DevOps CI/CD on GKE

Table of Contents

What is DevOps?

개발자와 운영자가 함께 새 소프트웨어 기능 및 서비스를

빠르게 개발하고 제공하는 방법론

DevOps Capabilities

DevOps 연구 결과 및 평가 (DORA)

기술적 역량 기능

- 버전 제어
- 지속적 통합
- 배포 자동화
- 코드 유지 관리성

프로세스 기능

- 소규모 배치 작업
- 변경 승인 간소화

측정 기능

- 모니터링
- 오류 사전 알림

문화적 기능

- 직무 만족도
- 배포 학습

What is CI/CD?

CI

지속적인 통합 (Continuous Integration) 입니다.

어플리케이션의 **새로운 코드의 변경 사항이 정기적으로 빌드 및 테스트 되어 레파지토리에 통합하는 것을 의미**합니다.

여러 개발자가 동시의 작업하며 발생할 수 있는 **충돌 문제를** 수시로 확인하고 해결할 수 있습니다. 버그를 빠르게 찾아 해결하고, 소프트웨어 품질을 개선하며 새로운 업데이트와 출시 시간을 단축 시키는 것 입니다.

CD

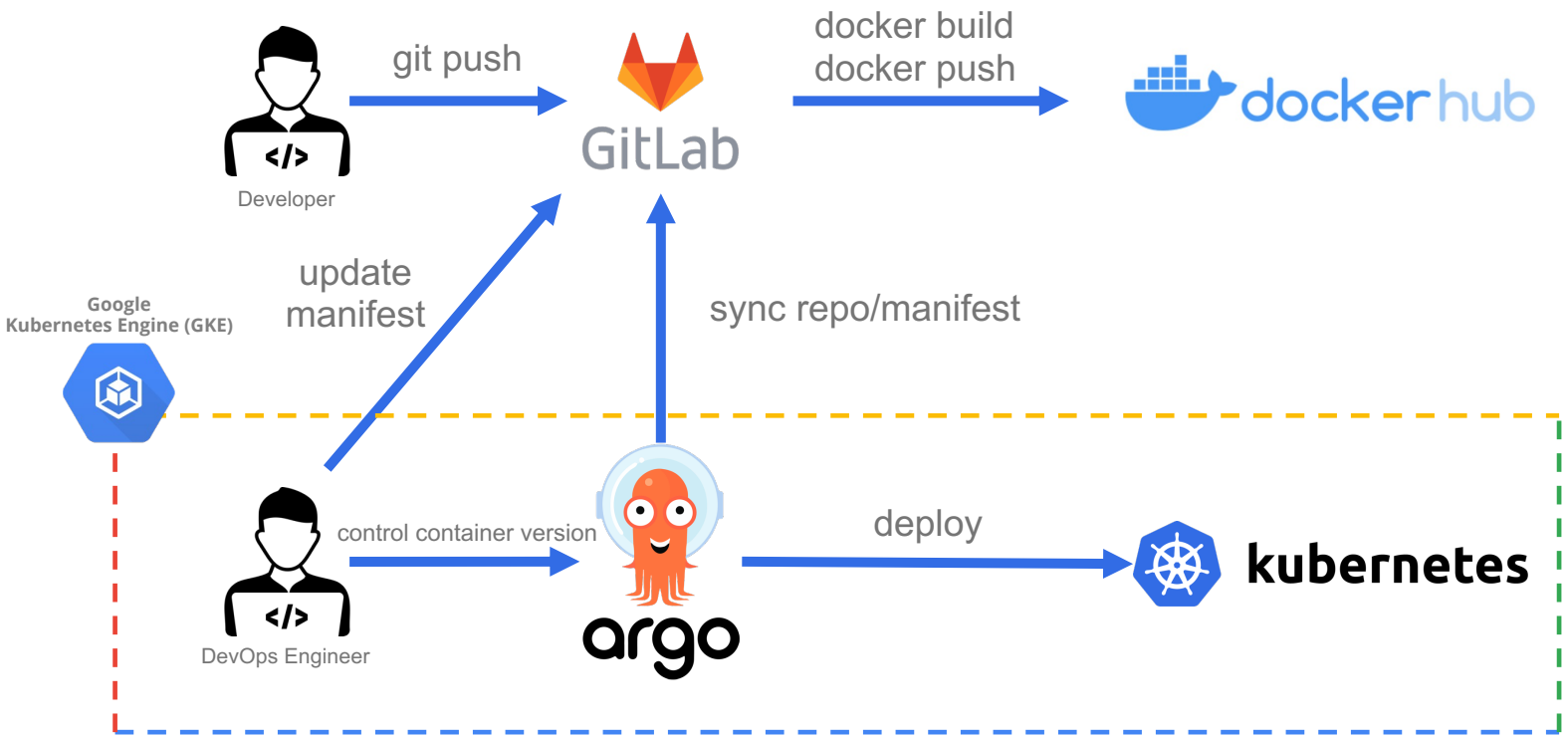
지속적인 제공(Continuous Delivery) 또는 **지속적인 배포(Continuous Deployment) 두 가지** 의미로 같이 사용됩니다.

지속적인 제공이란 개발자들이 애플리케이션에 적용한 변경 사항이 버그 테스트를 거쳐 레파지토리에 **자동으로 업로드되는 것**을 뜻하며, 운영팀은 이 레파지토리에서 애플리케이션을 **실시간으로 배포할 수 있습니다.**

지속적인 배포란 개발자의 변경 사항을 **레파지토리에서 실제 서비스까지 자동으로 릴리스하는 것**을 의미합니다. 이는 애플리케이션 제공 속도를 저해하는 **수동 프로세스로 인한 운영팀의 프로세스 과부하 문제를 해결할 수 있습니다.**

Diagram

Diagram


git push
GitLab
docker build
docker push
docker hub
Developer
update
manifest
sync repo/manifest
Google
Kubernetes Engine (GKE)
control container version
deploy
kubernetes
DevOps Engineer
argo

