

DevOps

CI/CD 파이프라인 자동화 구축

CONTENTS

01 E-ON 서비스 소개

02 개발 및 협업 방식

03 Rolling Update 기반
CI/CD 파이프라인 구축

04 BLUE-GREEN 기반
CI/CD 파이프라인 구축

05 트러블 슈팅

06 질의응답

01.

E-ON

서비스 소개

E-ON 서비스 소개

E-ON

초중등 학교 밖 청소년을 위한
학업 지속 및 사회성 발달 지원 서비스

E-ON 서비스 소개

주요 기능

학사 일정

학교별 학사 일정
평균 학사 일정

챌린지

비교과 활동
개설, 신청, 출석, 후기

활동 추천

개인별 맞춤 추천
시기별 맞춤 추천

커뮤니티

주제별 게시판

02.

E-ON

개발 및 협업 방식

개발 및 협업 방식

깃허브 활용



GITHUB ORGANIZATION

e-on-deployee Public

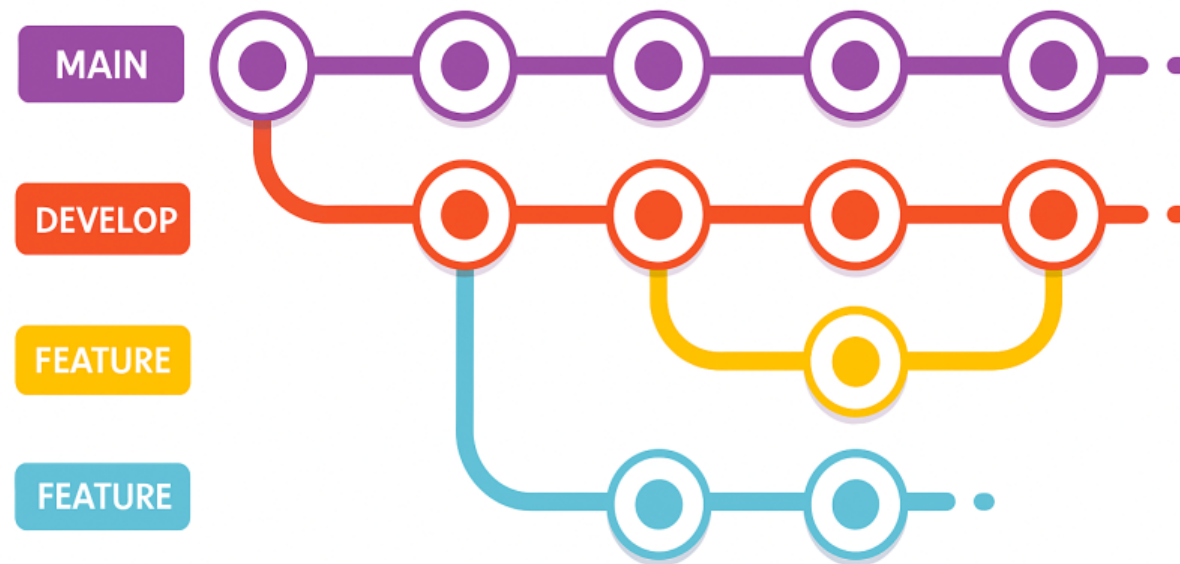
Edit Pins Watch 0

main Go to file + Code

ownue	Merge pull request #80 from Education-ON/calendar	16b8773 · 4 months ago
backend	feat(back): 보안 설정 추가	4 months ago
frontend	chore(front): 탈퇴/비활성화 소셜 전용 페...	4 months ago
node_modules	메인과 머지중	4 months ago
python-server	fix:에러수정	4 months ago
.gitignore	Merge branch 'main' into community	4 months ago
README.md	Update README.md	4 months ago
docker-compose.yml	init: E-ON 프로젝트 전체 초기 업로드	8 months ago
package.json	feat(back): 평균 학사일정	4 months ago

개발 및 협업 방식

브랜치 활용



- **MAIN 브랜치: 배포 가능한 안정 버전**
- **DEVELOP 브랜치: 기능 통합, 테스트 진행**
- **FEATURE 브랜치: 기능 단위 개발**
- **Merge Flow: Feature → Develop → Main**

개발 및 협업 방식

이슈

🔗 chore: nginx 백엔드 프록시 설정 추가 ✓
#18 by hyomee2 was merged last week

🔗 chore: 로그인 엔드포인트 수정 ✓
#17 by hyomee2 was merged last week

🔗 chore: 로그인 엔드포인트 수정 ✓
#16 by hyomee2 was merged last week

🔗 fix: schedules가 배열이 아닐 때 발생하는 filter 에러 처리 ✓
#14 by hyomee2 was merged last week

🔗 fix: 메인 화면 문제 해결 ✓
#13 by hyomee2 was merged last week

🔗 chore: 에러 처리 추가 ✓
#11 by hyomee2 was merged last week

🔗 chore: 회원 정보 조회 예외 처리 ✓
#10 by hyomee2 was merged last week

🔗 chore: URL에서 localhost 제거 ✓
#9 by hyomee2 was merged last week

🔗 chore: URL에서 localhost 삭제 ✓
#8 by hyomee2 was merged last week

🔗 refactor: Jenkinsfile에서 빌드 과정 삭제 ✓
#6 by hyomee2 was merged last week

- ISSUE로 작업 관리
- 담당자 지정 & 브랜치 연결
- 이슈에 라벨 부여 (feat, fix 등)

개발 및 협업 방식

Good Commit

feat: GKE Ingress 설정 파일 추가

fix: Jenkins 보안 경고 해결 및 파싱 문제 해결

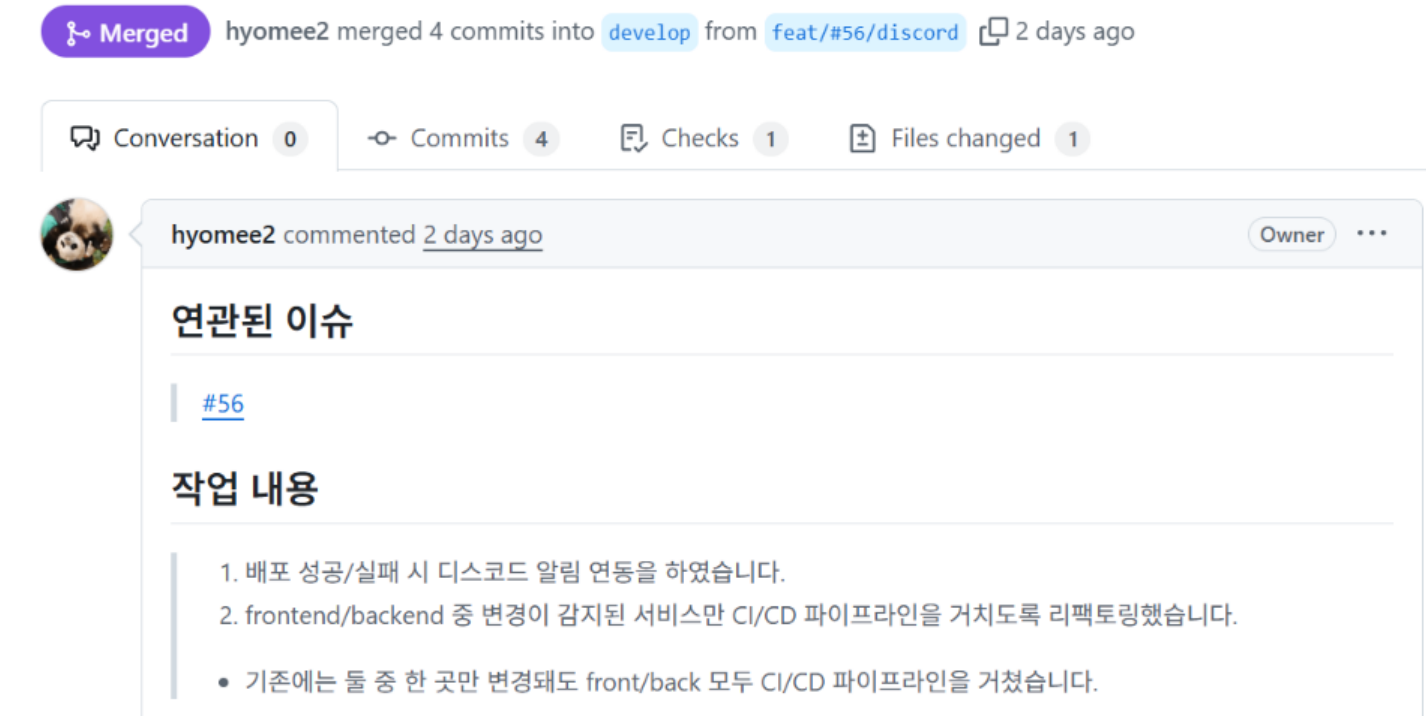
chore: 불필요한 줄바꿈 제거

feat	새로운 기능 추가
fix	버그 수정
chore	단순 코드 수정
docs	문서 수정
style	스타일 관련 수정
refactor	코드 리팩토링

- 커밋 메시지 컨벤션
- 태그 붙이기 -> 의도 파악 가능
- 변경 내용 기재

개발 및 협업 방식

PR



Merged hyomee2 merged 4 commits into develop from feat/#56/discord 2 days ago

Conversation 0 Commits 4 Checks 1 Files changed 1

hyomee2 commented 2 days ago Owner ...

연관된 이슈

[#56](#)

작업 내용

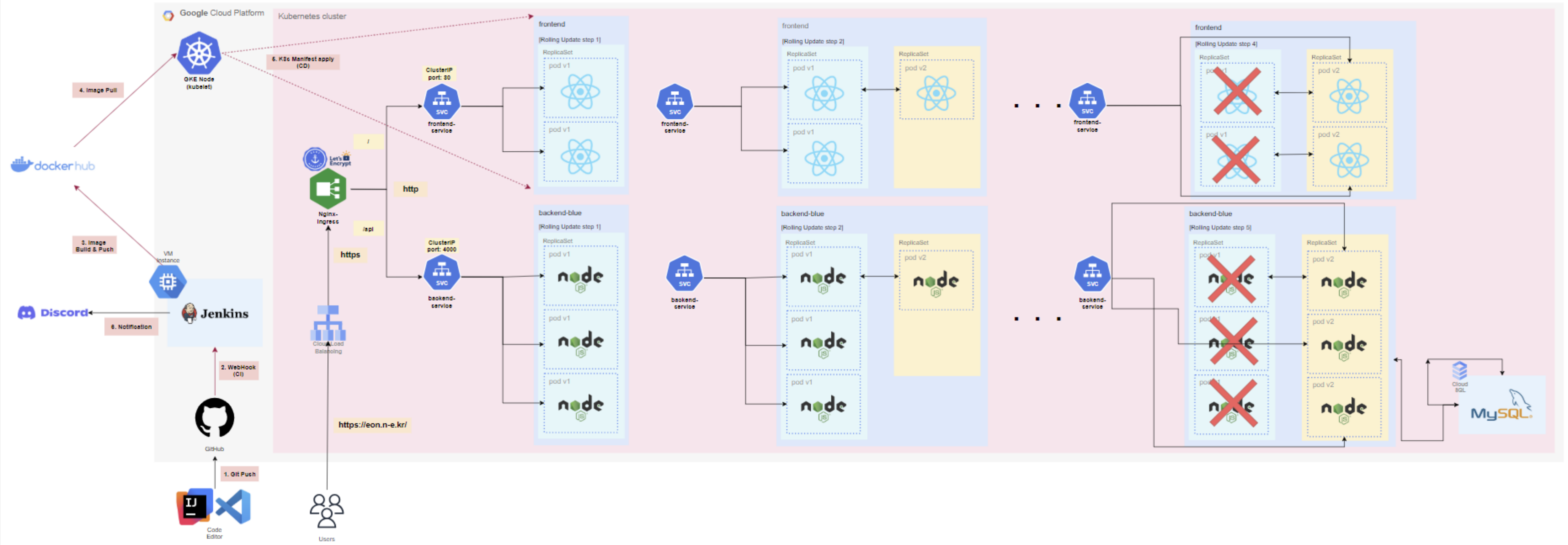
- 배포 성공/실패 시 디스코드 알림 연동을 하였습니다.
- frontend/backend 중 변경이 감지된 서비스만 CI/CD 파이프라인을 거치도록 리팩토링했습니다.

- 기존에는 둘 중 한 곳만 변경돼도 front/back 모두 CI/CD 파이프라인을 거쳤습니다.

