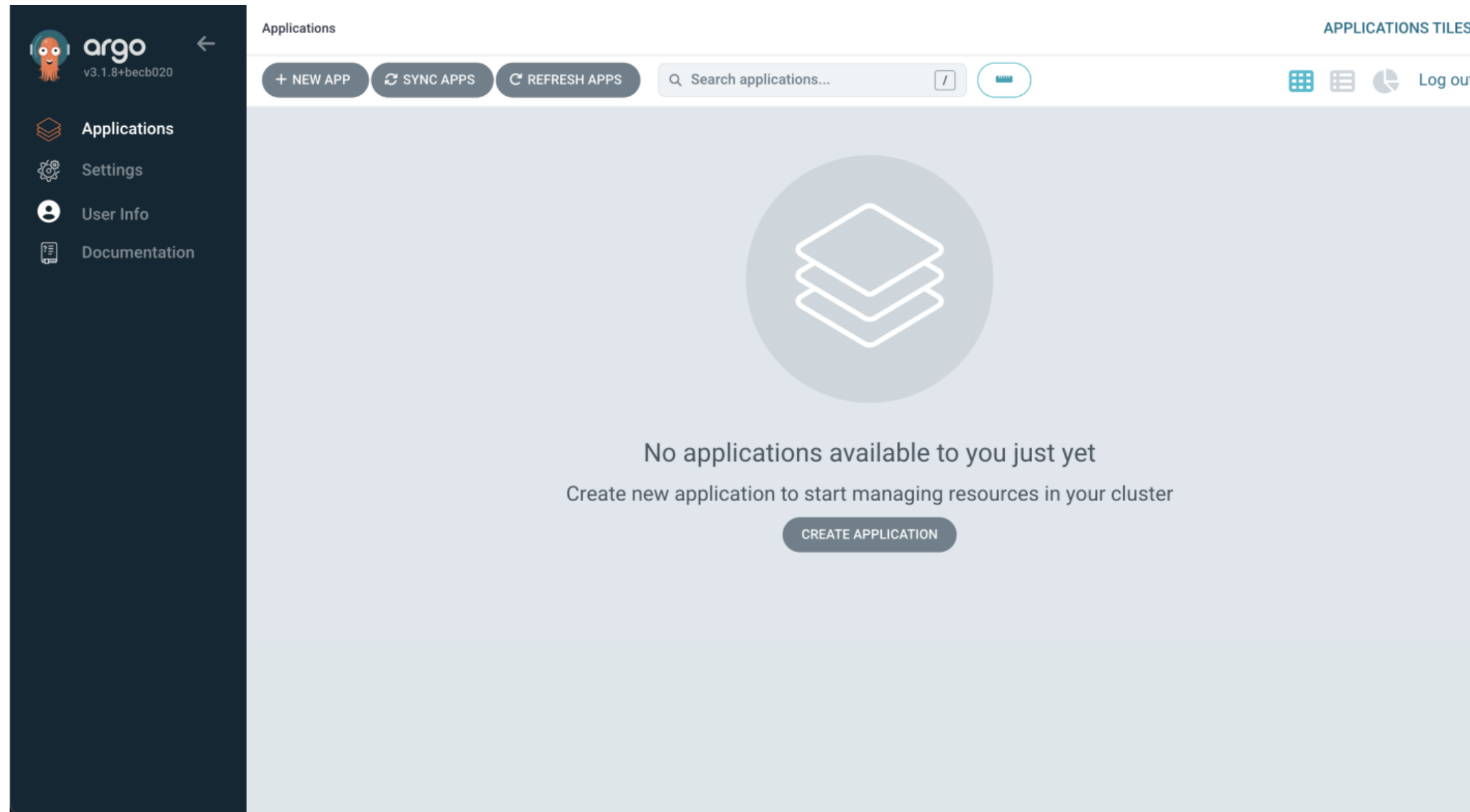
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/argocd-applicationset-controller	ClusterIP	10.105.156.202	<none>	7000/TCP	166m
service/argocd-dex-server	ClusterIP	10.108.15.42	<none>	5556/TCP,5557/TCP	166m
service/argocd-redis	ClusterIP	10.96.34.212	<none>	6379/TCP	166m
service/argocd-repo-server	ClusterIP	10.109.249.87	<none>	8081/TCP	166m
service/argocd-server	NodePort	10.98.143.248	<none>	80:30080/TCP,443:30443/TCP	166m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/argocd-applicationset-controller	1/1	1	1	166m
deployment.apps/argocd-dex-server	1/1	1	1	166m
deployment.apps/argocd-notifications-controller	1/1	1	1	166m
deployment.apps/argocd-redis	1/1	1	1	166m
deployment.apps/argocd-repo-server	1/1	1	1	166m
deployment.apps/argocd-server	1/1	1	1	166m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/argocd-applicationset-controller-5df7d666fb	1	1	1	166m
replicaset.apps/argocd-dex-server-7cd5d4988c	1	1	1	166m
replicaset.apps/argocd-notifications-controller-669f79bfd7	1	1	1	166m
replicaset.apps/argocd-redis-5fcd8bd66d	1	1	1	166m
replicaset.apps/argocd-repo-server-68877c5db9	1	1	1	166m
replicaset.apps/argocd-server-67d79d7d87	1	1	1	166m

NAME	READY	AGE
statefulset.apps/argocd-application-controller	1/1	166m

Session 1. ArgoCD Demo



ArgoCD는 Application(CRD로 선언 가능) 단위로 소스를 지정받아 GitOps CD를 진행함

Session 1. ArgoCD Demo

```
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: demo-prod
5    namespace: argocd
6  spec:
7    project: demo-proj
8    source:
9      repoURL: https://github.com/eocndp/argocd-kustomize-demo
10     targetRevision: main
11     path: overlays/prod
12  destination:
13     server: https://kubernetes.default.svc
14     namespace: prod
15  syncPolicy:
16     automated:
17       prune: true
18       selfHeal: true
19  syncOptions:
20     - CreateNamespace=true
21     - timeout.reconcile=30s
```

```
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: demo-dev
5    namespace: argocd
6  spec:
7    project: demo-proj
8    source:
9      repoURL: https://github.com/eocndp/argocd-kustomize-demo
10     targetRevision: main
11     path: overlays/dev
12  destination:
13     server: https://kubernetes.default.svc
14     namespace: dev
15  syncPolicy:
16     automated:
17       prune: true
18       selfHeal: true
19  syncOptions:
20     - CreateNamespace=true
21     - timeout.reconcile=30s
```

Kustomize에서 프로덕션 오버레이(prod)와 개발용 오버레이(dev)를 경로를 기반으로 나눠뒀기 때문에 overlays/ path 지정

Session 1. ArgoCD Demo

```
• > kubectl get application -n argocd -o wide
```

NAME	SYNC STATUS	HEALTH STATUS	REVISION	PROJECT
demo-dev	Synced	Healthy	72f4386252027d34ef8262f37dd61841b4965538	demo-proj
demo-prod	Synced	Healthy	72f4386252027d34ef8262f37dd61841b4965538	demo-proj

Synced = 바라보는 소스코드와 현재 클러스터 상태가 일치함 !

Healthy = 동기화된 클러스터 상태가 문제가 없음 !

Session 1. ArgoCD Demo

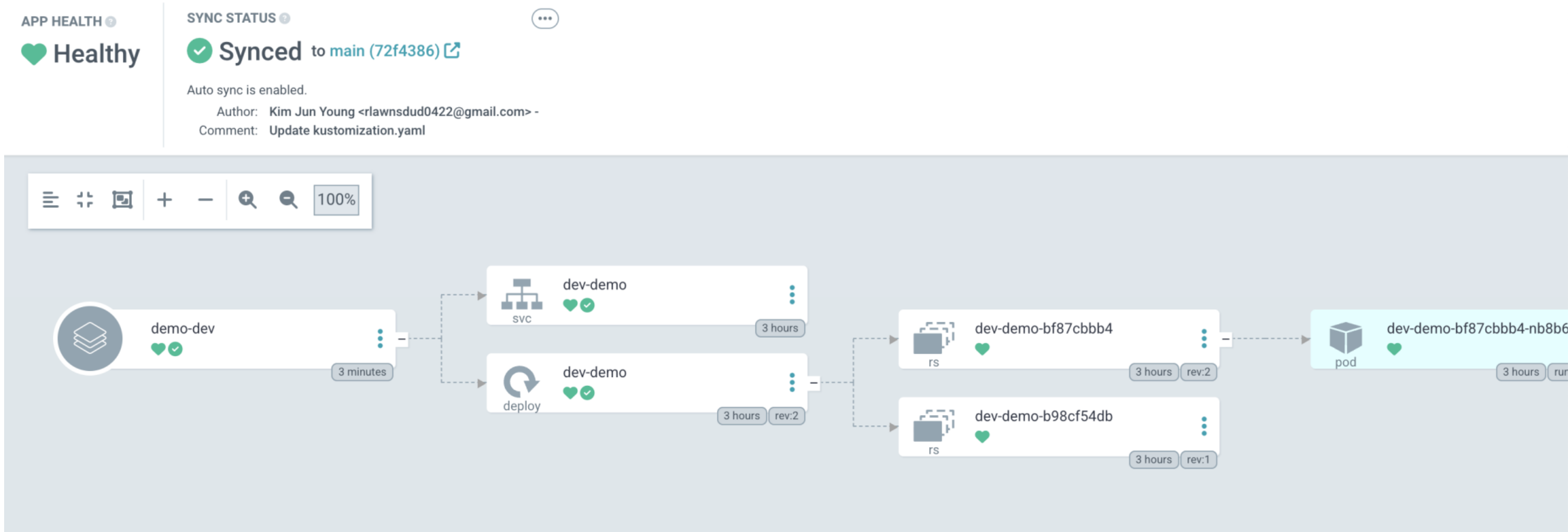
The screenshot displays the ArgoCD Dashboard with two application cards side-by-side. Each card represents an application and contains the following information:

- demo-dev** (left card):
 - Project: demo-proj
 - Labels:
 - Status: ♥ Healthy ✔ Synced
 - Repository: <https://github.com/eocndp/argocd-kustomize-demo>
 - Target Revision: main
 - Path: overlays/dev
 - Destination: in-cluster
 - Namespace: dev
 - Created At: 10/04/2025 22:20:04 (a minute ago)
- demo-prod** (right card):
 - Project: demo-proj
 - Labels:
 - Status: ♥ Healthy ✔ Synced
 - Repository: <https://github.com/eocndp/argocd-kustomize-demo>
 - Target Revision: main
 - Path: overlays/prod
 - Destination: in-cluster
 - Namespace: prod
 - Created At: 10/04/2025 22:20:06 (a minute ago)

At the bottom of each card, there are three buttons: **SYNC** (with a circular arrow icon), **REFRESH** (with a circular arrow icon), and **DELETE** (with a trash can icon).

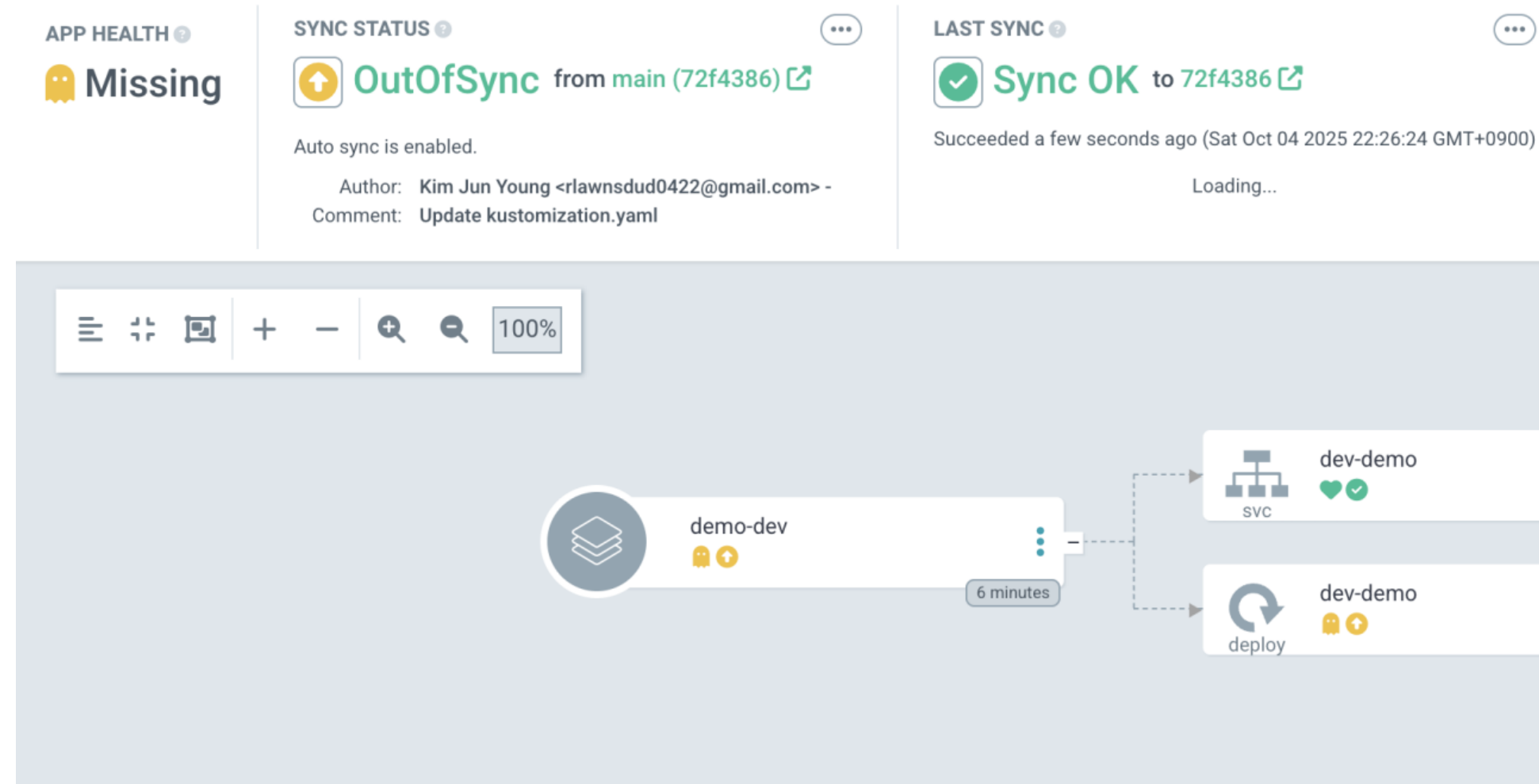
ArgoCD Dashboard에서도 확인 가능함, 특정 애플리케이션을 클릭하면
해당 리소스의 토폴로지를 확인해볼 수 있음

Session 1. ArgoCD Demo



만약 리소스를 삭제하여 Drift를 강제로 유발한다면?

Session 1. ArgoCD Demo



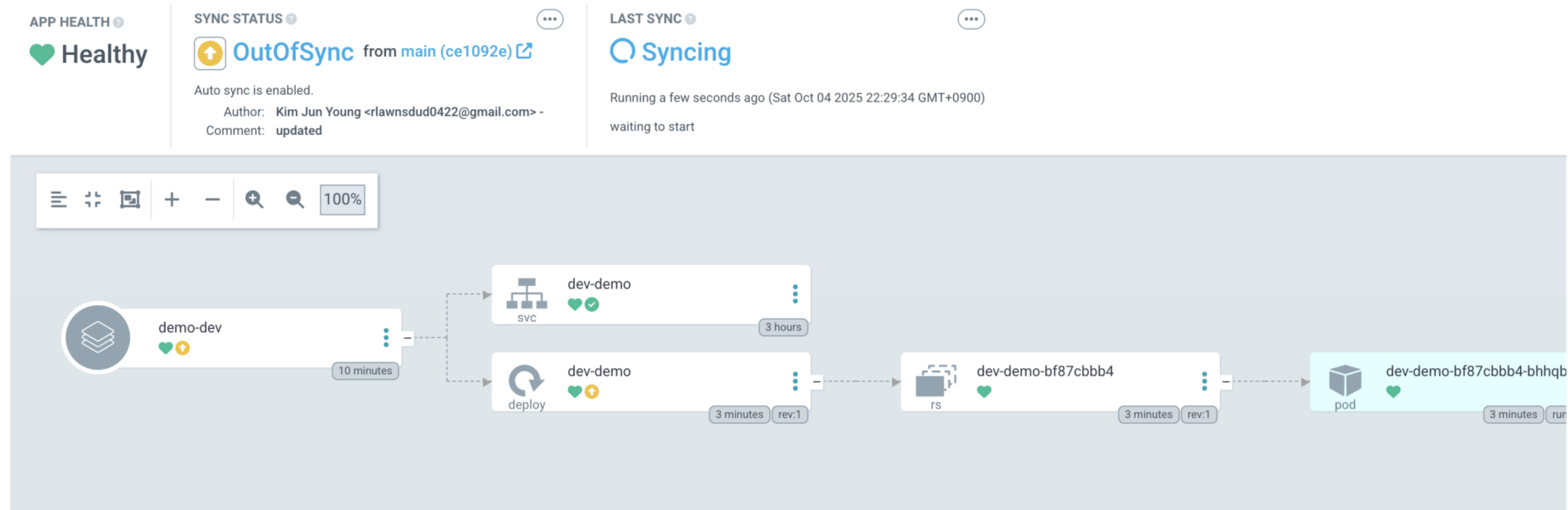
OutOfSync라는 메시지와 함께 ArgoCD가 원하는 상태에 맞게 복구되도록 조정(Reconcile)함

Session 1. ArgoCD Demo

```
diff --git a/overlays/dev/kustomization.yaml b/overlays/dev/kustomization.yaml
index 257ba01..f5ec659 100644
--- a/overlays/dev/kustomization.yaml
+++ b/overlays/dev/kustomization.yaml
@@ -12,4 +12,4 @@ patches:
     path: /spec/template/spec/containers/0/env/-
     value:
       name: APP_NAME
-     value: "Dev"
+     value: "Dev Application"
diff --git a/overlays/prod/kustomization.yaml b/overlays/prod/kustomization.yaml
index a97932b..e5274e7 100644
--- a/overlays/prod/kustomization.yaml
+++ b/overlays/prod/kustomization.yaml
@@ -12,4 +12,4 @@ patches:
     path: /spec/template/spec/containers/0/env/-
     value:
       name: APP_NAME
-     value: "Prod"
+     value: "Prod Application"
```

매니페스트 내용을 변경하여 ArgoCD에서 적용이 되는지 확인

Session 1. ArgoCD Demo



Reconcile(Pulling) 간격을 30초로 설정해뒀기 때문에
30초 간격으로 소스를 Pulling하여 상태를 확인, 상태가 변경되어 있다고 판단하여 Syncing

Session 1. ArgoCD Demo

REASON	MESSAGE	COUNT	FIRST OCCURRED	LAST OCCURRED
ResourceUpdated	Updated health status: Progressing -> Healthy	1	1m ago Today at 10:29 PM	1m ago Today at 10:29 PM
OperationCompleted	Sync operation to ce1092ec151b6166ae57b8e65a63f312a00db0ea succeeded	1	1m ago Today at 10:29 PM	1m ago Today at 10:29 PM
ResourceUpdated	Updated sync status: OutOfSync -> Synced	1	1m ago Today at 10:29 PM	1m ago Today at 10:29 PM
ResourceUpdated	Updated health status: Healthy -> Progressing	1	1m ago	1m ago

/Users/workspace5/blog-example-demo/k8s-argocd-example [git::main] [ky0422@MacBookPro] [8:58]

•> kubectl -n dev exec dev-demo-55f8477474-5rvz2 -- curl -s dev-demo; echo

Hello from Dev Application

/Users/workspace5/blog-example-demo/k8s-argocd-example [git::main] [ky0422@MacBookPro] [8:58]

•> kubectl -n prod exec prod-demo-fb8c54c4-hgkn2 -- curl -s prod-demo; echo

Hello from Prod Application

ResourceUpdated	Updated health status: Progressing -> Healthy	1	5m ago Today at 10:26 PM	5m ago Today at 10:26 PM
-----------------	---	---	-----------------------------	-----------------------------

그런데 ArgoCD는 CI/CD 중 CD(지속적 배포)에서 해당된다..

그럼 쿠버네티스 ArgoCD에 대한 **CI**(Continuous Integration, 지속적 통합: 빌드, 테스트, 린팅 등등)는?

(아키텍처 한계로 ArgoCD CI/CD 파이프라인을 하나의 레포지토리에서 진행하는건 어려움이 있음)

(CI 레포지토리에서 CI가 완료되면 CD 레포지토리로 PR을 보내는 등으로 진행하면 될 듯)

Session 02

ArgoCD가 CD(Continuous Deployment)라면,



kubernetes 에서 CI(Continuous Integration)는 어떻게 할까?

#CI #Github_Actions #Helm #KinD

CI(지속적 통합) 파이프라인 없이 수동으로 테스트하는 등.. 괜찮을까?

(ArgoCD 배포 실패) ... 아까 스테이징 환경에 배포해보니깐 됐었던거 같았는데...

(기억 안남)



휴먼 에러, 코드 충돌, 정기적인 테스트가 없어 버그 및 오류 발견의 지연, 품질/피드백 저하 등등...

특히 ArgoCD는 배포(CD)에 문제가 있을 시 직접 ArgoCD를 확인해야 하기 때문에 CI가 있어야 함

- A)** CI 파이프라인에서 임시적인 쿠버네티스 클러스터를 구축하여 배포 성공/Health 체크
- B)** 별도의 스테이징/개발 쿠버네티스 클러스터에 직접 배포하여 상태 확인

B의 경우 CI/CD 파이프라인 외부의 실제 스테이징/개발 클러스터에 접근해야 하지만
파이프라인 전용 **임시적인(Ephemeral)** 환경을 만들어 CI를 구축하는것이 정석임

환경 충돌(오염), 재현성 문제, 격리 불가, 병렬 처리의 어려움 등등 ...



???: 근데 쿠버네티스 클러스터는 무겁잖아요..? 그걸 파이프라인 내에서??

실제로 EKS 클러스터 만드는데 10~15분 정도 소요됨 + 쿠버네티스 자체가 먹는 리소스양이 너무 많음

그런 무거운 실제 쿠버네티스 클러스터를 띄우는건 파이프라인 내에선 무리가 많음

Session 2. What is KinD?



???: 그럼 MiniKube 같은 가벼운거 쓰면 되지 않나요?

MiniKube는 단일 노드(VM 등)을 중심으로 운영/실습 할 수 있도록 하는 구현체라 CI/CD 파이프라인에서 운영하기 어려움

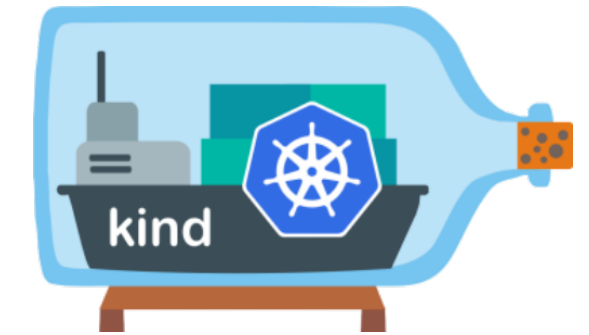


그럼 쿠버네티스를 도커에서 돌릴 수 있을까? (쿠버네티스 컨테이너화)

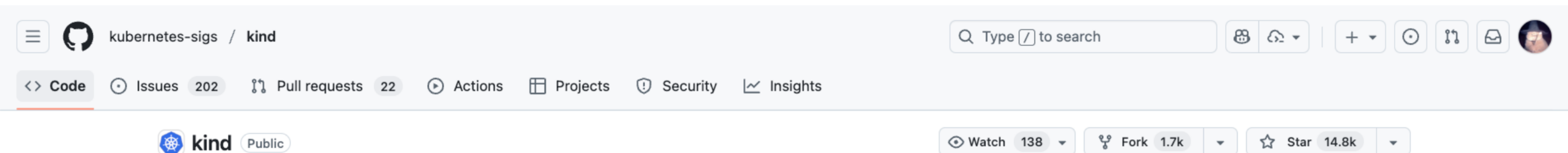
Session 2. What is KinD?

KinD(Kubernetes in Docker) = 쿠버네티스 클러스터를 도커 컨테이너 위에서 가볍게 돌릴 수 있는 도구

(단, 도커 런타임 위에서 동작하기 때문에 네트워킹, Ingress/Gateway(LB)나 성능 등의 한계가 있음)

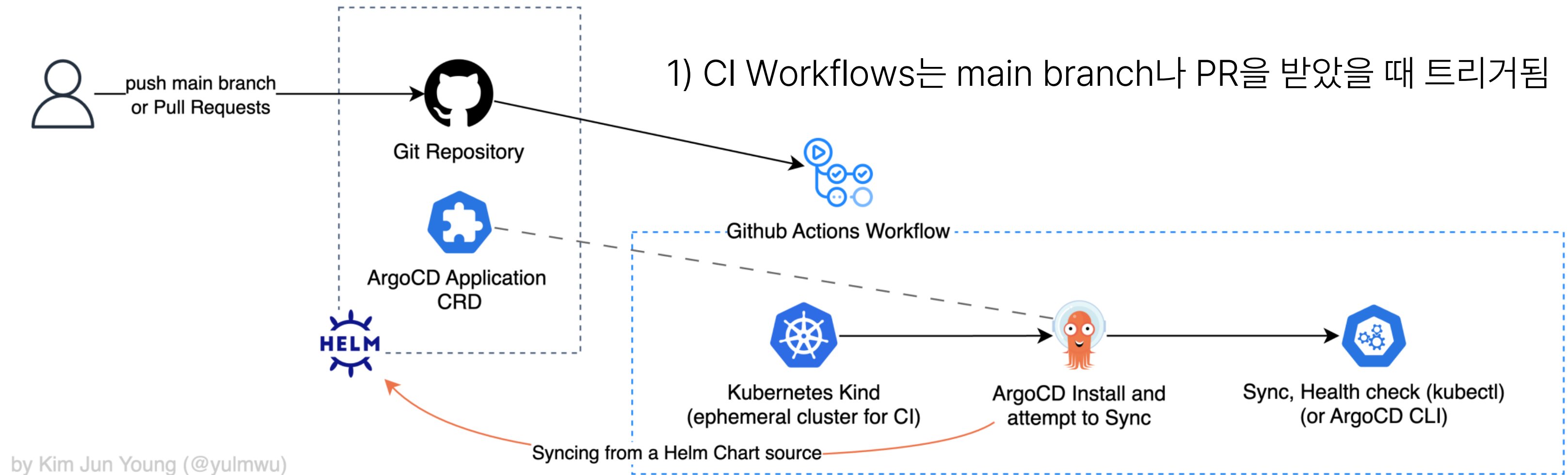


도커 컨테이너로도 동작할 수 있기 때문에 CI/CD 파이프라인 환경에서 돌리기 매우 적합함



Session 2. ArgoCD CI with Github Actions Demo

이번엔 Helm 차트를 통해 애플리케이션을 패키징하고(소스) Kind + ArgoCD CI를 구축해보겠습니다



2) 워크플로우의 VM에 KinD를 설치하여 쿠버네티스 클러스터를 구성함

3) ArgoCD 설치 및 클러스터 상태 동기화

4) Sync, Healthy 체크 (ArgoCD CLI or kubectl)

Session 2. ArgoCD CI with Github Actions Demo (Helm Chart)

```
.
├── .github
│   └── workflows
│       └── ci-argocd-kind.yaml
├── charts
│   └── demo-app
│       ├── Chart.yaml
│       ├── values.yaml
│       └── templates
│           ├── _helpers.yaml
│           ├── deployment.yaml
│           └── service.yaml
└── argocd
    └── application.yaml
```

Github Actions Workflows, Helm 차트 디렉토리 구조

Helm 차트는 Chart.yaml (차트 메타데이터), values.yaml (템플릿 기본 값)

templates 디렉토리(실제 템플릿이 적용된 매니페스트)로 구성되어 있음

(_helpers.yaml : 재사용이 가능한 함수나 변수를 선언하는 유틸리티 파일)

Chart.yaml

```
1  apiVersion: v2
2  name: demo-app
3  description: A demo application
4  type: application
5  version: 0.1.0
6  appVersion: "1.0.0"
```

templates/_helpers.yaml

```
1  {{- define "demo-app.name" -}}
2    {{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
3  {{- end }}
4
5  {{- define "demo-app.fullname" -}}
6    {{- if .Values.fullnameOverride }}
7      {{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" -}}
8    {{- else }}
9      {{- $name := include "demo-app.name" . -}}
10     {{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
11   {{- end }}
12 {{- end }}
```

Session 2. ArgoCD CI with Github Actions Demo (Helm Chart)

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: {{ include "demo-app.fullname" . }}
5    labels:
6      app.kubernetes.io/name: {{ include "demo-app.name" . }}
7      app.kubernetes.io/instance: {{ .Release.Name }}
8  spec:
9    replicas: {{ .Values.replicaCount }}
10   selector:
11     matchLabels:
12       app.kubernetes.io/name: {{ include "demo-app.name" . }}
13       app.kubernetes.io/instance: {{ .Release.Name }}
14   template:
15     metadata:
16       labels:
17         app.kubernetes.io/name: {{ include "demo-app.name" . }}
18         app.kubernetes.io/instance: {{ .Release.Name }}
19     annotations:
20       {{- with .Values.podAnnotations }}
21       {{- toYaml . | nindent 8 }}
22       {{- end }}
23   spec:
24     containers:
25       - name: {{ include "demo-app.name" . }}
26         image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
27         imagePullPolicy: {{ .Values.image.pullPolicy }}
28         ports:
29           - containerPort: 80
30             name: http
31     resources:
32       {{- toYaml .Values.resources | nindent 12 }}
```

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: demo-app
5  spec:
6    selector:
7      app: demo-app
8    ports:
9      - name: http
10        port: {{ .Values.service.port }}
11        targetPort: 80
12    type: {{ .Values.service.type }}
```

templates/deployment.yaml

templates/service.yaml

Deployment, Service 템플릿화 (Go Template 기반)