Used Tech

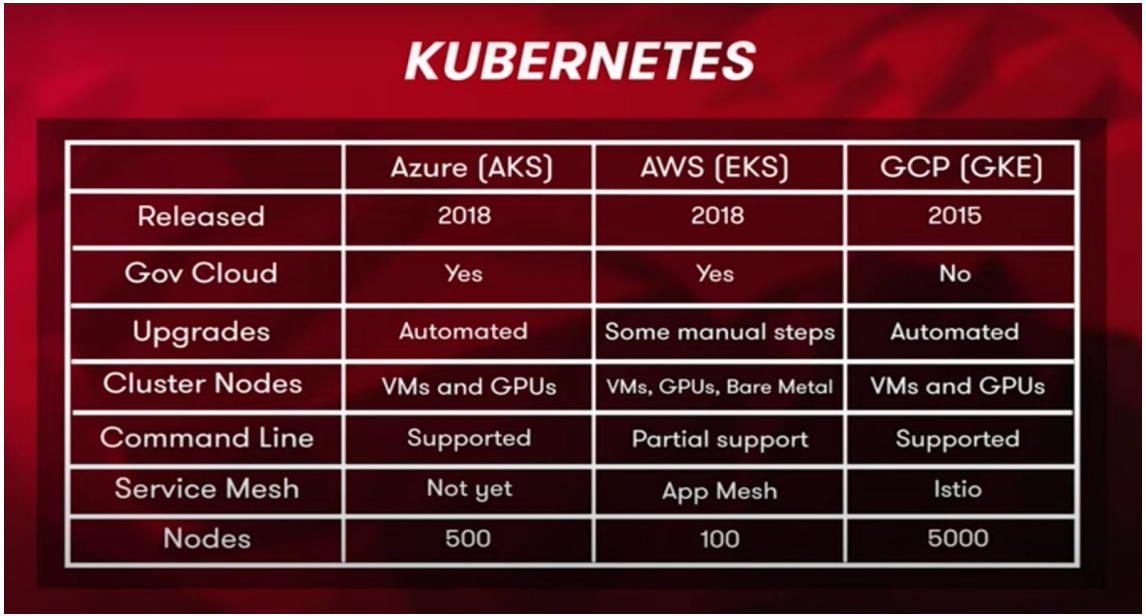
Used Tech

GitLab

argo

kubernetes

AKS vs EKS vs GKE



구글 GCP가 쿠버네티스를 가장 오래 서비스 했으며,
쿠버네티스가 안정적으로 지원할 것으로 생각되었으며,
결정적으로 타 클라우드 플랫폼보다 UI가 상당히 보기 편했습니다.

Jenkins vs Gitlab

Jenkins

Java 기반 오픈소스로 구성되어있고, 가장 대중적으로 사용되고 있으며, 다양한 플러그인을 지원하고 문서화가 잘 되어있습니다. 호스팅을 직접 하기 때문에 CI/CD 프로세스를 직접 관리할 수 있지만, 호스팅 비용이 발생하게 됩니다. 그리고 플러그인이 많기도 하고, 간혹 플러그인 간에 의존성 문제도 발생하게 됩니다.

GitLab

Gitlab에는 CI/CD 파이프라인을 빠르게 설정하는 데 도움이 되는 CI/CD 기능이 내장되어 있습니다. Auto DevOps는 소프트웨어 제공을 쉽고 효율적이며 자동화하는 Gitlab의 또 다른 강력한 기능입니다. 코드 언어를 스캔하고 작업을 자동으로 빌드, 테스트 및 배포하는 사전 정의된 기본 CI/CD 템플릿을 사용합니다.
Gitlab은 400분/월 무료로 이용 가능합니다

What is ArgoCD?

Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes.

쿠버네티스를 위한 CD(Continuous Delivery)툴입니다.
GitOps방식으로 관리되는 **Manifest 파일의 변경사항을 감시**하며, 현재 배포된 환경의 상태와 Git에 정의된 Manifest 상태를 **동일하게 유지**하는 역할을 수행 합니다.

What is FastAPI?

FastAPI framework, high performance, easy to learn, fast to code, ready for production

파이썬 3.6 부터 제공되는 파이썬 프레임워크입니다.
요즘 트렌드는 마이크로서비스가 트렌드입니다. 그러기 위해선 많은 데이터를 처리해야합니다. FastAPI는 기본적으로 **비동기**를 지원하며, **결론적으로 빠르게 만들 수 있기 때문입니다.**

Create FastAPI File

Install FastAPI

Command

```
1       $ pip3 install fastapi

2       $ pip3 install uvicorn
```

Code

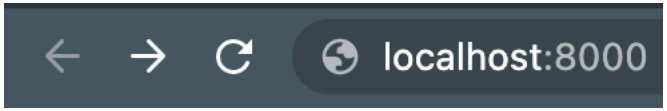
main.py

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
return {”HELLO": ”CI/CD"} #출력 화면
```

Start Server

main.py

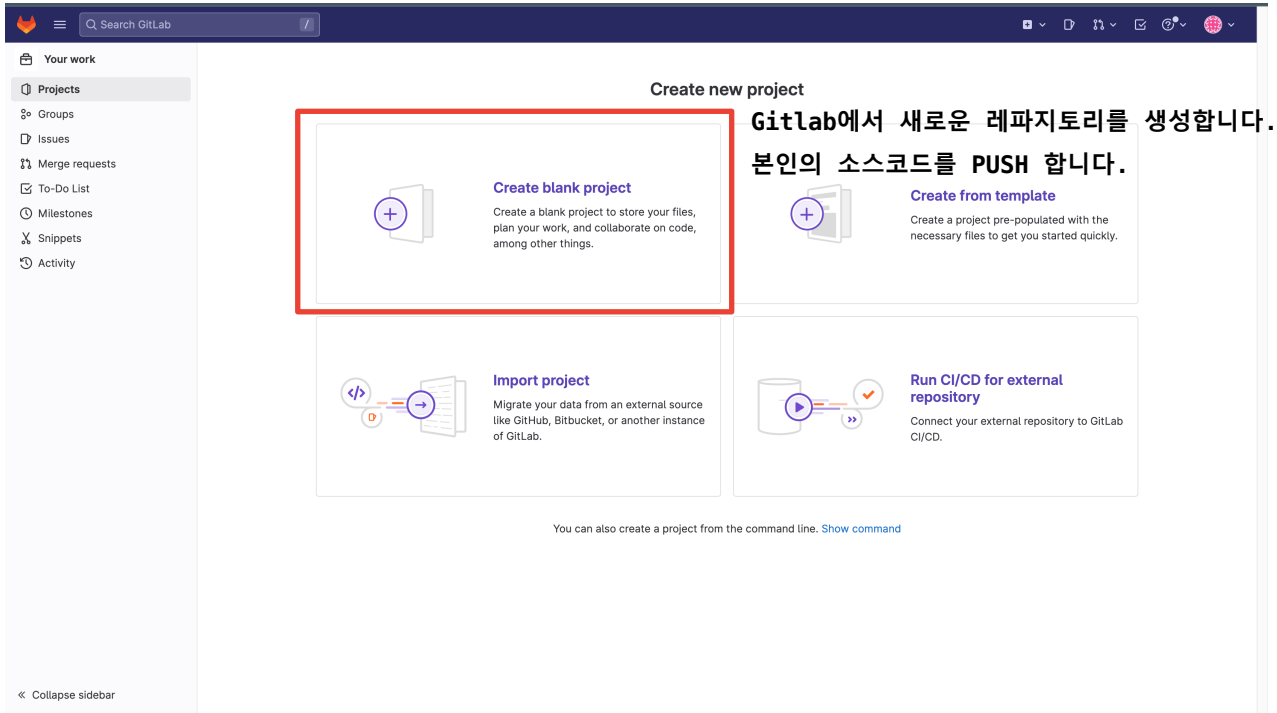
```
1    $ uvicorn --host=0.0.0.0 --port 8000 main:app
```



{"HELLO":"CI/CD"}

Gitlab Pipeline

Create Repository



Gitlab에서 새로운 레파지토리를 생성합니다.

본인의 소스코드를 PUSH 합니다.

Create Repository

CI/CD templates

Get started with GitLab CI/CD and your favorite programming language or framework by using a `.gitlab-ci.yml` template.

When you create a `.gitlab-ci.yml` file in the UI, you can choose one of these templates:

- Android (`Android.gitlab-ci.yml`)
- Android with fastlane (`Android-Fastlane.gitlab-ci.yml`)
- Bash (`Bash.gitlab-ci.yml`)
- C++ (`C++.gitlab-ci.yml`)
- Chef (`Chef.gitlab-ci.yml`)
- Clojure (`Clojure.gitlab-ci.yml`)
- Composer `Composer.gitlab-ci.yml`
- Crystal (`Crystal.gitlab-ci.yml`)
- Dart (`Dart.gitlab-ci.yml`)
- Django (`Django.gitlab-ci.yml`)
- Docker (`Docker.gitlab-ci.yml`)
- dotNET (`dotNET.gitlab-ci.yml`)
- dotNET Core (`dotNET-Core.gitlab-ci.yml`)
- Elixir (`Elixir.gitlab-ci.yml`)
- Flutter (`Flutter.gitlab-ci.yml`)
- Golang (`Go.gitlab-ci.yml`)
- Gradle (`Gradle.gitlab-ci.yml`)
- Grails (`Grails.gitlab-ci.yml`)
- iOS with fastlane (`iOS-Fastlane.gitlab-ci.yml`)
- Julia (`Julia.gitlab-ci.yml`)
- Laravel (`Laravel.gitlab-ci.yml`)
- LaTeX (`LaTeX.gitlab-ci.yml`)
- Maven (`Maven.gitlab-ci.yml`)
- Mono (`Mono.gitlab-ci.yml`)
- npm (`npm.gitlab-ci.yml`)
- Node.js (`Nodejs.gitlab-ci.yml`)
- OpenShift (`OpenShift.gitlab-ci.yml`)
- Packer (`Packer.gitlab-ci.yml`)
- PHP (`PHP.gitlab-ci.yml`)
- Python (`Python.gitlab-ci.yml`)
- Ruby (`Ruby.gitlab-ci.yml`)
- Rust (`Rust.gitlab-ci.yml`)
- Scala (`Scala.gitlab-ci.yml`)
- Swift (`Swift.gitlab-ci.yml`)
- Terraform (`Terraform.gitlab-ci.yml`)
- Terraform (`Terraform.latest.gitlab-ci.yml`)

Gitlab에서 CI/CD의 기능을 사용하기 위해서 “.gitlab-ci.yml”이라는 파일에서 JOB을 코딩하여 CI/CD기능을 이용할 수 있습니다.

https://docs.gitlab.com/ee/ci/examples

Create .gitlab-ci.yml

.gitlab-ci.yml