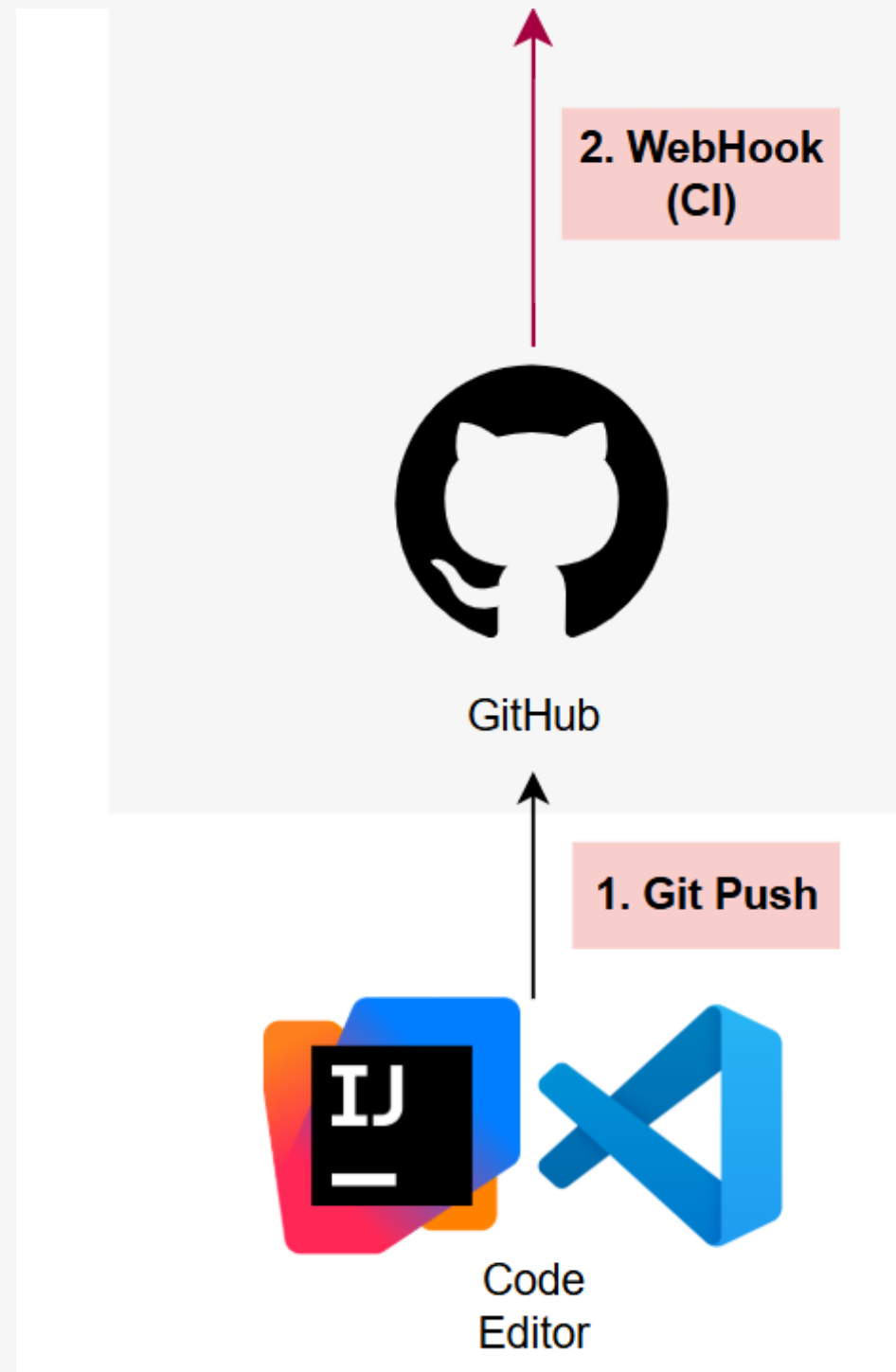
- 기능 완료 후 PR 생성
- 연결 이슈, 구현 내용 명시
- merge 진행

03.

**ROLLING-UPDATE 기반
CI/CD 파이프라인 구축**

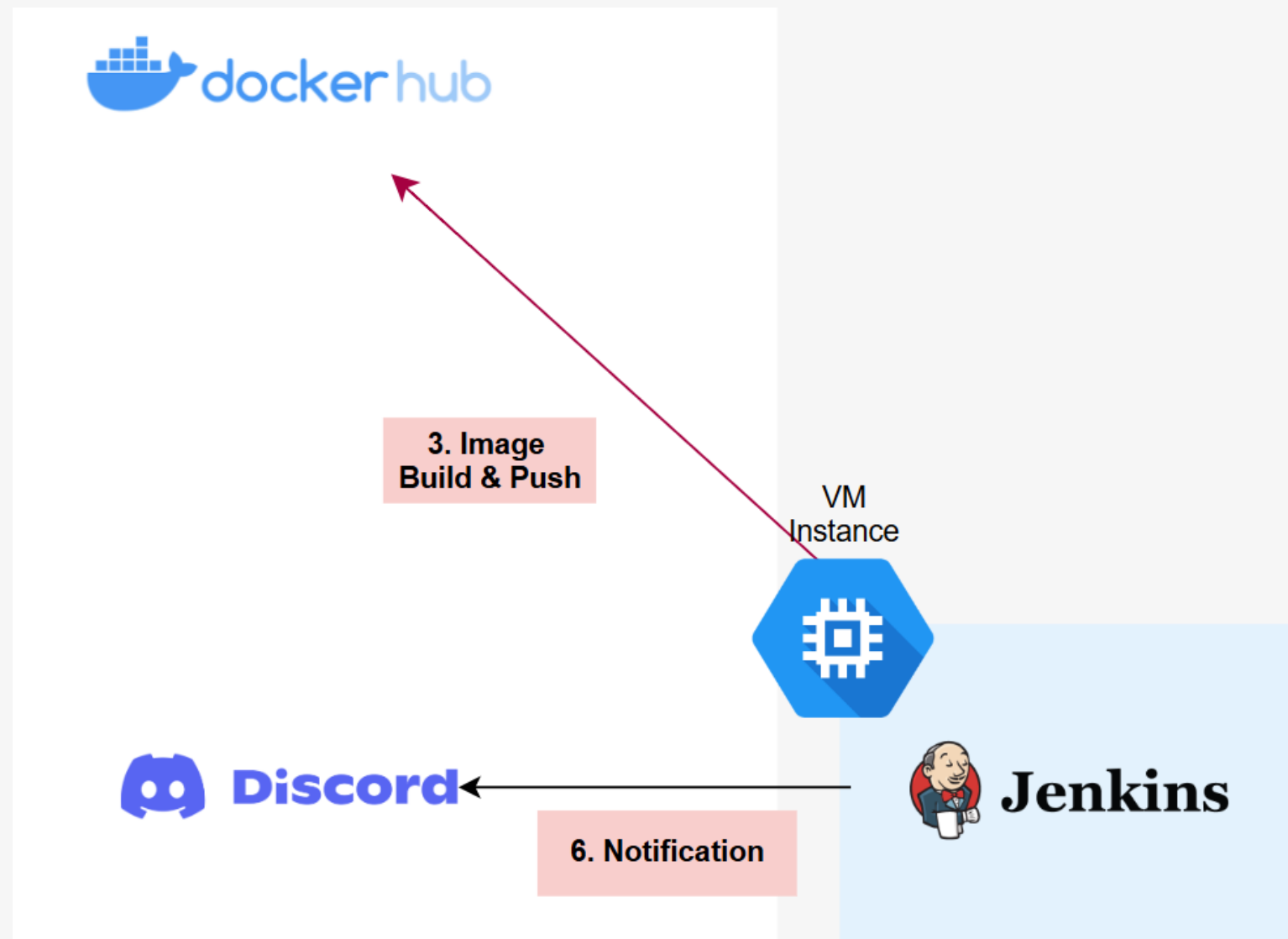


CI 파이프라인 구축



파일을 수정 후 깃허브에 푸시하면
GitHub의 Webhook이 Jenkins에게 전달

CI 파이프라인 구축



**Jenkins는 Multibranch Pipeline 방식으로
브랜치에 맞는 Jenkinsfile을 읽고 빌드를 시작**

CI 파이프라인 구축

```
stage('Deploy to GKE') {
  when {
    branch 'main' // main 에서만 배포
  }
  steps {
    echo "=== Deploy to GKE (Rolling Update) ==="

    step([
      $class: 'KubernetesEngineBuilder',
      projectId: env.PROJECT_ID,
      clusterName: env.CLUSTER_NAME,
      location: env.LOCATION,
      manifestPattern: 'k8s/*.yaml', // 배포 yaml 경로
      credentialsId: env.CREDENTIALS_ID,
      verifyDeployments: true
    ])
  }
}
```



e-on-deployee-pipeline-k8s

원본 e-on-deployee 프로젝트에 Jenkins와 Docker를 활용한 CI/CD 자동화 파이프라인을 구축하는 학습용 저장소입니다.

Branches (5)

Pull Requests (0)

S	W	Name ↓	최근 성공	최근 실패
✓	☀	develop	3 min 32 sec #1	—
✓	☀	feature/docker-cicd-setup	10 days #2	—
✓	☀	feature/gke-deployment	1 day 17 hr #41	10 days #2
✓	☀	feature/vm-deploy-setup	10 days #2	—
✓	☀	main	1 day 16 hr #47	6 days 0 hr #5

Multibranch Pipeline 방식



브랜치마다 자동으로 파이프라인 생성 및
각 브랜치의 Jenkinsfile을 기준 독립적인 CI 수행

CI 파이프라인 구축

```
stage('Deploy to GKE') {
  when {
    branch 'main' // main 에서만 배포
  }
  steps {
    echo "=== Deploy to GKE (Rolling Update) ==="

    step([
      $class: 'KubernetesEngineBuilder',
      projectId: env.PROJECT_ID,
      clusterName: env.CLUSTER_NAME,
      location: env.LOCATION,
      manifestPattern: 'k8s/*.yaml', // 배포 yaml 경로
      credentialsId: env.CREDENTIALS_ID,
      verifyDeployments: true
    ])
  }
}
```



e-on-deployee-pipeline-k8s

원본 e-on-deployee 프로젝트에 Jenkins와 Docker를 활용한 CI/CD 자동화 파이프라인을 구축하는 학습용 저장소입니다.

Branches (5)

Pull Requests (0)

S	W	Name ↓	최근 성공	최근 실패
✓	☀	develop	3 min 32 sec #1	—
✓	☀	feature/docker-cicd-setup	10 days #2	—
✓	☀	feature/gke-deployment	1 day 17 hr #41	10 days #2
✓	☀	feature/vm-deploy-setup	10 days #2	—
✓	☀	main	1 day 16 hr #47	6 days 0 hr #5

Multibranch Pipeline 방식



develop 브랜치에서는 빌드만 하고,
main 브랜치에서는 CI → CD까지 이어지도록 분리

CI 파이프라인 구축 - Jenkinsfile

기존 방식

```
stages {
  stage('Checkout') {
    steps {
      checkout scm
    }
  }

  stage('Build Backend') {
    steps {
      sh """
        echo "=== Build backend image ==="
        docker build -t ${BE_IMAGE_NAME}:latest -f backend/Dockerfile ./backend
      """
    }
  }

  stage('Build Frontend') {
    steps {
      sh """
        echo "=== Build frontend image ==="
        docker build --build-arg VITE_API_URL=${VITE_API_URL} \
          -t ${FE_IMAGE_NAME}:latest -f frontend/Dockerfile ./frontend
      """
    }
  }
}
```

프론트와 백엔드 이미지를 매번 동시에 빌드

CI 파이프라인 구축 - Jenkinsfile

변경된 방식

```
stages {
  /* 1. 코드 체크아웃 */
  stage('Checkout') {
    steps {
      checkout scm
      echo "BRANCH_NAME = ${env.BRANCH_NAME}"

      script {
        // 변경된 파일 목록 가져오기 (프론트/백 중 변경된 서비스만 빌드/배포)
        def changed = sh(
          script: "git diff --name-only HEAD~1 HEAD || true",
          returnStdout: true
        ).trim()

        env.FRONT_CHANGED = changed.contains("frontend/") ? "true" : "false"
        env.BACK_CHANGED = changed.contains("backend/") ? "true" : "false"
      }
    }
  }
}
```

**Git diff 기반으로 변경된 서비스만
선택적으로 빌드하고 push**

CI 파이프라인 구축 - Frontend

frontend/Dockerfile

```
3 # =====
4 # STAGE 1: Build Stage
5 # =====
6 # React 앱 빌드 환경
7 FROM node:20-alpine AS builder
8 WORKDIR /app
9 COPY package*.json ./
10 RUN npm install
11 COPY . .
12 # React 앱 빌드 실행 -> /app/dist 폴더에 결과물 생성
13 RUN npm run build
14
15 #빌드 결과물 확인
16 RUN ls -l
17
18
19 # =====
20 # STAGE 2: Final Stage
21 # =====
22
23 # 빌드된 정적 파일을 서빙할 Nginx 서버
24 FROM nginx:stable-alpine
25
26
27 # Builder 스테이지의 /app/dist 폴더에 있는 모든 파일을
28 # Nginx의 기본 웹 루트 폴더(/usr/share/nginx/html)로 복사
29 COPY --from=builder /app/dist /usr/share/nginx/html
30
```

React 앱을 정적으로 빌드 + Nginx로 서빙

CI 파이프라인 구축 - Frontend

frontend/nginx.conf

```
frontend > nginx.conf
1  server {
2      listen 80;
3      server_name localhost;
4
5      # 웹사이트의 루트 디렉토리 설정
6      root /usr/share/nginx/html;
7      index index.html index.htm;
8
9      location / {
10         # 1. 요청받은 파일($uri)이 있는지 확인하고,
11         # 2. 파일이 없고 디렉토리($uri/)이면, 그 디렉
12         # 3. 둘 다 없으면, 그냥 index.html을 보여준다
13         try_files $uri $uri/ /index.html;
14     }
15 }
```

try_files 설정을 통해 SPA 라우팅 안정적으로 동작

CI 파이프라인 구축 - Backend

backend/Dockerfile

```
backend > Dockerfile > ...
3 # =====
4 # STAGE 1: Build Stage (빌드 전용 스테이지, 이름은 builder)
5 # =====
6 FROM node:20-alpine AS builder
7
8 WORKDIR /app
9 COPY package*.json ./
10 # 개발용, 프로덕션용 모든 의존성 설치
11 RUN npm install
12 COPY . .
13
14
15 # =====
16 # STAGE 2: Final Stage (최종 실행용 스테이지)
17 # =====
18 FROM node:20-alpine
19
20 # netcat(nc) 도구 설치 (DB 연결 확인용)
21 RUN apk add --no-cache netcat-openbsd
22
23 WORKDIR /app
24
25 # --- 완성된 builder 스테이지에서 필요한 파일만 가져온다---
26 COPY --from=builder /app/package*.json ./
27 # 프로덕션에 필요한 의존성만 설치해서 용량을 줄인다
28 RUN npm install --omit=dev
29 # 빌드된 소스코드 전체를 가져온다
30 COPY --from=builder /app .
```