Session 2. ArgoCD CI with Github Actions Demo (Github Actions Workflows)

```
1  name: ArgoCD CI with Kind
2
3  on:
4    push:
5      branches: [ main ]
6    pull_request:
7      branches: [ main ]
```

Workflows 트리거 조건: main Branch Push 또는 PR 요청 시

```
9  jobs:
10    argocd-kind-test:
11      runs-on: ubuntu-latest
12
13      steps:
14        - name: Checkout
15          uses: actions/checkout@v4
16
17        - name: Set up kubectl
18          uses: azure/setup-kubectl@v4
19          with:
20            version: 'latest'
21
22        - name: Set up kind
23          uses: helm/kind-action@v1
24          with:
25            cluster_name: argocd-ci-cluster
26            kubectl_version: 'v1.33.0'
```

kubectl, KinD Actions 구성 (쿠버네티스 1.33 사용)

Session 2. ArgoCD CI with Github Actions Demo (Github Actions Workflows)

```
28     - name: Install ArgoCD
29       run: |
30         kubectl create namespace argocd
31         kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
32
33     - name: Wait for ArgoCD components to be ready
34       run: |
35         kubectl wait --namespace argocd \
36           --for=condition=Available deployment/argocd-redis \
37           --timeout=120s
38         kubectl wait --namespace argocd \
39           --for=condition=Available deployment/argocd-server \
40           --timeout=120s
41         kubectl wait --namespace argocd \
42           --for=condition=Available deployment/argocd-repo-server \
43           --timeout=120s
44         kubectl wait --namespace argocd \
45           --for=condition=Available deployment/argocd-dex-server \
46           --timeout=120s
47         kubectl wait --namespace argocd \
48           --for=condition=Available deployment/argocd-applicationset-controller \
49           --timeout=120s
50         kubectl wait --namespace argocd \
51           --for=condition=Available deployment/argocd-notifications-controller \
52           --timeout=120s
```

ArgoCD 설치 및 ArgoCD 구성까지 대기 (kubectl)

Session 2. ArgoCD CI with Github Actions Demo (Github Actions Workflows)

```
54         - name: Apply ArgoCD Application
55           run: |
56             kubectl apply -f k8s-argocd-ci-example/argocd/application.yaml
```

ArgoCD Application CRD 적용 (매니페스트는 레포지토리 내에서 그대로 사용)

(kubectl Context는 Kind 설치 시 자동으로 반영됨)

Session 2. ArgoCD CI with Github Actions Demo (Github Actions Workflows)

```
58     - name: Wait for ArgoCD Application Sync & Healthy
59       run: |
60         APP_NAME="demo-app"
61         NAMESPACE="argocd"
62
63         echo "Waiting for ArgoCD Application/${APP_NAME} to be Synced and Healthy..."
64
65         for i in {1..10}; do
66           SYNC_STATUS=$(kubectl get application ${APP_NAME} -n ${NAMESPACE} -o jsonpath='{.status.sync.status}' || echo "Unknown")
67           HEALTH_STATUS=$(kubectl get application ${APP_NAME} -n ${NAMESPACE} -o jsonpath='{.status.health.status}' || echo "Unknown")
68
69           echo "Try ${i}: sync=${SYNC_STATUS}, health=${HEALTH_STATUS}"
70
71           if [ "$SYNC_STATUS" = "Synced" ] && [ "$HEALTH_STATUS" = "Healthy" ]; then
72             echo "Application is Synced and Healthy"
73             exit 0
74           fi
75
76           sleep 5
77         done
78
79         echo "Application did not become Synced/Healthy in time"
80         kubectl get application ${APP_NAME} -n ${NAMESPACE} -o yaml || true
81         kubectl get pods -A || true
82
83         exit 1
```

5초 간격으로 10초 동안 kubectl로 상태(Synced, Healthy) 확인, 실패 시 exit 1로 파이프라인 Fail

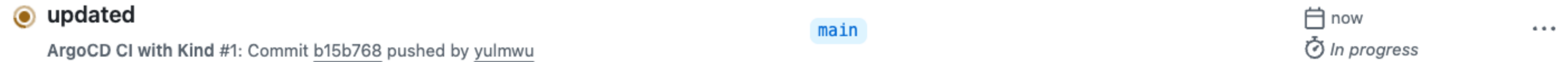
Session 2. ArgoCD CI with Github Actions Demo (Github Actions Workflows)

```
- name: Clean up
  if: always()
  run: |
    kind delete cluster --name argocd-ci-cluster
```

마지막으로 KinD 클러스터 정리

(if: always() 옵션을 달아주면 전 워크플로우가 Fail이 나던 말던 상관 없이 무조건 실행함)

Session 2. ArgoCD CI with Github Actions Demo (Testing)



(Github Actions 장점으로 깃허브 레포지토리와 별도의 연동 없이 바로 Workflows를 구성할 수 있음)

커밋 후 깃 레포지토리에 main Branch Push 시 Workflows가 바로 트리거됨

Session 2. ArgoCD CI with Github Actions Demo (Testing)

← ArgoCD CI with Kind

✔ updated #5

Re-run all jobs

...

Summary

Jobs

argocd-kind-test

Run details

Usage

Workflow file

argocd-kind-test

succeeded 1 minute ago in 1m 39s

Search logs

Refresh

Settings

> ✔ Set up job

2s

> ✔ Checkout

0s

> ✔ Set up kubectl

1s

> ✔ Set up kind

47s

> ✔ Install ArgoCD

3s

> ✔ Wait for ArgoCD components to be ready

30s

> ✔ Apply ArgoCD Application

0s

> ✔ Wait for ArgoCD Application Sync & Healthy

11s

> ✔ Clean up

2s

> ✔ Post Set up kind

0s

> ✔ Post Checkout

0s

> ✔ Complete job

0s

ArgoCD (+ Kind 클러스터) CI 파이프라인 구축 끝.

Session 2. ArgoCD CI with Github Actions Demo (Testing)

✓ Wait for ArgoCD Application Sync & Healthy

11s

```
1  ▶ Run APP_NAME="demo-app"
27  Waiting for ArgoCD Application/demo-app to be Synced and Healthy...
28  Try 1: sync=, health=
29  Try 2: sync=Synced, health=Progressing
30  Try 3: sync=Synced, health=Healthy
31  Application is Synced and Healthy
```

간단한 Deployment + Service 오브젝트 몇 개만 적용했을 때

위와 같이 4번째 시도(Pull/Reconcile 간격이 30초로 설정되어 있으니 3번이 최소 간격), 총합 1분 40초 정도 소요됨

물론 실제 애플리케이션에선 10분 이상 걸릴 수 있음

(필자 팀에서 테스트 해봤을 때 이미지 Pulling(300MB 정도 되는 이미지 2개) 시간 때문에 7분 정도 걸렸었음)

There's no compression algorithm for experience — AWS

THANKS FOR YOUR ATTENTION

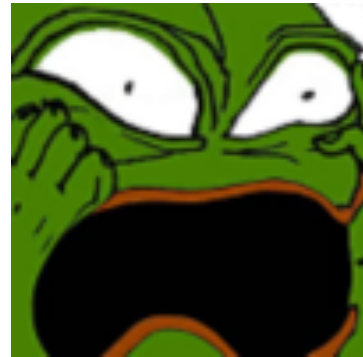
Speaker / Presenter by Kim Jun Young

Contact 010-2980-1336

(주제 하나 더 있음)



분량 조절 실패로 인한 10분만에 끝날 시



분량 계산 실패했다..

다른 주제로 짧게 5분만 더 해보자

이미지 리사이징을 통한 LCP 최적화

근데 이제  CloudFront + Lambda@Edge를 곁들인

Feat. 누가 LCP(Largest Contentful Paint)를 화나게 했을까?

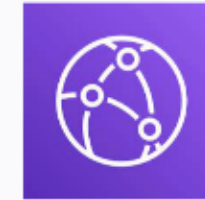


Image Resizing using AWS S3 + CloudFront + Lambda@Edge

[AWS Misc] Image Resizing with AWS CloudFront + Lambda@Edge

Web Optimization with Image Resizing using AWS S3 + CloudFront + Lambda@Edge

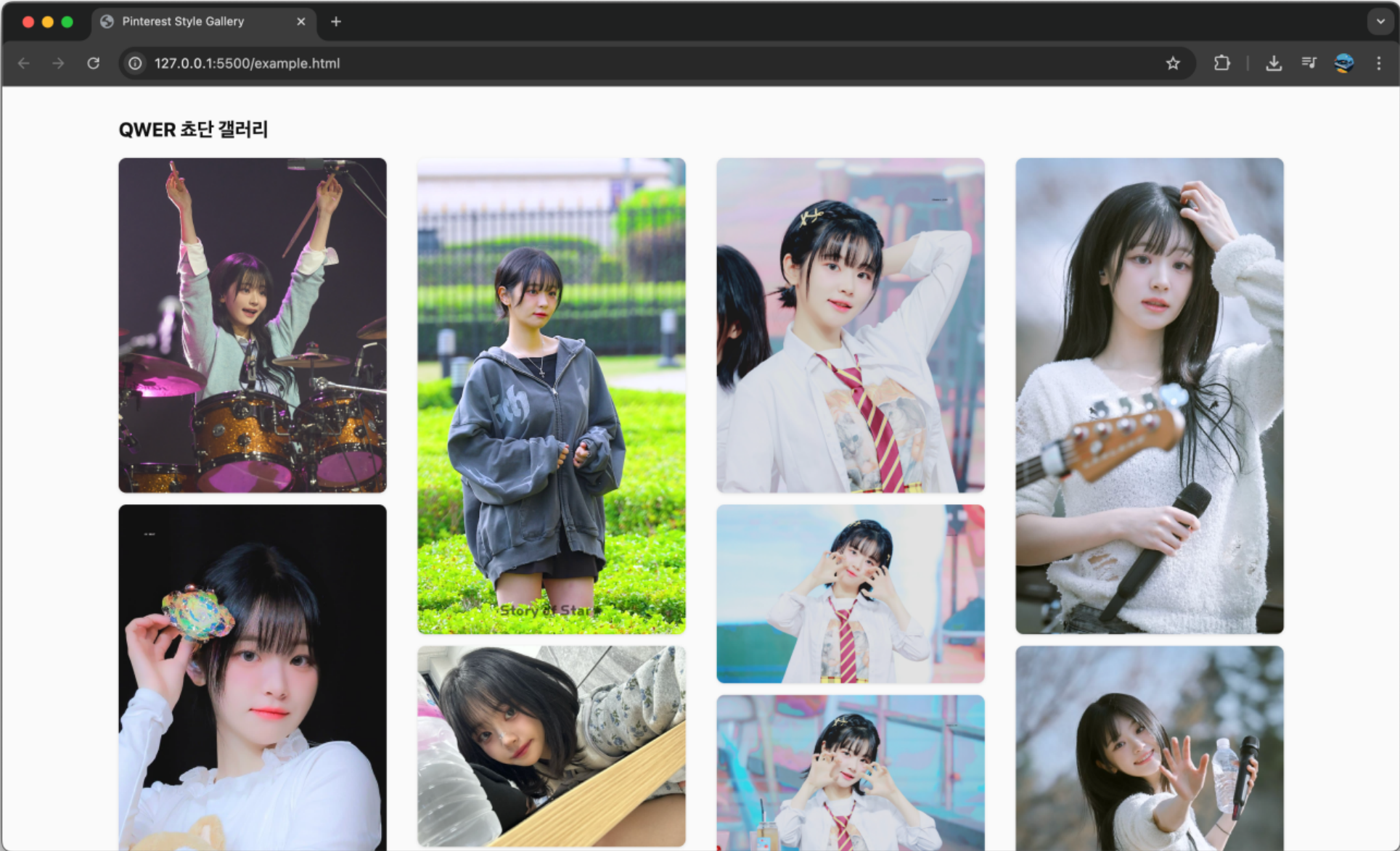
Misc

aws

2025년 8월 22일 · 0개의 댓글 · ♥ 0

<https://velog.io/@yulmwu/aws-cloudfront-lambda-image-resizing>

(목차 생략)



QWER 초단 갤러리(디시아님) 제작
(의뢰받음)

QWER 초단 갤러리



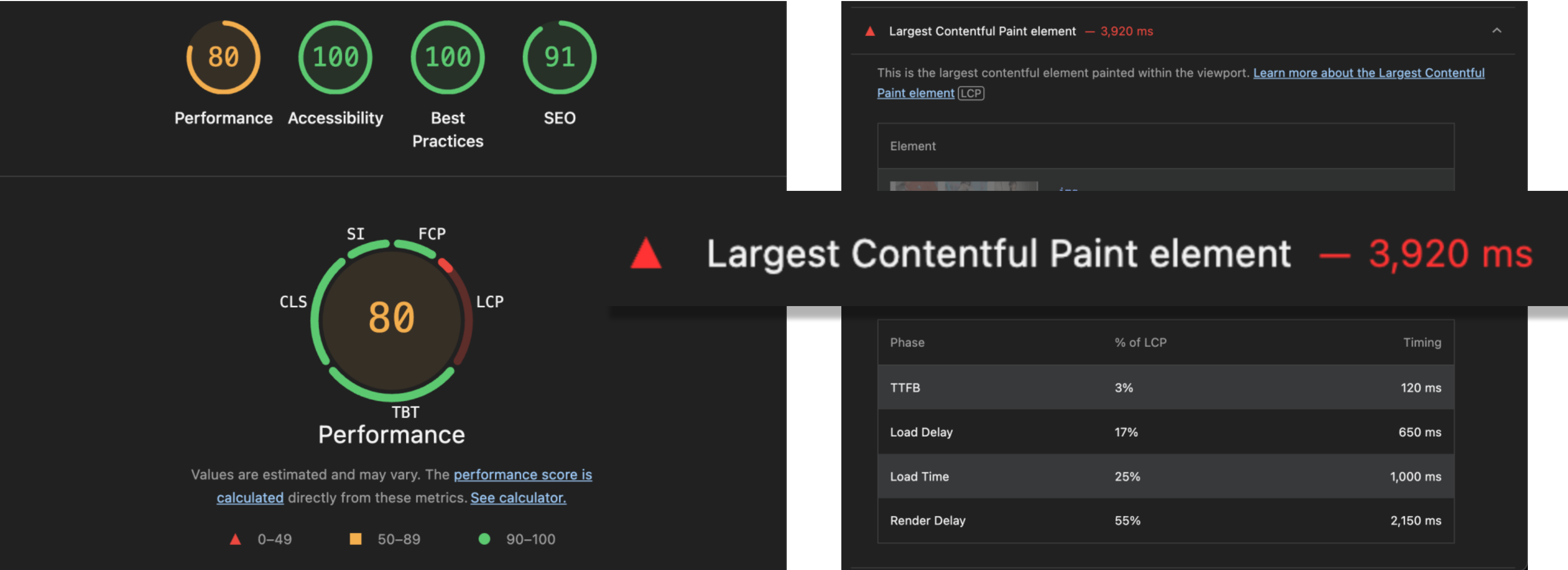
Network

19 requests | 4.5 MB transferred | 4.5 MB resources | Finish: 4.79 s

Name	Status	Type	Size	Time	Waterfall
ws	101	websocket	0.0 kB	Pending	
example.html	200	document	4.9 kB	177 ms	
1.jpg	200	jpeg	330 kB	3.43 s	
2.jpg	200	jpeg	70.6 kB	1.37 s	
3.jpg	200	jpeg	135 kB	2.11 s	
4.jpg	200	jpeg	106 kB	1.81 s	
5.jpg	200	jpeg	325 kB	3.41 s	
6.jpg	200	jpeg	99.5 kB	1.75 s	
7.jpg	200	jpeg	69.2 kB	1.35 s	
8.jpg	200	jpeg	190 kB	2.55 s	
9.jpg	200	jpeg	401 kB	3.71 s	
10.jpg	200	jpeg	400 kB	3.71 s	
11.jpg	200	jpeg	489 kB	3.89 s	
12.jpg	200	jpeg	82.8 kB	1.55 s	
13.jpg	200	jpeg	79.9 kB	1.51 s	
14.jpg	200	jpeg	137 kB	2.13 s	
15.jpg	200	jpeg	90.1 kB	1.66 s	
png1.png	200	png	1,200 kB	4.59 s	
png2.png	200	png	250 kB	0.00 s	

(4G Throttling) 4.59 Seconds??

4.5MB??



LightHouse = 구글에서 만든 성능 측정 도구(DevTools에 있음)

" LCP가 뭐길래 호들갑임?? "

LCP(Largest Contentful Paint)

크기가 가장 큰 데이터를 가져오고 랜더링하는 시간
일반적으로 2.5초 이하여야 최적화된 사이트라 칭함

⇒ 최대한 짧아야 사용자 입장에선 좋음

Phase	% of LCP	Timing
TTFB	3%	120 ms
Load Delay	17%	650 ms
Load Time	25%	1,000 ms
Render Delay	55%	2,150 ms

LCP 4초가 찍히는 초단 갤러리



(정상화 필요)

LCP(Largest Contentful Paint)

어떻게 줄이지?



512×512



256×256

사이즈 조정 ⇒ 이미지 GET 시간 단축
랜더링 시간 단축(LCP에 포함)



리사이징된 이미지의 크기는 일반적으로
표시 크기(CSS width/height) * DPR(Display Pixel Ratio)로 계산

Image Resizing

어떻게 줄여서 클라이언트로 보내지?

- 이미지 업로드 후 서버에서 리사이징 후 여러개의 이미지 저장
- 이미지 요청 시 즉석에서 동적으로 리사이징 후 응답(On the Fly)

캐싱(CDN)이 될 경우 유리해짐

다만 첫 요청은 Cache Miss라 리사이징하는데
시간 소요 (한번만)

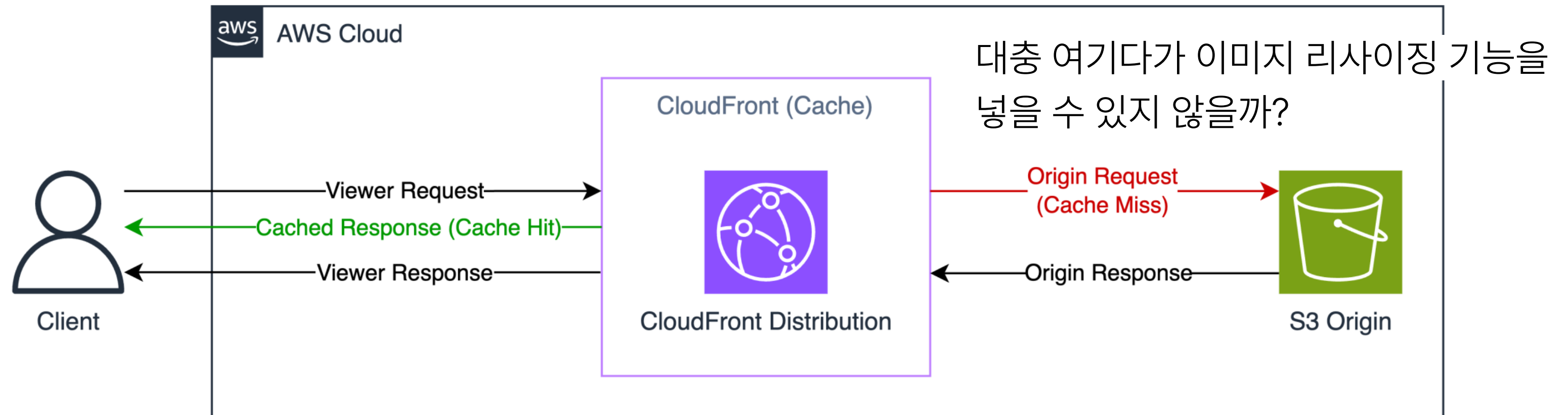


CDN을 어떻게 구현하지?



AWS Architecture

CloudFront 동작 과정

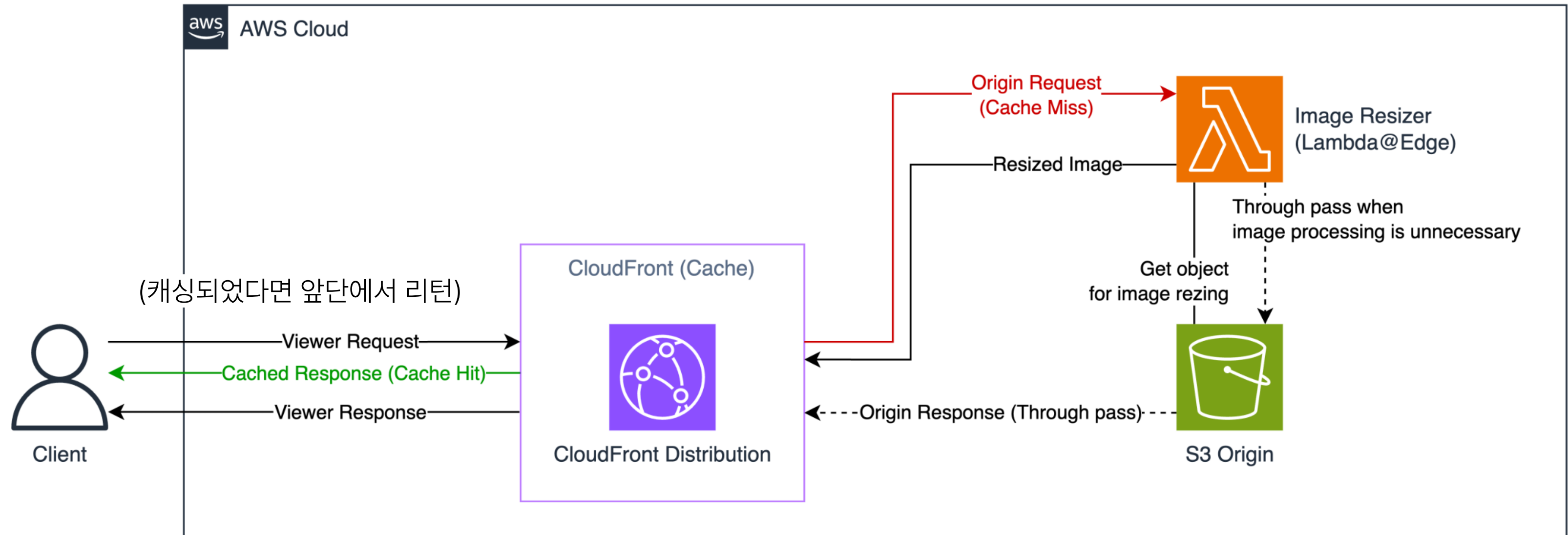


by Kim Jun Young (@yulmwu)

" Lambda@Edge "

전세계 엣지 로케이션에 람다를 두는 것

AWS Architecture



Let's write the Code

코드 전체 설명하는건 의미없음..

```
"devDependencies": {
  "@types/aws-lambda": "^8.10.152",
  "@types/node": "^24.3.0",
  "@smithy/types": "^4.3.2",
  "esbuild": "^0.20.0",
  "aws-lambda": "^1.0.7",
  "typescript": "^5.4.0"
},
"dependencies": {
  "@aws-sdk/client-s3": "^3.864.0",
  "sharp": "^0.34.3"
}
```

NodeJS Sharp 라이브러리 사용
(이미지 프로세싱 라이브러리)

Let's write the Code

```
switch (p.extension) {  
  case 'jpg':  
  case 'jpeg':  
    stream = p.quality ? stream.jpeg({ quality: p.quality }) : stream.jpeg()  
    break  
  case 'png': {  
    stream = p.quality  
      ? stream.png({ compressionLevel: this.pngCompressionLevel(p.quality) })  
      : stream.png()  
    break  
  }  
  case 'webp':  
    stream = p.quality ? stream.webp({ quality: p.quality }) : stream.webp()  
    break  
  case 'gif':  
    stream = stream.gif()  
    break  
}
```

PNG는 컴프레싱하기엔 애매한 포맷임
(가능하긴 한데 람다 CPU 사용률이 어마어마)

Let's write the Code

```
return {  
  status: '200',  
  statusDescription: 'OK',  
  bodyEncoding: 'base64',  
  body: body.toString('base64'),  
  headers: this.headers(contentType),  
}
```

이미지 바이너리를 Base64로 인코딩 후 반환

다만 Base64 인코딩의 경우 약 1.3배 증가, + 응답 바디 크기가 1.3MB로 제한

(더 큰 파일은 S3에 저장해서 리다이렉션 방식으로 추천)

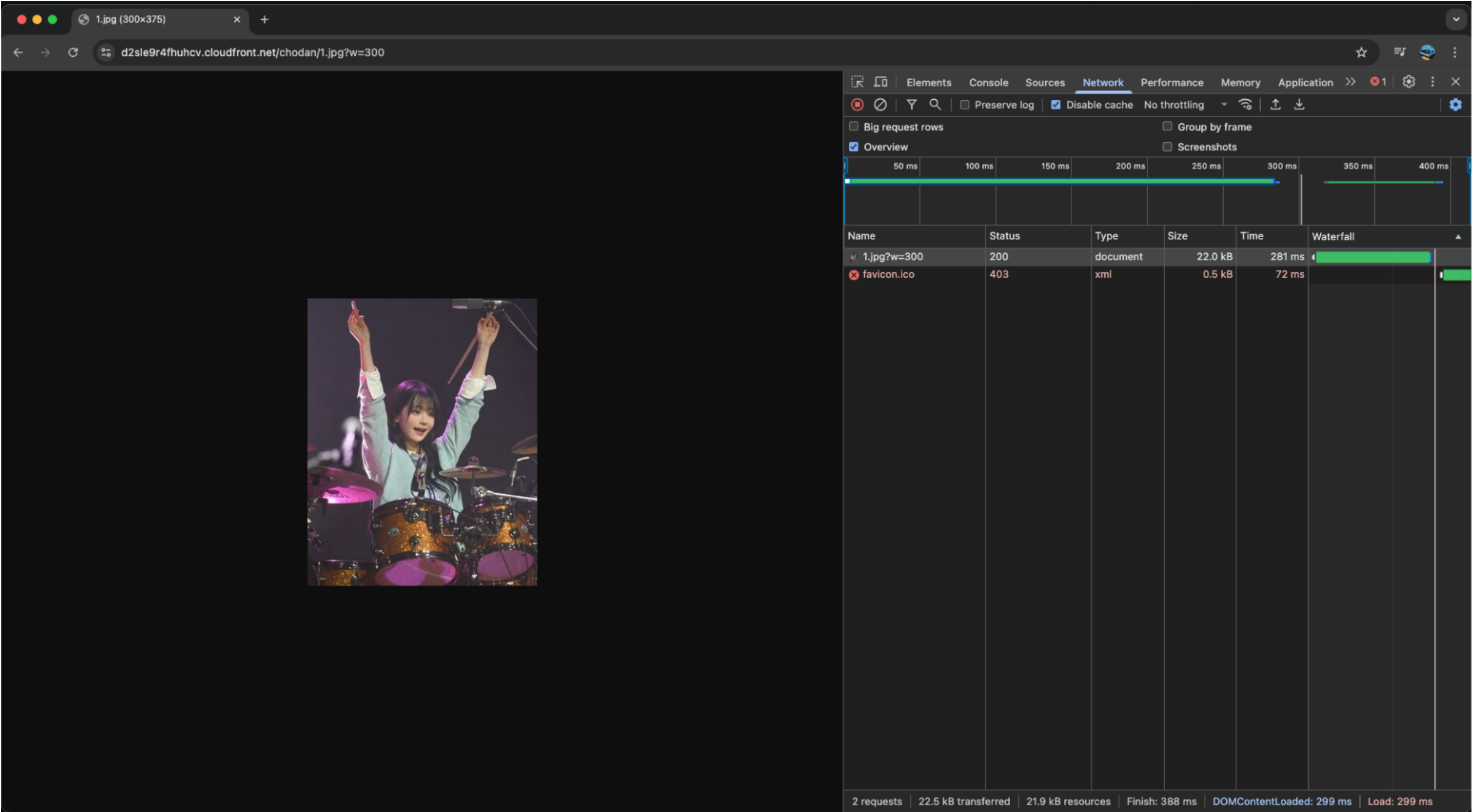
Let's write the Code



나머지 코드는 깃허브 레포지토리 확인 바람

분량으로 욕 먹을 것을 예상, 시간 관계 상 AWS 인프라 만드는건 생략합니다.