```
docker-build:
# Use the official docker image.
image: docker:latest
stage: build
services:
- docker:dind
before_script:
- echo "$CI_REGISTRY_PASSWORD" | docker login --username "$CI_REGISTRY_USER" --
password-stdin
script:
- |
if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
tag=""
echo "Running on default branch '$CI_DEFAULT_BRANCH': tag = 'latest'"
else
tag=":$CI_COMMIT_REF_SLUG"
echo "Running on branch '$CI_COMMIT_BRANCH': tag = $tag"
fi
- docker build --pull -t "$CI_REGISTRY_IMAGE${tag}" .
- docker push "$CI_REGISTRY_IMAGE${tag}"
# Run this job in a branch where a Dockerfile exists
rules:
- if: $CI_COMMIT_BRANCH
exists:
- Dockerfile
```

기본 docker 템플릿입니다.

미리 정의된 변수들을 활용하여 코딩을 할 수 있습니다.

다른 변수들은 여기서 추가로 볼 수 있습니다

그냥 쓰면 docker 이미지버전이 latest가 형상관리하기

어렵기 때문에 버전을 바꿔줄 필요가 있다고 생각했습니다.

Edit .gitlab-ci.yml

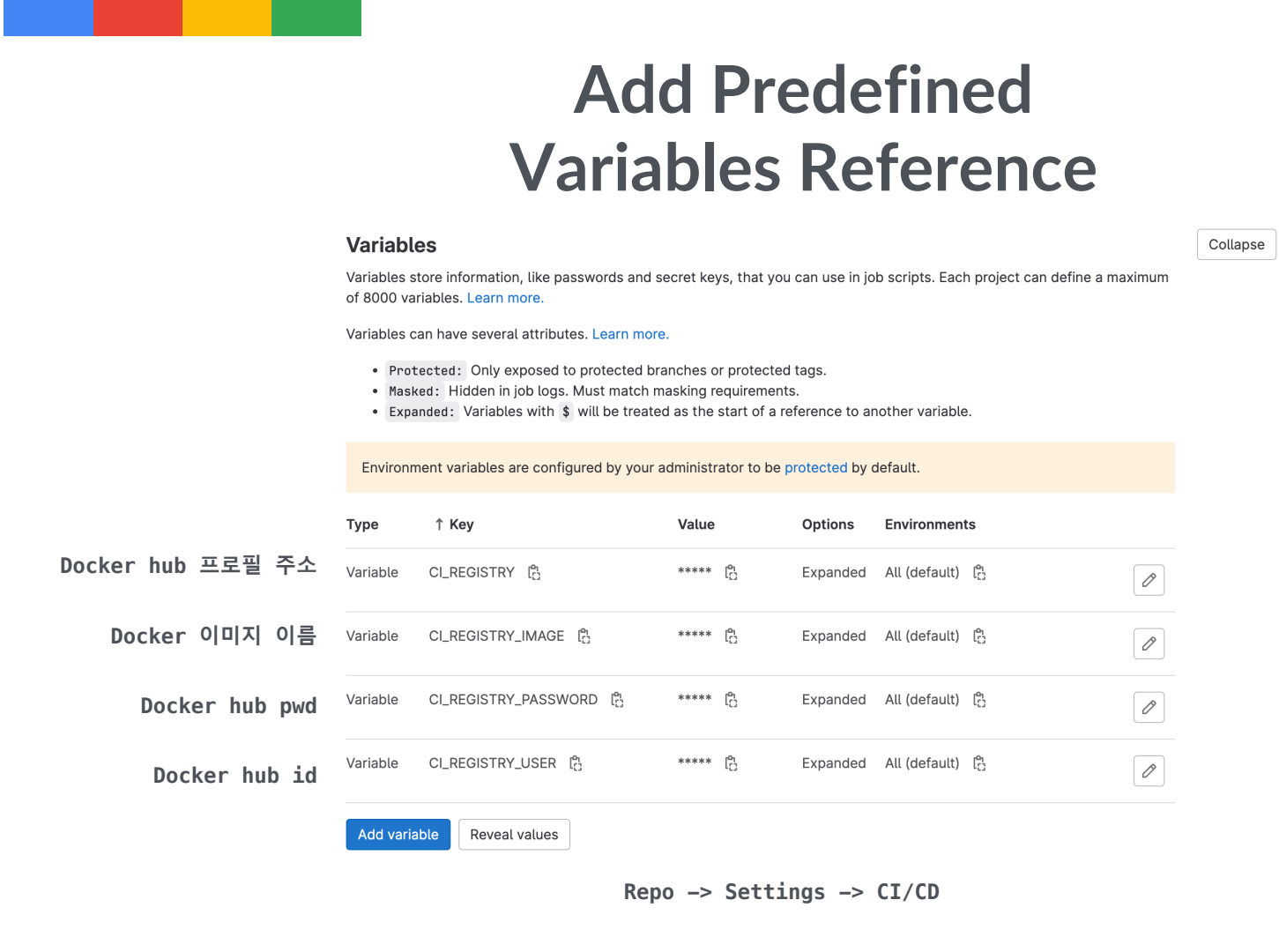
.gitlab-ci.yml