**Node 실행 환경으로 이미지 생성 및
entrypoint 스크립트 가장 먼저 실행**

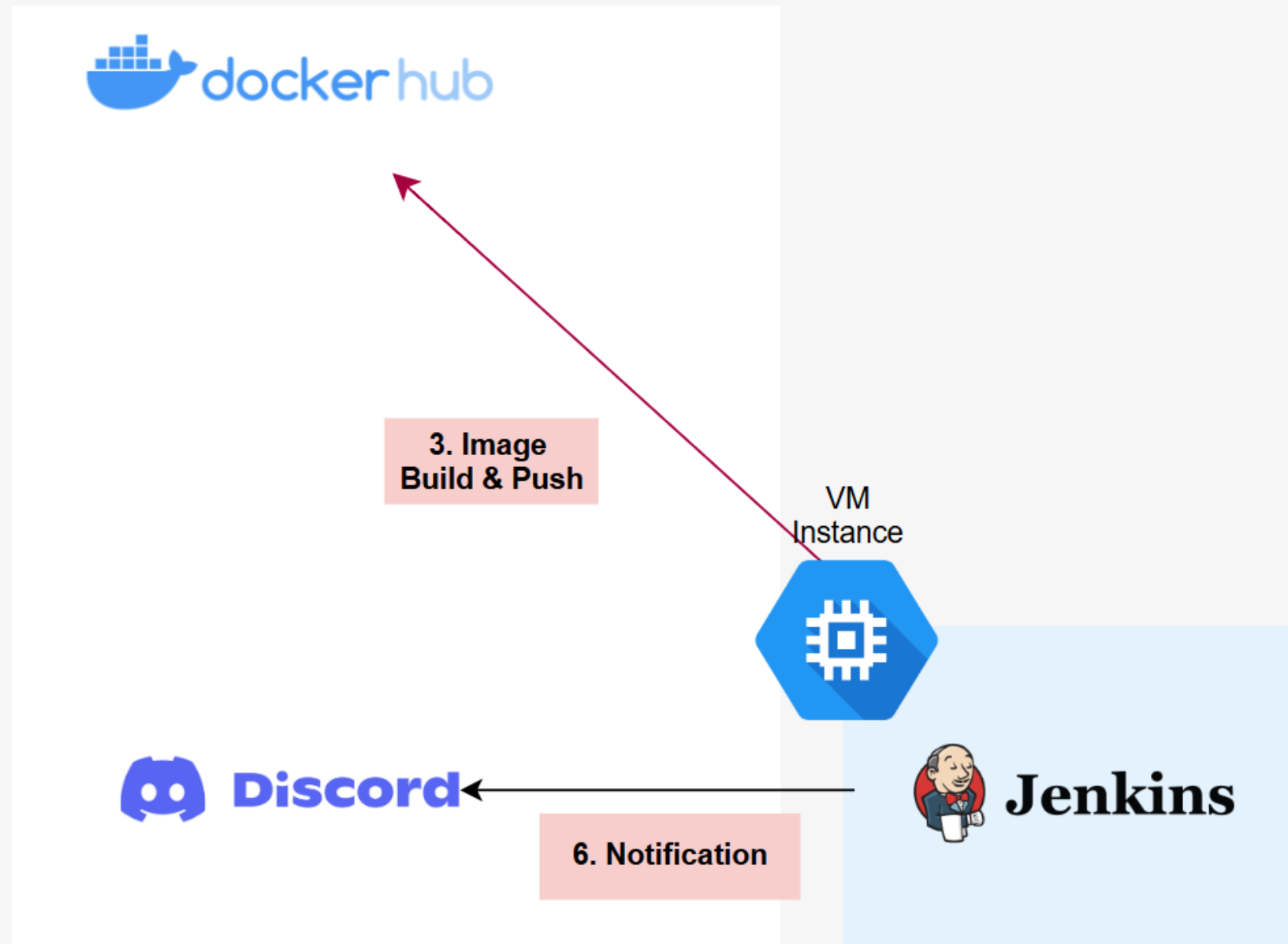
CI 파이프라인 구축 - Backend

backend/entrypoint.sh

```
backend > Dockerfile > ...
3 # =====
4 # STAGE 1: Build Stage (빌드 전용 스테이지, 이름은 builder)
5 # =====
6 FROM node:20-alpine AS builder
7
8 WORKDIR /app
9 COPY package*.json ./
10 # 개발용, 프로덕션용 모든 의존성 설치
11 RUN npm install
12 COPY . .
13
14
15 # =====
16 # STAGE 2: Final Stage (최종 실행용 스테이지)
17 # =====
18 FROM node:20-alpine
19
20 # netcat(nc) 도구 설치 (DB 연결 확인용)
21 RUN apk add --no-cache netcat-openbsd
22
23 WORKDIR /app
24
25 # --- 완성된 builder 스테이지에서 필요한 파일만 가져온다---
26 COPY --from=builder /app/package*.json ./
27 # 프로덕션에 필요한 의존성만 설치해서 용량을 줄인다
28 RUN npm install --omit=dev
29 # 빌드된 소스코드 전체를 가져온다
30 COPY --from=builder /app .
```

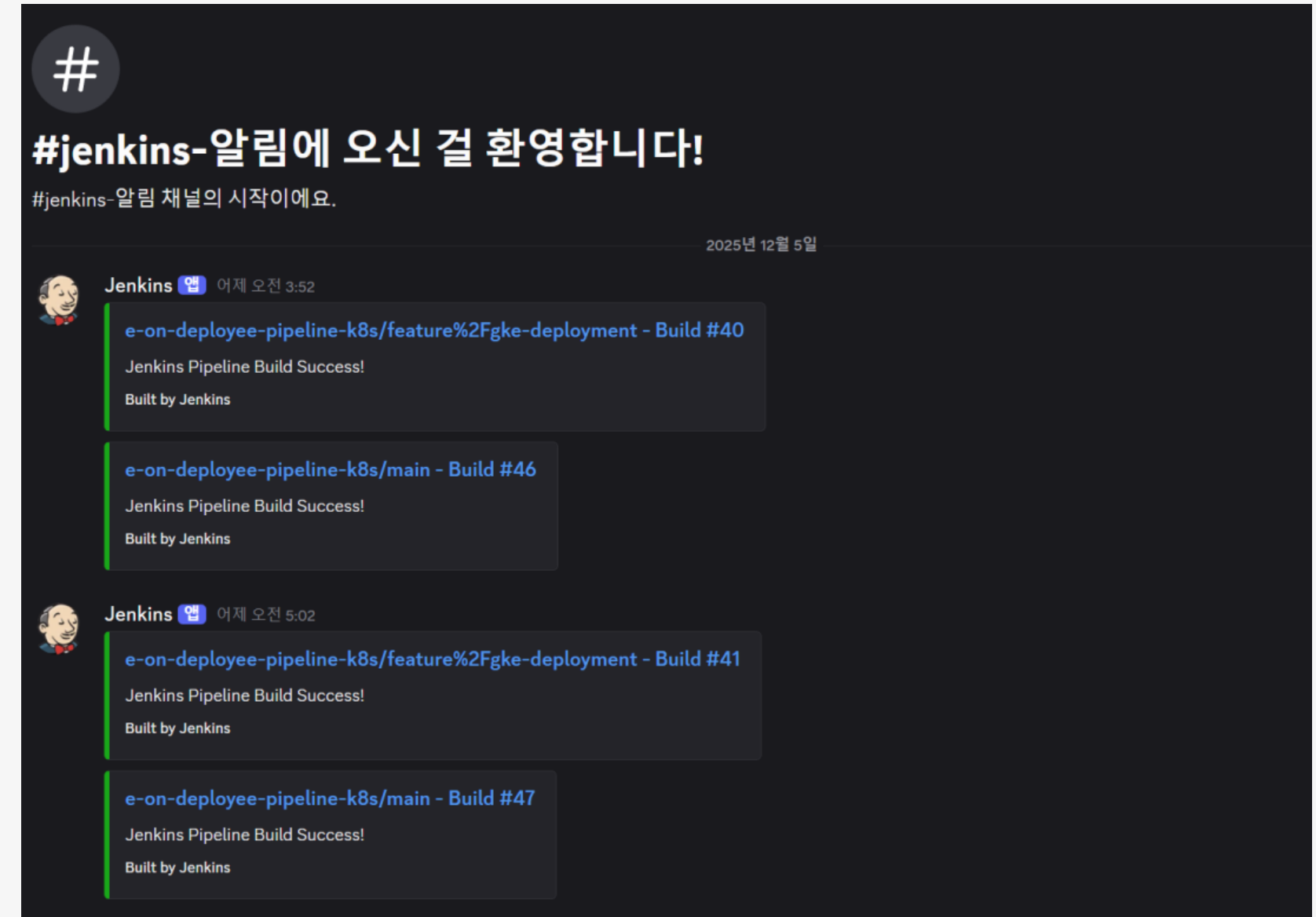
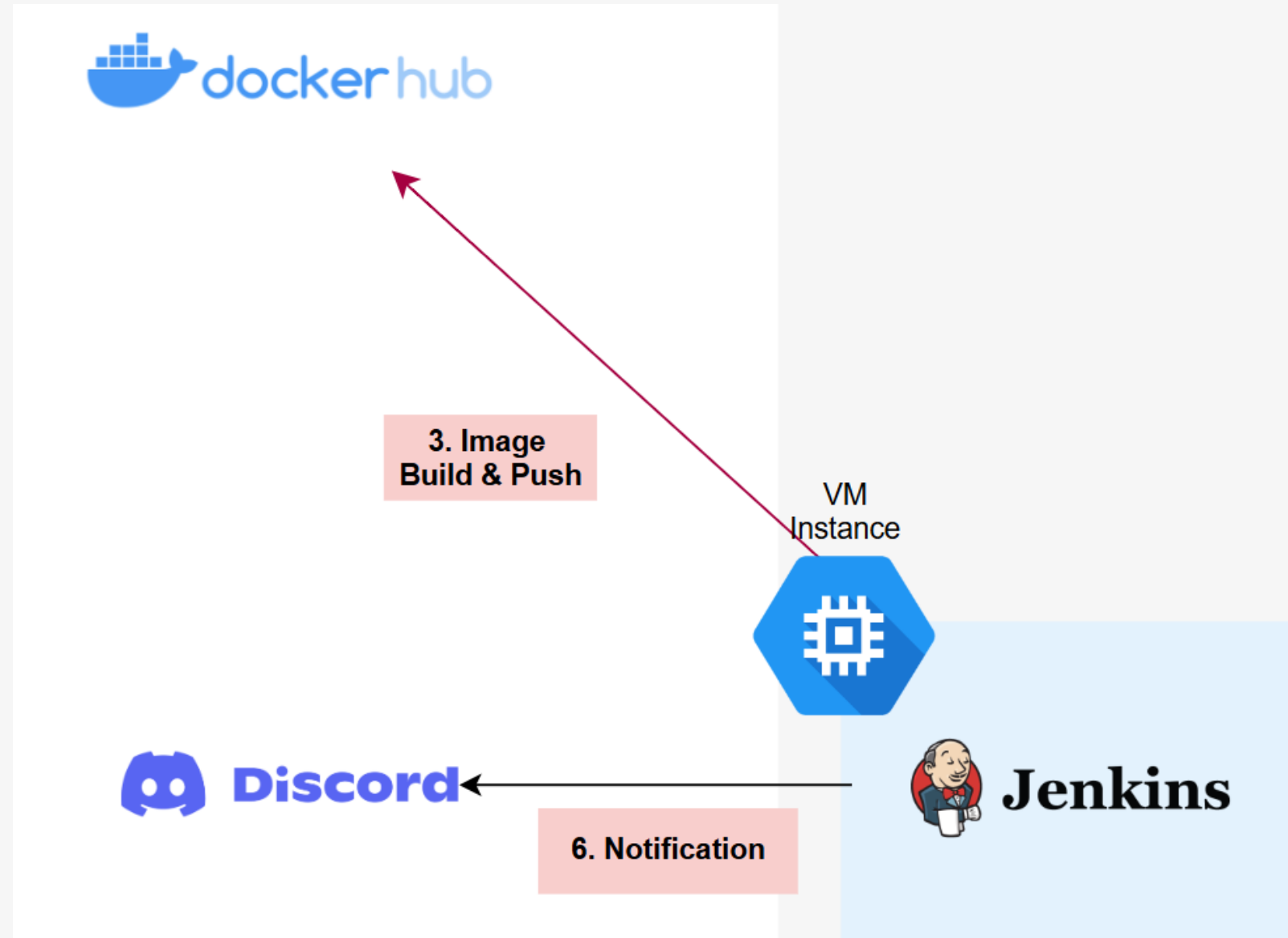
**DB가 준비될 때까지 대기 및
Sequelize 마이그레이션과 Seeder를 자동으로 적용한 후 서버 시작**