CloudFront URL 뒤에 ?w=300 등의 쿼리를 붙여 잘 동작하는지 확인



Name	Status	Type	Size	Time	Waterfall	
 1.jpg?w=300	200	document	22.0 kB	281 ms		
 favicon.ico	403	xml	0.5 kB	72 ms		

첫 로딩에 대해선 이미지 리사이징 프로세스 때문에 레이턴시가 있긴 한데,
새로고침해서 캐싱된 이미지를 가져와보면 매우 짧아진 모습을 볼 수 있음

Name	Status	Type	Size	Time	Waterfall	
 1.jpg?w=300 	200	document	22.0 kB	20 ms		
 favicon.ico	403	xml	0.5 kB	153 ms		

Pinterest Style Gallery

localhost:3000/example_w600

QWER 초단 갤러리



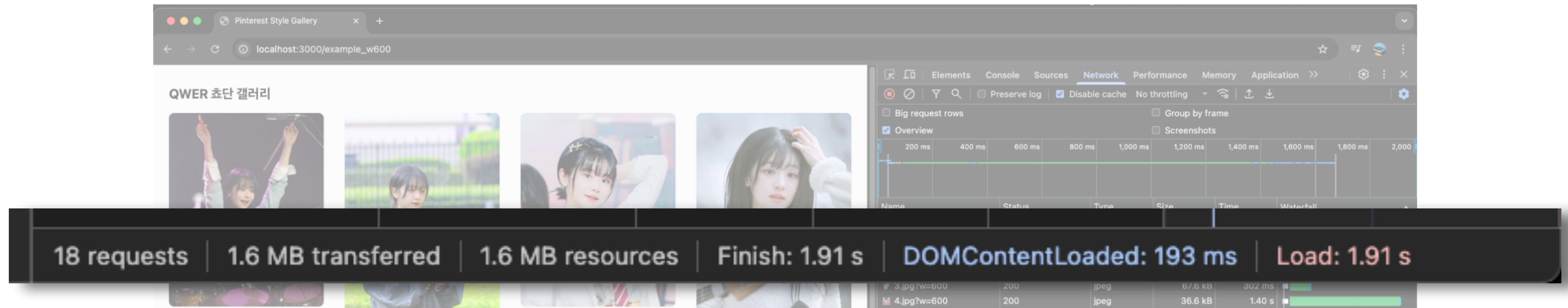
Elements Console Sources Network Performance Memory Application

Network

Big request rows Overview

Name	Status	Type	Size	Time	Waterfall
example_w600.html	301	document / ...	0.2 kB	7 ms	
example_w600	200	document	1.2 kB	17 ms	
1.jpg?w=600	200	jpeg	67.3 kB	1.41 s	
2.jpg?w=600	200	jpeg	55.6 kB	1.33 s	
3.jpg?w=600	200	jpeg	67.6 kB	302 ms	
4.jpg?w=600	200	jpeg	36.6 kB	1.40 s	
5.jpg?w=600	200	jpeg	113 kB	1.51 s	
6.jpg?w=600	200	jpeg	53.1 kB	1.50 s	
7.jpg?w=600	200	jpeg	50.8 kB	1.37 s	
8.jpg?w=600	200	jpeg	89.2 kB	1.47 s	
9.jpg?w=600	200	jpeg	48.9 kB	1.40 s	
10.jpg?w=600	200	jpeg	22.7 kB	1.55 s	
11.jpg?w=600	200	jpeg	28.6 kB	1.54 s	
12.jpg?w=600	200	jpeg	32.9 kB	1.47 s	
13.jpg?w=600	200	jpeg	65.2 kB	1.53 s	
14.jpg?w=600	200	jpeg	99.3 kB	1.52 s	
15.jpg?w=600	200	jpeg	61.6 kB	1.54 s	
png1.png?w=600	200	png	365 kB	1.62 s	
png2.png?w=600	200	png	302 kB	1.46 s	

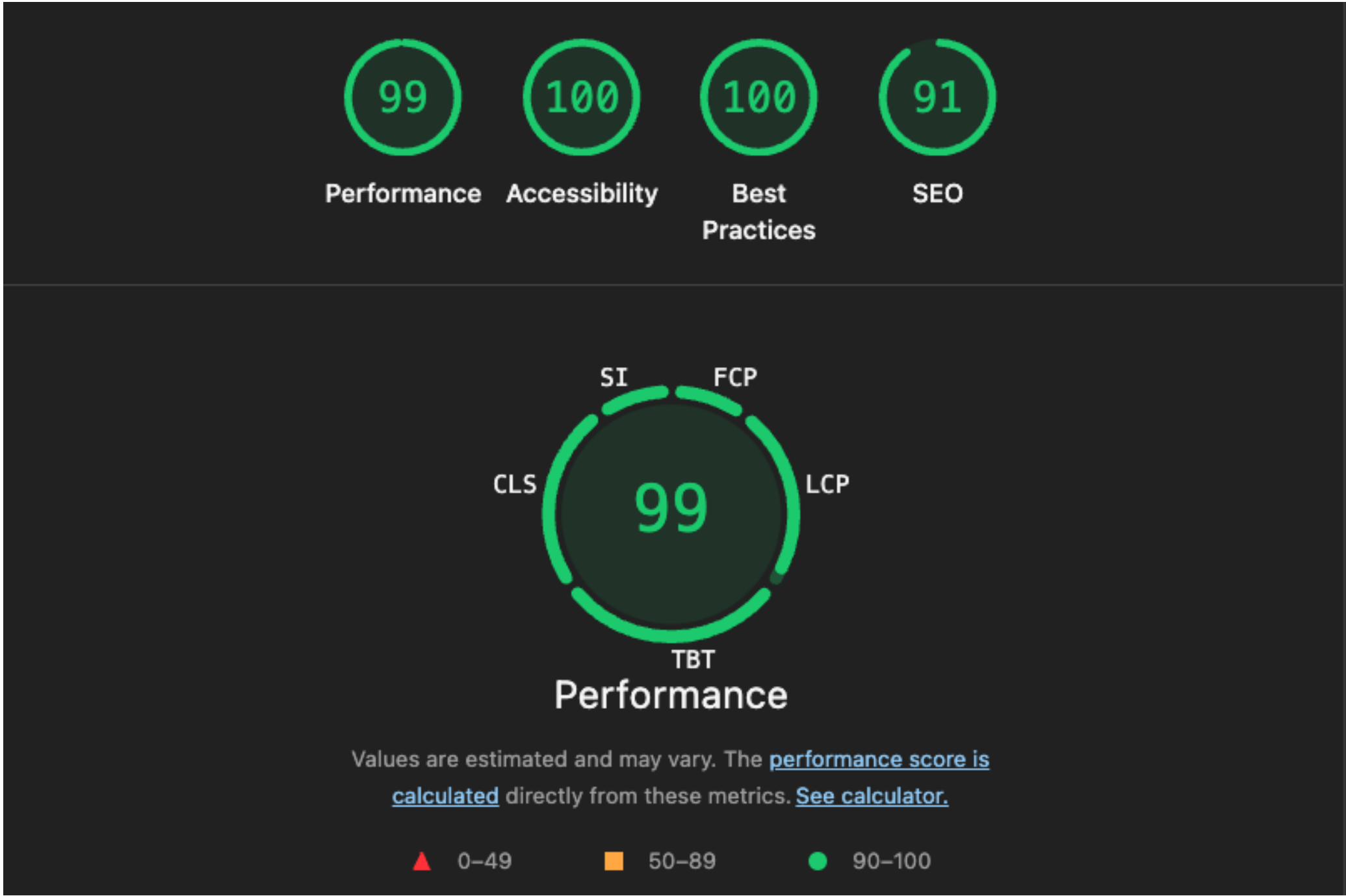
19 requests | 1.6 MB transferred | 1.6 MB resources | Finish: 1.66 s | DOMContentLoaded: 28 ms | Load: 1.66 s



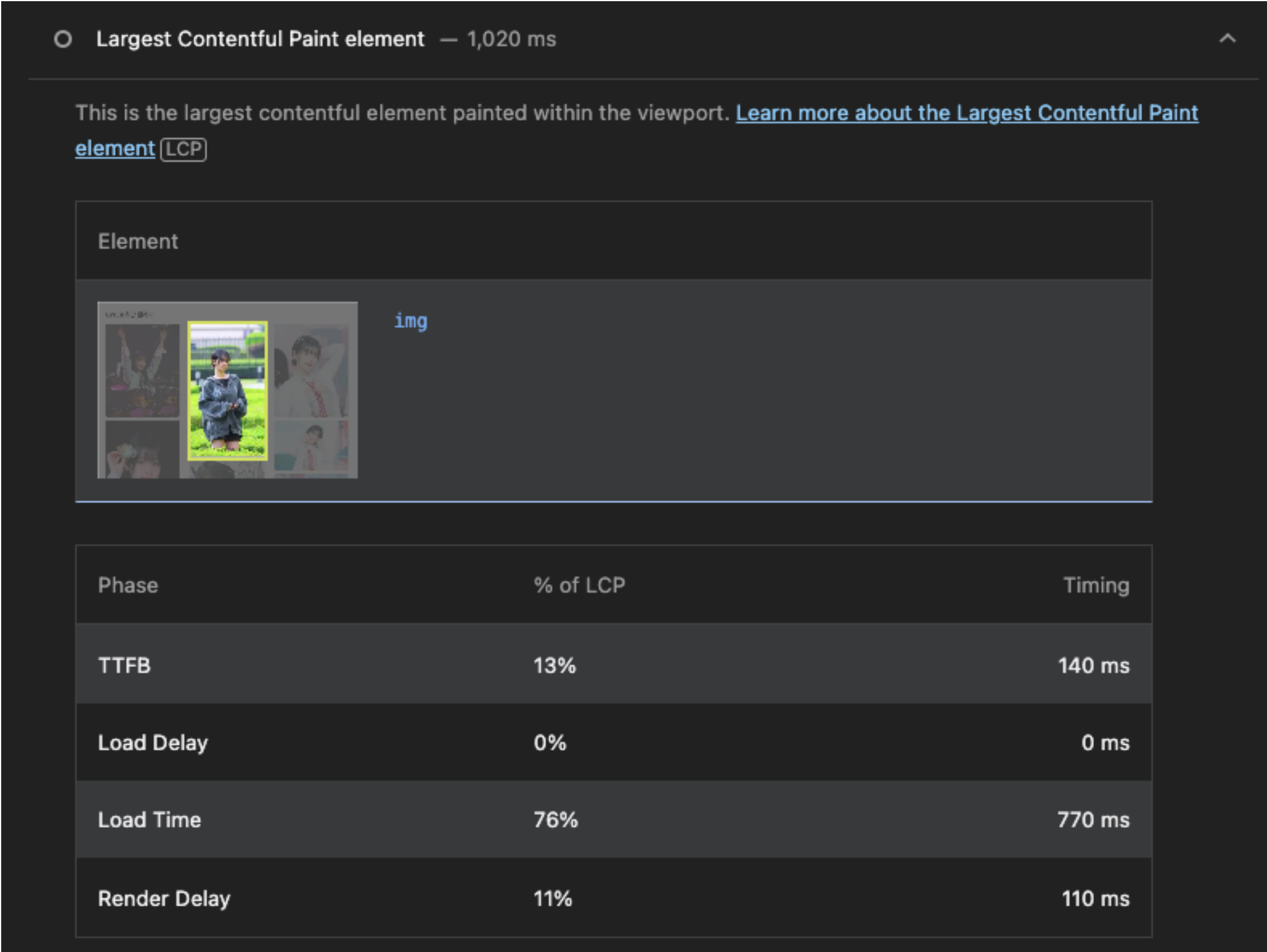
리사이징 적용 전엔 4.8초 만에 리소스를 불러왔으나, 적용 후 1.9초 정도로 2배 이상 줄어든 것을 볼 수 있음.

또한 불러온 데이터의 크기 또한 4.5MB에서 1.6MB로 2~3배 가량 줄어든 것을 볼 수 있음.

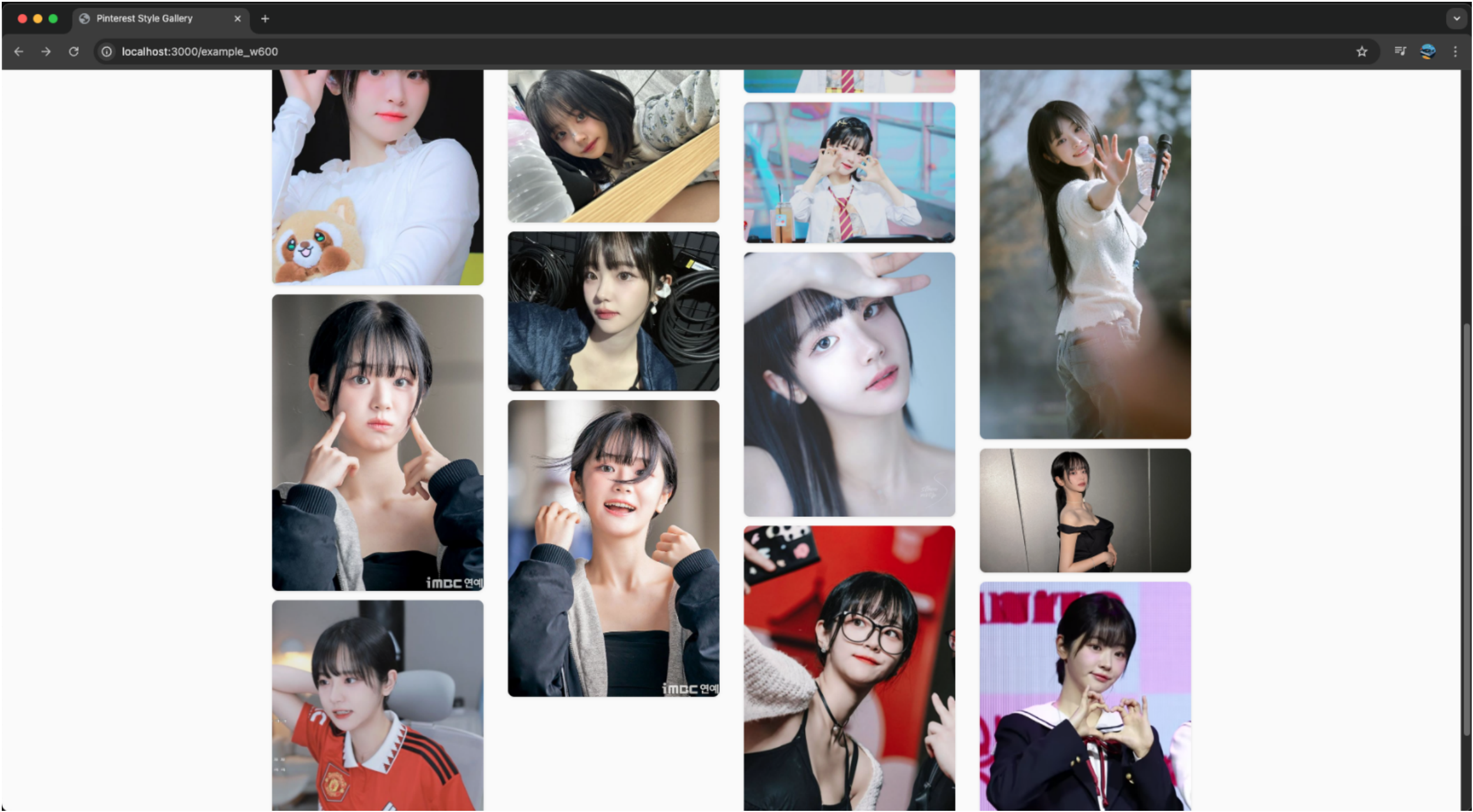




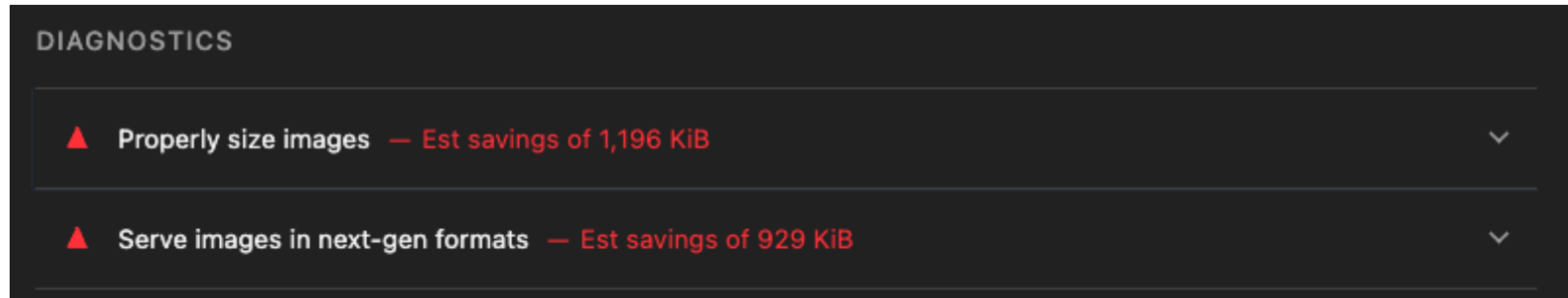
정상화 바로 ON



4초에서 1초로 최적화



리사이징이 되어도 뷰포트에서 육안으로 보기엔 문제가 없음



심지어 더 줄여도 된다고 함.

이건 CSS 사이즈, DPR, 이미지 사이즈 등등 고려해서 알잘딱

- 월 1,000,000회 의 이미지 요청(CloudFront 요청)
- 클라이언트에게 응답되는 이미지의 평균 크기: 200KB
- CloudFront 캐시 히트 비율: 80% (200,000회의 S3 GET, 람다 실행)
- 람다 메모리: 1GB (1024MB)
- 람다 평균 실행 시간: 1초

만약 리사이징 없이 원본 사이즈를 전송한다면

$$381GB \times 0.120\$ = 45.72\$$$

여태 발생했던 비용들을 요약하자면 아래와 같다.

- S3 GET Request: **0.07\$**
- CloudFront Internet Data Transfer: **24\$**
- CloudFront HTTPS Request: **1.2\$**
- Lambda@Edge Execution: **10\$**
- Lambda@Edge Request: **0.12\$**

정도가 발생, 리사이징을 하는게 더 효율적이라는 말임

자세한 계산 방법은 블로그 참고 바람.

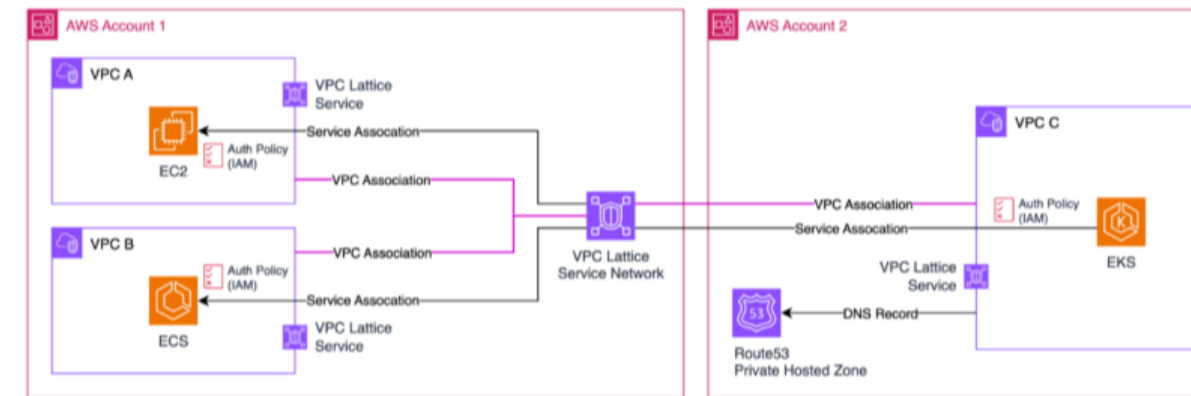
그래서 모두 합산한다면 월 **35.39\$**의 요금이 부과되게 된다.



K₈UBERNETES

GATEWAY API

with AWS VPC Lattice



by Kim Jun Young (@yulmwu)

[Kubernetes] [EKS] Gateway API (Feat. AWS VPC Lattice)

Gateway API instead of Ingress, and AWS Lattice Demo

aws

eks

kubernetes

networking

2025년 9월 9일 · 0개의 댓글 · ♥ 0

<https://velog.io/@yulmwu/kubernetes-gateway>

00

Overview

- whoami
- Hello, DevOps!
- What is **Kubernetes/EKS** and why use it?

01

Kubernetes Networking

- Overview
- Service
- Ingress
- **Gateway API**

02

AWS VPC

- Overview
- Peering
- Transit Gateway
- PrivateLink
- **Lattice**

03

Example Demo

- Overview
- EKS Cluster
- Deployment & Service
- Pod Identity(IAM)
- SG Lattice Prefix List
- Gateway API Controller
- Lattice Service Network
- Gateway, HTTPRoute
- Route 53 Private Hosted Zone
- Testing

04

Calculate Price

- VPC Lattice Pricing
- Lattice vs. ALB Comparison

05

Conclusion



KIM JUN YOUNG

010-2980-1336

normal8781@gmail.com

Discord @rlawnsdud

Github @yulmwu



세명컴퓨터고등학교
Semyeong Computer High School



새로운 기능 만들었는데 배포하고 출시해주셈

ㄴㄴ 지금 안됨; 테스트 제대로 해보심? 안정성이 최고임

FE/BE 개발팀



에이...
전 싫어요

인프라 운영팀



신기능 나온다면서 언제나와요??????

유저



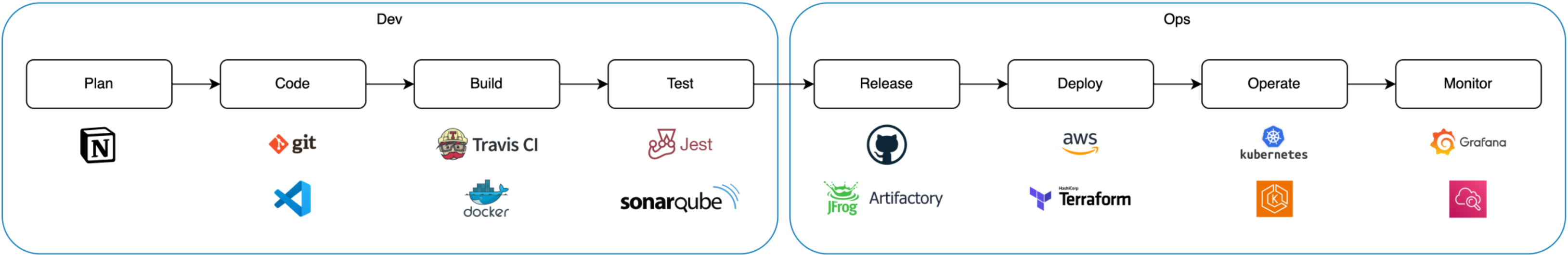
이거 버그나는데요???

유저

(서비스 종료)

DevOps = Development + Operations + α

이번 주제인 AWS, Kuberentes



단순 개발 + 인프라 운영 뿐만 아니라 CI/CD, 테스트, 모니터링, 보안 등 통합



- 장점: 빠른 개발부터 배포, 커뮤니케이션/협업 쉬워짐, 자동화 등
- 단점: 러닝 커브, 자동화 의존 등

전통적으로 서버를 배포하려면 어떻게 해야할까? (상남자식 배포)



(대충 복잡한 애플리케이션 구조)

서버 접속 > 코드 수정 > 애플리케이션 관리 등등, 복잡함 + 유지보수에 효율적이지 못함, 기존의 의존성 충돌 등

전통적으로 서버를 배포하려면 어떻게 해야할까? (VM 사용 시)

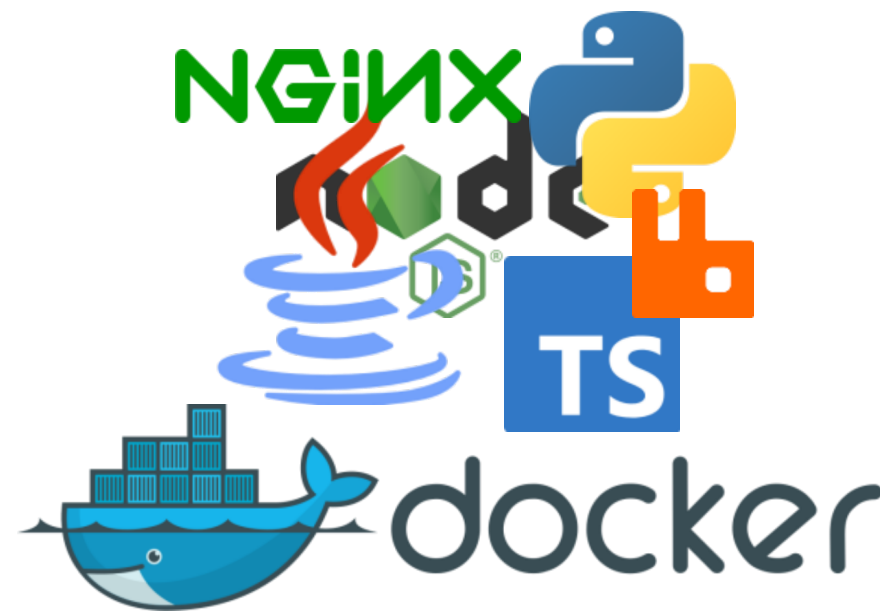


(가상머신 사용)

가상머신 사용: 독립된 환경에 분리되어 충돌 적음, 효율적인 자원 활용, 환경 분리 용이

단점: 무거움(OS 자체를 가상화) ⇒ 라이브러리, 패키지, 아키텍트만 올려둘 순 없을까?

그러한 단점들로 컨테이너(Container) 기술이 등장함

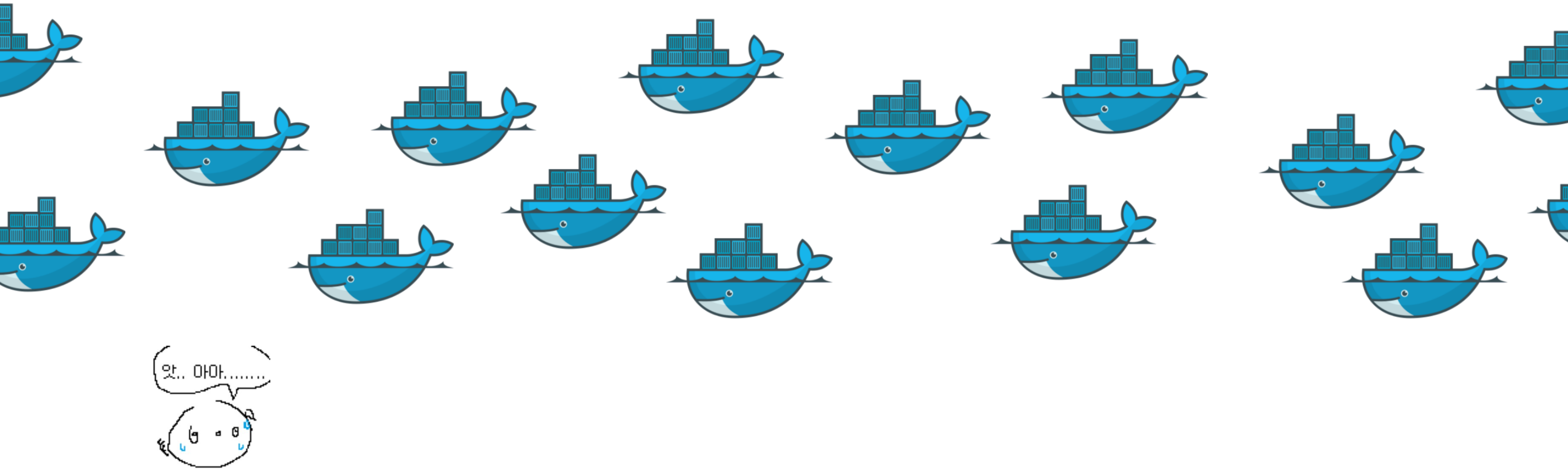


(도커, Containerd 등의 컨테이너 기술)

컨테이너 사용: 필요한 애플리케이션/런타임/라이브러리만 패키징, 호스트 OS(커널) 공유

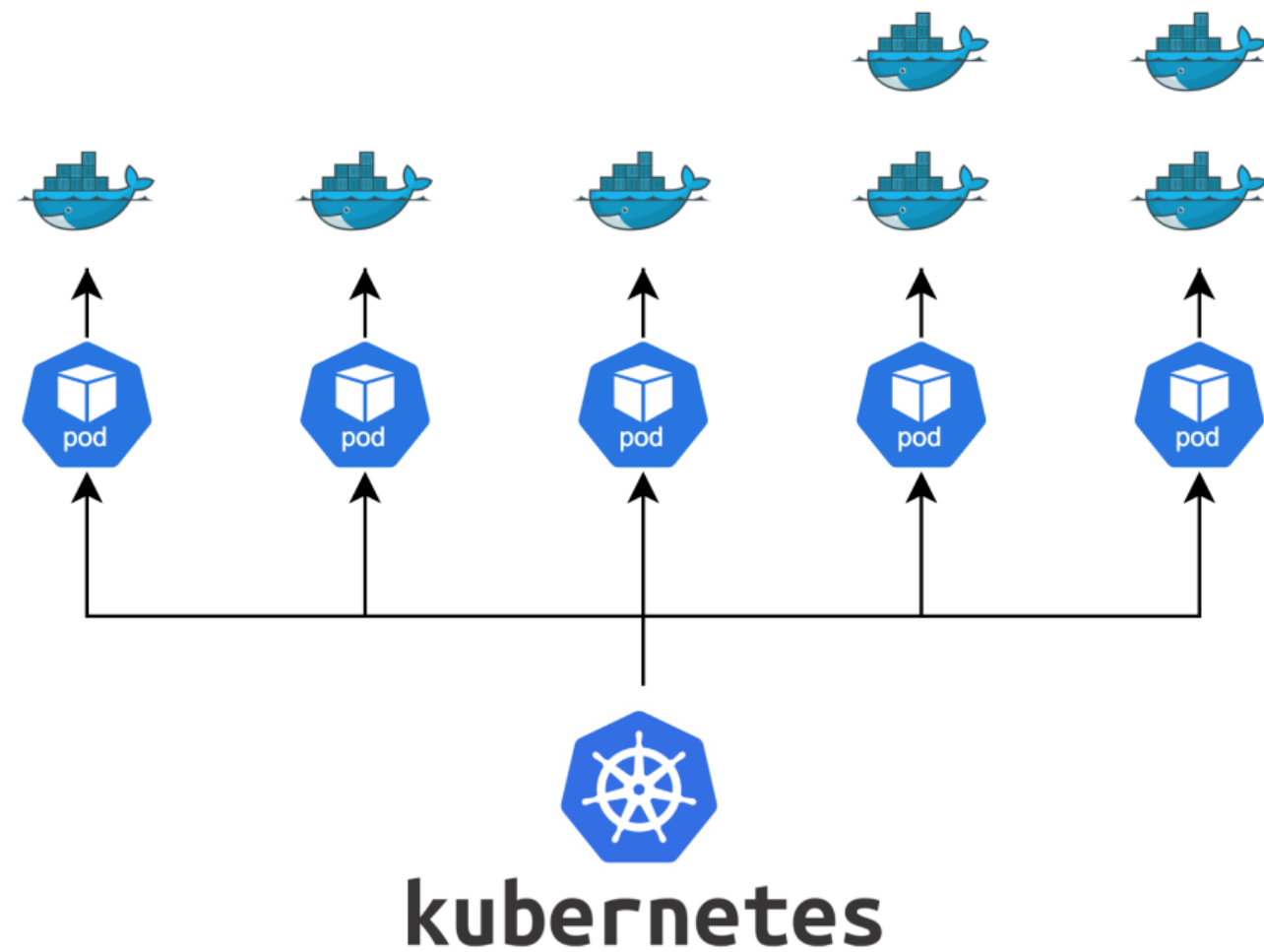
어디서든 똑같이 실행된다, VM보다 빠르게 확장가능, 가벼움 등

그런데 수많은 컨테이너들을 어떻게 관리하지?