```
docker-build:
# Use the official docker image.
image: docker:latest
stage: build
services:
- docker:dind
before_script:
- echo "$CI_REGISTRY_PASSWORD" | docker login --username "$CI_REGISTRY_USER" --
password-stdin
script:
- |
if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
tag=""
echo "Running on default branch '$CI_DEFAULT_BRANCH': tag = 'latest'"
else
tag=":$CI_COMMIT_REF_SLUG"
echo "Running on branch '$CI_COMMIT_BRANCH': tag = $tag"
fi
- docker build --pull -t "$CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA" .
- docker push "$CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA"

# Run this job in a branch where a Dockerfile exists
rules:
- if: $CI_COMMIT_BRANCH
exists:
- Dockerfile
```

"CI_COMMIT_SHORT_SHA"는 commit했을 때 commit hash의 앞 8자리로 이미지를 만들게 코딩했습니다.

Docker 이미지버전을 commit hash로 하면 어떤 업데이트가 이루어 졌는지 형상관리하기 편할 것이라고 생각했습니다.

Add Predefined Variables Reference



Docker hub 프로필 주소

Docker 이미지 이름

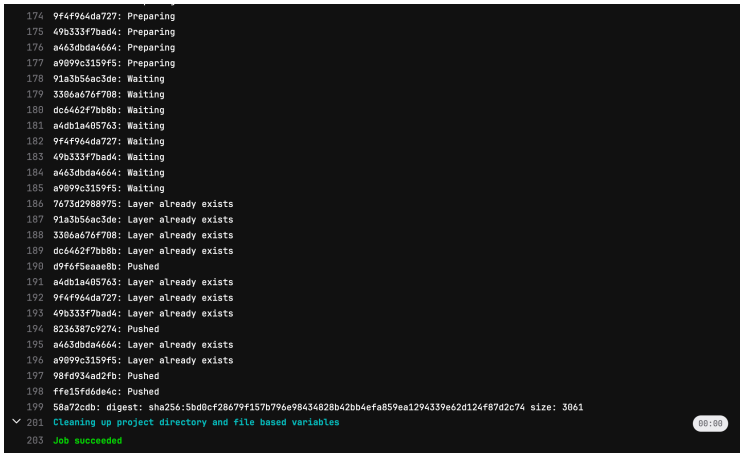
Docker hub pwd

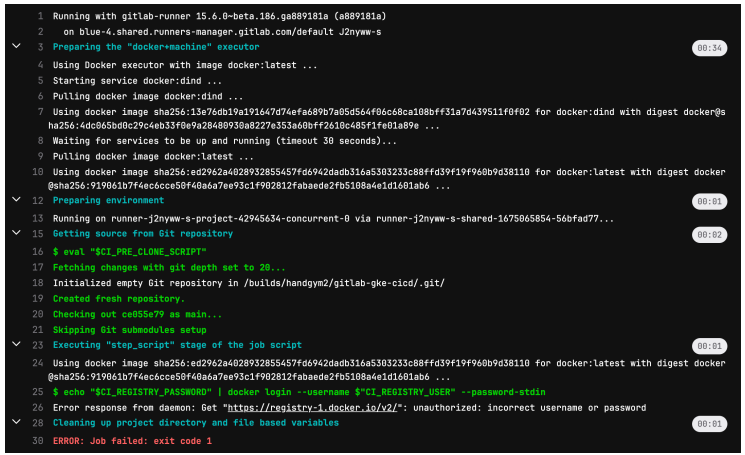
Docker hub id

Repo -> Settings -> CI/CD

Check CI/CD

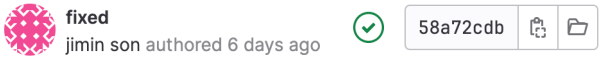
Commit을 했을 때 파이프라인 내부 빌드 과정을 확일 할 수 있으며
오류가 생겼을 때 어디서 오류가 났는지 확인할 수 있습니다.



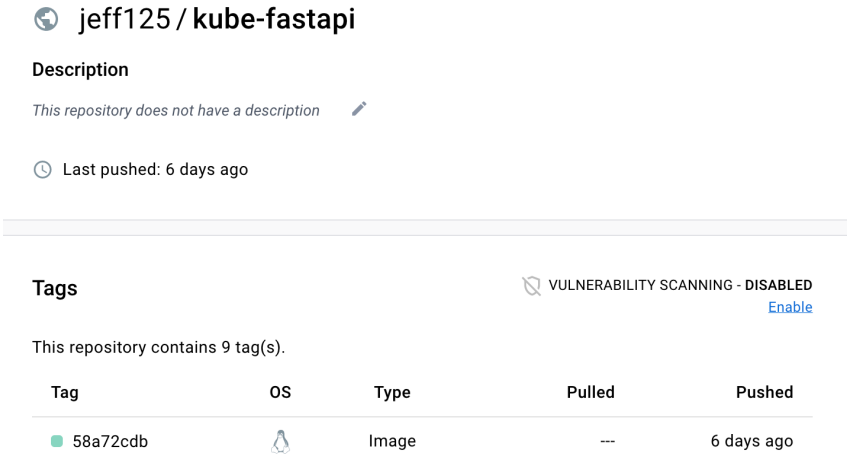


Repo -> CI/CD -> Pipelines

Check CI/CD

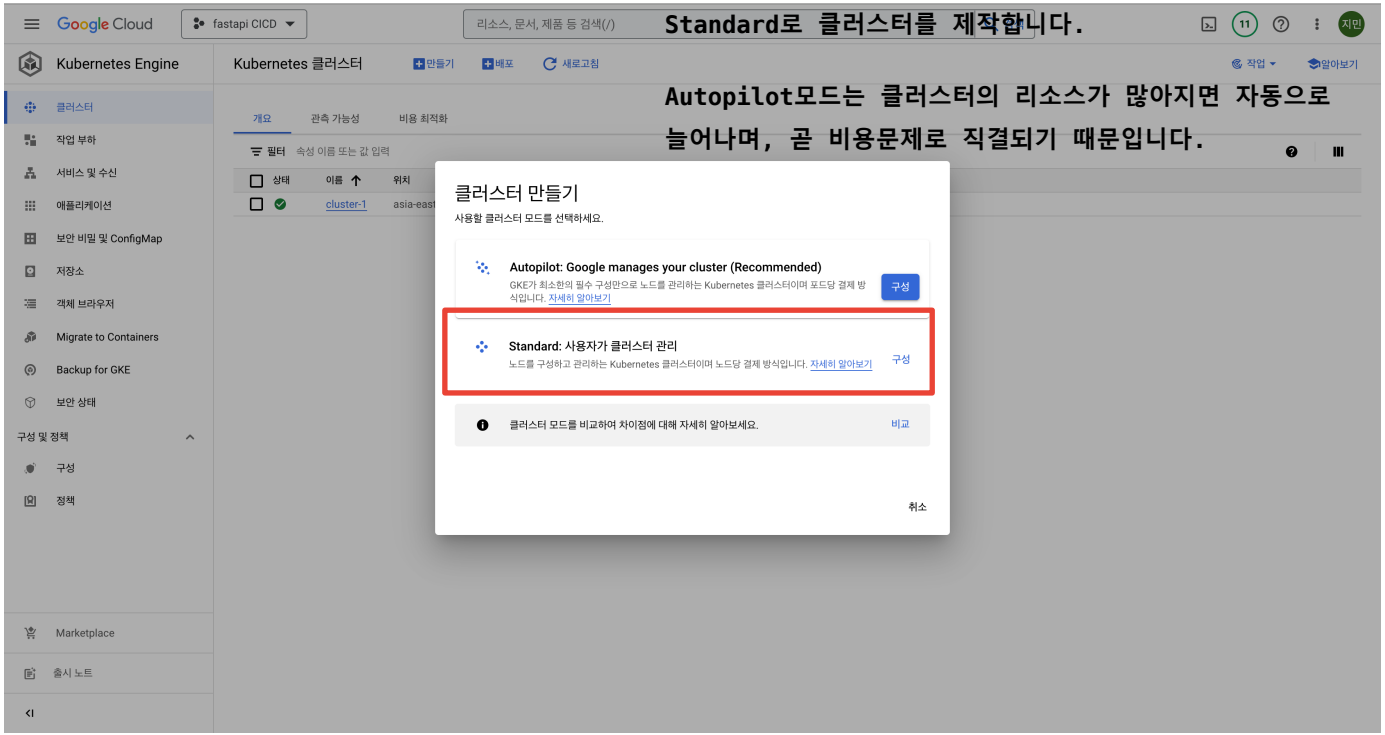


Commit을 했을 때 commit hash 값과
이미지버전이 같은 모습을 확인 할 수 있습니다.



Create Cluster

Create Cluster



Standard로 클러스터를 제작합니다.

Autopilot모드는 클러스터의 리소스가 많아지면 자동으로 늘어나며, 곧 비용문제로 직결되기 때문입니다.

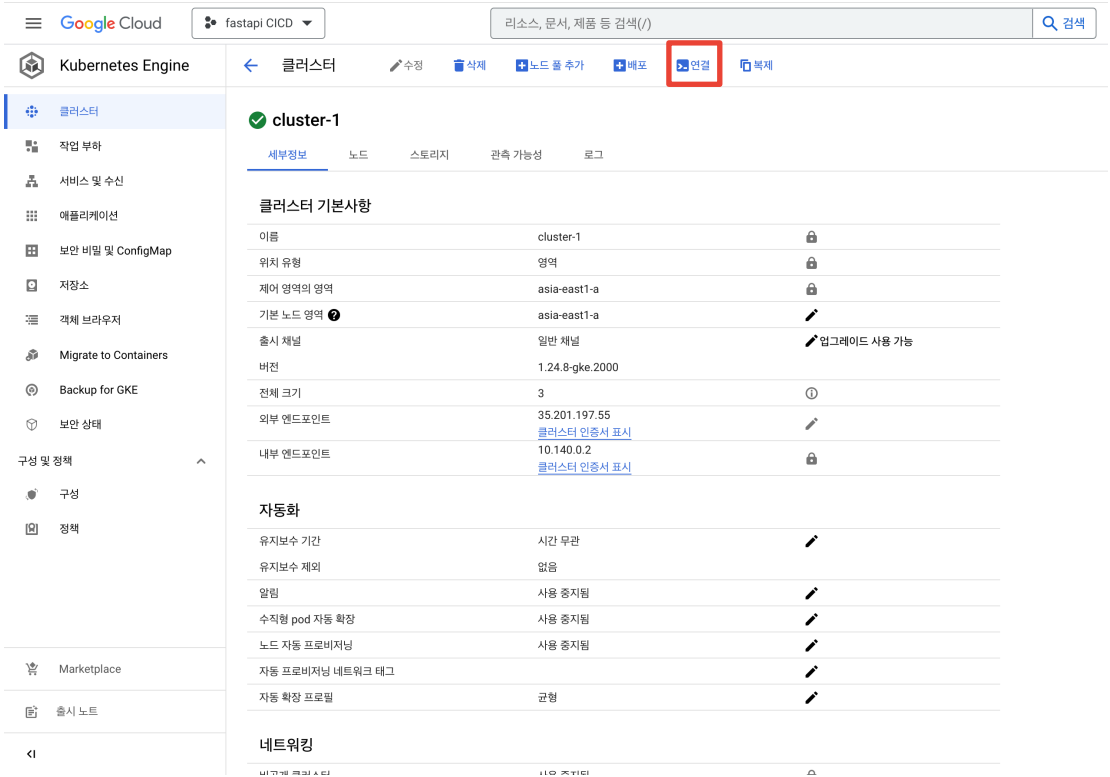
Create Cluster

이름, 영역 외에 건들이지 않았습니다.