CI 파이프라인 구축



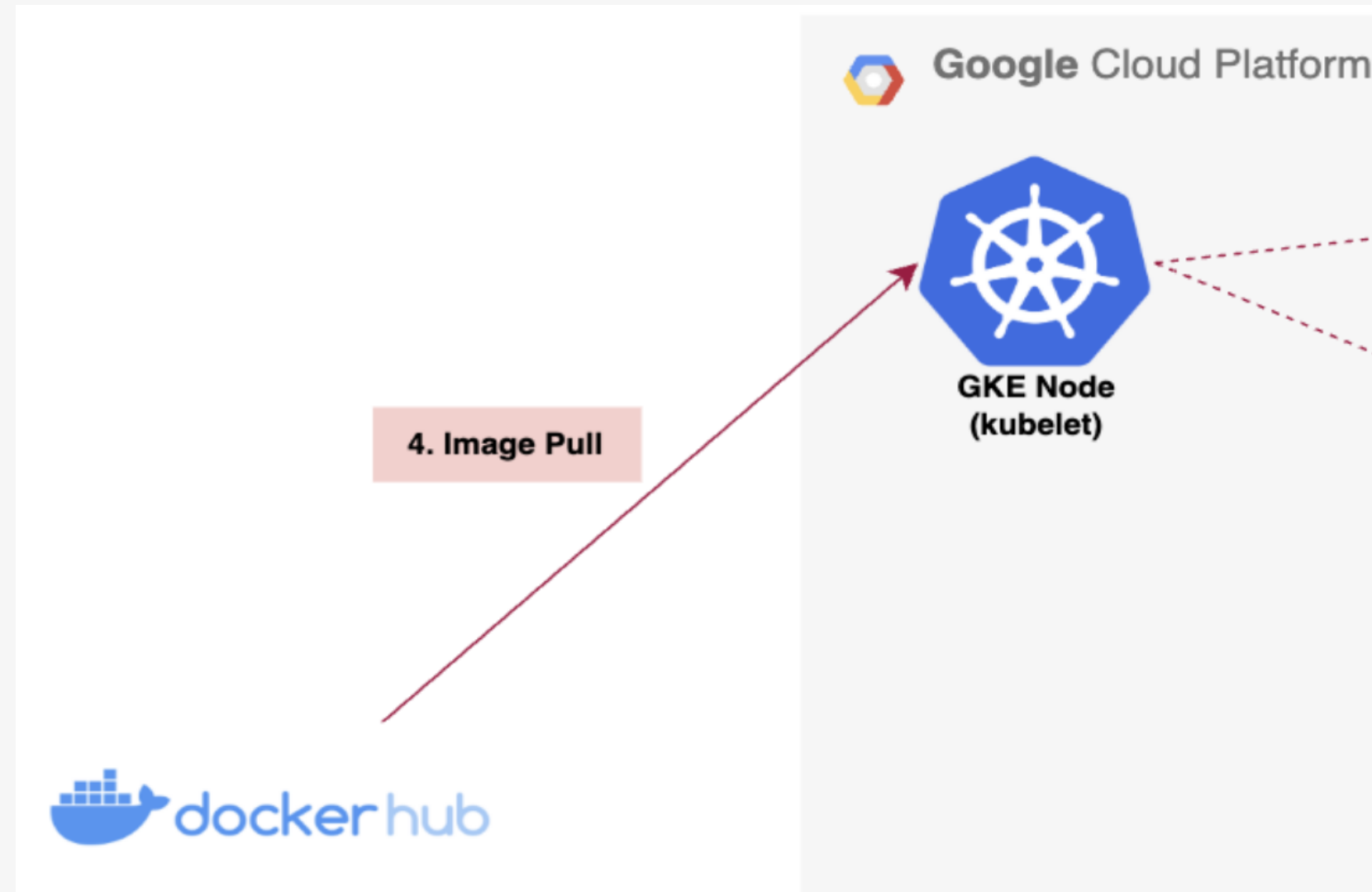
이미지 빌드가 완료되면
Docker Hub로 푸시

CI 파이프라인 구축 - Discord Webhook



빌드 혹은 배포 시에 Discord Webhook을 이용해 알림이 옵니다

CD 파이프라인

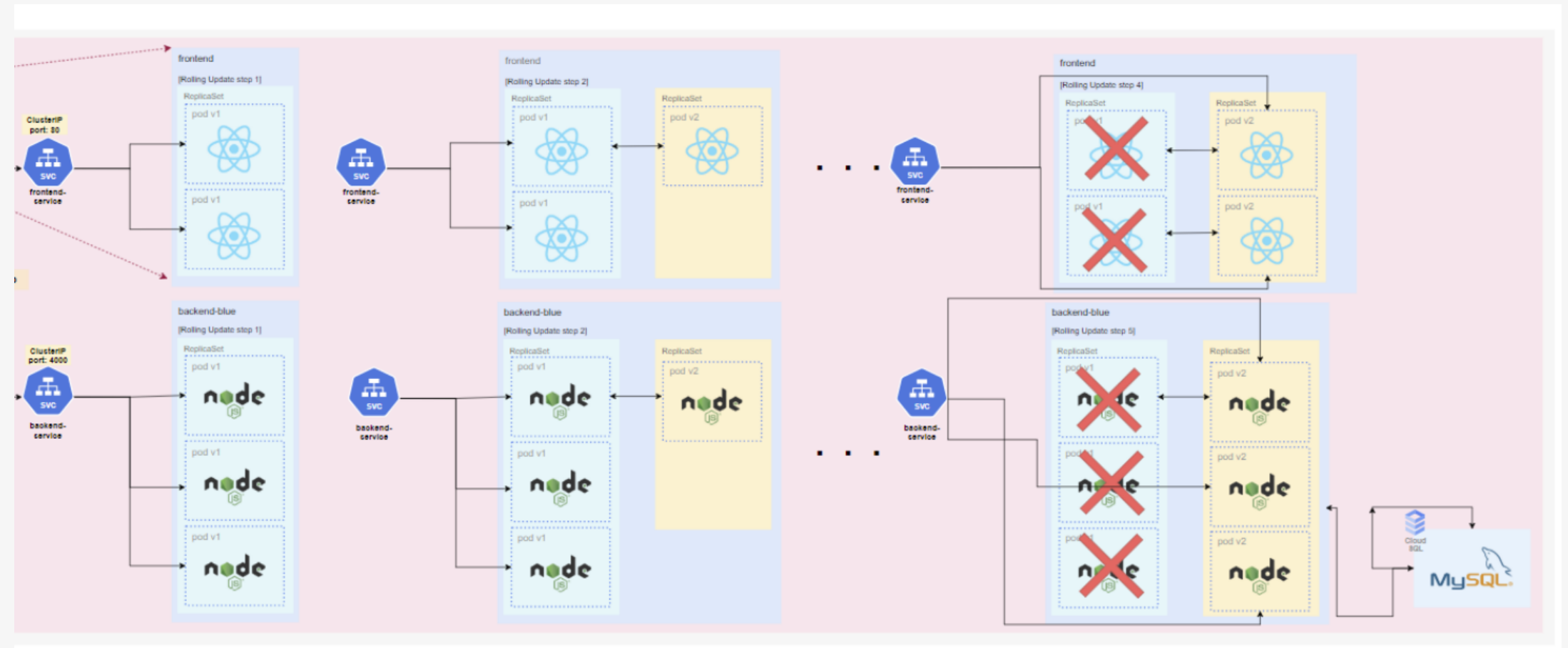


**CI 과정에서 만들어진 이미지를 활용해
쿠버네티스에서 롤링 업데이트 방식의 CD 진행**

CD 파이프라인

RollingUpdate란?

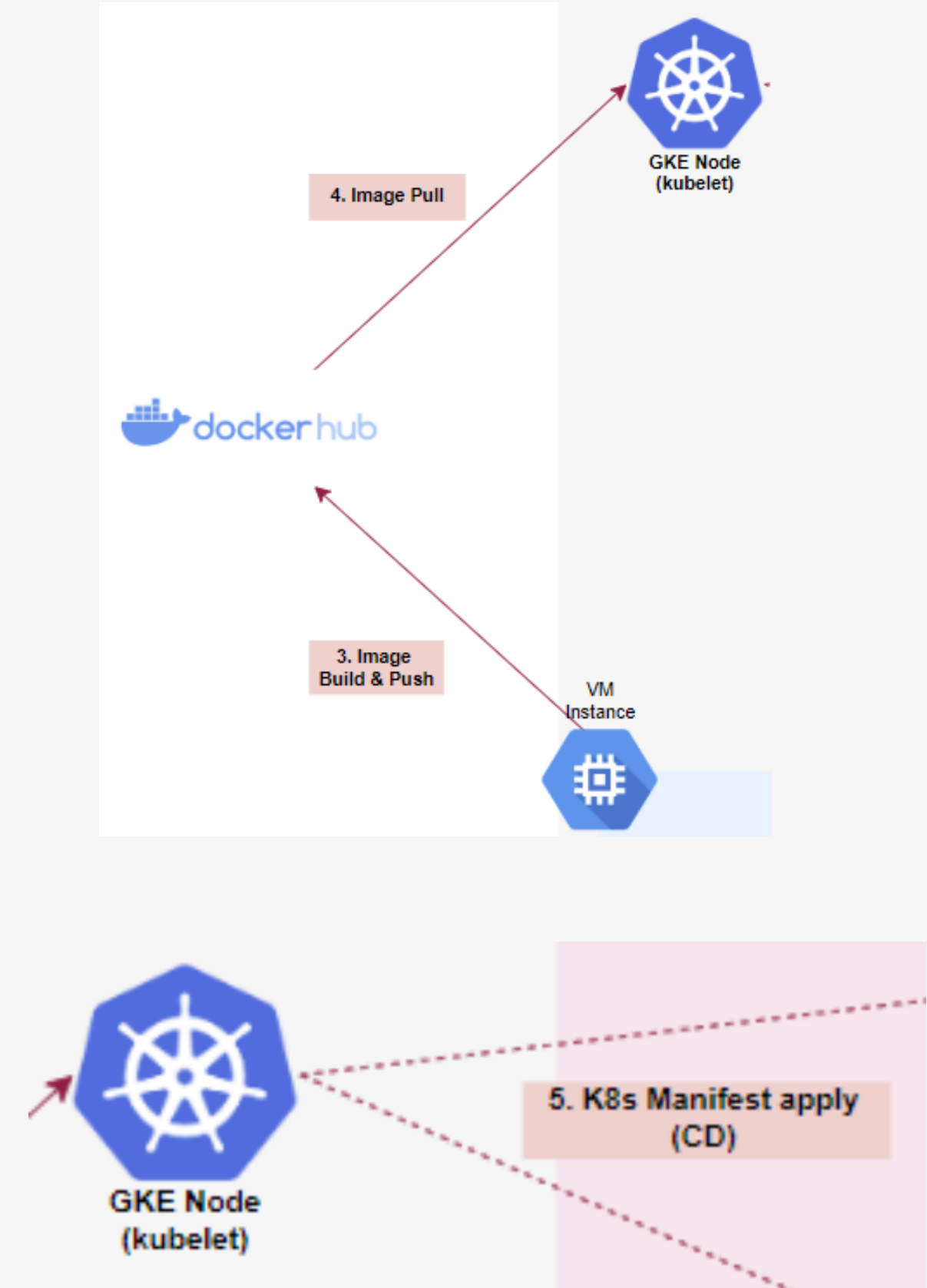
- Kubernetes의 기본 배포 전략
- Pod를 순차적으로 교체하여 서비스 중단 없이 업데이트
- 점진적 배포로 안정성 확보



CD 파이프라인

배포 과정

- 1 Jenkins CI 완료 (이미지 빌드 & Push)
- 2 `kubectl apply -f k8s/` (CD 시작)
- 3 Kubernetes 가 새 버전 배포
- 4 사용자 서비스 중단 없이 업데이트 완료



CD 파이프라인 - Kubernetes 배포 방식

기본값의 한계

기본값(25%)으로는 배포 중 가용 Pod 수 예측 한계

개선 시도 - 명시적으로 설정

- maxSurge: 1 → 업데이트 중 최대 4개 Pod까지
동시 실행 가능
- maxUnavailable: 1 → 최소 2개 Pod는 항상
서비스 가능 상태 유지
- 배포 과정에서 일정한 가용 Pod 수 보장

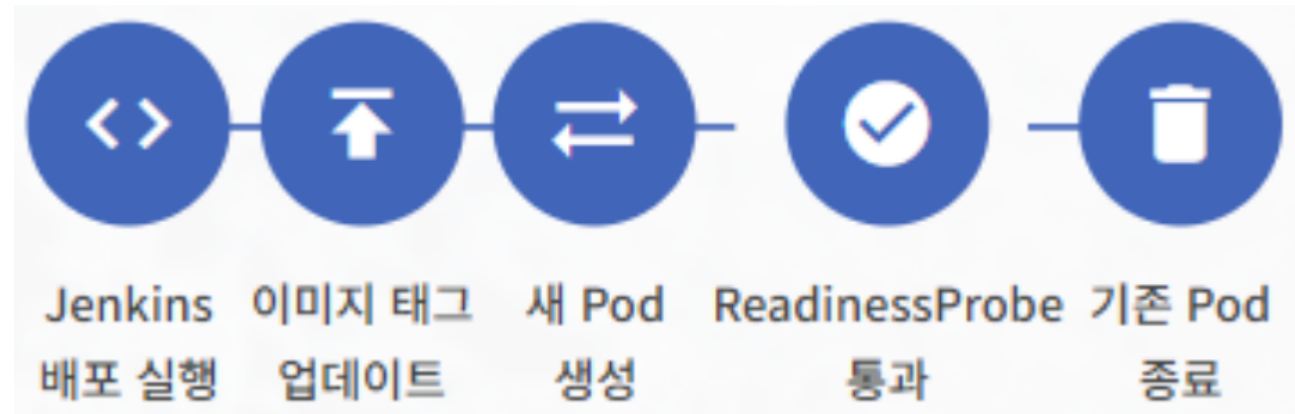
```
kimjh0937@cloudshell:~ (open-source)
strategy:
  rollingUpdate:
    maxSurge: 25%
    maxUnavailable: 25%
  type: RollingUpdate
template:
```

```
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
```