(기본적으로 컨테이너는 "소모품" 형태로 언제든지 생성/확장=스케일링/삭제될 수 있음)

그런데 수많은 컨테이너들을 어떻게 관리하지?



쿠버네티스: 컨테이너화된 애플리케이션을 배포/확장/관리하는 선언형 컨테이너 오케스트레이션 플랫폼 (지금은 기본 Containerd 런타임)

파드: 기본 단위, 1개 이상의 컨테이너 이미지를 가짐

노드: 파드를 실행하는데 필요한 컴퓨팅 리소스 단위 (워커노드)

클러스터: 노드들을 모아둔 것

서비스: 파드들에 대해 안정적인 네트워크 접근을 제공함



설명할 기초 개념은 많지만 다 설명하면 1~2시간은 넘으니 패스

Elastic Kubernetes Service (EKS)

클라우드(AWS)의 유연한 자원 활용성 + 컨테이너 오케스트레이션 기능 = 확장성, 고가용성, 이식성, 자동화 등등
단점은 러닝 커브, 비용 밖에 없음 (뇌피셜)



AWS 쿠버네티스 서비스(완전 관리형 = 컨트롤 플레인/워커 노드, ELB, IAM 등 AWS 서비스 연동 등)

" EKS "

와 정말 알고싶던
내용이었어요 감
사합니다! ^^*



01

Kubernetes Networking

Service, Ingress, Gateway API

파드 사이(Service Discovery), 혹은 쿠버네티스 클러스터 외부와 어떻게 통신할까?

기본적으로 클러스터 안에선 노드의 kubelet을 통해 서로 통신 자체는 가능
다만 외부로 노출 불가 + Private IP를 못찾을 뿐(IP 찾는걸 Discovery라고 칭함)

" Service "



파드 사이에 안정적인 엔드포인트를 만들어주는 리소스 (요약: 파드끼리 통신 쉽게 도와주고 외부로 노출시키는 애)

용도에 따라 3가지의 타입이 있음

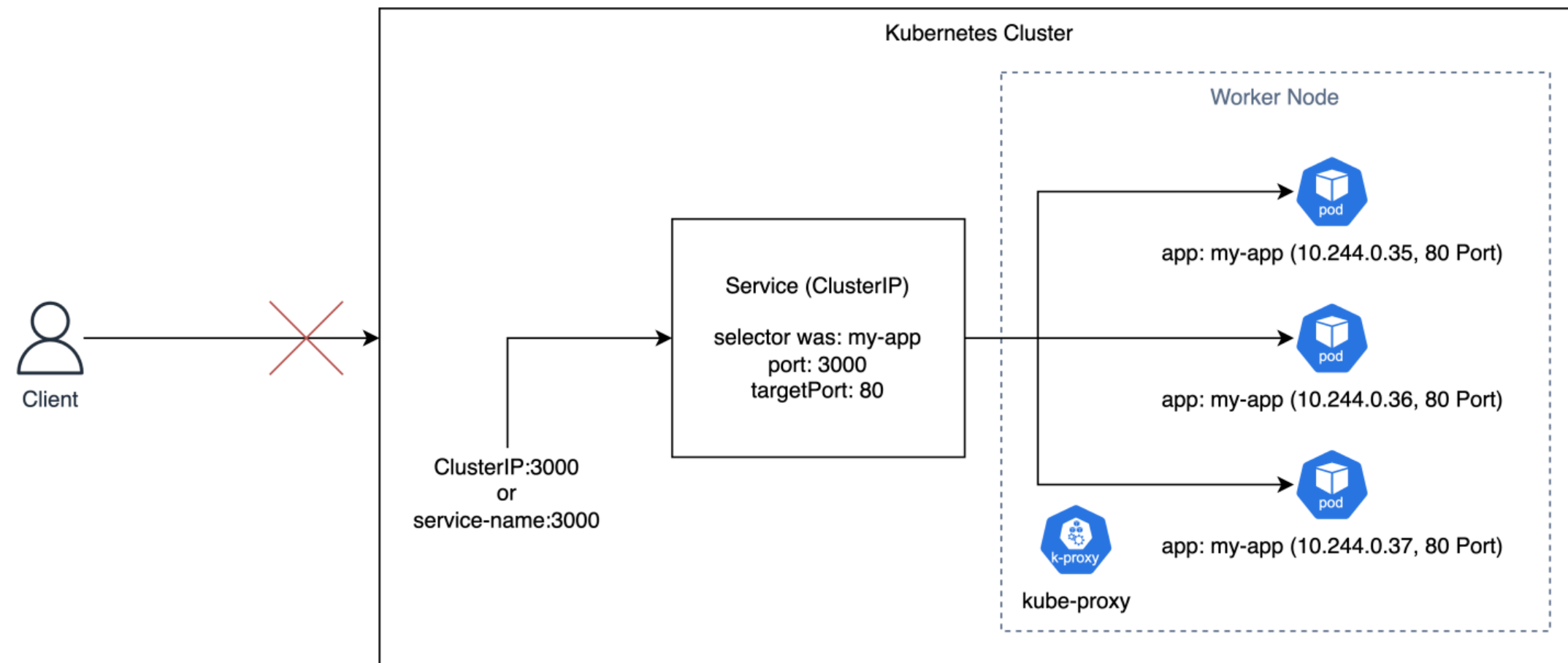
ClusterIP NodePort LoadBalancer

ClusterIP

파드에 붙는 클러스터 CIDR을 가진 Private IP는 동적으로 할당되기 때문에 서로 Discovery 불가

그래서 클러스터 안에 가상의 IP를 만들고 파드 IP + Port로 매핑함.

(ReplicaSet, Deployment 등으로 파드가 여러개라면 자동으로 로드밸런싱이 됨)



ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: app-clusterip-svc
spec:
  type: ClusterIP
  selector:
    app: app1
  ports:
    - name: http
      port: 3000    ClusterIP 포트
      targetPort: 8080    대상 파드 포트
```

우린 이걸 보고 Manifest라고 부르기로 했어요



ClusterIP

```

/Users/Workspace4/00/kubernetes/eks_test/service_clusterip [Ky0422@macbookpro] [15.02]
•> kubectl get svc -o wide
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE      SELECTOR
app-clusterip-svc   ClusterIP    172.20.145.82 <none>         3000/TCP        4m23s    app=app1
kubernetes           ClusterIP    172.20.0.1     <none>         443/TCP        132m     <none>

/ # for i in $(seq 1 10); do curl 172.20.145.82:3000; echo; done
Hello from app1-deploy-597f4cf788-ncbf6
Hello from app1-deploy-597f4cf788-lq98k
Hello from app1-deploy-597f4cf788-zb6qr
Hello from app1-deploy-597f4cf788-ncbf6
Hello from app1-deploy-597f4cf788-ncbf6
Hello from app1-deploy-597f4cf788-ncbf6
Hello from app1-deploy-597f4cf788-ncbf6
Hello from app1-deploy-597f4cf788-ncbf6
Hello from app1-deploy-597f4cf788-zb6qr
Hello from app1-deploy-597f4cf788-zb6qr
Hello from app1-deploy-597f4cf788-lq98k
/ # █

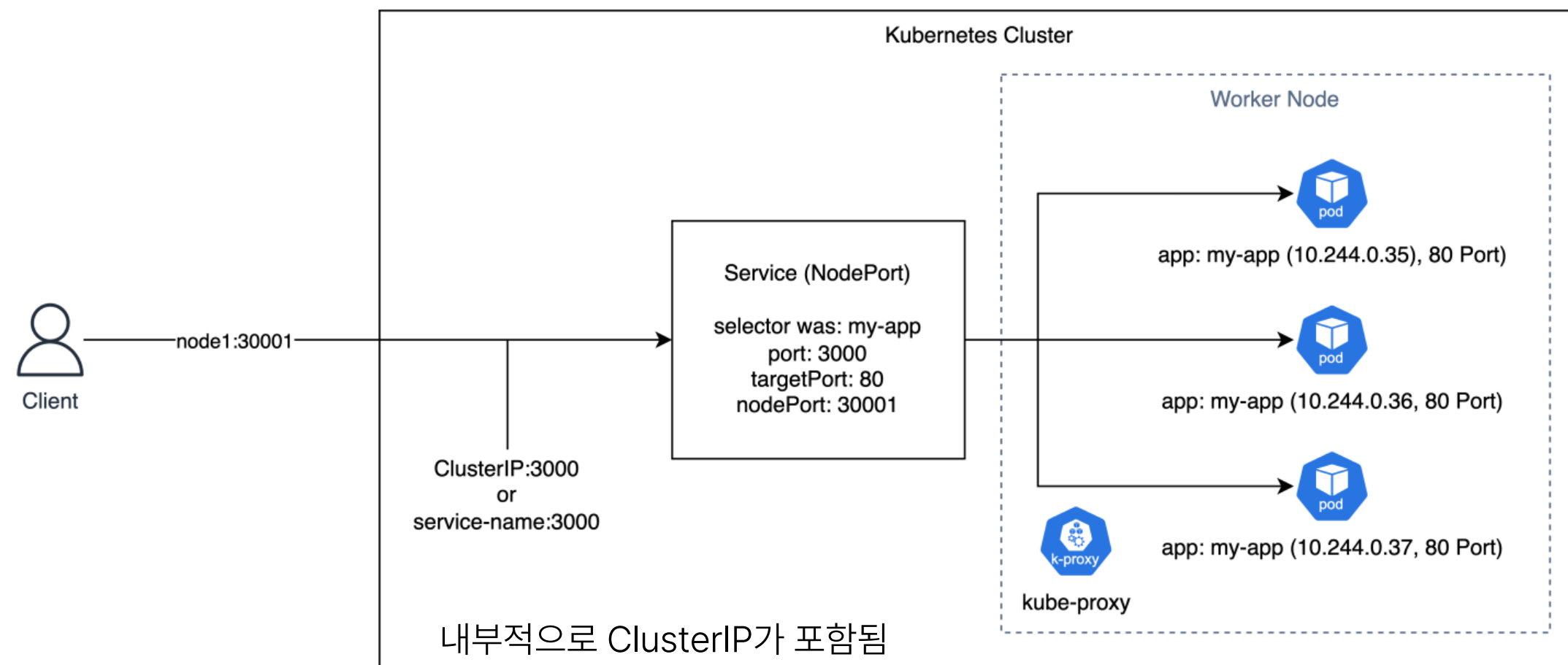
```

고정된 클러스터 내 Private IP를 통해 여러 파드로 로드밸런싱 (Service Discovery 용도)

NodePort

ClusterIP는 클러스터 외부와 통신이 불가능함, 그래서 노드의 Public IP와 특정 포드를 열어서 노출

다만 일반적인 프로덕션 환경에선 특정 노드에 Public IP를 할당해서 쓰진 않음



NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: app-nodeport-svc
spec:
  type: NodePort
  selector:
    app: app1
  ports:
    - name: http
      port: 3000
      targetPort: 8080
      nodePort: 30001
```

ClusterIP 서비스와 동일

NodePort (30000 ~ 32767)

노드의 특정 포트를 열어주는 것이기 때문에 클러스터 외부에서 접근하려면 해당 노드의 Public IP가 있어야함

일반적인 프로덕션 환경에선 잘 안씀, 디버깅 등등

NodePort

```
/Users/workspace4/dd/kubernetes/eks_test/service_nodeport [ky0422@MacBookPro] [19:27]
• > kubectl get svc -o wide
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE    SELECTOR
app-nodeport-svc    NodePort    172.20.25.185 <none>         3000:30001/TCP   33s    app=app1
kubernetes          ClusterIP   172.20.0.1    <none>         443/TCP          153m   <none>
```



```
/Users/workspace4/dd/kubernetes/eks_test/service_nodeport [ky0422@MacBookPro] [19:29]
• > kubectl get nodes -o wide
NAME                                STATUS    ROLES    AGE    VERSION    INTERNAL-IP    EXTERNAL-IP    OS-IMAGE             KERNEL-VERSION    CONTAINER-RUNTIME
ip-10-0-117-17.ap-northeast-2.compute.internal Ready    <none>    148m   v1.30.14-eks-3abbe1 10.0.117.17    <none>         Amazon Linux 2023.8.20250818 6.1.147-172.266.amzn2023.x86_64 containerd://1.7.27
ip-10-0-135-26.ap-northeast-2.compute.internal Ready    <none>    148m   v1.30.14-eks-3abbe1 10.0.135.26    <none>         Amazon Linux 2023.8.20250818 6.1.147-172.266.amzn2023.x86_64 containerd://1.7.27
ip-10-0-27-240.ap-northeast-2.compute.internal Ready    <none>    148m   v1.30.14-eks-3abbe1 10.0.27.240    13.209.73.50   Amazon Linux 2023.8.20250818 6.1.147-172.266.amzn2023.x86_64 containerd://1.7.27
```

Public EC2 노드 IP로 접속

```
/Users/workspace4/dd/kubernetes/eks_test/service_nodeport [ky0422@MacBookPro] [19:34]
• > curl 13.209.73.50:30001
Hello from app1-deploy-597f4cf788-ncbf6%

/Users/workspace4/dd/kubernetes/eks_test/service_nodeport [ky0422@MacBookPro] [19:34]
• > curl 13.209.73.50:30001
Hello from app1-deploy-597f4cf788-lq98k%

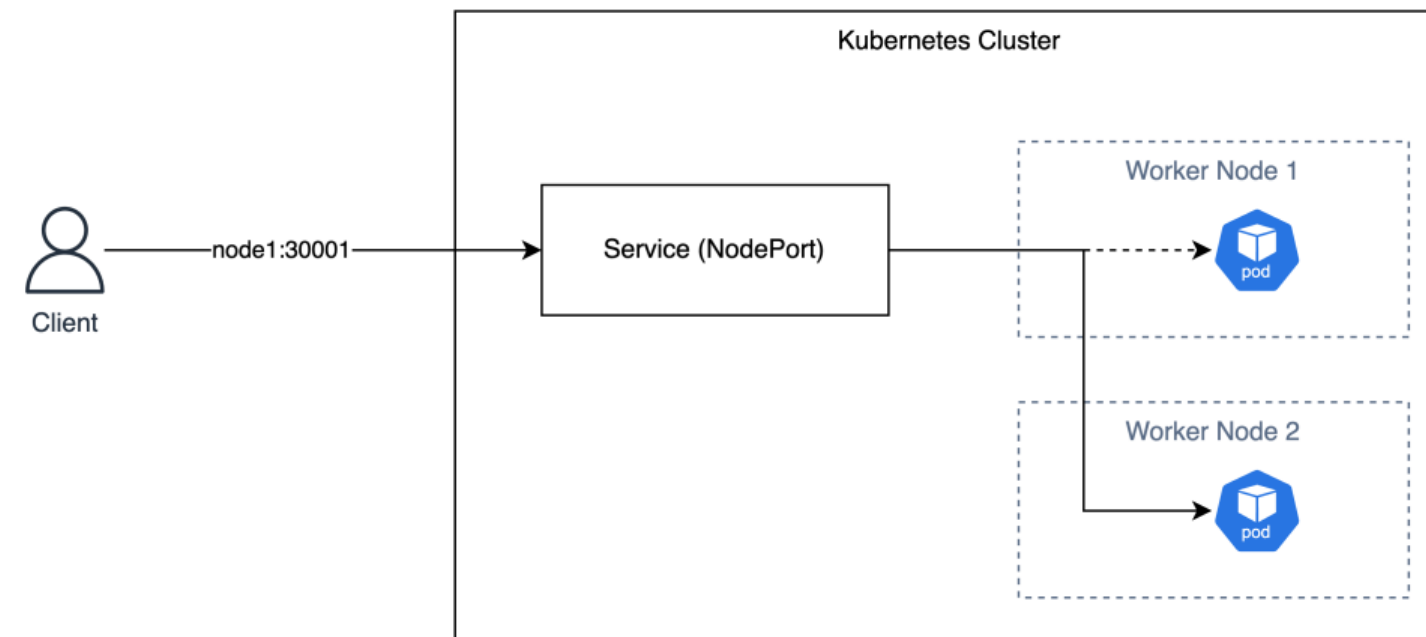
/Users/workspace4/dd/kubernetes/eks_test/service_nodeport [ky0422@MacBookPro] [19:35]
• > curl 13.209.73.50:30001
Hello from app1-deploy-597f4cf788-lq98k%

/Users/workspace4/dd/kubernetes/eks_test/service_nodeport [ky0422@MacBookPro] [19:35]
• > curl 13.209.73.50:30001
Hello from app1-deploy-597f4cf788-zb6qr%
```

NodePort - externalTrafficPolicy

NodePort로 받은 트래픽은 kube-proxy(iptables/ipvs)를 통해 엔드포인트(파드)를 특정

⇒ 기본적으로 상황에 따라 트래픽이 다른 노드의 파드로 갈 수 도 있음



by Kim Jun Young (@yulmwu)

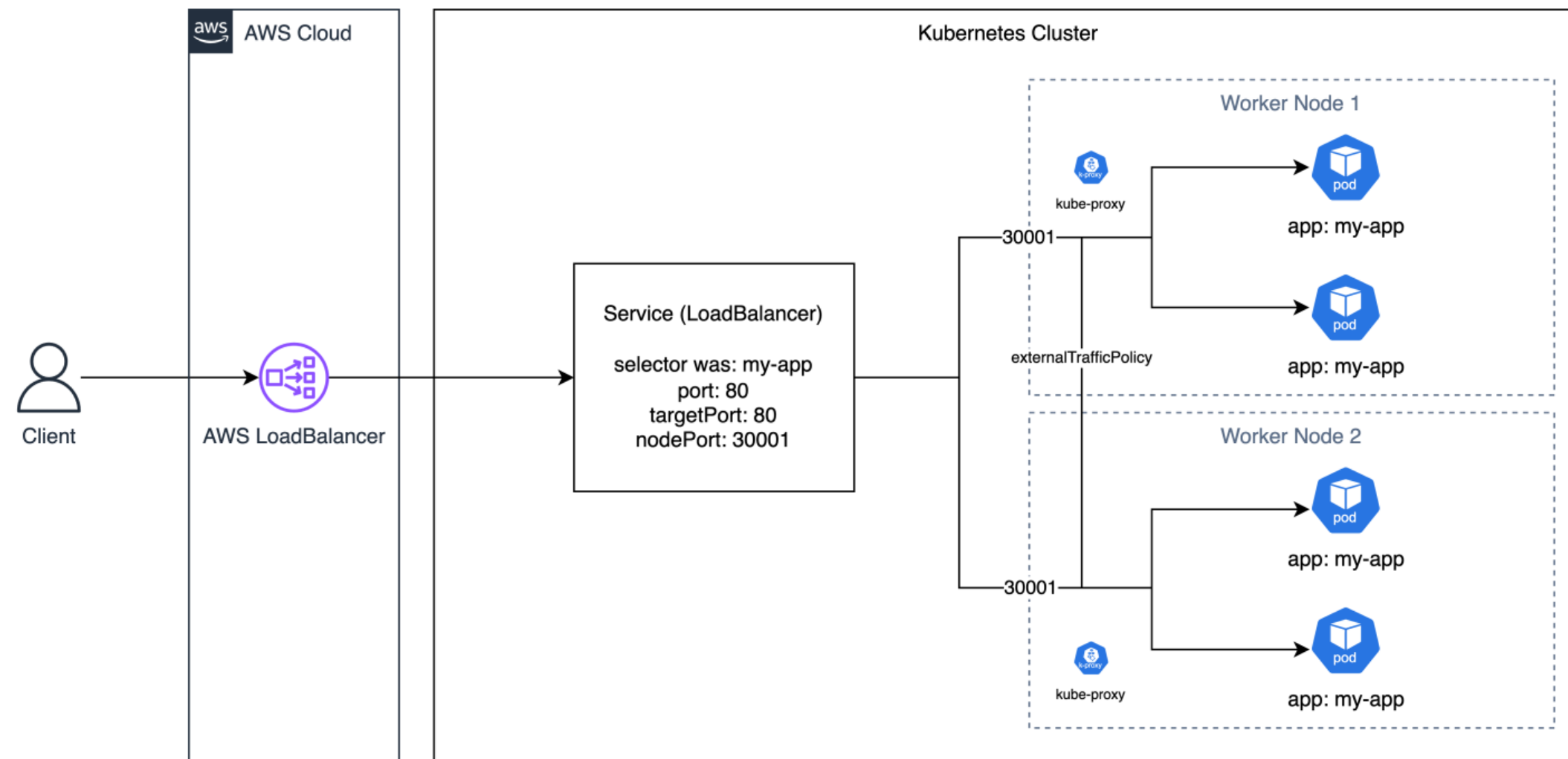
효율적일 순 있지만 상황에 따라 Cross AZ 비용, SNAT으로 인한 클라이언트 IP 보존 불가, 네트워크 홉 등의 단점

externalTrafficPolicy 옵션에서 Local로 설정 시 해당 NodePort로 다이렉트로 감 (기본값 Cluster)

LoadBalancer

여러 노드가 있는데 특정 NodePort를 가지고 접근하는건 실사용하기 어려움

그래서 앞에 로드밸런서를 뒀서 여러 노드로 트래픽을 분산하여 외부로 노출함 (AWS ELB L4/L7)



LoadBalancer

```
apiVersion: v1
kind: Service
metadata:
  name: app-lb-svc
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
spec:
  type: LoadBalancer
  selector:
    app: app1
  ports:
    - name: http
      port: 80
      targetPort: 8080
```

사용하는 로드밸런서 서비스(AWS ELB, Nginx 등)에 따라 다른 어노테이션이 필요

LoadBalancer

```
/Users/workspace4/dd/kubernetes/eks_test/service_loadbalancer [ky0422@MacBookPro] [19:37]
• > kubectl get svc -o wide
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE      SELECTOR
app-lb-svc    LoadBalancer 172.20.87.92    a8860f716b26049219a28d3912d4c59f-58e11c1aa4ebb5a6.elb.ap-northeast-2.amazonaws.com 80:30982/TCP 9s      app=app1
kubernetes    ClusterIP      172.20.0.1      <none>           443/TCP          164m     <none>
```

로드 밸런서 (1)

로드 밸런서 생성 ▼

Elastic Load Balancing은 수신 트래픽의 변화에 따라 자동으로 로드 밸런서 용량을 확장합니다.

로드 밸런서 필터링

< 1 > ⚙

☐	이름 ▼	상태 ▼	유형 ▼	체계 ▼	IP 주소 유형 ▼	VPC ID ↗
☐	a8860f716b26049219a...	✔️ 활성	network	Internet-facing	IPv4	vpc-09f72cfd9d9c

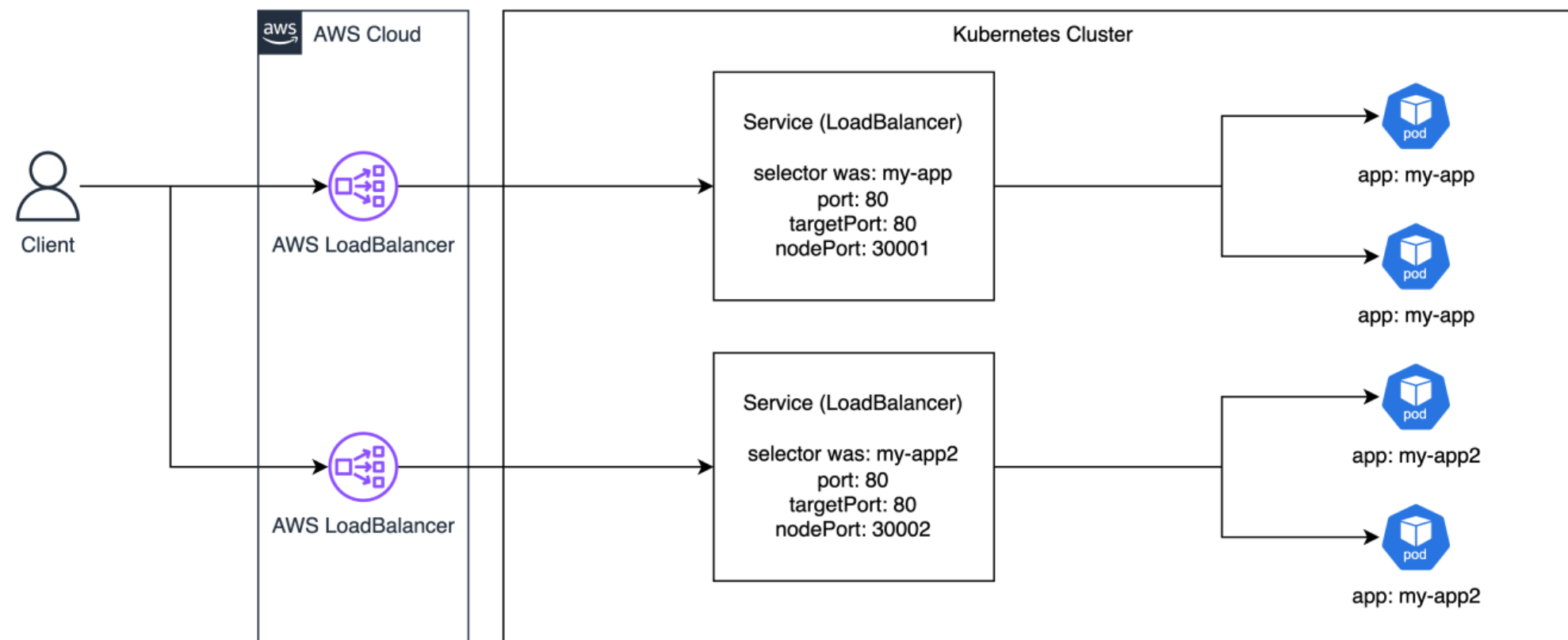
```
/Users/workspace4/dd/kubernetes/eks_test/service_loadbalancer [ky0422@MacBookPro] [19:38]
• > curl http://a8860f716b26049219a28d3912d4c59f-58e11c1aa4ebb5a6.elb.ap-northeast-2.amazonaws.com/
Hello from app1-deploy-597f4cf788-ncbf6%

/Users/workspace4/dd/kubernetes/eks_test/service_loadbalancer [ky0422@MacBookPro] [19:48]
• > curl http://a8860f716b26049219a28d3912d4c59f-58e11c1aa4ebb5a6.elb.ap-northeast-2.amazonaws.com/
Hello from app1-deploy-597f4cf788-zb6qr%

/Users/workspace4/dd/kubernetes/eks_test/service_loadbalancer [ky0422@MacBookPro] [19:48]
• > curl http://a8860f716b26049219a28d3912d4c59f-58e11c1aa4ebb5a6.elb.ap-northeast-2.amazonaws.com/
Hello from app1-deploy-597f4cf788-lq98k%
```


Ingress

만약 여러개의 Deployment(또는 마이크로서비스)를 외부로 노출하는 방법??



by Kim Jun Young (@yulmwu)

로드밸런서 여러개 만들기?

로드밸런서 비용 생각 안함?

인증서, 도메인 설정을 로드밸런서마다 설정함?

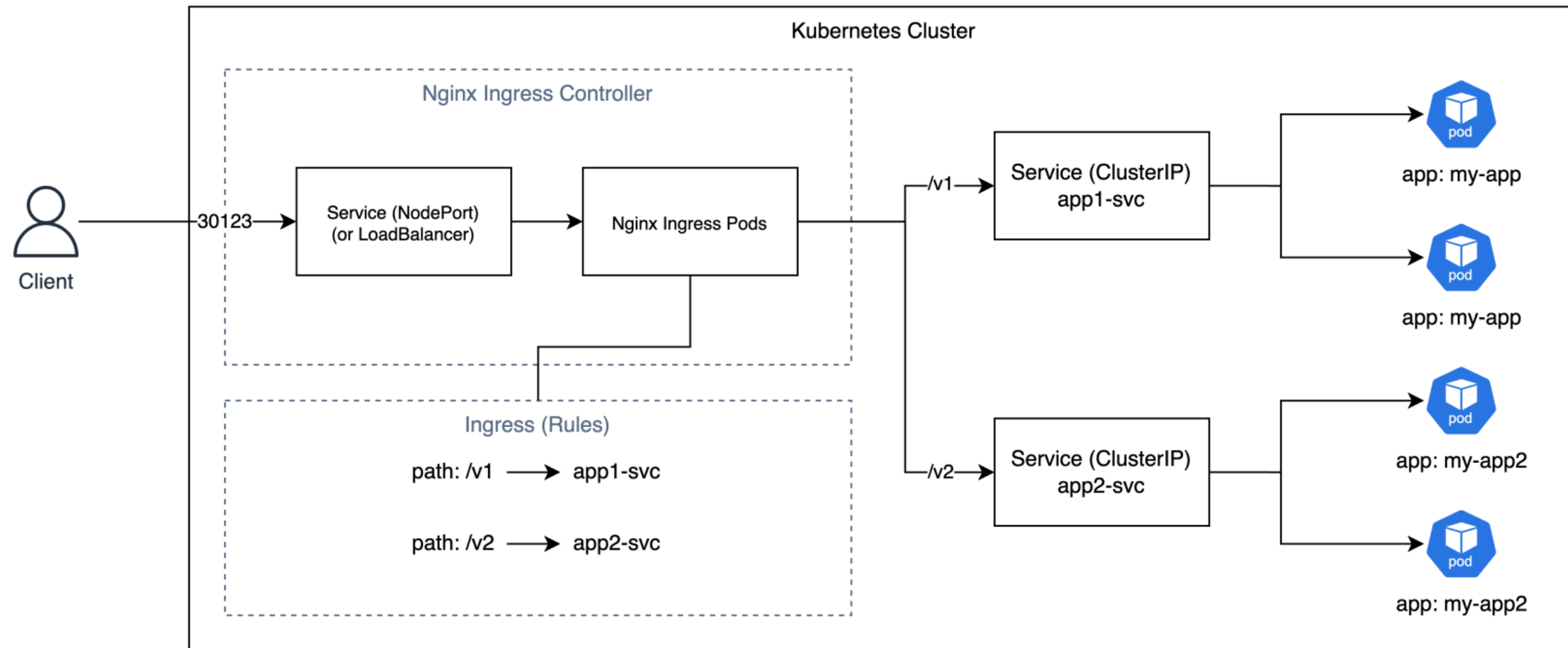
로드밸런서 엔드포인트가 여러개인데 어케سم??