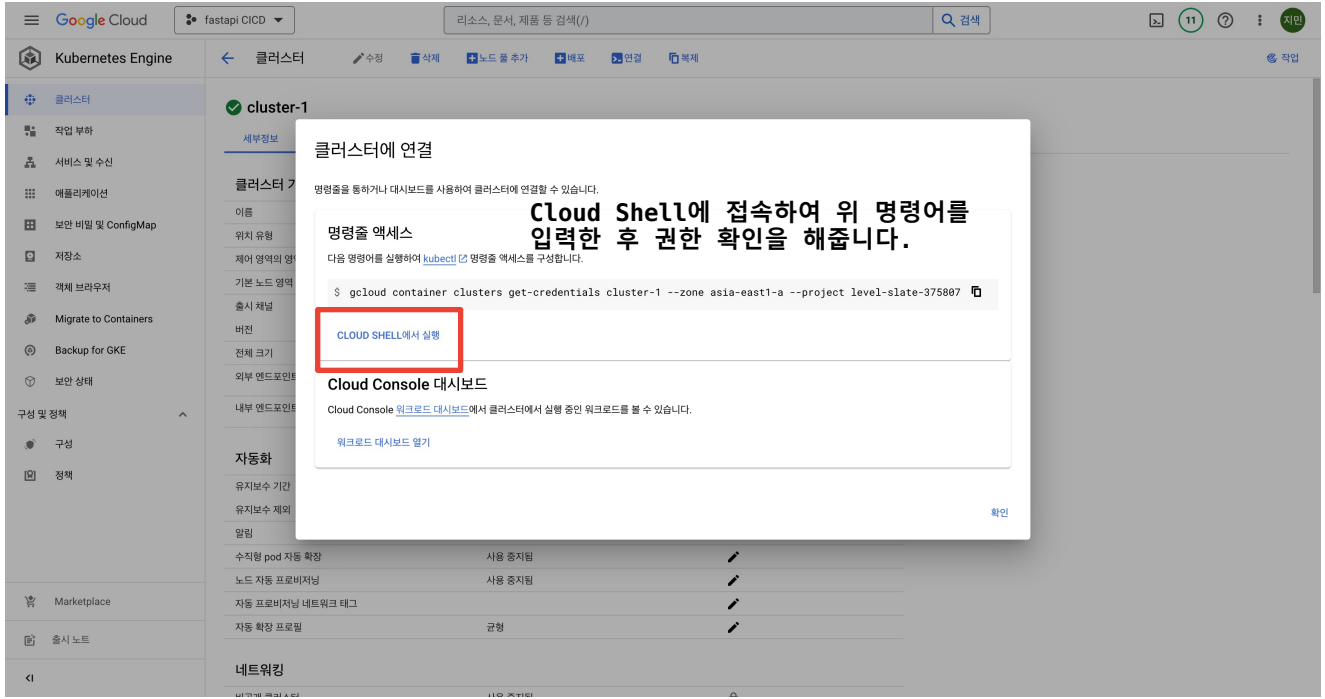
그 이유는 성능, 노드 개수 증가 등 있어 비용으로 청구됩니다.

Settings Kubernetes

Settings Kubernetes



Settings Kubernetes



Install ArgoCD

Patch yaml ArgoCD Service

CloudShell

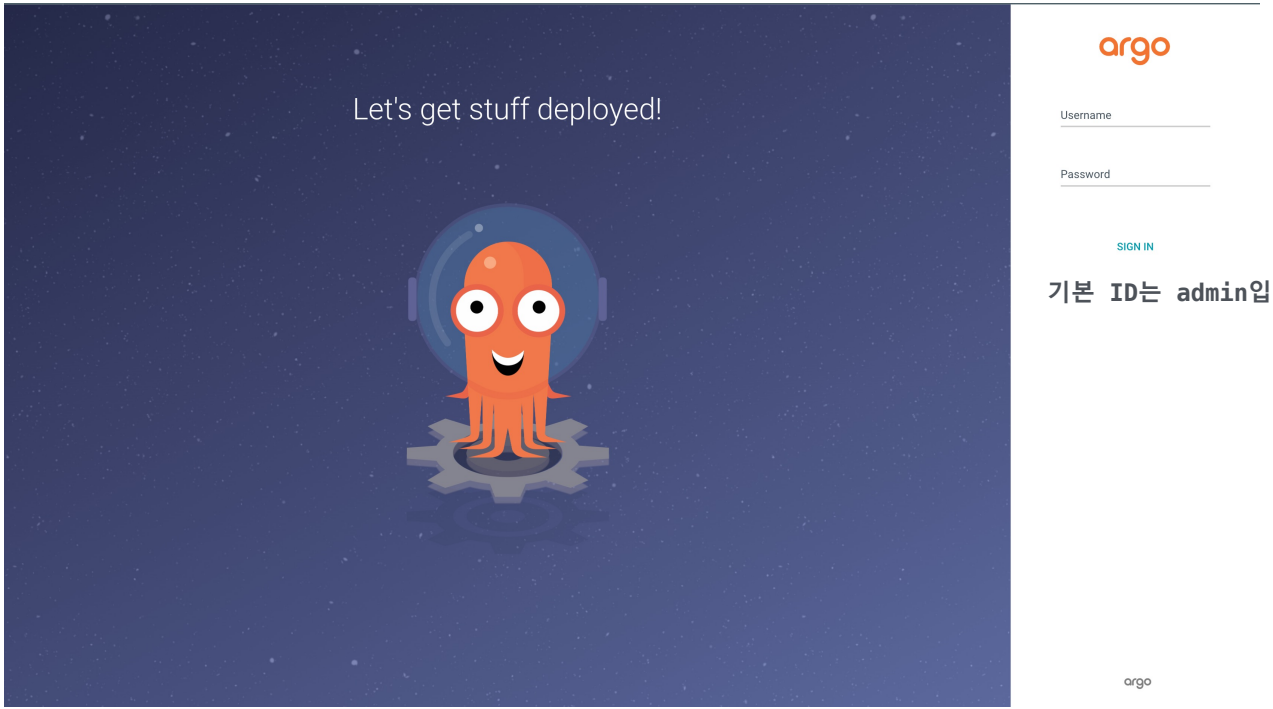
```
username@cloudshell:~ (project ID) $ kubectl get -n argocd
NAME                                  TYPE        CLUSTER-IP     EXTERNAL-IP     PORT(S)         AGE
argo-controller                       ClusterIP   10.76.0.137      <none>          9090/TCP        60s
argo-server                           ClusterIP   10.76.8.11       <none>          2746/TCP        60s
argo-workflows-postgresql-exporter-svc  ClusterIP   10.76.5.1        <none>          9187/TCP        60s
argo-workflows-1-postgresql-svc         ClusterIP   10.76.13.151     <none>          5432/TCP        60s