CD 파이프라인 - Jenkins CD 스테이지 실행 과정

Jenkins CD 스테이지 실행 과정

Jenkins CD 스테이지가 실행되면 새로운 이미지 태그로 Deployment 리소스 업데이트



readinessProbe를 설정하여 새로운 파드가 트래픽을 받을 준비가 완벽히 되었을 때만 기존 파드를 종료하도록 구성

```
readinessProbe:  
  httpGet:  
    path: /health  
    port: 4000  
  initialDelaySeconds: 3  
  periodSeconds: 5  
  timeoutSeconds: 2  
  failureThreshold: 3
```

CD 파이프라인 - Jenkins CD 스테이지 실행 과정

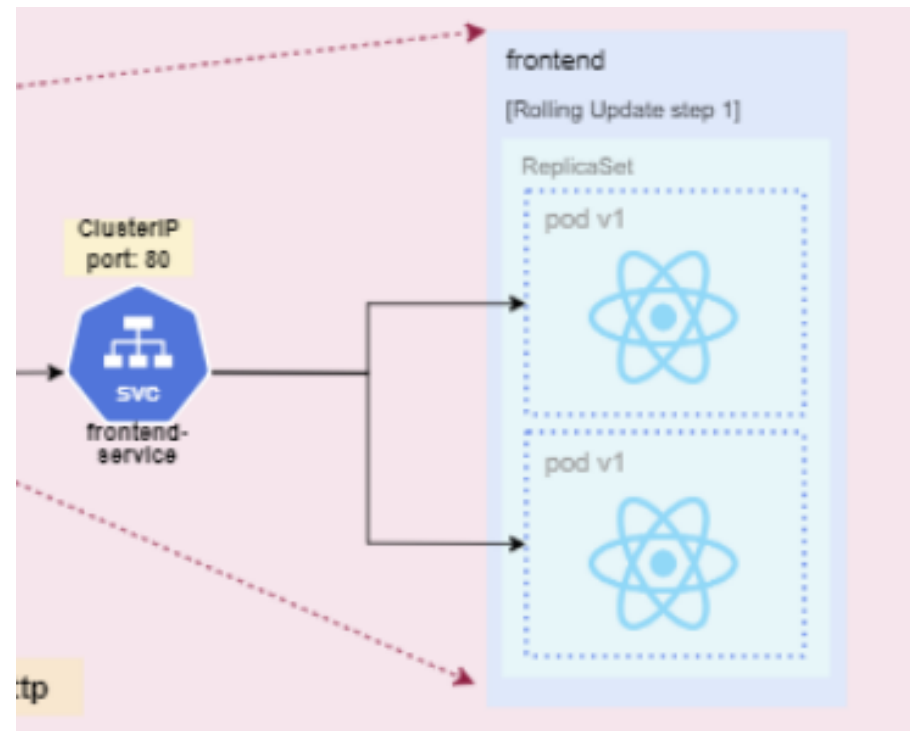
새 버전의 파드가 Running 상태가 된 후에야 기존 파드가 Terminating 되는 것을 확인할 수 있습니다.

```
kimjh0937@cloudshell:~ (open-source-gcp)$ kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
backend-587599c77f-8c47q             1/1    Running   0          30h
backend-587599c77f-95c4v             1/1    Running   0          30h
backend-587599c77f-m4ckz             1/1    Running   0          30h
frontend-6f98884df7-87xsp           1/1    Running   0          30h
frontend-6f98884df7-gk2pv           1/1    Running   0          30h
mysql-997c5499d-9xqz8               1/1    Running   0          30h
^[[Dbackend-57bb7864b8-btnvr          0/1    Pending   0          0s
backend-57bb7864b8-btnvr             0/1    Pending   0          0s
backend-57bb7864b8-btnvr             0/1    ContainerCreating   0          0s
backend-57bb7864b8-btnvr             1/1    Running   0          23s
backend-587599c77f-m4ckz             1/1    Terminating      0          30h
backend-57bb7864b8-khljl             0/1    Pending   0          0s
backend-57bb7864b8-khljl             0/1    Pending   0          0s
backend-57bb7864b8-khljl             0/1    ContainerCreating   0          0s
backend-57bb7864b8-khljl             1/1    Running   0          10s
backend-587599c77f-95c4v             1/1    Terminating      0          30h
backend-57bb7864b8-zxz5j             0/1    Pending   0          1s
backend-57bb7864b8-zxz5j             0/1    Pending   0          1s
backend-57bb7864b8-zxz5j             0/1    ContainerCreating   0          1s
backend-57bb7864b8-zxz5j             1/1    Running   0          12s
backend-587599c77f-8c47q             1/1    Terminating      0          30h
```

CD 파이프라인 - 배포 단계별 동작

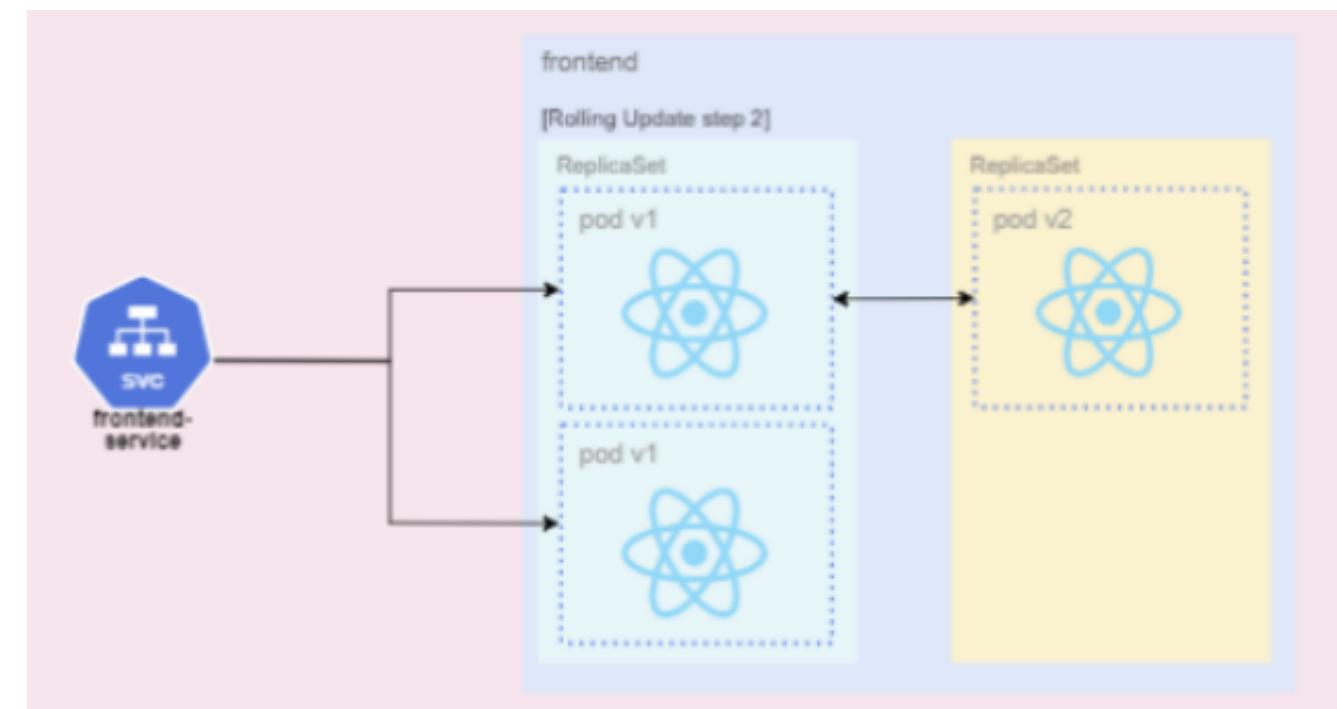
1 초기 상태

- 2개의 v1 Pod 실행 중



2 새 Pod 생성

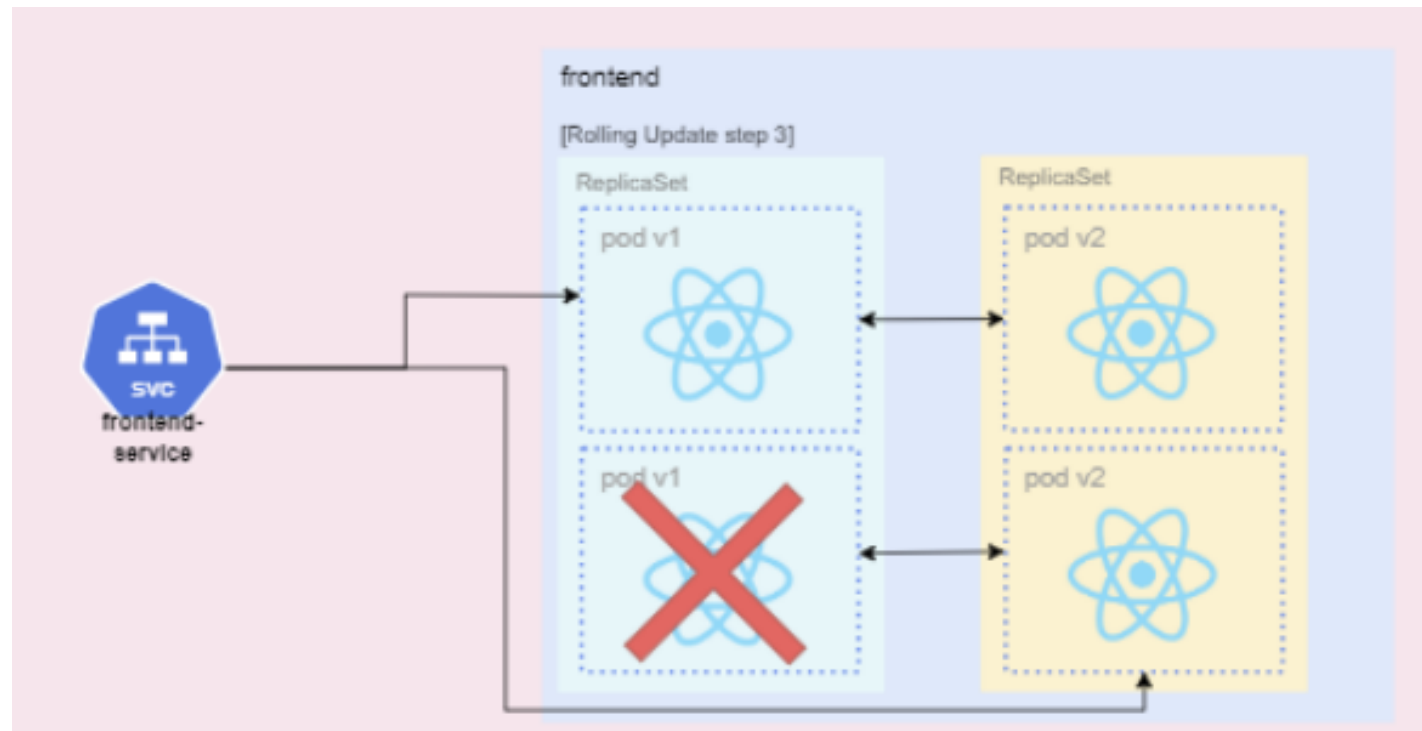
- maxSurge: 1 → 총 4개 Pod 실행



CD 파이프라인 - 배포 단계별 동작

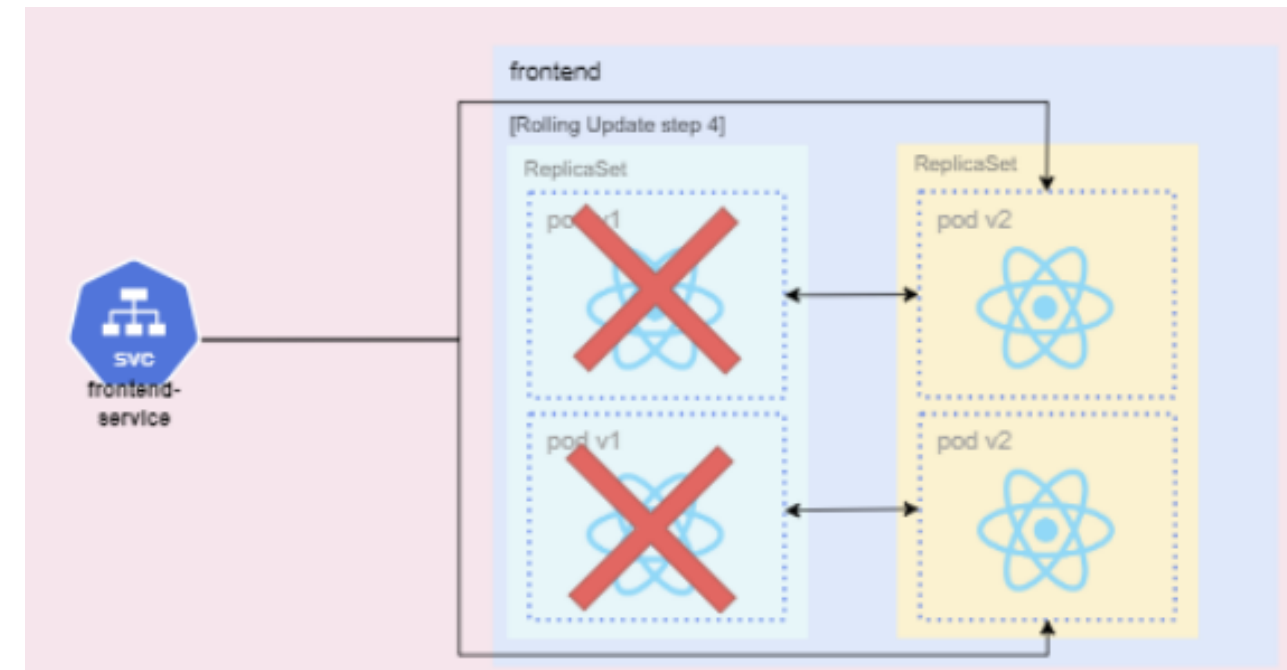
3 기존 Pod 제거

- `maxUnavailable: 1` → 기존 Pod 종료



4 완료

- 모든 Pod가 v2 버전으로 업데이트



Rolling Update 방식의 장점

안정성

순차적 교체로 리스크 최소화



무중단 배포

Service가 Ready Pod에만
트래픽 전달



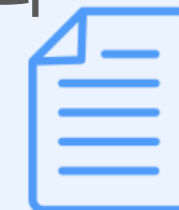
자동 Rollback

잘못된 배포 시 기존
Pod 유지



간편한 관리

별도 인프라 불필요,
YAML 파일로 선언적 관리



04.