Ingress

클러스터 외부에서 들어오는 HTTP/HTTPS L7 트래픽을 호스트/경로 기반으로 라우팅

실질적으론 Ingress Controller(AWS ALB Controller, Nginx 등)가 트래픽 분산을 담당함



Ingress

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
spec:
  controller: ingress.k8s.aws/alb
```

리스너 및 규칙 (1) 정보

리스너 필터링

< 1 >

프로토콜: 포트

기본 작업

규칙

ARN

보안 정책

기본 SSL/TLS 인증서

mTLS

고정 응답 반환

응답 코드: 404

응답 본문

응답 콘텐츠 유형: text/plain

3개 규칙

ARN

해당되지 않음

해당되지 않음

해당되지 않음

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /v1/*
            pathType: Prefix
            backend:
              service:
                name: app1-svc
                port:
                  number: 3000
          - path: /v2/*
            pathType: Prefix
            backend:
              service:
                name: app2-svc
                port:
                  number: 3000
```

```
/Users/workspace4/dd/kubernetes/eks_test/ingress_alb [ky0422@MacBookPro] [12:11]
•> for i in $(seq 1 3); do
  curl k8s-default-appingre-c982983f2b-1710225052.ap-northeast-2.elb.amazonaws.com/v1
  echo
done
Hello from app1-deploy-7d5bbbf897-r8qmd
Hello from app1-deploy-7d5bbbf897-q8xjb
Hello from app1-deploy-7d5bbbf897-cztfq

/Users/workspace4/dd/kubernetes/eks_test/ingress_alb [ky0422@MacBookPro] [12:11]
•> for i in $(seq 1 3); do
  curl k8s-default-appingre-c982983f2b-1710225052.ap-northeast-2.elb.amazonaws.com/v2
  echo
done
Hello from app2-deploy-b89cd8c9d-8fbnx
Hello from app2-deploy-b89cd8c9d-6nvk5
Hello from app2-deploy-b89cd8c9d-mcz45

/Users/workspace4/dd/kubernetes/eks_test/ingress_alb [ky0422@MacBookPro] [12:11]
•> kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE                                NOMINATED NODE   READINESS GATES
app1-deploy-7d5bbbf897-cztfq        1/1     Running   0           26m   10.0.127.77     ip-10-0-110-150.ap-northeast-2.compute.internal   <none>            <none>
app1-deploy-7d5bbbf897-q8xjb        1/1     Running   0           26m   10.0.43.107     ip-10-0-43-197.ap-northeast-2.compute.internal   <none>            <none>
app1-deploy-7d5bbbf897-r8qmd        1/1     Running   0           26m   10.0.141.19     ip-10-0-133-212.ap-northeast-2.compute.internal   <none>            <none>
app2-deploy-b89cd8c9d-6nvk5        1/1     Running   0           26m   10.0.134.55     ip-10-0-133-212.ap-northeast-2.compute.internal   <none>            <none>
app2-deploy-b89cd8c9d-8fbnx        1/1     Running   0           26m   10.0.108.75     ip-10-0-110-150.ap-northeast-2.compute.internal   <none>            <none>
app2-deploy-b89cd8c9d-mcz45        1/1     Running   0           26m   10.0.56.25      ip-10-0-43-197.ap-northeast-2.compute.internal   <none>            <none>

/Users/workspace4/dd/kubernetes/eks_test/ingress_alb [ky0422@MacBookPro] [12:11]
o>
```


Ingress

EC2 > 대상 그룹 > k8s-default-app1svc-de25870554

k8s-default-app1svc-de25870554

작업

세부 정보

arn:aws:elasticloadbalancing:ap-northeast-2:986129558966:targetgroup/k8s-default-app1svc-de25870554/104227bd8242c7ff

대상 유형

IP

대상 주소 유형

IPv4

프로토콜 : 포트

HTTP: 8080

로드 밸런서

k8s-default-appingre-c982983f2b

프로토콜 버전

HTTP1

VPC

vpc-0cc03657e518f2027

3

대상 합계

3

정상

0 이상

0

비정상

0

사용되지 않음

0

초기

0

드레이닝

가용 영역별 대상 배포

아래의 등록된 대상 테이블에 적용된 해당 필터를 보려면 이 테이블에서 값을 선택합니다.

대상

모니터링

상태 검사

속성

태그

등록된 대상 (3)

정보

이상 완화: 해당되지 않음

등록 취소

대상 등록

대상 그룹은 지정한 프로토콜 및 포트 번호를 사용하여 등록된 개별 대상으로 요청을 라우팅합니다. 상태 확인은 대상 그룹의 상태 확인 설정에 따라 등록된 모든 대상에 대해 수행됩니다. 이상 탐지는 정상 대상이 3개 이상 있는 HTTP/HTTPS 대상 그룹에 자동으로 적용됩니다.

대상 필터링

IP 주소

포트

영역

상태 확인

상태 확인 세부 정보

관리상 재정의

재정의 ...

이상 탐지 결과

10.0.141.19

8080

ap-northeast-2a (ap...

Healthy

-

No override

No overri...

Normal

10.0.127.77

8080

ap-northeast-2b (ap...

Healthy

-

No override

No overri...

Normal

10.0.43.107

8080

ap-northeast-2a (ap...

Healthy

-

No override

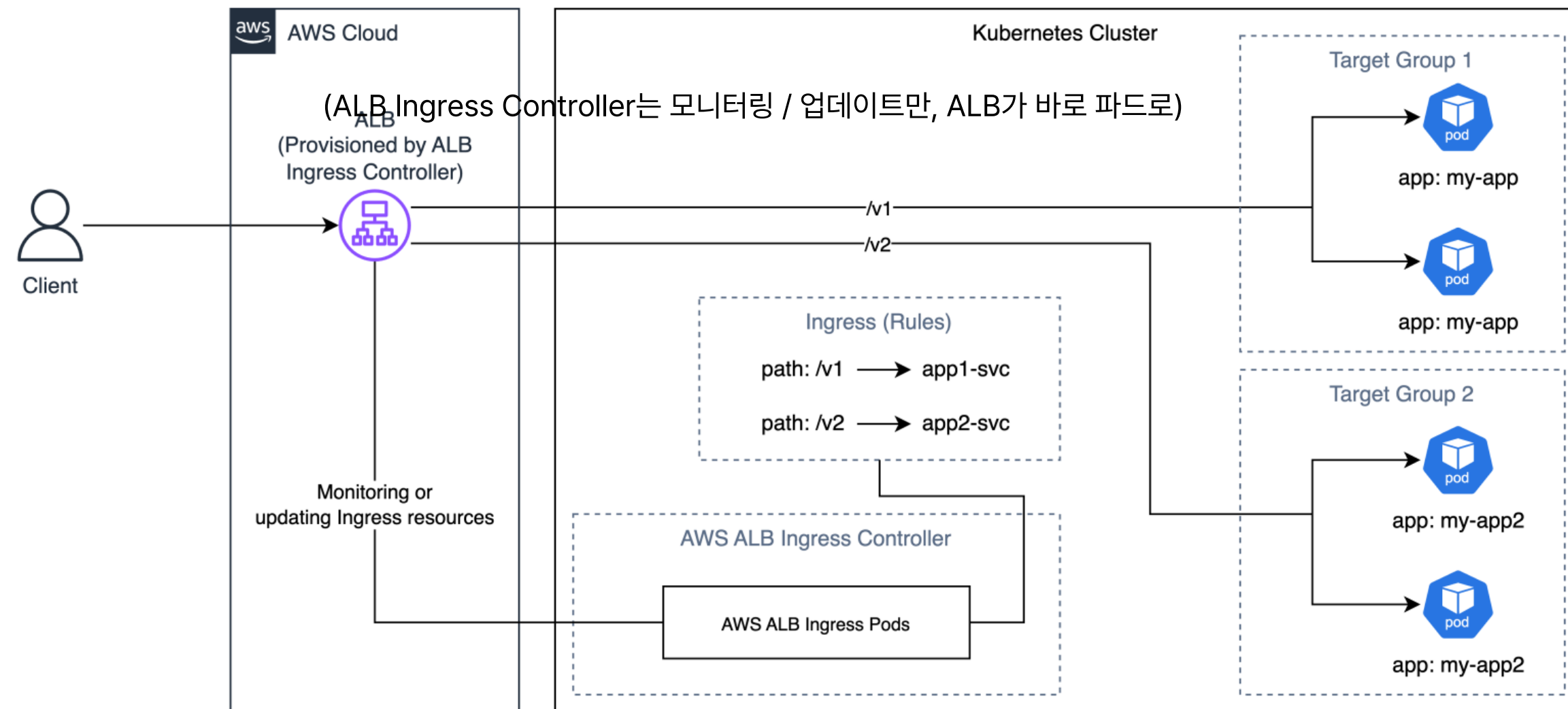
No overri...

Normal

로드밸런서 타겟 그룹에 노드가 아니라 파드가? ⇒ target-type: ip

Ingress - IP Target Type

LoadBalancer 서비스, ALB Ingress Controller에선 로드밸런서의 타겟 타입을 파드의 IP로 직접 지정 가능



by Kim Jun Young (@yulmwu)

`alb.ingress.kubernetes.io/target-type: ip`

⇒ 이 경우엔 `externalTrafficPolicy`가 필요가 없어짐 (ALB 업데이트를 위한 ClusterIP 서비스만 있으면 됨)

Ingress

그런데 TCP/UDP 등의 L4 트래픽은 어떻게 처리하지? (Ingress는 L7용)

+ 컨트롤러마다 어노테이션(메타데이터)가 달라서 일관성이 없고 표준성도 없음

```
annotations:
  kubernetes.io/ingress.class: alb
  alb.ingress.kubernetes.io/scheme: internet-facing
  alb.ingress.kubernetes.io/actions.response-503: >
    {"Type":"fixed-response","FixedResponseConfig":{"ContentType":"text/plain","StatusCode":"503","MessageBody":"503 error text"}}
  alb.ingress.kubernetes.io/actions.redirect-to-eks: >
    {"Type":"redirect","RedirectConfig":{"Host":"aws.amazon.com","Path":"/eks/","Port":"443","Protocol":"HTTPS","Query":"k=v","StatusCode":"HTTP_302"}}
  alb.ingress.kubernetes.io/actions.forward-single-tg: >
    {"Type":"forward","TargetGroupArn": "arn-of-your-target-group"}
  alb.ingress.kubernetes.io/actions.forward-multiple-tg: >
    {"Type":"forward","ForwardConfig":{"TargetGroups":[{"ServiceName":"service-1","ServicePort":"80","Weight":20},{ "ServiceName":"service-2","ServicePort":"80","Weight":20},{ "TargetGroupArn":"arn-of-y
```

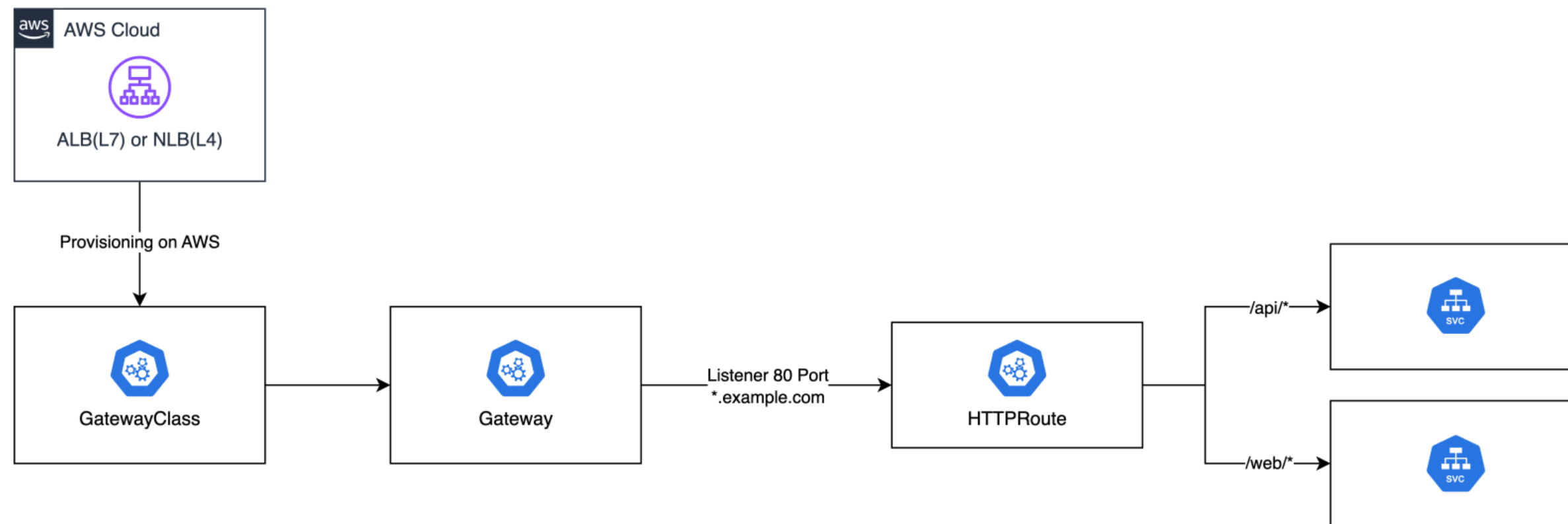


(전부 ALB Ingress Controller에서 사용되는 어노테이션)

Name	Type	Default	Location
alb.ingress.kubernetes.io/actions.\$(action-name)	json	N/A	ingress
alb.ingress.kubernetes.io/auth-idp-cognito	json	N/A	ingress,service
alb.ingress.kubernetes.io/auth-idp-oidc	json	N/A	ingress,service
alb.ingress.kubernetes.io/auth-on-unauthenticated-request	authenticate allow deny	authenticate	ingress,service
alb.ingress.kubernetes.io/auth-scope	string	openid	ingress,service
alb.ingress.kubernetes.io/auth-session-cookie	string	AWSAuthSessionCookie	ingress,service
alb.ingress.kubernetes.io/auth-session-timeout	integer	'604800'	ingress,service
alb.ingress.kubernetes.io/auth-type	none oidc cognito	none	ingress,service
alb.ingress.kubernetes.io/backend-protocol	HTTP HTTPS	HTTP	ingress,service
alb.ingress.kubernetes.io/certificate-arn	stringList	N/A	ingress
alb.ingress.kubernetes.io/conditions.\$(conditions-name)	json	N/A	ingress
alb.ingress.kubernetes.io/healthcheck-interval-seconds	integer	'15'	ingress,service
alb.ingress.kubernetes.io/healthcheck-path	string	'/'	ingress,service
alb.ingress.kubernetes.io/healthcheck-port	integer traffic-port	traffic-port	ingress,service
alb.ingress.kubernetes.io/healthcheck-protocol	HTTP HTTPS	HTTP	ingress,service
alb.ingress.kubernetes.io/healthcheck-timeout-seconds	integer	'5'	ingress,service
alb.ingress.kubernetes.io/healthy-threshold-count	integer	'2'	ingress,service
alb.ingress.kubernetes.io/inbound-cidrs	stringList	0.0.0.0/0	ingress
alb.ingress.kubernetes.io/ip-address-type	ipv4 dualstack	ipv4	ingress
alb.ingress.kubernetes.io/listen-ports	json	{["HTTP": 80]} {["HTTPS": 443]}	ingress
alb.ingress.kubernetes.io/load-balancer-attributes	stringMap	N/A	ingress
alb.ingress.kubernetes.io/scheme	internal internet-facing	internal	ingress
alb.ingress.kubernetes.io/security-groups	stringList	N/A	ingress
alb.ingress.kubernetes.io/shield-advanced-protection	boolean	N/A	ingress
alb.ingress.kubernetes.io/ssl-policy	string	ELBSecurityPolicy-2016-08	ingress
alb.ingress.kubernetes.io/subnets	stringList	N/A	ingress
alb.ingress.kubernetes.io/success-codes	string	'200'	ingress,service
alb.ingress.kubernetes.io/tags	stringMap	N/A	ingress
alb.ingress.kubernetes.io/target-group-attributes	stringMap	N/A	ingress,service
alb.ingress.kubernetes.io/target-type	instance ip	instance	ingress,service
alb.ingress.kubernetes.io/unhealthy-threshold-count	integer	'2'	ingress,service
alb.ingress.kubernetes.io/waf-acl-id	string	N/A	ingress
alb.ingress.kubernetes.io/wafv2-acl-arn	string	N/A	ingress

Gateway API

HTTP(S) 뿐만 아니라 TCP/UDP, gRPC 등의 여러 프로토콜을 지원함 + 여러 라우팅 옵션 등등



by Kim Jun Young (@yulmwu)

⇒ 표준화된 어노테이션 + 역할 분리(TargetGroupPolicy 등) + 일관성있게 사용 가능한 Ingress 상위호환 리소스

Gateway API

Ingress와 마찬가지로 Gateway API Controller가 실질적으로 트래픽을 처리

컨트롤러는 AWS EKS 환경이라면 AWS ALB(L7)/NLB(L4), VPC Lattice 등을 사용

GatewayClass: 어떤 Gateway API Controller를 사용할지 정의

Gateway: 어느 Route로 보낼지 리스너, 인증서 설정 등 실질적인 인스턴스 정의

Route: 각 프로토콜 별로 트래픽을 어느 Service로 전달할지 정의

자세한 내용은 블로그에서



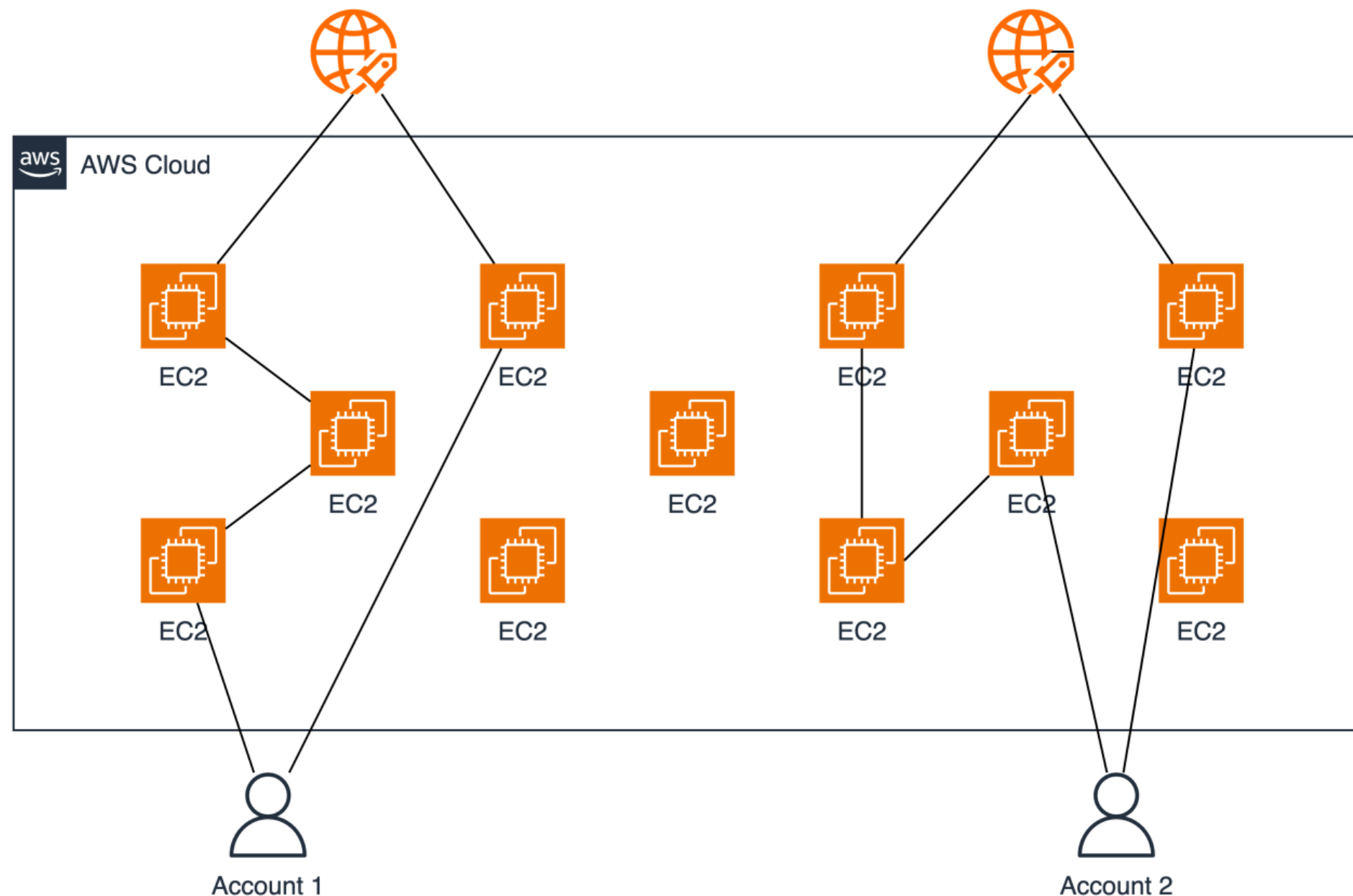
02

AWS VPC

Peering, Transit Gateway, PrivateLink, Lattice

AWS VPC

여러 AWS 서비스간 네트워크 통신을 하려면 어떻게 해야할까?

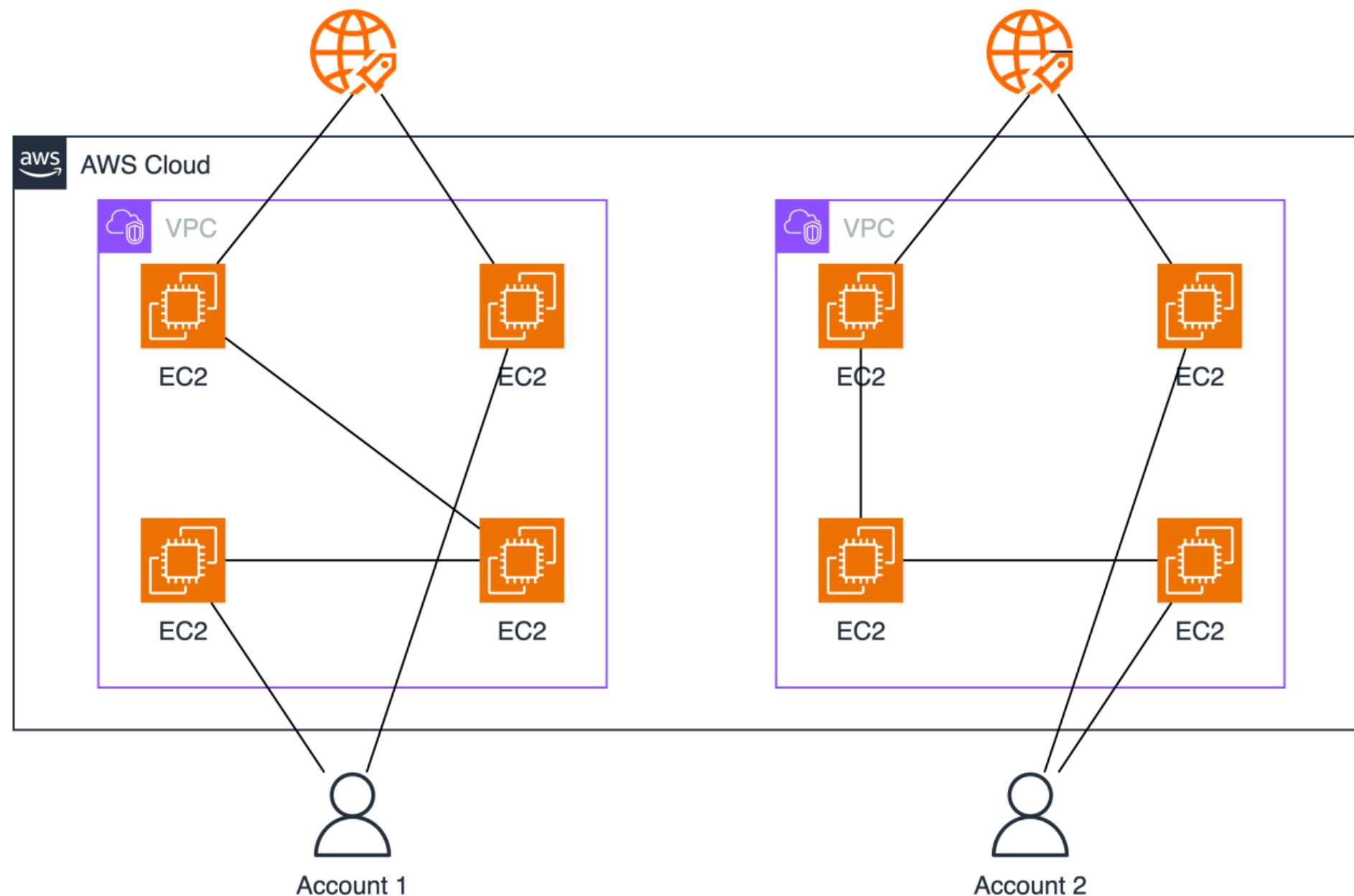


계정 별로 여러 EC2를 가지고 있고 통신을 해야한다면?
Account 1이 Account 2의 EC2에 접근하면 안되는데..

계정 별로 네트워크를 논리적으로 격리시키면 어떨까?

AWS VPC

여러 AWS 서비스간 네트워크 통신을 하려면 어떻게 해야할까?



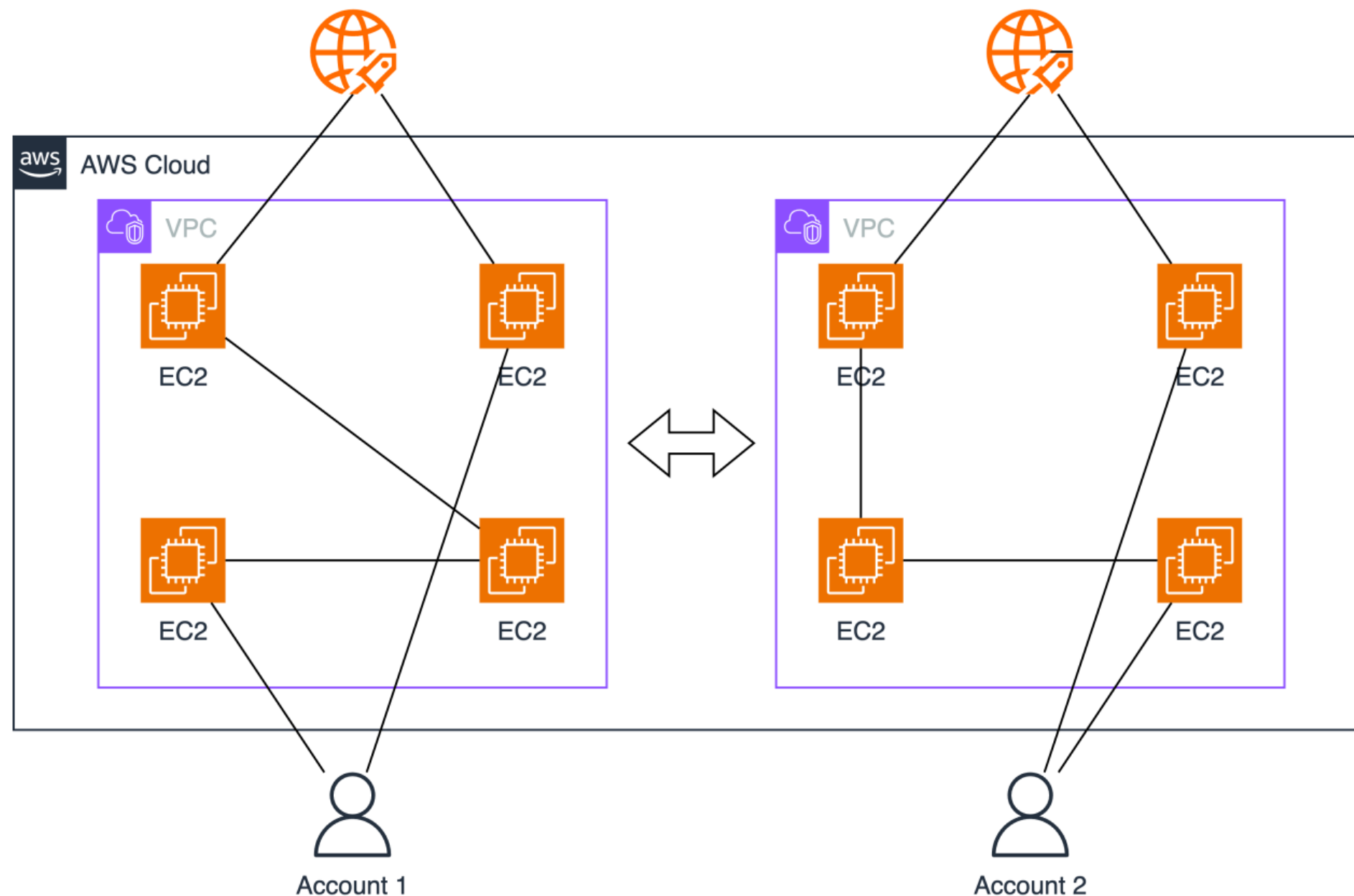
VPC = 계정 단위로 논리적으로 격리된 가상 네트워크

AWS 네트워킹의 핵심, Global 서비스 등을 제외하면
대부분 VPC에 의존됨

⇒ 여러 VPC 간 네트워크 통신은 불가능(제작자의 의도)

AWS VPC

그런데 나는 분산된 계정/리전에 여러 VPC를 두고 하나의 서비스로 이용하고 싶은데?