username@cloudshell:~ (project ID) $ kubectl patch svc argo-server -n argocd -p '{"spec" : {"type" :
"LoadBalancer"}}' #argo-server의 yaml의 spec/type/ClusterIP를 LoadBalncer 또는 NodePort로 수정합니다. 그 이유는 외부에서 접속을 해야하기 때문입니다.
service/argo-server patched

username@cloudshell:~ (project ID) $ kubectl get -n argocd
NAME                                TYPE          CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
argo-server                         LoadBalncer   10.76.8.11    <Your Extenal IP>   2746:31922/TCP    10m
```

Get ArgoCD Admin PWD

CloudShell

```
username@cloudshell:~ (project ID) $ kubectl get secrets argocd-initial-admin-secret -n argocd
--template={{.data.password}} | base64 -d; echo
<decode your argocd password>
```

Login ArgoCD



Your argo-server Extenal IP

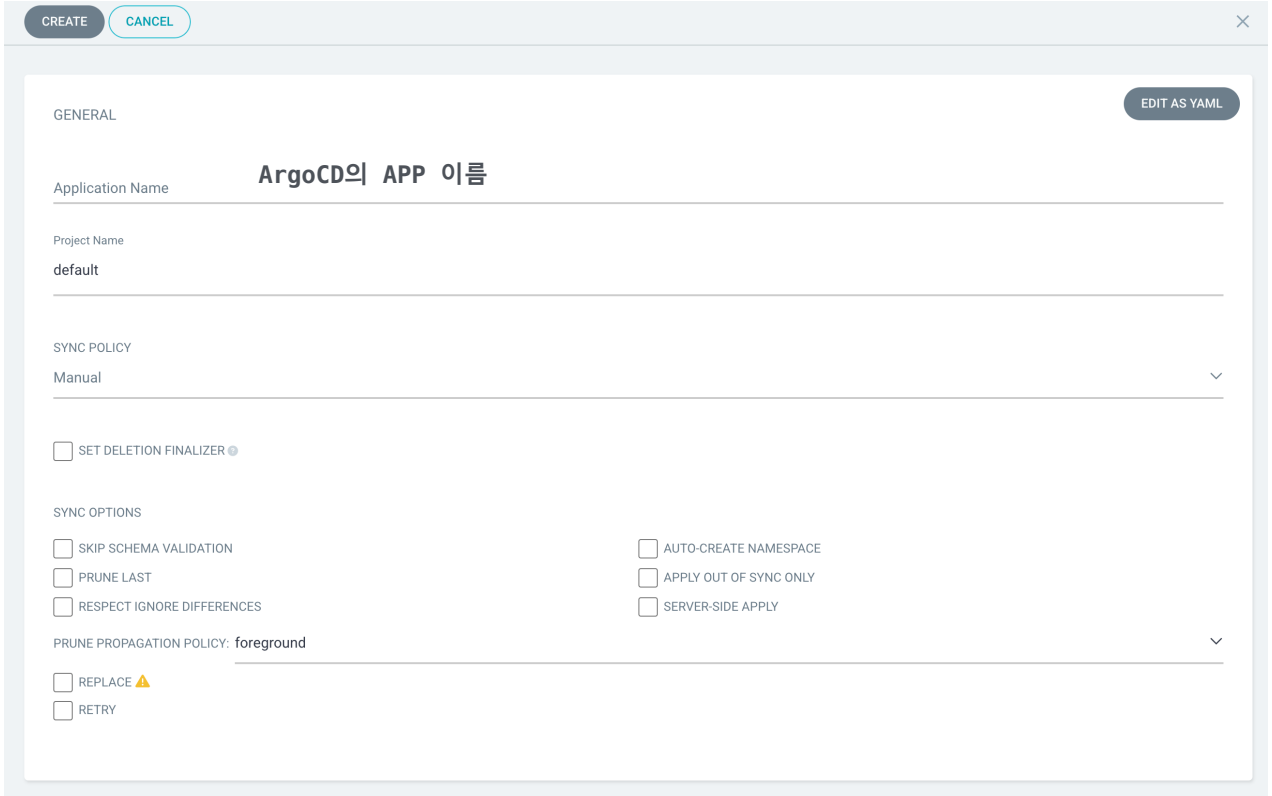
Settings ArgoCD

Create Manifest

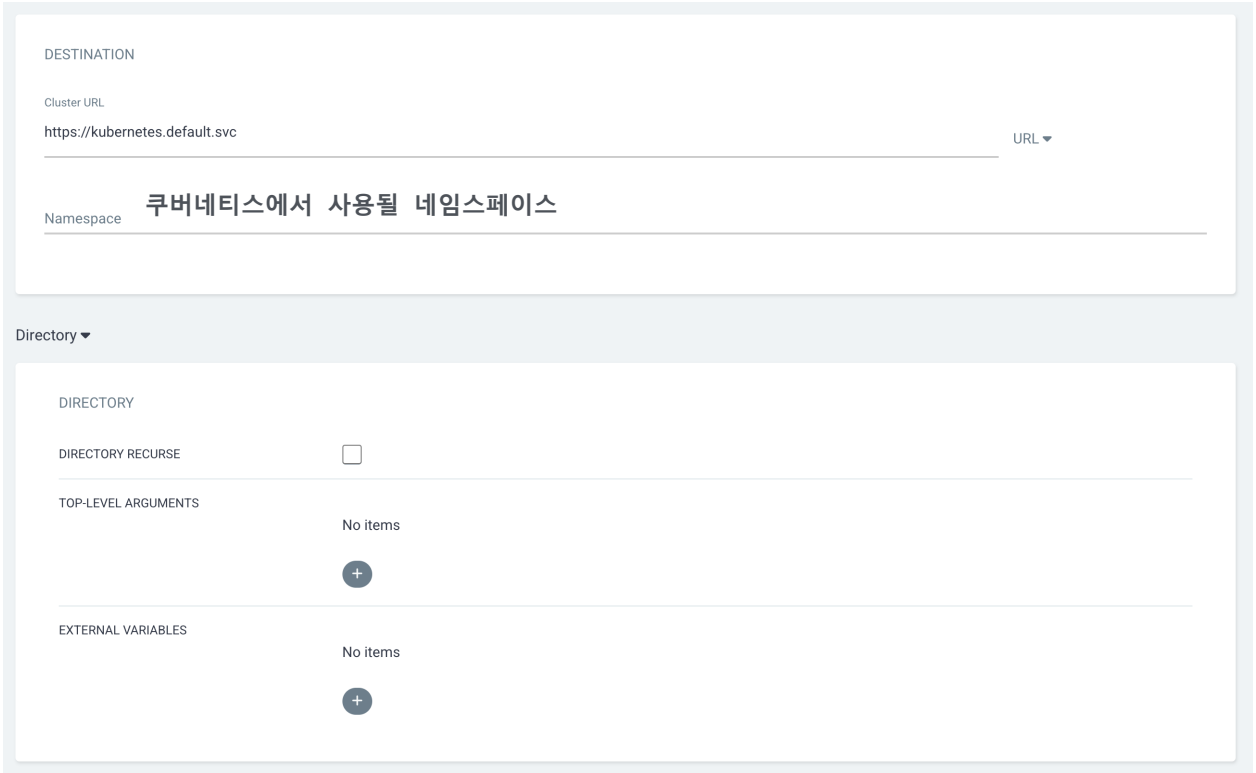
deploy-fastapi.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: fastapi
  name: fastapi
spec:
  selector:
    matchLabels:
      app: fastapi
  template:
    metadata:
      labels:
       app: fastapi
    spec:
      containers:
      - name: fastapi
        image: jeff125/kube-fastapi:58a72cdb
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 8000
                    .
                    .
                    .
---
apiVersion: v1
kind: Namespace
metadata:
  name: fastapi
---

apiVersion: v1
kind: Service
metadata:
  name: fastapi-service
spec:
  type: LoadBalancer
  selector:
    app: fastapi
  ports:
  - port: 8000
    targetPort: 8000
```

Create APP


CREATE
CANCEL
GENERAL
EDIT AS YAML
Application Name
ArgoCD의 APP 이름
Project Name
default
SYNC POLICY
Manual
SET DELETION FINALIZER
SYNC OPTIONS
SKIP SCHEMA VALIDATION
PRUNE LAST
RESPECT IGNORE DIFFERENCES
AUTO-CREATE NAMESPACE
APPLY OUT OF SYNC ONLY
SERVER-SIDE APPLY
PRUNE PROPAGATION POLICY:
foreground
REPLACE
RETRY

Create APP


DESTINATION
Cluster URL
https://kubernetes.default.svc
URL
Namespace
쿠버네티스에서 사용될 네임스페이스
Directory
DIRECTORY
DIRECTORY RECURSE
TOP-LEVEL ARGUMENTS
No items
EXTERNAL VARIABLES
No items