**BLUE-GREEN 기반
CI/CD 파이프라인 구축**

Rolling Update 방식의 문제점



문제 발생 시
느린 롤백 속도

Deployment를 이전 이미지로 되돌린 후
Pod가 순차적으로 재생성



여러 버전이
동시에 서비스

기존 pod와 새 pod가 섞여 있어
충돌 발생 가능



실서버에서
배포 전 테스트 불가

새 버전이 실제 운영 환경에서
정상 동작하는지 미리 확인 불가

따라서, 이러한 문제들을 해결하기 위해
Blue-Green Deployment을 구축했습니다.

Blue-Green 배포

운영 중인 버전(Blue)과 새 버전(Green)을 **완전히 분리된 환경**으로 두고,
검증을 마친 뒤 **트래픽 라우팅만 전환**하는 방식

다운타임이 적은
무중단 배포



운영 환경에서
새 버전
사전 검증 가능

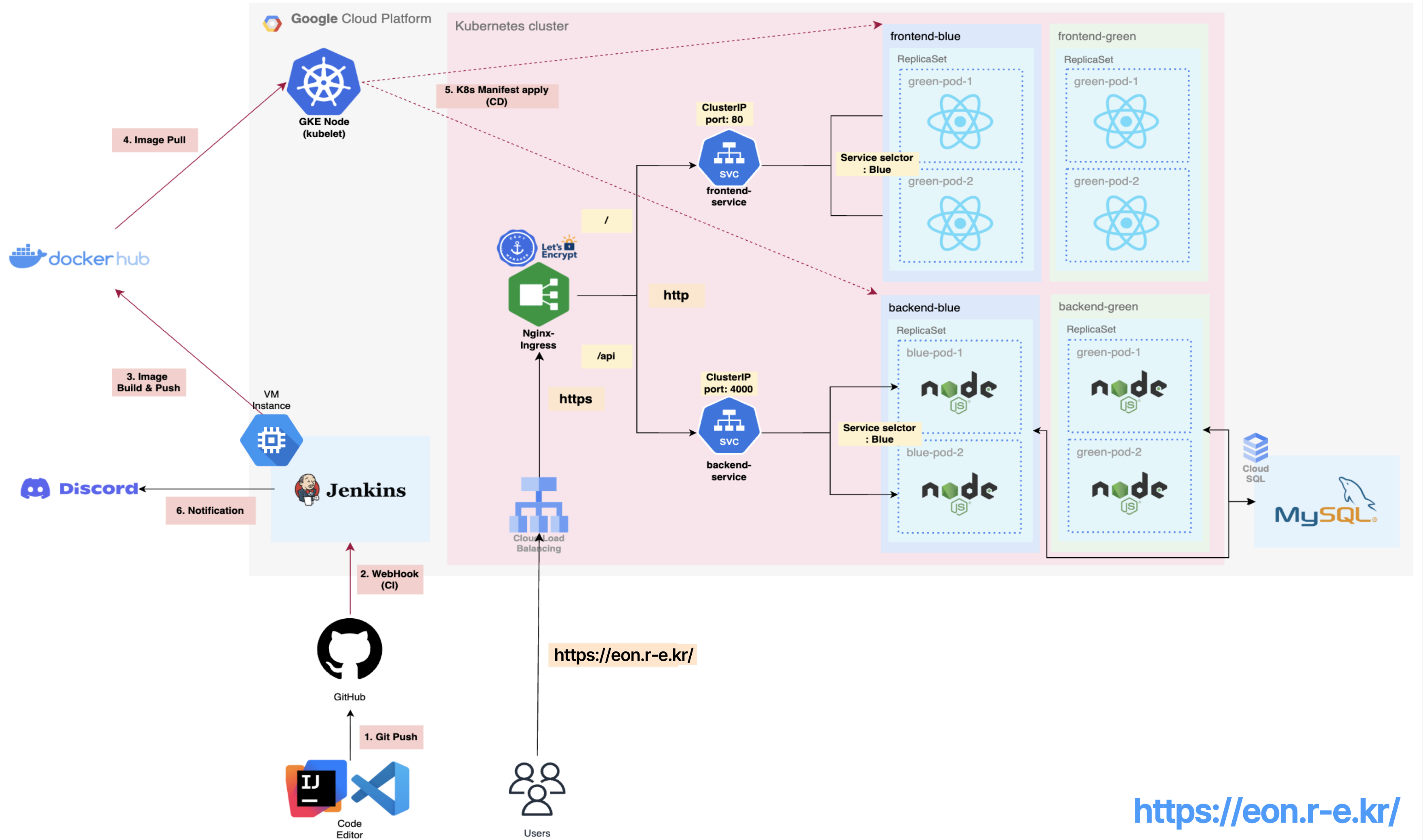


즉각적이고 안전한
Rollback



하나의 버전만 서비스하여
서비스의 안전성과
일관성 확보





CD 파이프라인

2)

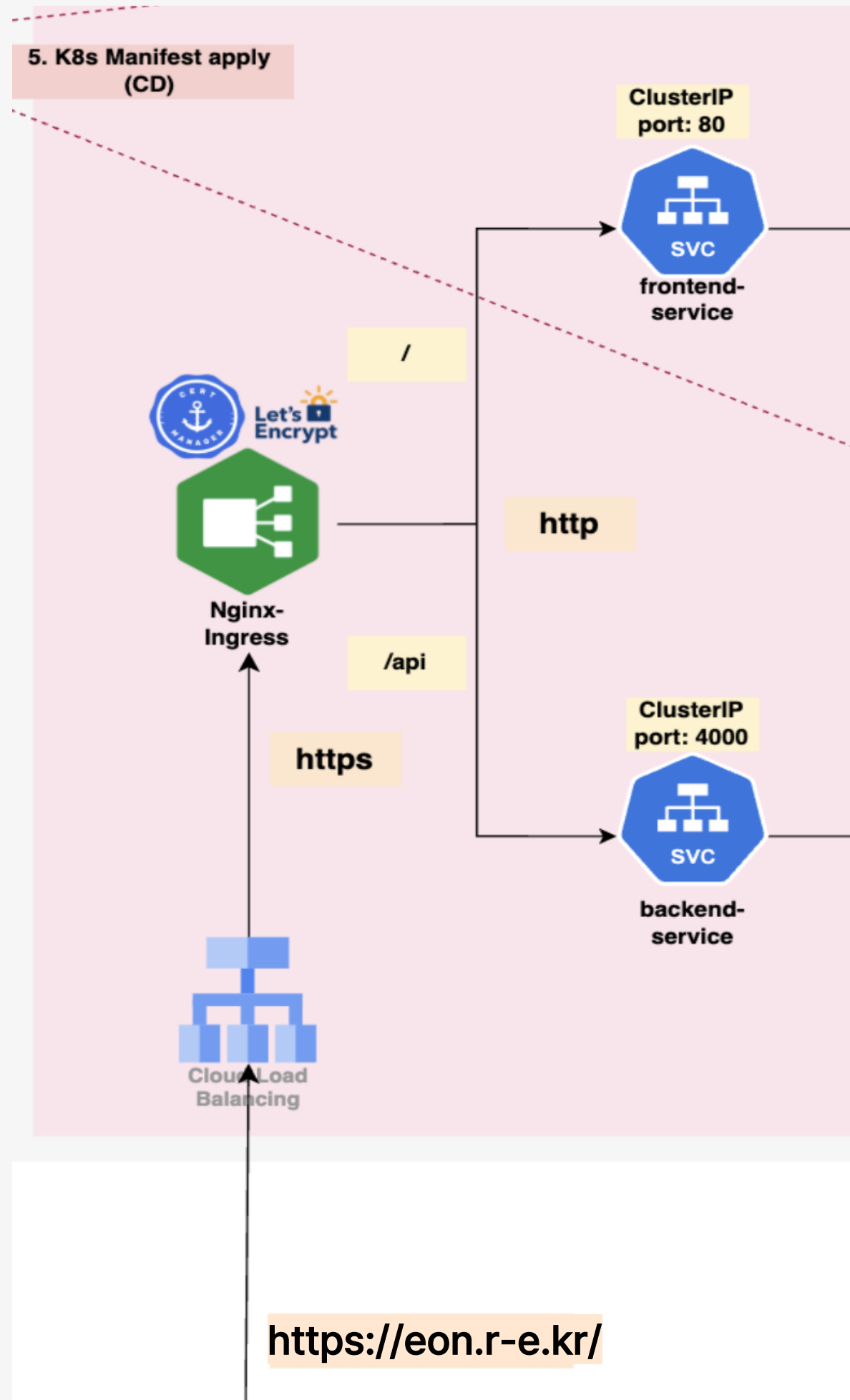
Nginx Ingress

TLS 종료 및
백엔드/프론트엔드 분기

1)

Cloud Load Balancer

외부에서 GKE 클러스터 접근



CD 파이프라인

2)

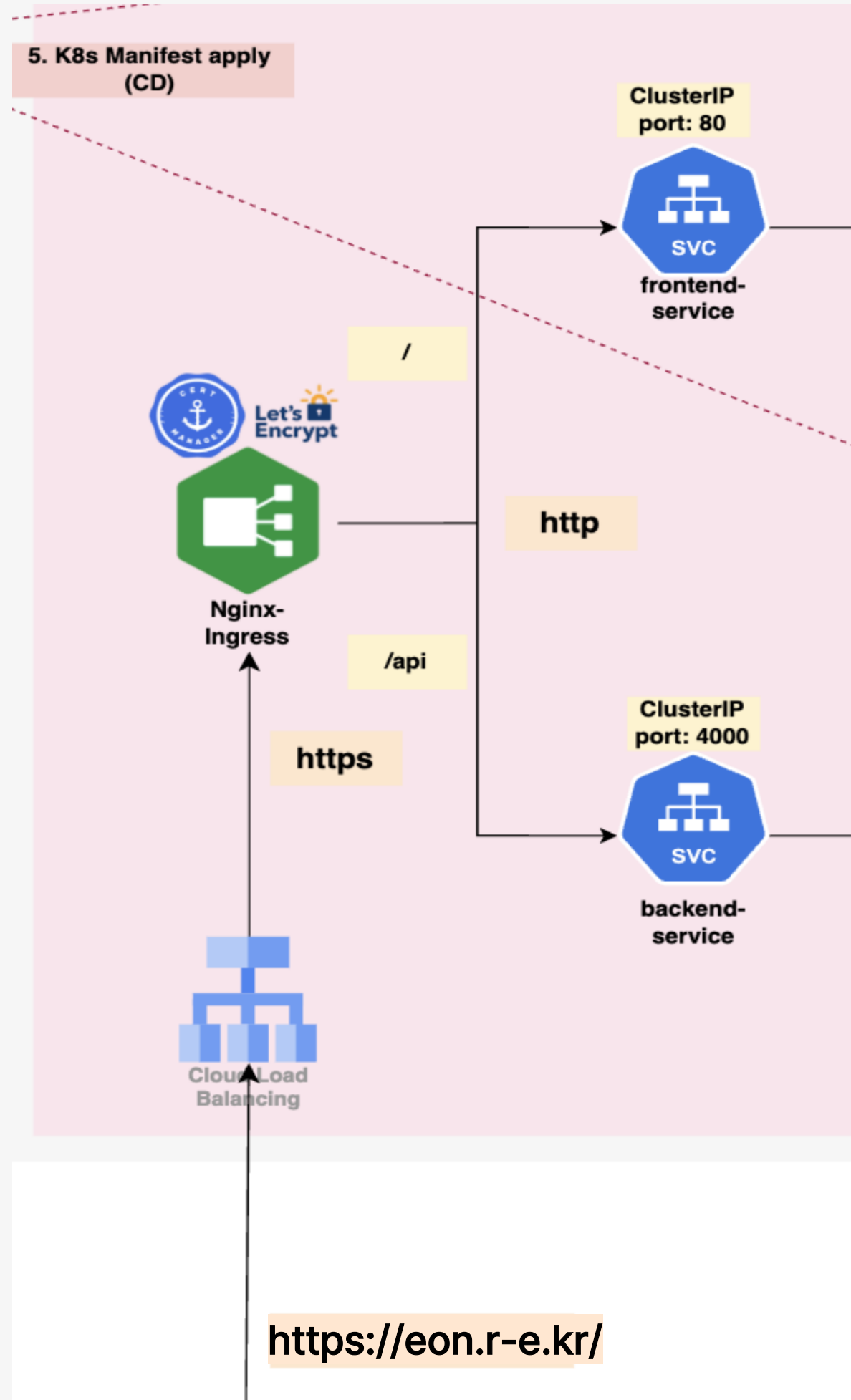
Nginx Ingress

TLS 종료 및
백엔드/프론트엔드 분기

1)