가용성을 위해 Cross Region에 여러 VPC 등
⇒ 어떻게 VPC 간 통신을 하지?

VPC Peering

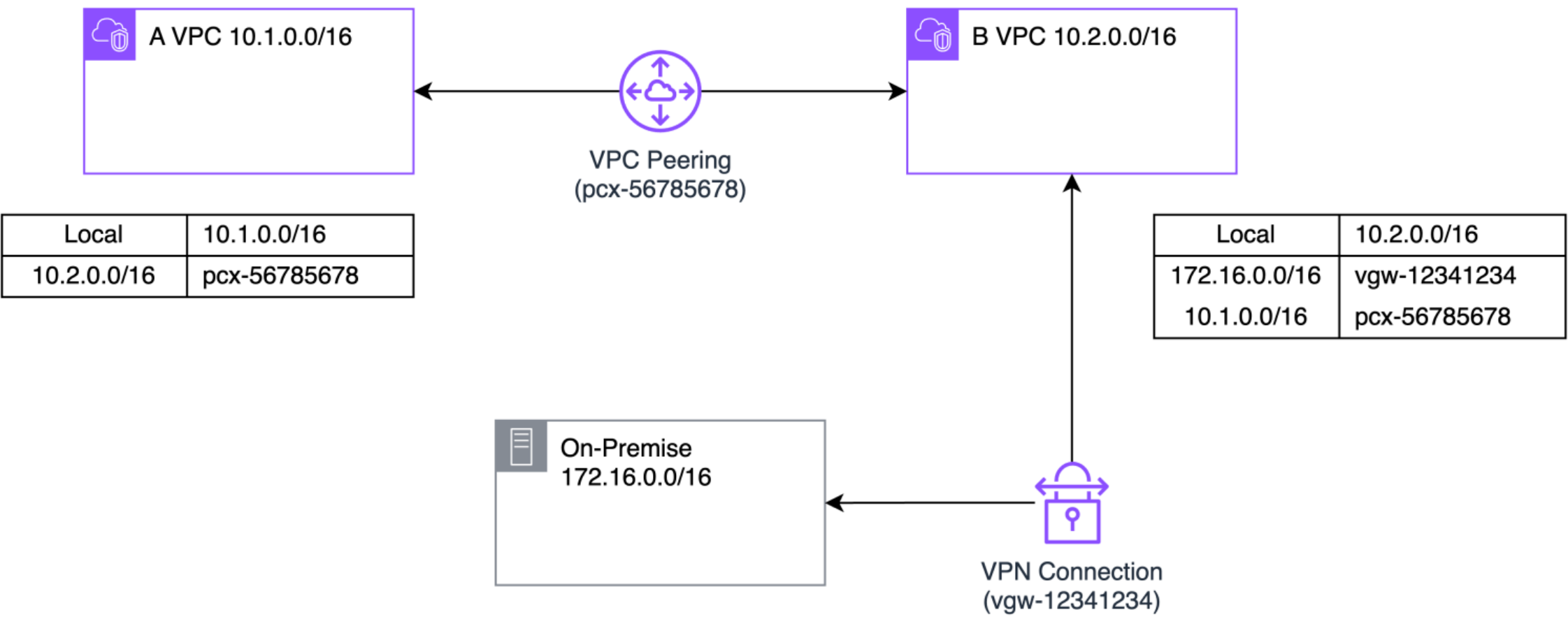
VPC Transit Gateway(TGW)

VPC PrivateLink (Endpoint)

VPC Lattice

VPC Peering

두 VPC 간의 통신을 위한 일종의 통로를 뚫어줌

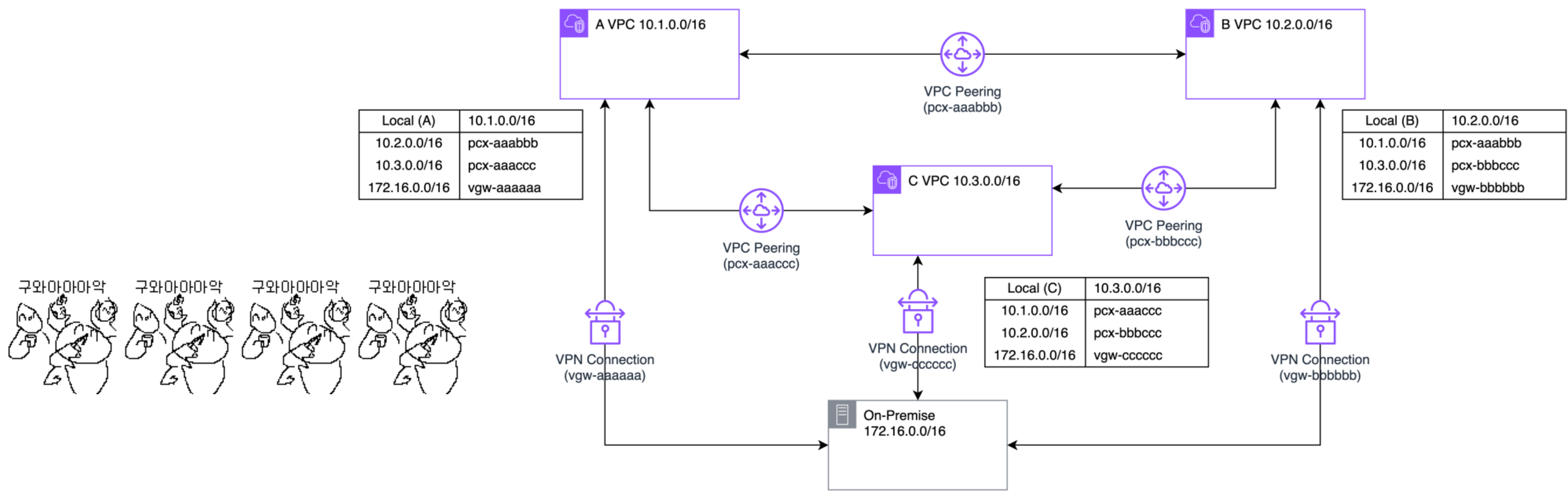


by Kim Jun Young (@yulmwu)

VPC Peering을 설정하고 pcx-xxx를 각 VPC 서브넷 라우팅 테이블에 설정

VPC Peering

" 너, VPC 쌓여있잖아 "



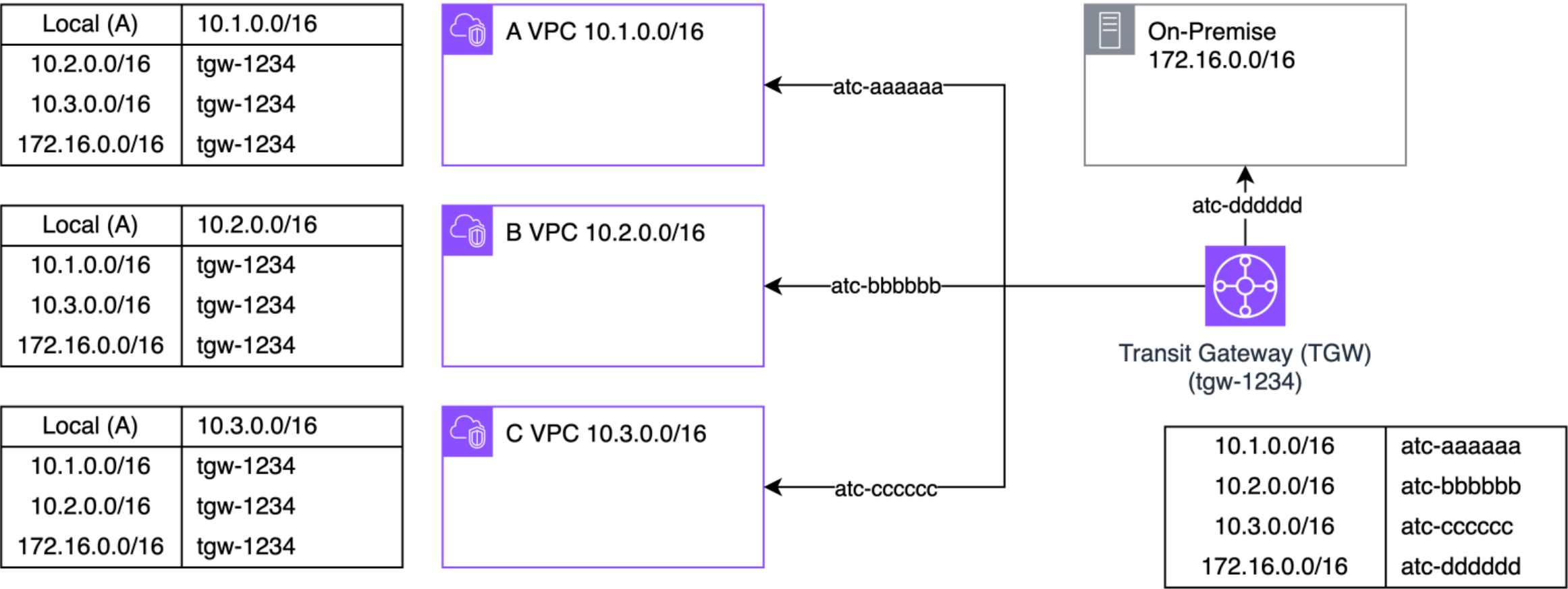
by Kim Jun Young (@yulmwu)

필요한 Peering 리소스 수 = $\frac{N \times (N-1)}{2}$

VPC 마다 Peering을 설정하고 라우팅 테이블을 관리해야함, Full Mesh 형태라 유지보수의 어려움

VPC Transit Gateway

중간에 일종의 라우팅 허브를 두고 VPC를 연결하도록 하면 어떨까? (Hub and Spoke)

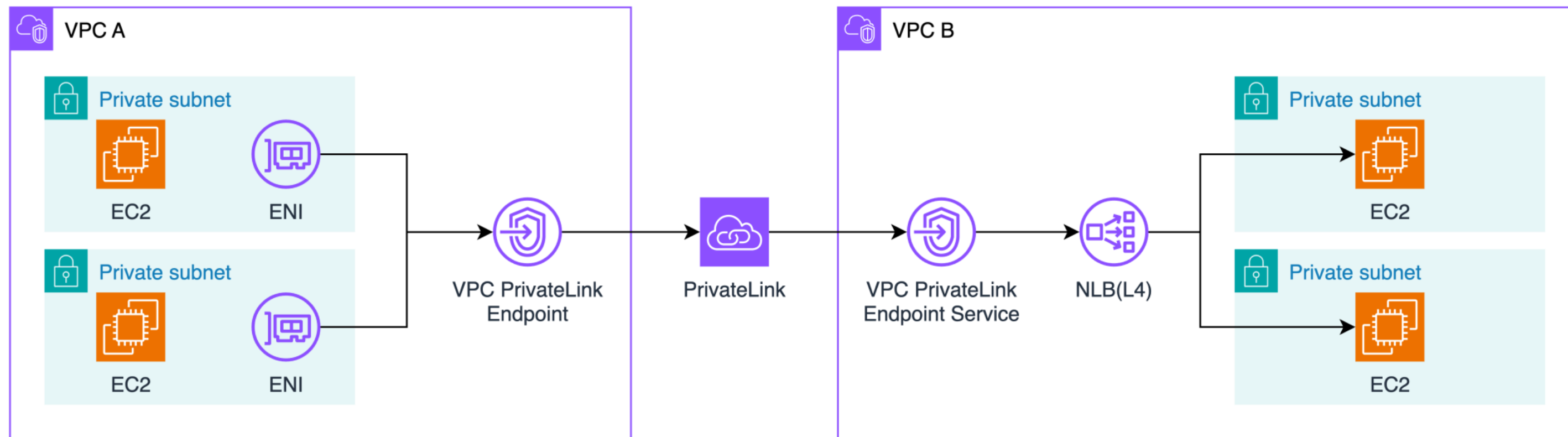


by Kim Jun Young (@yulmwu)

중앙에 Transit Gateway를 두고, 간단한 라우팅 테이블 설정으로 VPC 간 통신 가능(라우팅 허브 거침)

VPC PrivateLink

" 그런데 나는 VPC의 특정 서비스만 다른 VPC로 노출하고 싶은데? " (NACL 등만 가지고는 어려움)



by Kim Jun Young (@yulmwu)

서비스 제공자(프로바이더)에서 전용 NLB를 만들고(Endpoint Service) 컨슈머에선 엔드포인트와 전용 ENI 연결

⇒ 이 기술을 통틀어 PrivateLink라 칭함, 내부적으로 AWS Backbone을 통해 통신

VPC Peering, TGW, PrivateLink



더 알고싶다면? ⇒ <https://velog.io/@yulmwu/aws-vpc-peering-transit-privatelink>

VPC Lattice

" 다른 계정/리전의 VPC 간 통신 + 마이크로 서비스 네트워크 통합이 필요한데? "

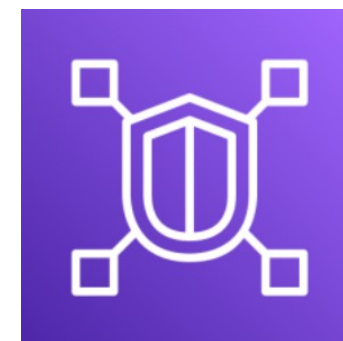
VPC Peering: 두 VPC가 직접 연결되는 Point to Point 방식, 유지보수의 어려움

Transit Gateway: 네트워크 계층으로 애플리케이션(서비스) 단위의 라우팅은 아님

PrivateLink: 서비스마다 NLB 연결 필요, 대규모 서비스에선 사용하기 어려움

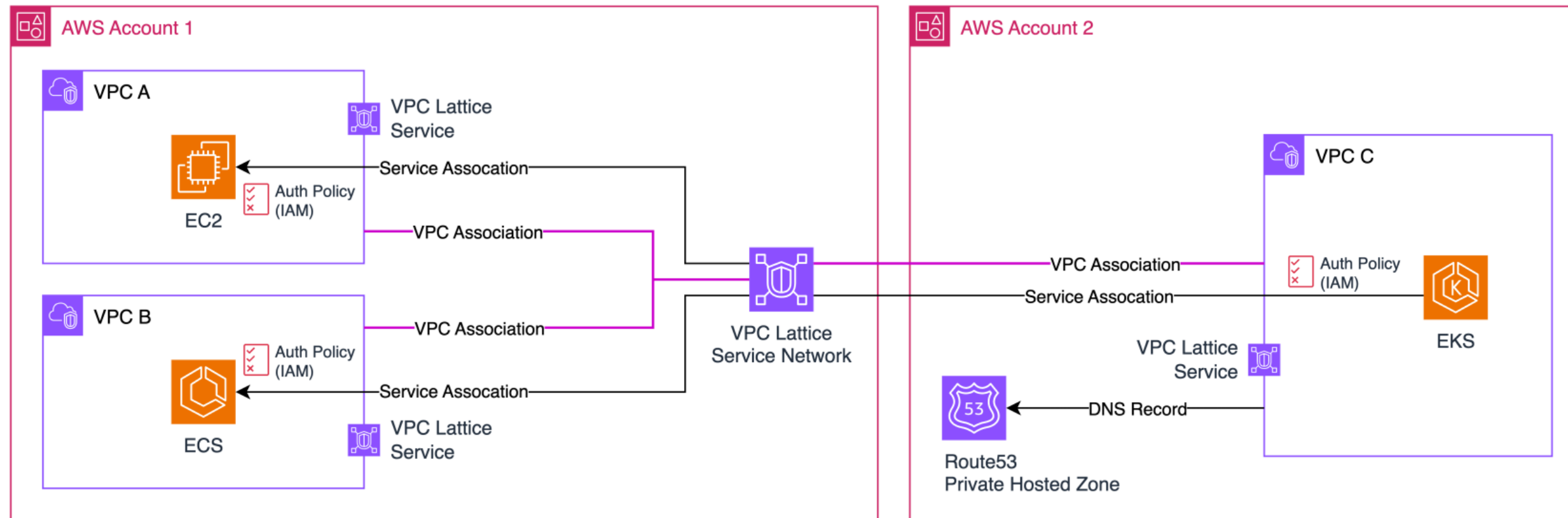
결정적으로 L3/L4 계층에서 동작하는 애들이라 애플리케이션(서비스) 단위의 라우팅이나 유지보수가 어려움

" VPC Lattice "



VPC Lattice

⇒ 분산된 여러 VPC나 계정 간 서비스(애플리케이션) 네트워킹/라우팅을 위해 설계된 서비스
L4/L7, HTTP/HTTPS, gRPC, TLS, TCP/UDP 등 다양한 프로토콜 지원



by Kim Jun Young (@yulmwu)

추상화된 형태는 Hub and Spoke 처럼 보일 순 있지만 내부적으로 Service Mesh 형태로 동작 + 사이드카 프록시 없이 동작

VPC Lattice

Internet Facing ALB/ELB와는 다르게 인터넷 노출 목적이 아닌 내부 네트워킹이 목적

때문에 필요 시 앞단에 로드밸런서나 API Gateway 등의 서비스 필요

Service: 실제 서비스(애플리케이션)를 대표하는 단위의 리소스, 타겟 그룹과 연결됨

Target Group: ELB의 TG와 마찬가지로 실제 트래픽을 받는 컴퓨팅 서비스의 그룹

Service Network: 여러 서비스와 VPC를 논리적으로 묶고 액세스 제어, VPC 연결, 모니터링 등 중앙에서 제공

Auth Policy: 서비스나 서비스 네트워크 단위의 액세스 제어 권한, 모두 허용이나 IAM 기반(Zero Trust)

Observability: Service Network 중앙에서 모니터링(CloudWatch, Access Logs 등)

근데 쿠버네티스 설명하다가 갑자기 왜 이걸 설명함???



VPC Lattice

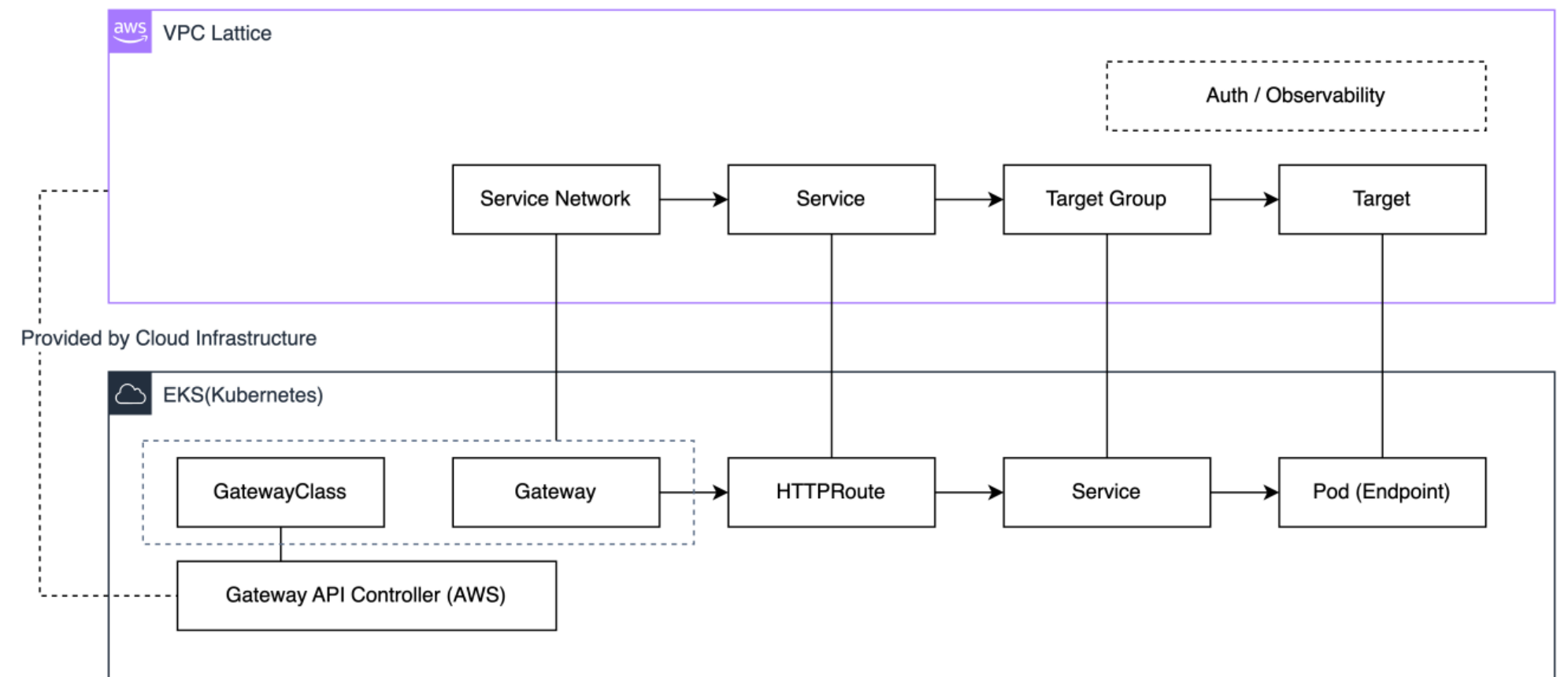
사실 K8s Gateway API엔 ALB/NLB를 붙일 순 있음, 그런데 만약 대규모 서비스라 이런 상황이 있다면?

- 다수의 계정/VPC와 다수의 클러스터를 가짐 (가용성 등)
- 서비스 간 통신에 IAM과 같은 일관된 AuthN/Z가 필요
- EC2, ECS, EKS, 람다 등의 다양한 컴퓨팅 서비스가 있는 경우
- 중앙에서 모니터링 등이 필요한 경우

⇒ 이거 완전 Lattice 쓰라는거잖아??

심지어 Gateway API, Lattice
추상화 구조가 비슷하잖아??

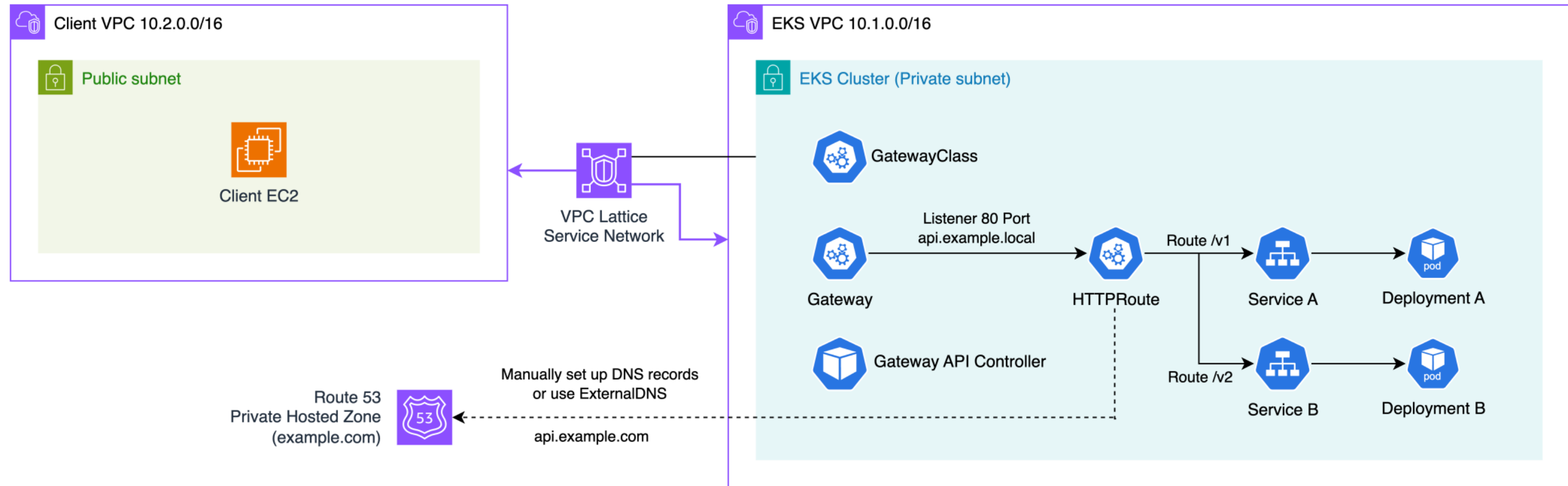
(100% 똑같은건 아님)



03

Example Demo

Example Demo



by Kim Jun Young (@yulmwu)

EKS 클러스터는 eksctl로, 간단한 웹 서버(eks-vpc)에 다른 VPC의 Client EC2로 접속해보는 예제

+ Route 53 Private Hosted Zone을 통한 내부 도메인 설정후 접속

EKS Cluster

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: eks-demo
  region: ap-northeast-2
  version: "1.33" (표준 지원 버전으로)
vpc:
  cidr: 10.1.0.0/16
  nat:
    gateway: Single # NAT Gateway
managedNodeGroups:
  - name: ng-1
    instanceType: t3.medium
    desiredCapacity: 2
    privateNetworking: true
    iam:
      (Private Subnet)
      withAddonPolicies:
        ebs: true
addons:
  - name: vpc-cni
  - name: kube-proxy
  - name: coredns
```

```
eksctl create cluster -f cluster.yaml
aws eks update-kubeconfig --name eks-demo --region ap-northeast-2
```

클러스터 (1) 정보

클러스터 필터링

클러스터 이름

상태

Kubernetes 버전

지원 기간

정책 업그레이드

생성됨

공급자

eks-demo

활성

1.33

2026년 7월 29일까지 표준 지원

확장됨

23분 전

EKS

인스턴스 (2) 정보

인스턴스를 속성 또는 (case-sensitive) 태그로 찾기

모든 상태

최종 업데이트 날짜
less than a minute 전

연결

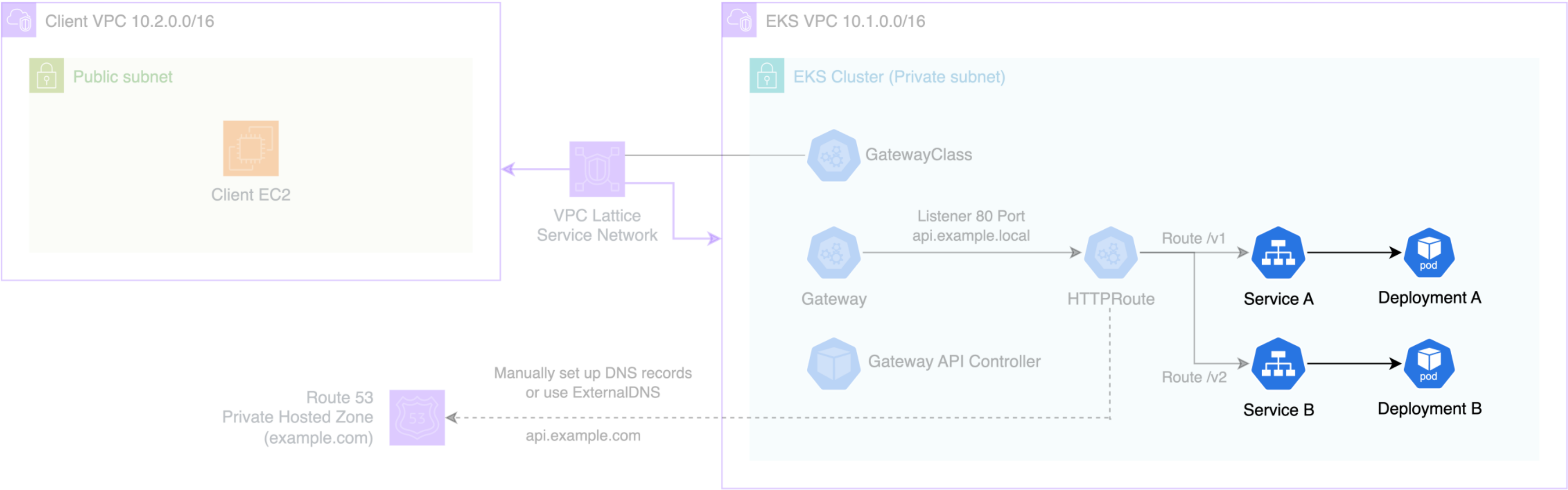
인스턴스 상태

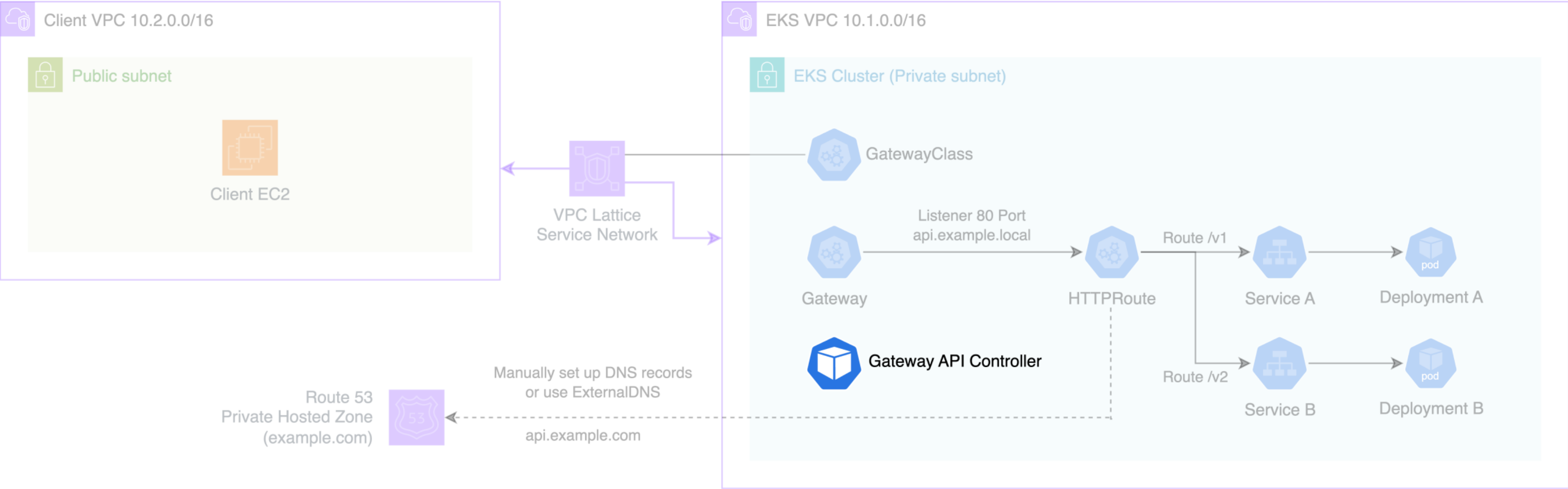
작업

인스턴스 시작

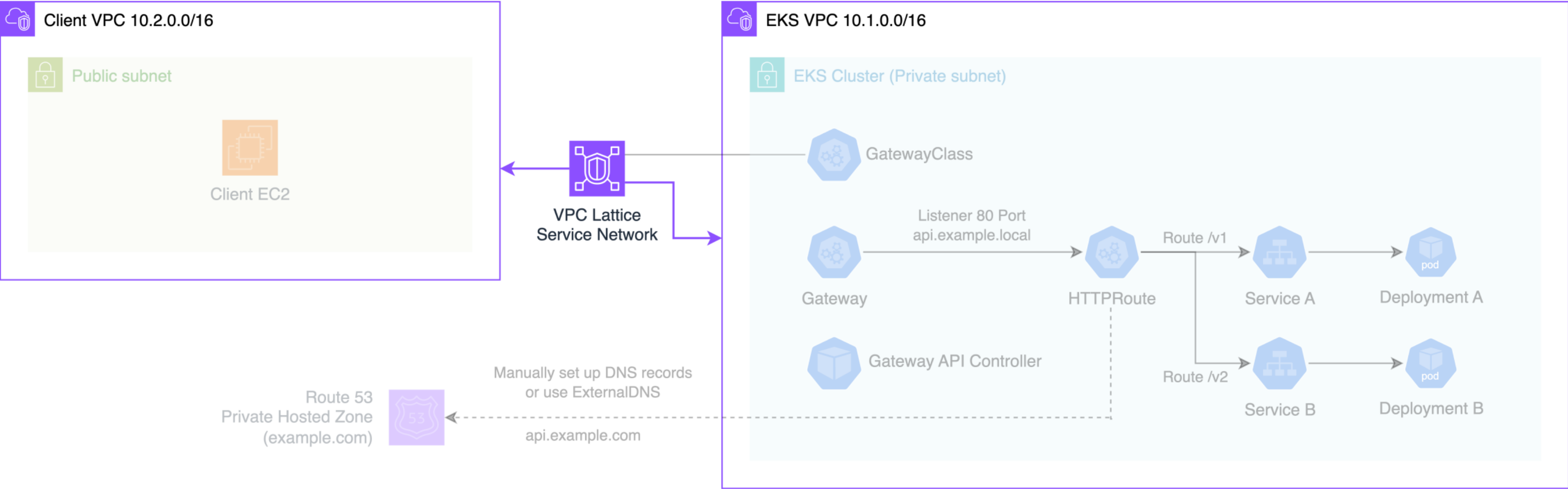
☐	Name	인스턴스 ID	인스턴스 상태	인스턴스 유형	상태 검사	경보 상태	가용 영역	퍼블릭 IPv4 DNS
☐	eks-demo-ng-1-Node	i-0f716243d520c866f	실행 중	t3.medium	3/3개 검사 통과	경보 보기 +	ap-northeast-2d	-
☐	eks-demo-ng-1-Node	i-0724c1cc7478c0e1d	실행 중	t3.medium	3/3개 검사 통과	경보 보기 +	ap-northeast-2a	-

eksctl 딸깍 (상남자 특: Terraform 안씀)
(사실 eksctl도 CloudFormation 기반이긴 함)

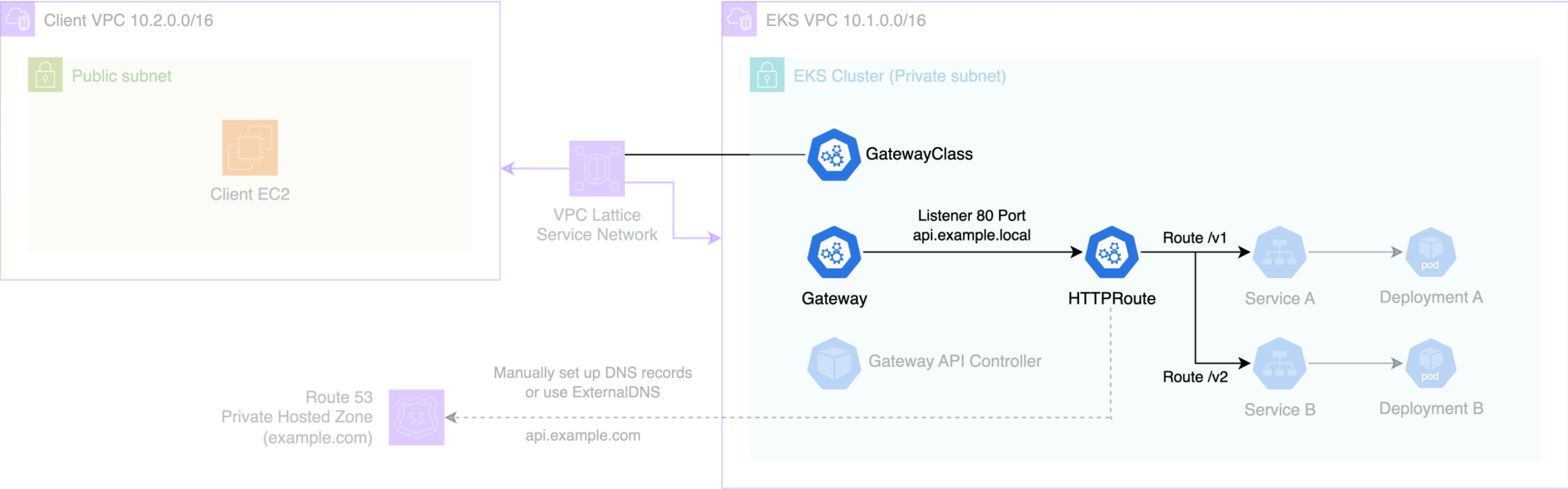


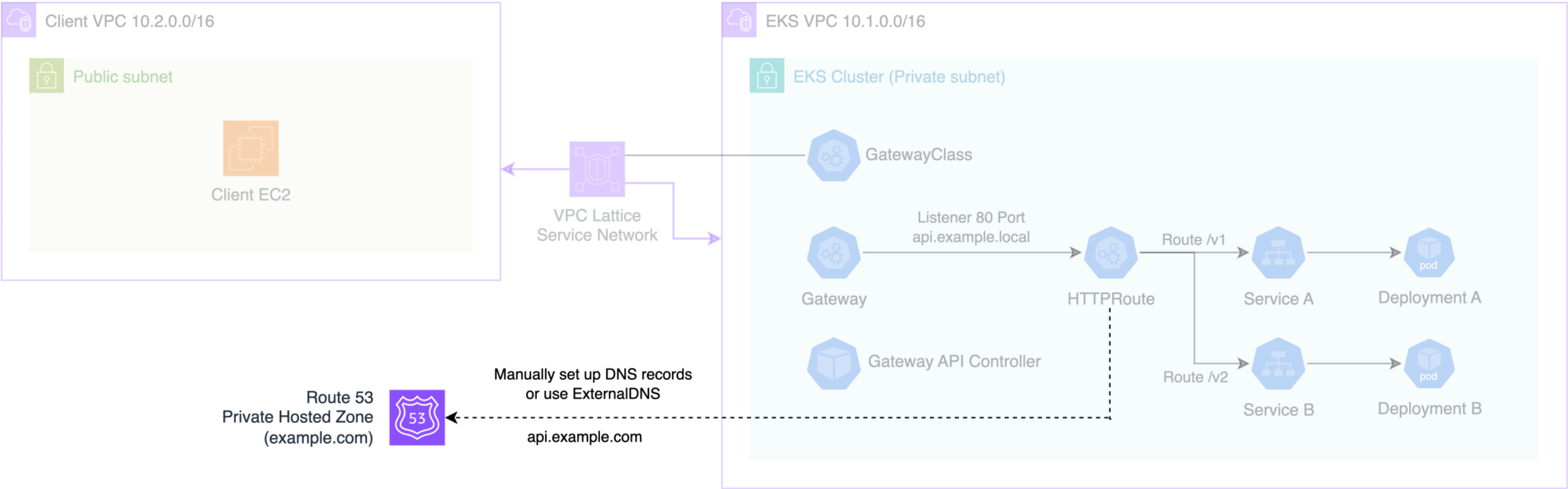


by Kim Jun Young (@yulmwu)

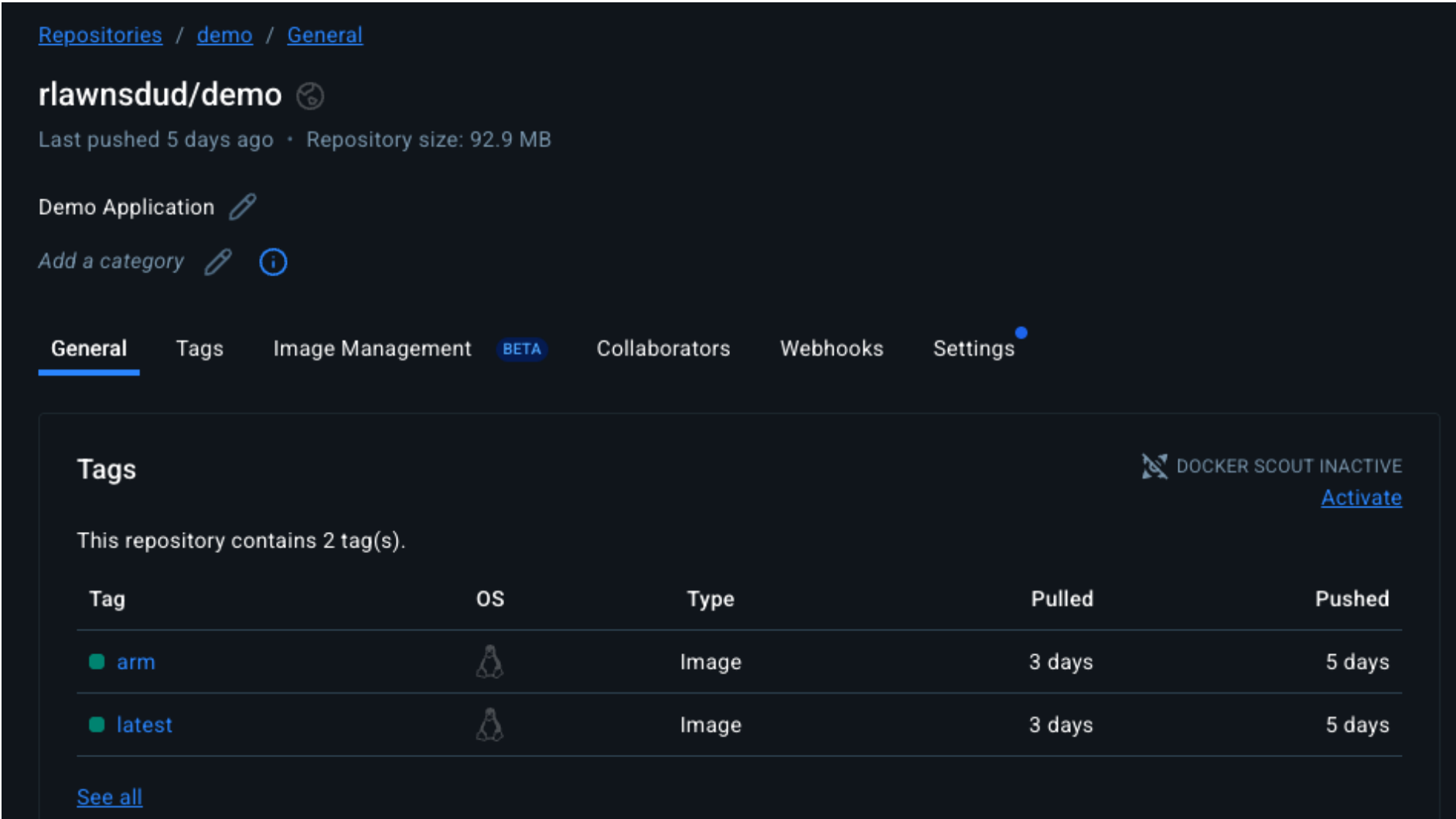


by Kim Jun Young (@yulmwu)





Deployment and Service



컨테이너 이미지는 간단하게 제작한 데모 앱 사용
⇒ <https://github.com/yulmwu/example-app>

Deployment and Service

```
containers:
- name: demo-app-a
  image: rlawnsdud/demo:latest
  imagePullPolicy: Always
  ports:
    - containerPort: 3000
  env:
    - name: HOST
      value: "0.0.0.0"
    - name: PORT
      value: "3000"
    - name: POD
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: ROUTE
      value: "v1"
    - name: GLOBAL_PREFIX
      value: "/v1"
```

같은 Deployment B를 또 만들어줌 (/v1, /v2)

```
readinessProbe:
  httpGet:
    path: /v1/health
    port: 3000
  initialDelaySeconds: 5
  periodSeconds: 10
livenessProbe:
  httpGet:
    path: /v1/health
    port: 3000
  initialDelaySeconds: 10
  periodSeconds: 20
```

쿠버네티스 파드 Health Check를 위한
readinessProbe, livenessProbe 추가

(Lattice TG Health Check랑 무관)

Deployment and Service

```
apiVersion: v1
kind: Service
metadata:
  name: demo-app-a
spec:
  selector:
    app: demo-app-a
  ports:
    - name: http
      port: 80
      targetPort: 3000
  type: ClusterIP
```

Gateway API Route 리소스에 연결을 하려면 ClusterIP를 포함한 서비스 필요

Deployment and Service

```
apiVersion: application-networking.k8s.aws/v1alpha1
kind: TargetGroupPolicy
metadata:
  name: tg-policy-demo-app-a
spec:
  targetRef:
    group: ""
    kind: Service
    name: demo-app-a
  protocol: HTTP
  protocolVersion: HTTP1
  healthCheck:
    enabled: true
    path: "/v1/health"
    port: 3000
    protocol: HTTP
    protocolVersion: HTTP1
    intervalSeconds: 10
    timeoutSeconds: 5
    healthyThresholdCount: 3
    unhealthyThresholdCount: 3
```

추가적으로 Lattice 타겟 그룹 Health Check를 위한 TargetGroupPolicy 적용

Deployment and Service

K9s(모니터링 도구)로 잘 만들어졌나 확인

pods(default)[4]

NAME↑	PF	READY	STATUS	RESTARTS	CPU	%CPU/R	%CPU/L	MEM	%MEM/R	%MEM/L	IP	NODE
demo-app-a-6479f8d7df-74lq6	●	1/1	Running	0	5	n/a	n/a	46	n/a	n/a	10.1.149.251	ip-10-1-152-201.ap-n
demo-app-a-6479f8d7df-wdj8x	●	1/1	Running	0	5	n/a	n/a	47	n/a	n/a	10.1.177.99	ip-10-1-181-208.ap-n
demo-app-b-d8798c8bd-9cd54	●	1/1	Running	0	92	n/a	n/a	47	n/a	n/a	10.1.187.223	ip-10-1-181-208.ap-n
demo-app-b-d8798c8bd-982xf	●	1/1	Running	0	107	n/a	n/a	49	n/a	n/a	10.1.156.125	ip-10-1-152-201.ap-n

🐙 > SVC

services(default)[3]

NAME↑	TYPE	CLUSTER-IP	EXTERNAL-IP	PORTS	AGE
demo-app-a	ClusterIP	172.20.85.152		http:80→0	110s
demo-app-b	ClusterIP	172.20.110.189		http:80→0	109s
kubernetes	ClusterIP	172.20.0.1		https:443→0	21m

Deployment and Service

K9s(모니터링 도구)로 잘 만들어졌나 확인

pods(default)[4]

NAME↑	PF	READY	STATUS	RESTARTS	CPU	%CPU/R	%CPU/L	MEM	%MEM/R	%MEM/L	IP	NODE
demo-app-a-6479f8d7df-74lq6	●	1/1	Running	0	5	n/a	n/a	46	n/a	n/a	10.1.149.251	ip-10-1-152-201.ap-n
demo-app-a-6479f8d7df-wdj8x	●	1/1	Running	0	5	n/a	n/a	47	n/a	n/a	10.1.177.99	ip-10-1-181-208.ap-n
demo-app-b-d8798c8bd-9cd54	●	1/1	Running	0	92	n/a	n/a	47	n/a	n/a	10.1.187.223	ip-10-1-181-208.ap-n
demo-app-b-d8798c8bd-982xf	●	1/1	Running	0	107	n/a	n/a	49	n/a	n/a	10.1.156.125	ip-10-1-152-201.ap-n

> SVC

services(default)[3]

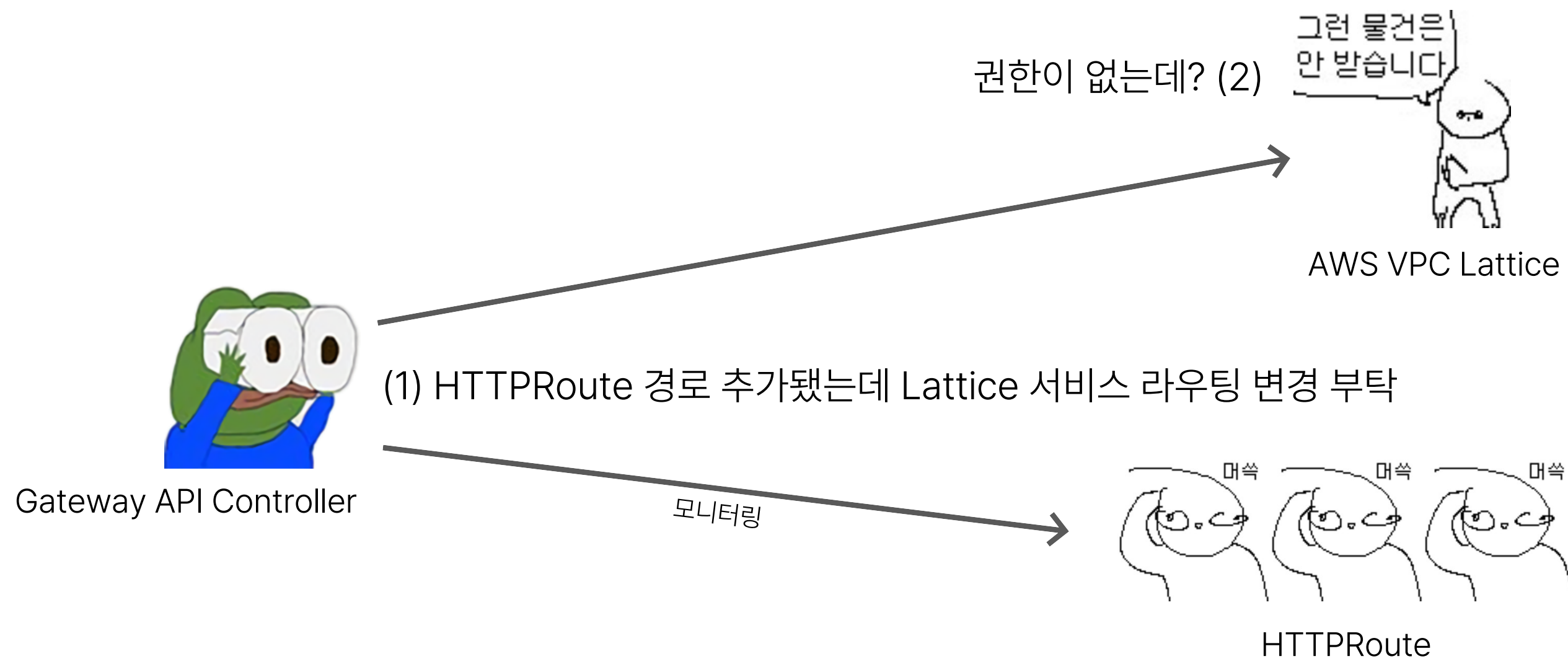
NAME↑	TYPE	CLUSTER-IP	EXTERNAL-IP	PORTS	AGE
demo-app-a	ClusterIP	172.20.85.152		http:80→0	110s
demo-app-b	ClusterIP	172.20.110.189		http:80→0	109s
kubernetes	ClusterIP	172.20.0.1		https:443→0	21m

/app # curl localhost:30000/env?select=POD,ROUTE
{ "POD": "demo-app-a-6479f8d7df-74lq6", "ROUTE": "v1" }/app #
/app # █

Pod Identity(IAM)

EKS 내 쿠버네티스 자체와 AWS 서비스는 기본적으로 권한이 없음

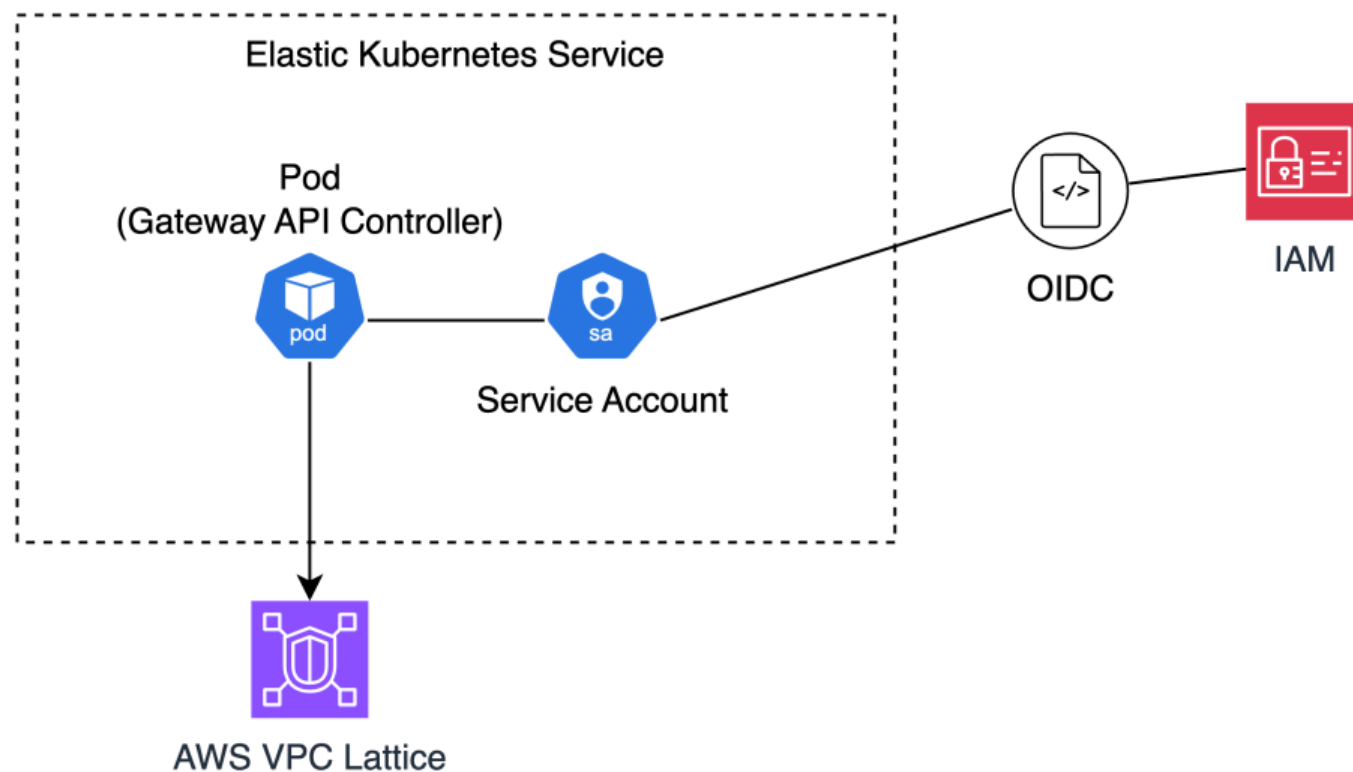
⇒ Gateway API Controller가 AWS VPC Lattice를 컨트롤 할 수 없음



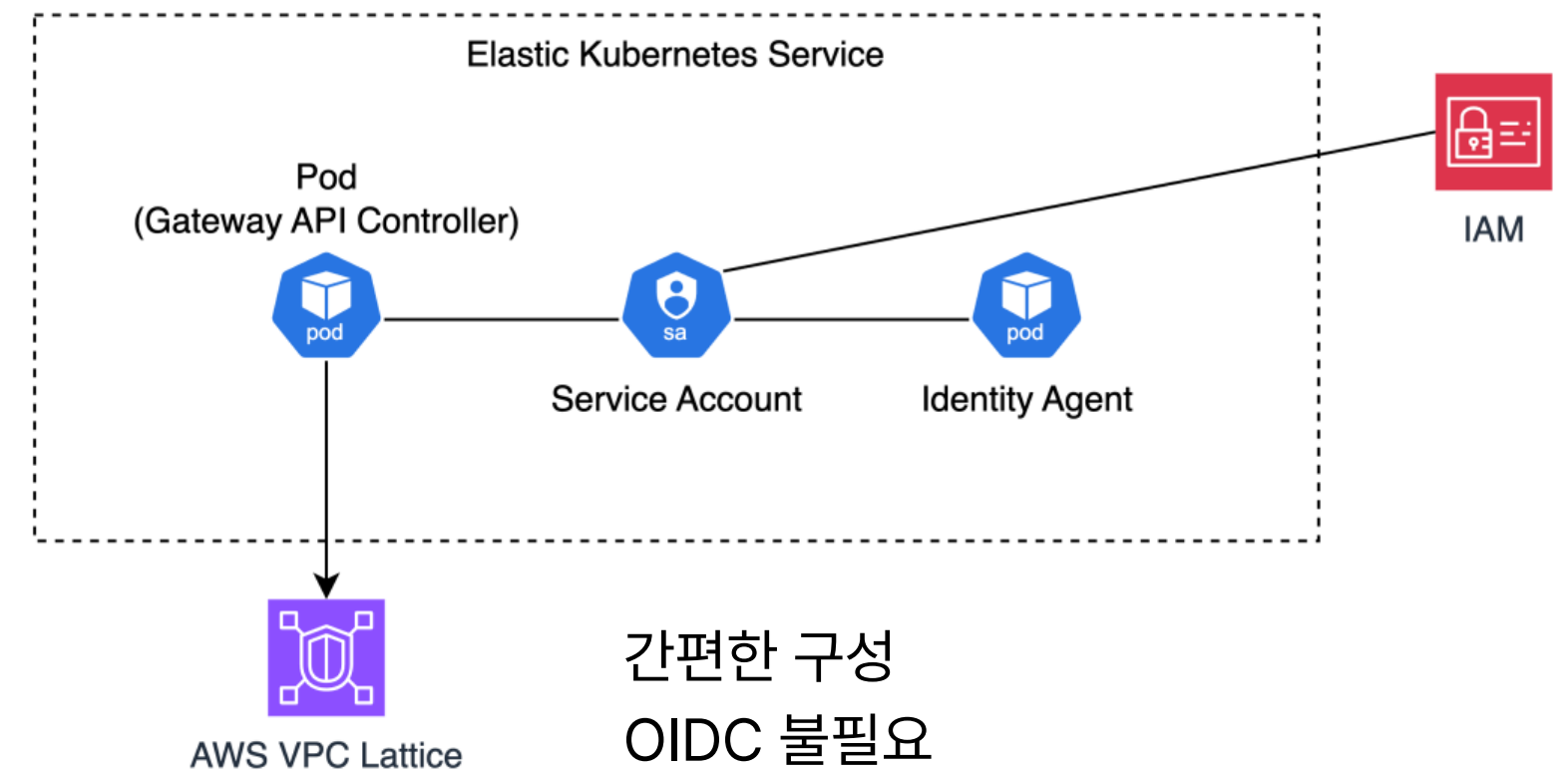
Pod Identity(IAM)

쿠버네티스 리소스와 AWS 리소스 간 권한을 부여할 수 있는 방법: IRSA, Pod Identity

IAM Roles for Service Accounts (IRSA)



Pod Identity



간편한 구성
OIDC 불필요
여러 클러스터에서 동일한 Role 재사용 가능



Pod Identity(IAM)

```
aws iam create-policy --policy-name VPCLatticeControllerIAMPolicy \
  --policy-document file:///recommended-inline-policy.json

export VPCLatticeControllerIAMPolicyArn=$(aws iam list-policies \
  --query 'Policies[?PolicyName==`VPCLatticeControllerIAMPolicy`].Arn' --output text)

# 컨트롤러 네임스페이스 & 서비스 어카운트
kubectl apply -f https://raw.githubusercontent.com/aws/aws-application-networking-k8s/main/example/gateway-api-controller-service-account.yaml

cat > gateway-api-controller-service-account.yaml <<'EOF'
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gateway-api-controller
  namespace: aws-application-networking-system
EOF
kubectl apply -f gateway-api-controller-service-account.yaml

# 파드 Identity 애드온 설치
aws eks create-addon --cluster-name ${CLUSTER_NAME} \
  --addon-name eks-pod-identity-agent --addon-version v1.0.0-eksbuild.1

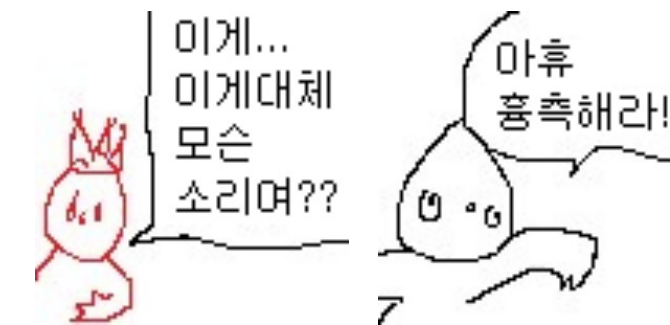
# IAM Role 생성 (Trust Relationship 설정)
cat > eks-pod-identity-trust-relationship.json <<'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
      "Effect": "Allow",
      "Principal": { "Service": "pods.eks.amazonaws.com" },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }
  ]
}
EOF
```