Cloud Load Balancer

외부에서 GKE 클러스터 접근



```
ingress.yml
spec:
  ingressClassName: nginx

  tls:
  - hosts:
    - eon.r-e.kr
    secretName: eon-tls-cert

  rules:
  - host: eon.r-e.kr
    http:
      paths:
        # Backend API
        - path: /api
          pathType: Prefix
          backend:
            service:
              name: backend-service
              port:
                number: 4000

        # Frontend
        - path: /
          pathType: Prefix
          backend:
            service:
              name: frontend-service
              port:
                number: 80
```

CD 파이프라인

2)

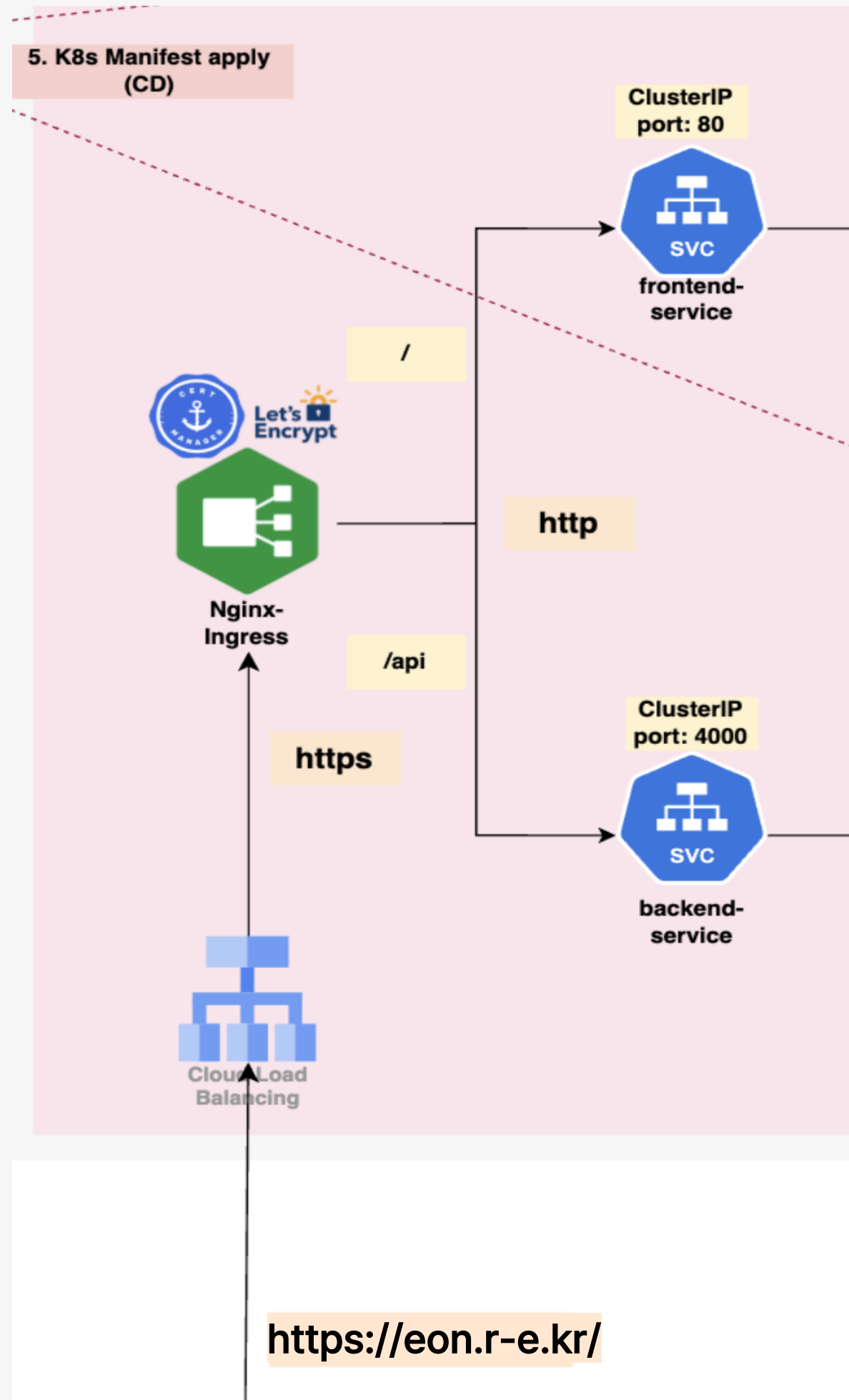
Nginx Ingress

TLS 종료 및
백엔드/프론트엔드 분기

1)

Cloud Load Balancer

외부에서 GKE 클러스터 접근



frontend-service.yml

```
spec:  
  type: ClusterIP  
  selector:  
    app: frontend  
    version: blue  
  ports:  
    - port: 80  
      targetPort: 80
```

backend-service.yml

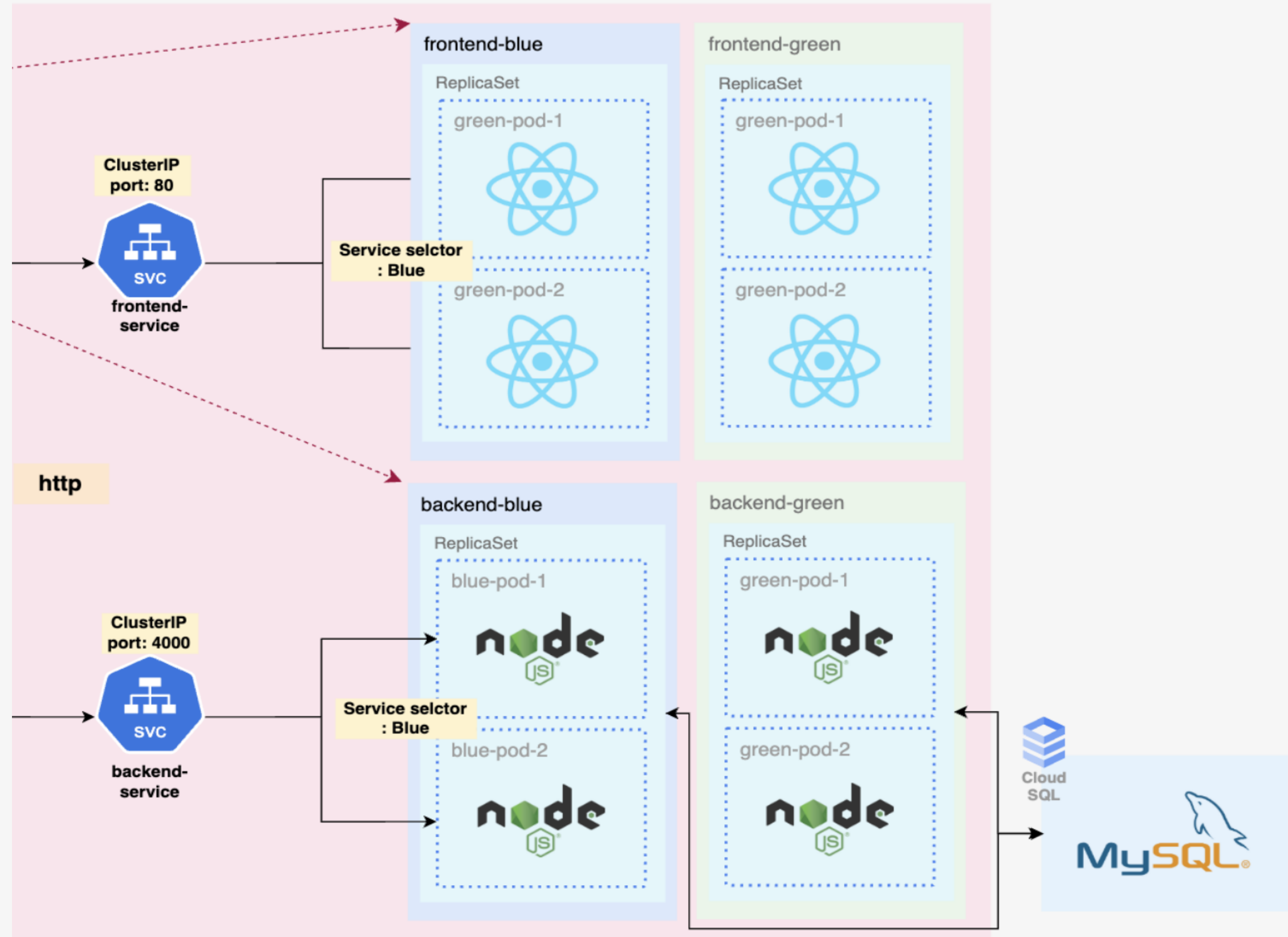
```
spec:  
  type: ClusterIP  
  selector:  
    app: backend  
    version: blue  
  ports:  
    - port: 4000  
      targetPort: 4000
```

CD 파이프라인

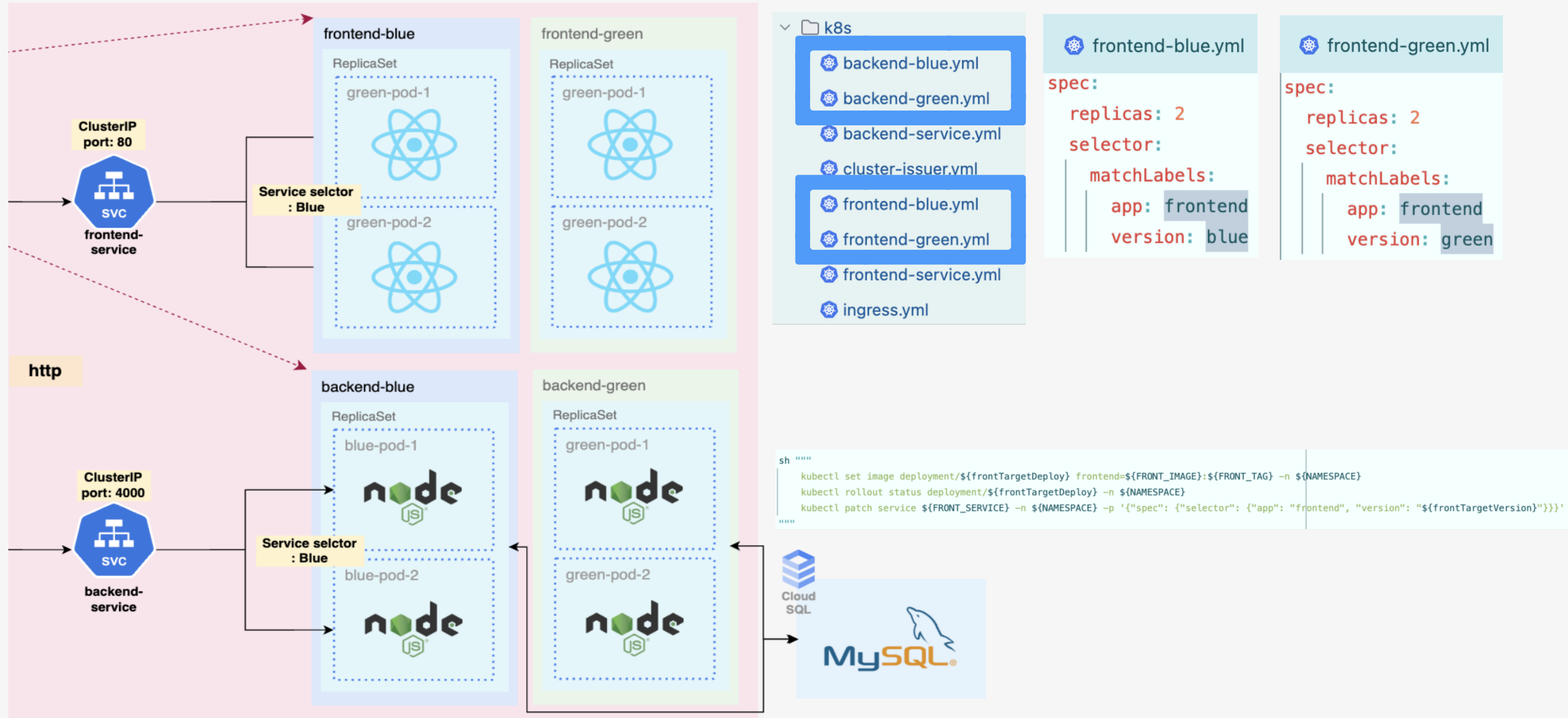
3)

Service

selector을 통해
적절한 파드를 찾아 트래픽 처리



CD 파이프라인



Blue-Green 배포의 단점

높은 인프라 비용
운영 환경과 동일한
두 배의 컴퓨팅 리소스
일시적 유지 필요



데이터베이스 관리 복잡성
두 버전이 공유하는
데이터베이스 스키마 변경이
있을 경우 전략 필요



따라서, 여러 배포 방식의 Trade-off를 고려하여
서비스에 가장 적합한 배포 전략을 수립해야 합니다.