```
aws iam create-role \
  --role-name VPCLatticeControllerIAMRole \
  --assume-role-policy-document file:///eks-pod-identity-trust-relationship.json \
  --description "IAM Role for AWS Gateway API Controller for VPC Lattice"

aws iam attach-role-policy \
  --role-name VPCLatticeControllerIAMRole \
  --policy-arn $VPCLatticeControllerIAMPolicyArn

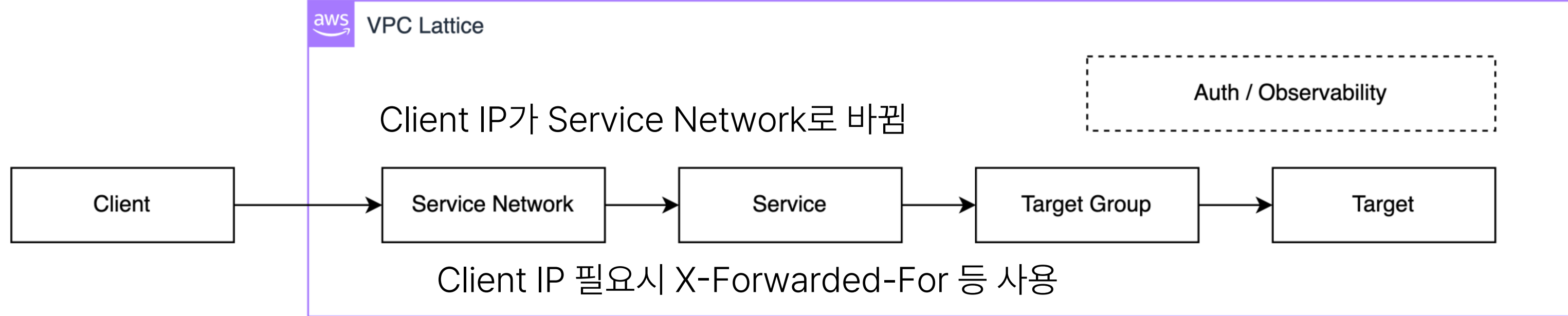
export VPCLatticeControllerIAMRoleArn=$(aws iam list-roles \
  --query 'Roles[?RoleName==`VPCLatticeControllerIAMRole`].Arn' --output text)

# 서비스 어카운트에 IAM Role 연결
aws eks create-pod-identity-association \
  --cluster-name ${CLUSTER_NAME} \
  --role-arn $VPCLatticeControllerIAMRoleArn \
  --namespace aws-application-networking-system \
  --service-account gateway-api-controller
```



상남자 특) 3줄 이상 안읽음

SG Lattice Prefix List



Service Network를 통해 트래픽을 받으려면 해당 트래픽을 보안 그룹(SG)에서 허용해야 함

SG Lattice Prefix List

Lattice CIDR 대신 Prefix List 제공(IP 대역 모아둔 것)

인바운드 규칙 정보

보안 그룹 규칙 ID

sgr-0d4678bfa740b66e7

sgr-088ef768959067ace

유형 정보

모든 트래픽

모든 트래픽

프로토콜 정보

전체

전체

포트 범위 정보

전체

전체

소스 정보

사용자...

사용자...

설명 - 선택 사항 정보

Q

pl-0cd842b438fee3b4b X

Q

pl-06da045b37607c8a5 X

삭제

삭제

노드나 클러스터 보안 그룹에 추가

Gateway API Controller

Helm(패키지 매니저)을 통해 Gateway API Controller 설치 & SA 연결

```
# Gateway API CRD 설치
kubectl kustomize "github.com/kubernetes-sigs/gateway-api/config/crd?ref=v1.1.0" | kubectl apply -f -

# ECR 로그인
aws ecr-public get-login-password --region us-east-1 | helm registry login --username AWS --password-stdin public.ecr.aws

# Helm Gateway API Controller 설치 (SA는 기존에 있는걸로)
helm install gateway-api-controller \
  oci://public.ecr.aws/aws-application-networking-k8s/aws-gateway-controller-chart \
  --version=v1.1.4 \
  --set=serviceAccount.create=false \
  --set=serviceAccount.name=gateway-api-controller \
  --namespace aws-application-networking-system \
  --set=log.level=info
```

Gateway API Controller

```

NAME↑
gateway-api-controller-aws-gateway-controller-chart-7f8796m5f2f ● 1/1 Running 0 1 2 1 13 20 10
gateway-api-controller-aws-gateway-controller-chart-7f8796zd826 ● 1/1 Running 0 3 6 3 18 28 14

```

Gateway API Controller가 Gateway/Route CRD 감지 후 VPC Lattice에 매핑 해줌

Lattice Service Network

Lattice 서비스, 타겟 그룹은 Gateway API Controller가 알아서 해주니 SN만 만들기

VPC 연결 - 권장 정보

지금 또는 생성 후 언제든지 VPC를 서비스 네트워크에 연결할 수 있습니다. 연결 후 VPC 내의 서비스는 서비스 네트워크의 서비스를 호출할 수 있습니다. 서비스 네트워크에 액세스할 수 있는 모든 VPC 소유자는 자신의 VPC를 해당 서비스 네트워크에 연결할 수 있습니다.

▼ VPC 연결: client-vpc

제거

VPC

연결된 VPC 내의 서비스는 서비스 네트워크의 서비스를 호출할 수 있습니다.

client-vpc
vpc-05170ca0393788fdc
ipv4: 10.2.0.0/16

↻

보안 그룹

VPC에서 서비스 네트워크로의 인바운드 트래픽을 제어할 보안 그룹을 선택합니다.

최대 5개의 보안 그룹을 선택합니다.

↻

default
sg-02e44945c5f54e697
vpc-05170ca0393788fdc

새 보안 그룹 생성

▼ VPC 연결: eksctl-eks-demo-cluster/VPC

제거

VPC

연결된 VPC 내의 서비스는 서비스 네트워크의 서비스를 호출할 수 있습니다.

eksctl-eks-demo-cluster/VPC
vpc-0f66037f5214bc205
ipv4: 10.1.0.0/16

↻

보안 그룹

VPC에서 서비스 네트워크로의 인바운드 트래픽을 제어할 보안 그룹을 선택합니다.

최대 5개의 보안 그룹을 선택합니다.

↻

eksctl-eks-demo-cluster-ClusterSharedNodeSecurityGroup-SOtKq76UBqQt
sg-069f2fb1061841653
vpc-0f66037f5214bc205

eksctl-eks-demo-cluster-ControlPlaneSecurityGroup-cypozlQWtqNk
sg-00f99e55f5c260733
vpc-0f66037f5214bc205

eks-cluster-sg-eks-demo-554178781
sg-06a3c2d3213b00fc3
vpc-0f66037f5214bc205

default
sg-07d810a41b8f8357b
vpc-0f66037f5214bc205

새 보안 그룹 생성

VPC 연결 추가

최대 498개의 VPC 연결을 추가할 수 있습니다.

▶ VPC 연결 태그 - 선택 사항

VPC 연결 (Client VPC, EKS VPC)

Lattice Service Network

네트워크 액세스 정보

서비스 네트워크에 대한 클라이언트 액세스를 구성하고 관리합니다.

인증 유형

네트워크에 대한 클라이언트 액세스를 관리하는 인증 및 권한 부여를 지정합니다.

☒ 없음

서비스 네트워크는 클라이언트 액세스를 인증하거나 승인하지 않습니다. 인증 정책이 있는 경우 비활성 상태입니다. 서비스 수준 정책이 액세스를 제한하지 않는 한 연결된 VPC 내의 리소스는 이 네트워크의 서비스에 액세스할 수 있습니다.

☐ AWS IAM

서비스 네트워크에 IAM 리소스 정책을 적용합니다. 이렇게 하면 관리자가 네트워크 내 서비스에 대해 인증을 적용하고 세부적인 권한을 작성할 수 있습니다.

▶ 인증 정책 - 선택 사항

인증 없음 = 모든 서비스 간 접속 허용

AWS IAM = Zero Trust, IAM 바탕으로 접근 설정 (권장)

Lattice Service Network

```
helm upgrade gateway-api-controller \  
  oci://public.ecr.aws/aws-application-networking-k8s/aws-gateway-controller-chart \  
  --version=v1.1.4 \  
  --reuse-values \  
  --namespace aws-application-networking-system \  
  --set=defaultServiceNetwork=demo-sn # Lattice 서비스 네트워크 이름
```

만든 Service Network 이름으로 기본값 변경

Gateway, HTTPRoute

```
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: amazon-vpc-lattice
spec:
  controllerName: application-networking.k8s.aws/gateway-api-controller
```

Helm을 통해 설치한 Gateway API Controller (Lattice)

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-sn
spec:
  gatewayClassName: amazon-vpc-lattice
  listeners:
    - name: http
      protocol: HTTP
      port: 80
      allowedRoutes:
        namespaces:
          from: All
        kinds:
          - kind: HTTPRoute
            group: gateway.networking.k8s.io
```

HTTP, 80 리스너

모든 네임스페이스 내 demo-sn 부모를 가진 HTTPRoute 리소스 자동 연결

Gateway, HTTPRoute

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: demo-http-route
spec:
  parentRefs:
    - name: demo-sn      부모 Gateway 리소스 지정
      sectionName: http
  rules:
    - matches:           엔드포인트(경로) 라우팅 지정 = 리스너 규칙
      - path:
          type: PathPrefix
          value: /v1
      backendRefs:        타겟 그룹(백엔드 애플리케이션/서비스)
        - kind: Service
          name: demo-app-a
          port: 80
    - matches:
      - path:
          type: PathPrefix
          value: /v2
      backendRefs:
        - kind: Service
          name: demo-app-b
          port: 80
```

Gateway, HTTPRoute

▼ 리스너: demo-http-route-default-80-http

리스너 이름

demo-http-route-default-80-http

프로토콜 : 포트

HTTP : 80

기본 작업

고정 응답 반환: 상태 코드 404

태그

1개 태그

ARN

arn:aws:vpc-lattice:ap-northeast-2:986129558966:service/svc-03f25c24e24e7f446/listener/listener-0733577171f536e5f

리스너 규칙 (3)

Name	조건(모두 일치)	작업	우선 순위	태그	ARN
k8s-1758088057-rule-1	• 경로 (접두사): /v1 (대/소문자 구분)	다음으로 전달: k8s-default-demo-app-a-bbeucnuqwl: 1 (100%)	1	1개 태그	ARN
k8s-1758088057-rule-2	• 경로 (접두사): /v2 (대/소문자 구분)	다음으로 전달: k8s-default-demo-app-b-rpzzcklpwe: 1 (100%)	2	1개 태그	ARN
리스너 기본 작업	다른 규칙이 적용되지 않는 경우	고정 응답 반환: 상태 코드 404	기본/마지막	-	-

Gateway, HTTPRoute

HTTPRoute (Lattice Service 리소스) DNS 확인

```
/Users/workspace4/k8s_gateway_lattice_demo [git::main *] [ky0422@MacBookPro] [14:49]
•> kubectl get httproute demo-http-route -o jsonpath='{.metadata.annotations.application-networking\.k8s\.aws/lattice-assigned-domain-name}'
demo-http-route-default-03f25c24e24e7f446.7d67968.vpc-lattice-svcs.ap-northeast-2.on.aws%

/Users/workspace4/k8s_gateway_lattice_demo [git::main *] [ky0422@MacBookPro] [14:50]
○> █
```

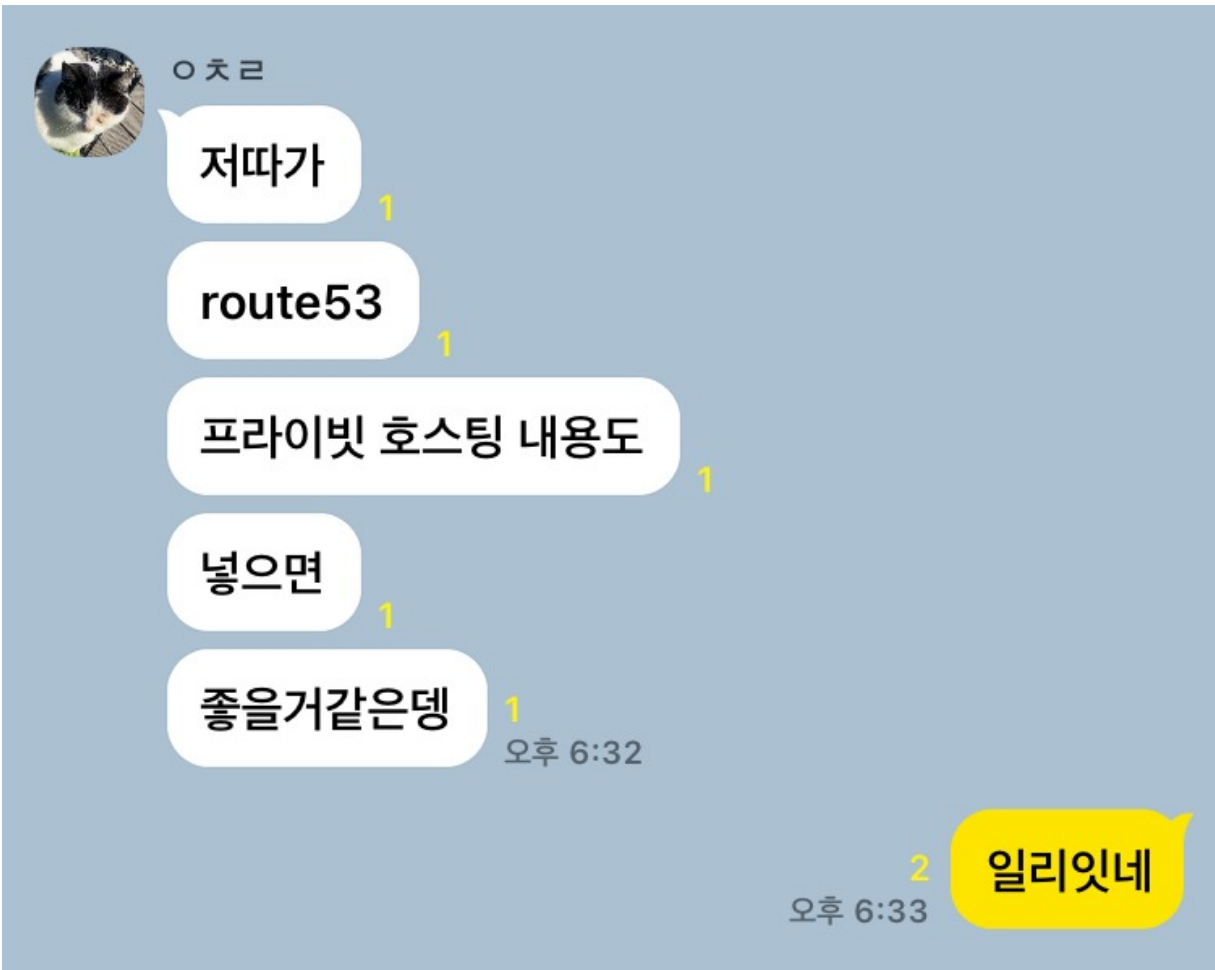
⇒ 이 DNS로 클라이언트에서 사용해도 무방함

```
[ec2-user@ip-10-2-8-65 ~]$ curl demo-http-route-default-03f25c24e24e7f446.7d67968.vpc-lattice-svcs.ap-northeast-2.on.aws/v1/env?select=POD,ROUTE && echo
{"POD":"demo-app-a-77c7bbc4b6-sb8rt","ROUTE":"v1"}
[ec2-user@ip-10-2-8-65 ~]$ █
```

" 발표 분량 못채우겠는데? "



Route 53



Route 53

AWS에서 제공하는 DNS 서비스

호스팅 영역(특정 도메인 이름에 대한 DNS 레코드들을 저장하는 곳)에 두가지 종류가 있음

Public Hosted Zone

돈주고 도메인 구매 & 연결
인터넷을 통해 접근하는 일반적인 DNS

Privated Host Zone

특정 VPC 내부 전용 DNS
해당 VPC 내에서만 레코드 해석 가능



Route 53 Private Hosted Zone

긴 HTTPRoute(Lattice Service)를 PHZ를 만들고 CNAME(별칭)으로 레코드 추가

⇒ PHZ api.example.com → CNAME(xxx.yyy.vpc-lattice-svcs.ap-northeast-2.on.aws)

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: demo-http-route
spec:
  parentRefs:
    - name: demo-sn
      sectionName: http
  hostnames:
    - api.example.com
```

HTTPRoute 리소스 hostnames 수정(해야 제대로 라우팅 가능)

Route 53 Private Hosted Zone

서비스 (1)

VPC Lattice 서비스는 연결된 서비스 네트워크에서 수신하는 네트워크 트래픽에 대한 액세스, 라우팅 및 모니터링을 정의합니다.

속성 또는 태그로 리소스 찾기

< 1 > | ⚙

▼ 도메인 이름	▼ 사용자 지정 도메인 이름	▼ 사용자 지정 인
 demo-http-route-default-0184fd20205eed05f.7d67968.vpc-lattice-svcs.ap-northeast-2.on.aws	 api.example.com	없음

DNS CNAME 추가 시 api.example.com으로도 라우팅이 될 수 있게 사용자 지정

Route 53 Private Hosted Zone

호스팅 영역 생성

호스팅 영역 구성

호스팅 영역은 example.com 같은 도메인과 관련 하위 도메인에 대한 트래픽을 라우팅하는 방식에 대한 정보를 포함하는 컨테이너입니다.

도메인 이름

PHZ 도메인이기 때문에 원하는 도메인 가능

트래픽을 라우팅할 도메인의 이름입니다.

example.com

유효한 문자: a-z, 0-9 및 ! " # \$ % & ' () * + , - / : ; < = > ? @ [\] ^ _ ` { | } . ~

설명 - 선택 사항

이 값을 사용하면 이름이 동일한 호스팅 영역을 구별할 수 있습니다.

호스팅 영역이 사용되는 경우...

설명은 최대 256자입니다. 0/256

유형

☐ 퍼블릭 호스팅 영역

퍼블릭 호스팅 영역은 인터넷에서 트래픽을 라우팅하는 방식을 결정합니다.

☒ 프라이빗 호스팅 영역

프라이빗 호스팅 영역은 Amazon VPC 내에서 트래픽을 라우팅하는 방식을 결정합니다.

호스팅 영역과 연결할 VPC

이 호스팅 영역을 사용하여 하나 이상의 VPC에 대한 DNS 쿼리를 확인하려면 VPC를 선택합니다. 다른 AWS 계정을 사용하여 VPC를 생성할 때 VPC를 호스팅 영역과 연결하려면 AWS CLI와 같은 프로그래밍 방식을 사용해야 합니다.

프라이빗 호스팅 영역과 연결하는 각 VPC에 대해 Amazon VPC 설정 enableDnsHostnames 및 enableDnsSupport 을(를) true로 설정해야 합니다.

리전

정보

아시아 태평양(서울)

VPC ID

정보

vpc-05170ca0393788fdc

VPC 제거

아시아 태평양(서울)

vpc-0f66037f5214bc205

VPC 제거

VPC 추가

Route 53 Private Hosted Zone

레코드 생성

정보

빠른 레코드 생성

마법사로 전환

▼ 레코드 1

삭제

레코드 이름

정보

api

.example.com

레코드 유형

정보

CNAME – 다른 도메인 이름과 일부 AWS 리소스로 트래픽 라우팅

루트 도메인에 대한 레코드를 생성하려면 비워 둡니다.

☐ 별칭

값

정보

demo-http-route-default-03f25c24e24e7f446.7d67968.vpc-lattice-svcs.ap-northeast-2.on.aws

HTTPRoute(Lattice Service) DNS 주소

별도의 줄에 여러 값을 입력합니다.

TTL(초)

정보

300

1분

1시간

1일

라우팅 정책

정보

단순 라우팅

권장 값: 60~172,800(2일)

다른 레코드 추가

Testing

```
[ec2-user@ip-10-2-8-65 ~]$ dig api.example.com

; <<>> DiG 9.18.33 <<>> api.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29943
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;api.example.com.                IN      A

;; ANSWER SECTION:
api.example.com.                300     IN      CNAME   demo-http-route-default-0184fd20205eed05f.7d67968.vpc-lattice-svcs.ap-northeast-2.on.aws.
demo-http-route-default-0184fd20205eed05f.7d67968.vpc-lattice-svcs.ap-northeast-2.on.aws. 60 IN A 169.254.171.1

;; Query time: 0 msec
;; SERVER: 10.2.0.2#53(10.2.0.2) (UDP)
;; WHEN: Wed Sep 17 09:22:46 UTC 2025
;; MSG SIZE rcvd: 162
```

도메인 CNAME 정보 확인

```
[ec2-user@ip-10-2-8-65 ~]$ curl api.example.com/v2/env?select=POD,ROUTE
{"POD":"demo-app-b-9f77d7c99-qs57b","ROUTE":"v2"}[ec2-user@ip-10-2-8-65 ~]$
[ec2-user@ip-10-2-8-65 ~]$ curl api.example.com/v2/env?select=POD,ROUTE
{"POD":"demo-app-b-9f77d7c99-mr9xs","ROUTE":"v2"}[ec2-user@ip-10-2-8-65 ~]$
[ec2-user@ip-10-2-8-65 ~]$ curl api.example.com/v2/env?select=POD,ROUTE
{"POD":"demo-app-b-9f77d7c99-qs57b","ROUTE":"v2"}[ec2-user@ip-10-2-8-65 ~]$
[ec2-user@ip-10-2-8-65 ~]$ █
```

PHZ + 내부 Lattice Service 주소라 Private IP

HTTPRoute(Lattice Service)를 통한 로드밸런싱 + 경로 지정

04

(Lattice) Calculate Price

VPC Lattice Pricing, Lattice vs. ALB(Ingress) Comparison

EKS 요금 = EKS 컨트롤 플레인 요금(시간당 표준 지원 0.1\$, 확장 지원 0.6\$)
+ 워커노드(EC2) 비용 (Fargate 예외)
+ 네트워킹, 스토리지 비용
+ α



AWS 예상 청구서 요약 정보			
총 요금 및 결제 정보			
계정 ID 986129558966	결제 기간 정보 9월 1일 - 2025년 9월 30일	결제 상태 정보 ⌚ 대기 중	
서비스 제공업체 Amazon Web Services Korea LLC		합계 USD USD 11.69	
		예상 총합계:	<u>USD 11.69</u>
▶ 결제 정보 정보			



2025년 9월

- Amazon Elastic Container Service for Kubernetes US\$6.69
- EC2 - Other US\$2.17
- Amazon Elastic Compute Cloud - Compute US\$1.35
- Tax US\$1.06
- Amazon Virtual Private Cloud US\$0.28
- Others US\$0.15

Calculate Price (Lattice)

VPC Lattice 비용

- 서비스 별 연결 시간당 비용
- 데이터 처리량(송수신 합계)
- HTTP 요청당 요금(HTTP/HTTPS 리스너)
- TCP 연결당 요금(TLS 리스너)

서비스별 요금 항목	요금
시간당 요금	시간당 USD 0.0325
처리된 데이터 GB당 요금	GB당 USD 0.0325
HTTP 요청당 요금(HTTP/HTTPS 리스너만 해당) 또는 TCP 연결당 요금(TLS 리스너만 해당) - 상시 프리 티어 후	100만 요청/연결당 USD 0.13

(ap-northeast-2 (서울) 기준)

Calculate Price (Lattice)

1. 서비스 별 연결 시간당 비용 $\Rightarrow$ 1시간당 서비스 1개를 기준으로 0.0325\$

$$\text{1달 기준} \quad 0.0325\$ \times 730h = 24.37\$$$

Calculate Price (Lattice)

2. 데이터 처리량 (송수신 합계) $\Rightarrow$ GB당 0.0325\$

$$\text{트래픽 100GB (송수신 합계)} \quad 0.0325\$ \times 100GB = 3.25\$$$

Calculate Price (Lattice)

3. HTTP 요청당 요금(HTTP/HTTPS 리스너) $\Rightarrow$ 100만 건의 요청/연결 당 0.13\$
(첫 30만 건의 요청/연결은 무료)

$$300\text{만 건의 HTTP/HTTPS 요청} \quad 0.13\$ \times 2.7M = 0.351\$$$

Calculate Price (Lattice)

총합 (월)

$$24.37\$ + 3.25\$ + 0.35\$ = 27.97$$



Calculate Price (Lattice)



이게 저렴한건가 아님 비싼건가..
그냥 Ingress랑 ALB 로드밸런서가 더 낫지 않을까?

Lattice vs. ALB(Ingress) Comparison

Lattice 비용 =	1달 기준	$0.0325\$ \times 730h = 24.37\$$	$\Rightarrow 27.97\$$
	트래픽 100GB (송수신 합계)	$0.0325\$ \times 100GB = 3.25\$$	
	300만 건의 HTTP/HTTPS 요청	$0.13\$ \times 2.7M = 0.351\$$	

같은 조건으로 ALB(L4 Ingress)는?

Lattice vs. ALB(Ingress) Comparison

AWS ELB(ALB, NLB)는 LCU 단위로 요금을 계산 (항목을 넘는다면 LCU 추가됨, 가장 큰 항목을 기준으로 함)
+ 로드밸런서 가동 시간

$$\text{LCU} = \max \left(\begin{array}{l} 1 \text{ LCU} = \text{초당 신규 연결 25개} \\ 1 \text{ LCU} = \text{분당 활성 연결(TLS) 3,000개 (mTLS의 경우 1,500개)} \\ 1 \text{ LCU} = \text{시간당 처리 바이트 (EC2, IP는 1GB/h, 람다는 0.4GB/h)} \\ 1 \text{ LCU} = \text{초당 요청 1000개(= 규칙 평가)} \end{array} \right)$$



저걸 직접 계산한다고? (GPT 딸깍)

Lattice vs. ALB(Ingress) Comparison

최종 계산

AWS 리전에서 Application Load Balancer를 사용하는 경우:

- Application Load Balancer 시간(또는 부분 시간)당 USD 0.0225
- LCU 시간당 USD 0.008
- 상호 TLS를 사용할 때 Application Load Balancer와 연결된 신뢰 스토어별 시간당 USD 0.005
- 예약된 LCU 시간당 USD 0.008

$$\text{ALB 가동 시간} = 0.0225\$ \times 730h = 16.43\$$$

$$\text{LCU} = 0.008\$ \times 1 \times 730h = 5.84\$$$

$$\text{합계 (월)} \quad 16.43\$ + 5.84\$ = 22.27\$$$

Lattice vs. ALB(Ingress) Comparison

Lattice (Gateway API)

$$24.37\$ + 3.25\$ + 0.35\$ \\ = 27.97\$$$

ALB (Ingress)

$$16.43\$ + 5.84\$ = 22.27\$$$



Lattice가 더 비싼데용

⇒ VPC Lattice는 다중 VPC/계정 통합에 HTTP(S) 말고도 여러 프로토콜 지원 (상황에 맞는걸 사용)

[-] Elastic Container Service for Kubernetes			USD 7.07
[-] Asia Pacific (Seoul)			USD 7.07
[-] Amazon Elastic Container Service for Kubernetes APN2-Fargate-GB-Hours			USD 0.02
[-] AWS Fargate - Memory - Asia Pacific (Seoul)		3.09 hours	USD 0.02
[-] Amazon Elastic Container Service for Kubernetes APN2-Fargate-vCPU-Hours			USD 0.06
[-] AWS Fargate - vCPU - Asia Pacific (Seoul)		1.196 hours	USD 0.06
[-] Amazon Elastic Container Service for Kubernetes CreateOperation			USD 2.35
[-] Amazon EKS cluster usage in Asia Pacific (Seoul)		23.483 Hours	USD 2.35
[-] Amazon Elastic Container Service for Kubernetes ExtendedSupport			USD 4.64
[-] Amazon EKS extended support usage in Asia Pacific (Seoul)		9.283 Hours	USD 4.64
[+] Elastic Compute Cloud			USD 3.92

Extended Support = 표준 지원이 끝난 뒤 12개월 동안 보안 패치 등을 지원해줌 (시간당 0.6\$로 상승)

Virtual Private Cloud			USD 0.28
Asia Pacific (Seoul)			USD 0.28
Amazon Virtual Private Cloud APN2-VpcPeering-In-Bytes			USD 0.00
Amazon Virtual Private Cloud APN2-VpcPeering-Out-Bytes			USD 0.00
Amazon Virtual Private Cloud Public IPv4 Addresses			USD 0.01
Amazon Virtual Private Cloud TransitGatewayVPC			USD 0.14
Amazon Virtual Private Cloud VpcEndpoint			USD 0.00
Amazon Virtual Private Cloud VpcLattice			USD 0.13
\$0 for the first 300,000 Requests per hour processed by VPC Lattice Servic		75 Requests	USD 0.00
\$0.0325 per GB Data Processed by VPC Lattice Service (Seoul)		0 GB	USD 0.00
\$0.0325 per Hour per VPC Lattice Service (Seoul)		4 Hrs	USD 0.13

Conclusion

Introduction

Gateway API is an official Kubernetes project focused on L4 and L7 routing in Kubernetes. This project represents the next generation of Kubernetes Ingress, Load Balancing, and Service Mesh APIs. From the outset, it has been designed to be generic, expressive, and role-oriented.

The overall resource model focuses on 3 separate *personas* and corresponding resources that they are expected to manage:

(오피셜) Ingress 차세대 어쩌고...

(요약) Ingress 대신 Gateway API 잡춰보श्य

Conclusion



... 근데 단일 VPC에 HTTP(S) 트래픽만 받는 일반적인 상황이면
굳이 필요가 없지 않나?

Conclusion

단일 VPC/리전/계정에만 있는 클러스터라면 굳이 쓸 필요가 없음 (대신 Ingress)

만약 기존의 Ingress 리소스를 꼭 필요한게 아니라면 마이그레이션 할 필요가 없다는 뜻

만약 구성을 처음부터 하는 상황이라면? ⇒ Gateway API 권장



감사합니다

..?