05.

트러블 슈팅

Jenkins OOM(Out Of Memory) 해결

문제 Frontend와 Backend **동시 빌드** 시 Jenkins 컨테이너 **강제 종료 및 워크스페이스 유실.**

원인 GCP 인스턴스의 **제한된 메모리** 환경에서 **병렬(Parallel) 빌드** 수행으로 인한 리소스 고갈.

해결 parallel 스테이지를 제거하고 **순차 실행**으로 구조 변경 → 메모리 피크 사용량 감소.

```
[Pipeline] End of Pipeline
ERROR: missing workspace /var/jenkins_home/workspace/e-on-deployee-pipeline-k8s_main on

GitHub has been notified of this commit's build result

Finished: FAILURE
```

```
#3 [internal] load metadata for docker.io/library/node:20-alpine
#3 ...

#17 exporting to image
#17 exporting layers
Resuming build at Sun Nov 30 11:40:06 UTC 2025 after Jenkins restart
Ready to run at Sun Nov 30 11:44:29 UTC 2025
[Pipeline] }
[Pipeline] }
[Pipeline] // stage
[Pipeline] // stage
[Pipeline] }
Failed in branch Build Backend
[Pipeline] }
Failed in branch Build Frontend
[Pipeline] // parallel
```

```
//병렬 제거하고 순차적으로 실행
stage('Build Backend') {
    steps {
        // 백엔드 Dockerfile로 이미지 빌드
        sh "docker build -t ${BE_IMAGE_NAME}:latest -f backend/Dockerfile ./backend"
    }
}
stage('Build Frontend') {
    steps {
        // 프론트엔드 Dockerfile로 이미지 빌드 (build-arg로 API 주소 주입)
        sh "docker build --no-cache --build-arg VITE_API_URL=${VITE_API_URL} -t ${DOCKERHUB_ID_TEXT}/e-on-frontend:${IMAGE_TAG} -f frontend/Dockerfile ./frontend"
    }
}
```

Kubernetes 배포가 반영되지 않는 '이미지 캐싱' 문제

문제 Jenkins 빌드 성공 후 재배포했으나, GKE 상의 서비스는 **구버전 그대로** 유지됨.

원인 K8s의 **이미지 캐싱** 정책: 동일한 태그 사용 시 노드에 저장된 **기존 이미지를 재사용**. 정적 태그 사용으로 인한 버전 관리 불가.

해결 동적 태그 적용: **빌드 번호**(\${BUILD_NUMBER})를 **이미지 태그**로 사용.
Manifest 자동화: **배포 단계**에서 sed 명령어로 Deployment 파일의 **태그를 실시간 치환**.

```
IMAGE_TAG = "${env.BUILD_NUMBER}"
```

```
stage('Build Frontend') {  
  steps {  
    // 프론트엔드 Dockerfile로 이미지 빌드 (build-arg로 API 주소 주입)  
    sh "docker build --no-cache --build-arg VITE_API_URL=${VITE_API_URL} -t ${DOCKERHUB_ID_TEXT}/e-on-frontend:${IMAGE_TAG} frontend/Dockerfile ./frontend"  
  }  
}
```

```
echo ">> Changing image tag to ${IMAGE_TAG} in deployment.yaml..."
```

```
# 'frontend:빌드번호'로 변경
```

```
sed -i 's|nye0817/e-on-frontend:.*|nye0817/e-on-frontend:${IMAGE_TAG}|' k8s/frontend-deployment.yaml
```


서비스 노출 방식의 진화 (ClusterIP → Ingress + NodePort)

해결

```
e-on-deployee > k8s > ! backend-deployment.yaml
120 ---
121 apiVersion: v1
122 kind: Service
123 metadata:
124   name: backend
125 spec:
126   type: LoadBalancer
127   selector:
128     app: backend
129   ports:
130     - port: 4000
131       targetPort: 4000
```

```
! ingress.yaml X
e-on-deployee > k8s > ! ingress.yaml
You, 5 hours ago | 1 author (You)
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: eon-ingress
5    annotations:
6      kubernetes.io/ingress.class: "gce"
7  spec:
8    defaultBackend:
9      service:
10       name: frontend
11       port:
12         number: 80
13  rules:
14  - http:
```

```
kind: Service
metadata:
  name: frontend
spec:
  type: NodePort
```

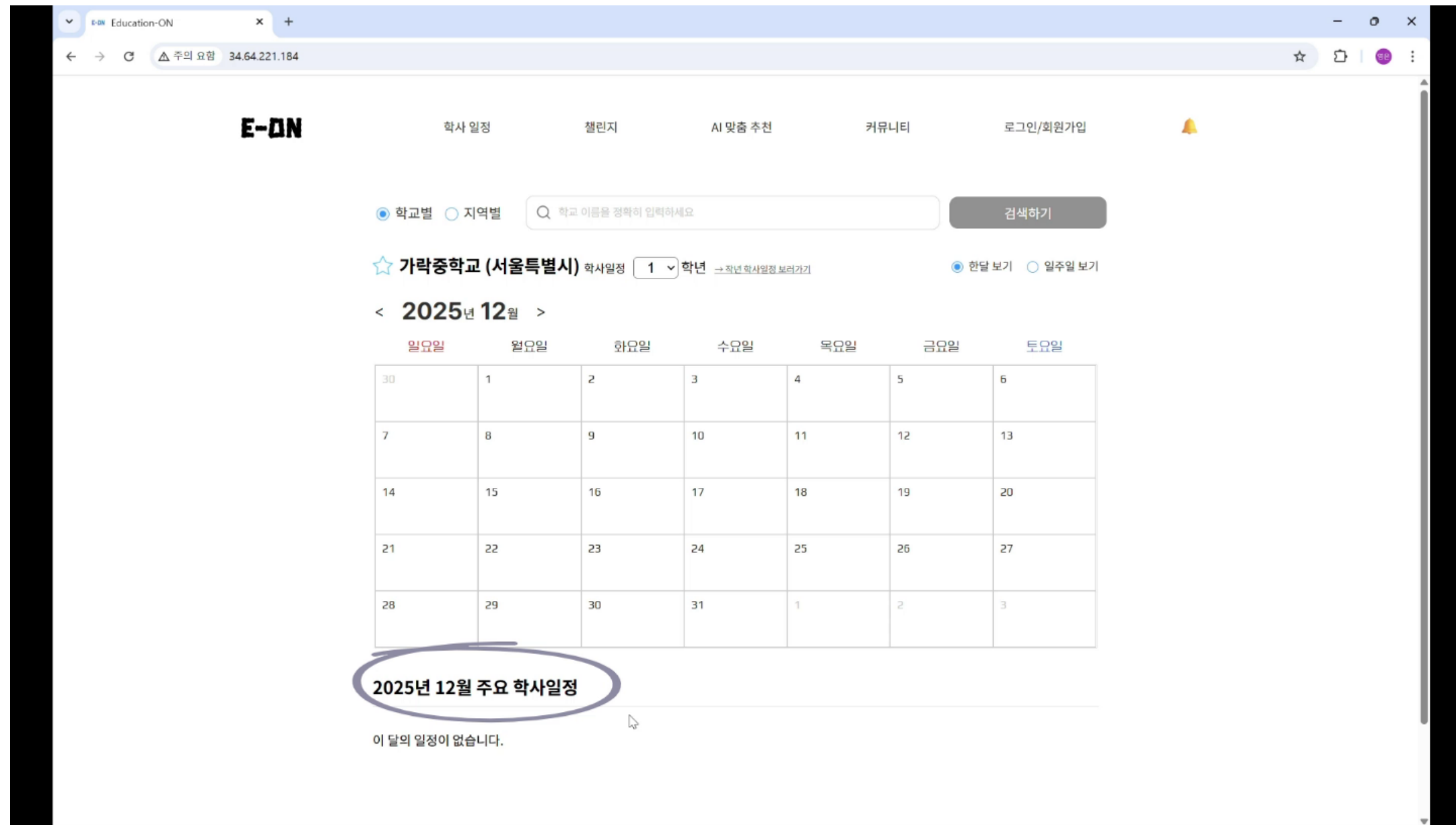
```
kind: Service
metadata:
  name: backend
spec:
  type: NodePort
```

처음: LoadBalancer 타입 변경 (공인 IP 할당, 보안 취약).

이후: Ingress 도입. 단일 진입점에서 라우팅 처리 및 보안 강화.

시연 영상

<https://www.youtube.com/watch?si=RAF-czTh8IPkL9y9&v=QKsKL9WsBvA&feature=youtu.be>



E-ON 학사 일정 캘린더 AI 맞춤 추천 커뮤니티 로그인/회원가입

학교별 지역별

☆ **가락중학교 (서울특별시)** 학사일정 학년 [→ 학년 학사일정 보러가기](#) 한달 보기 일주일 보기

< **2025년 12월** >

일요일	월요일	화요일	수요일	목요일	금요일	토요일
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

2025년 12월 주요 학사일정

이 달의 일정이 없습니다.

시연 영상

<https://youtu.be/Jp0wrMOG1No?si=dz8O0K57S6DgkZWS>

The screenshot displays a web application interface on the left and a code editor on the right. The web application shows a calendar for December 2025 with no events. The code editor shows the SchoolSearchBar component code.

2025년 12월 주요 학사일정

이 달의 일정이 없습니다.

```
1 import ...
2 // import extractCityName from "../utils/extractCityNameUtil"
3 import debounce from "lodash.debounce";
4
5 const SchoolSearchBar = () => {
6   const { searchType, setSearchType, schoolAddress, setSchoolAddress } =
7     useContext(SearchTypeContext);
8   const { setSelectedValue, setSchedules, setCurrentSchoolCode } =
9     useContext(ViewContext);
10   const [inputValue, setInputValue] = useState(initialState: "");
11   const [isFocused, setIsFocused] = useState(initialState: false);
12   const [suggestions, setSuggestions] = useState(initialState: []);
13   const [selectedSchool, setSelectedSchool] = useState(initialState: null);
14   const [selectedRegion, setSelectedRegion] = useState(initialState: null);
15
16   const placeholder: string =
17     searchType.type === "school"
18       ? "학교 이름을 입력하세요"
19       : "지역 이름을 정확히 입력하세요";
20
21   const debouncedSearch = useCallback(
22     debounce(async (value): Promise<> => {
23       if (!value.trim()) return setSuggestions(value: []); // 입력 값 없으면 비우기
24     }, 250),
25     [searchType.type]
26   );
27
28   return (
29     <div>
30       <input
31         type="text"
32         value={inputValue}
33         onChange={setInputValue}
34         onFocus={setIsFocused}
35         placeholder={placeholder}
36       />
37       <div style={isFocused ? {} : { display: none}}>
38         {suggestions.map((suggestion) => (
39           <div style={suggestion === selectedSchool ? {} : { background-color: "#f0f0f0", padding: 2px 5px; margin-bottom: 2px; border: 1px solid #ccc; border-radius: 4px; display: flex; align-items: center; gap: 5px;">
40             <span style="font-size: 0.8em; color: #666;">{suggestion}
41             <span style="font-size: 0.8em; color: #666;">> {selectedSchool}
42             <span style="font-size: 0.8em; color: #666;">> {selectedRegion}
43           </div>
44         ))}
45       </div>
46     </div>
47   );
48 };
```

터미널

```
(base) hyomee@hyomeeui-MacBookAir ~/main/2025/projects/e-on-deployee-blue-green
kubect get svc frontend-service -n eon -o jsonpath='{.spec.selector.version}'
```

현재 버전 (Blue/Green)을 확인합니다.

Lessons Learned

VM 기반의 배포에서 쿠버네티스 환경으로 확장해보며 **컨테이너 오케스트레이션의 실효성을 체감**할 수 있었다. CI 과정에서는 Webhook과 Jenkins를 연동해 변경된 서비스만 감지하여 빌드하는 방식을 도입하고, 런타임 환경까지 고려한 Docker 이미지를 구성하며 **리소스 최적화의 중요성**을 깊이 이해할 수 있었다.

특히 프로젝트 진행 중 흥미로웠던 점은 **'구현의 다양성'**이었다. 같은 CI/CD 기능을 구축하더라도 팀원마다 스크립트를 짜는 방식과 구조가 조금씩 다른 것을 공유하며 서로의 접근 방식에서 많은 것을 배울 수 있었다. 이러한 고도화 작업을 통해, 잘 구축된 데브옵스 환경이 실제 개발과 운영 과정을 얼마나 편하고 효율적으로 만들어주는지 몸소 경험했다.

나아가 K8s와 Ingress를 활용해 Blue-Green 배포를 직접 구현하고 트래픽을 제어해보며 실무적인 감각을 익힐 수 있었다. 이 과정에서 실제 발생하는 GCP 클라우드 비용을 분석해보며, 기술 도입 시에는 단순히 기능 구현뿐만 아니라 **비용과 서비스 특성을 고려한 전략적 판단이 필수적**임을 깨달을 수 있었다.

06.

질의응답

감사합니다.