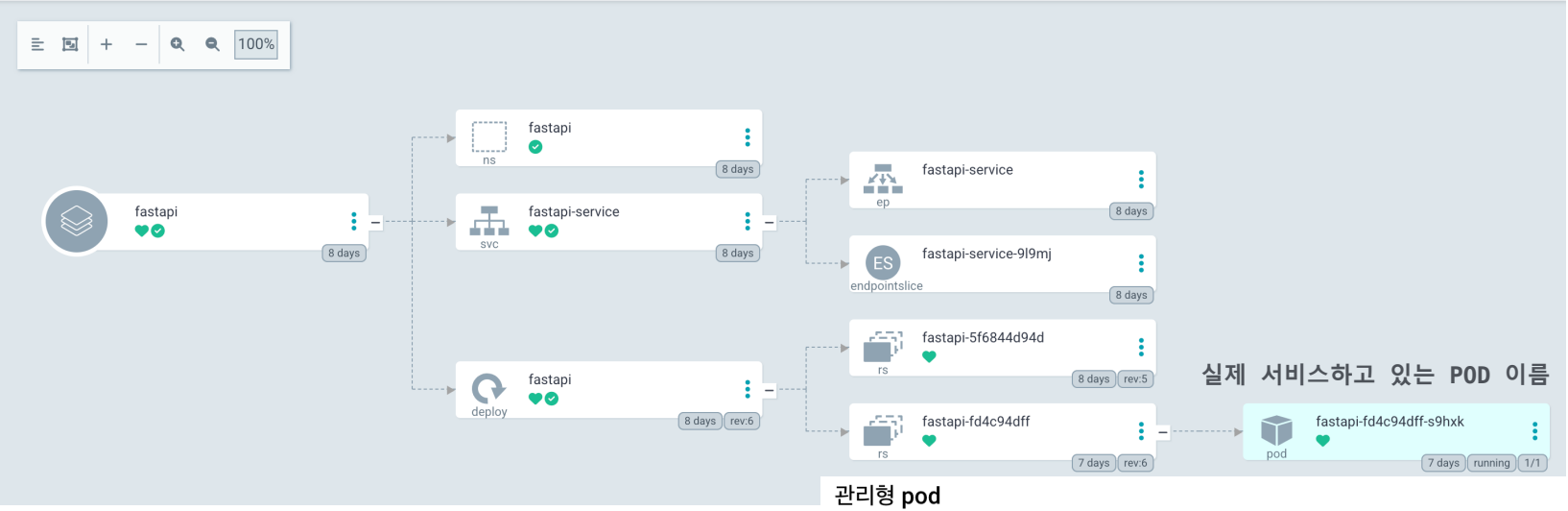
Check Sync

마지막 commit hash를 확인 할 수 있다.

APP HEALTH
Healthy

CURRENT SYNC STATUS
Synced
MORE
To HEAD (58a72cd)
Author: handgym2 <k36470125@gmail.com> -
Comment: fixed

LAST SYNC RESULT
Sync OK
MORE
To 58a72cd
Succeeded 7 days ago (Thu Feb 02 2023 13:52:43 GMT+0900)
Author: handgym2 <k36470125@gmail.com> -
Comment: fixed



실제 서비스하고 있는 POD 이름

관리형 pod

수정 버전	이름	상태	재시작	생성일 ↑
6	fastapi-fd4c94dff-s9hxk	Running	0	2023. 2. 2. PM 1:52:43

Monitoring GKE

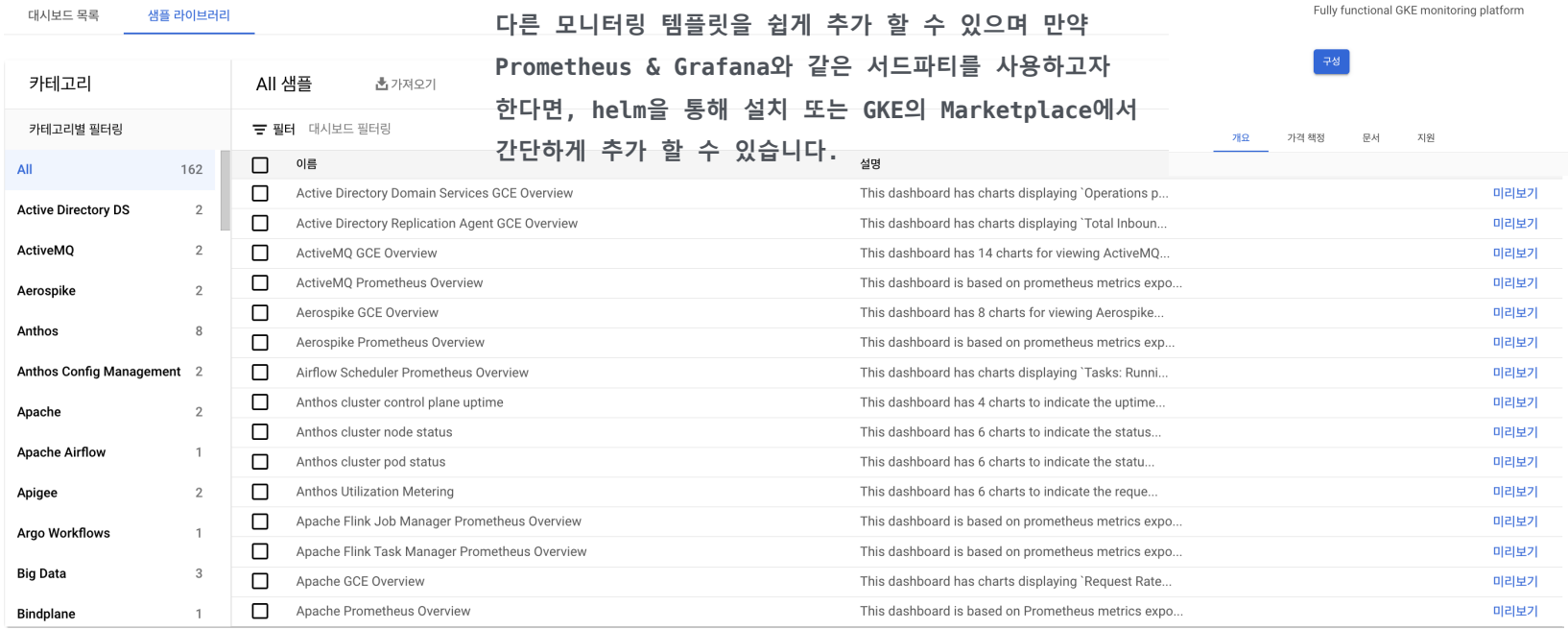
Default Dashboard

GKE에서 자주 쓰는 항목들을 대시보드로 템플릿을 미리 만들어서
사용하면 편합니다. 또한 Prometheus & Grafana를 써도 좋지면
리소스를 잡아 먹기 때문에 GKE에서 기본 지원하는 템플릿을 썼습니다.

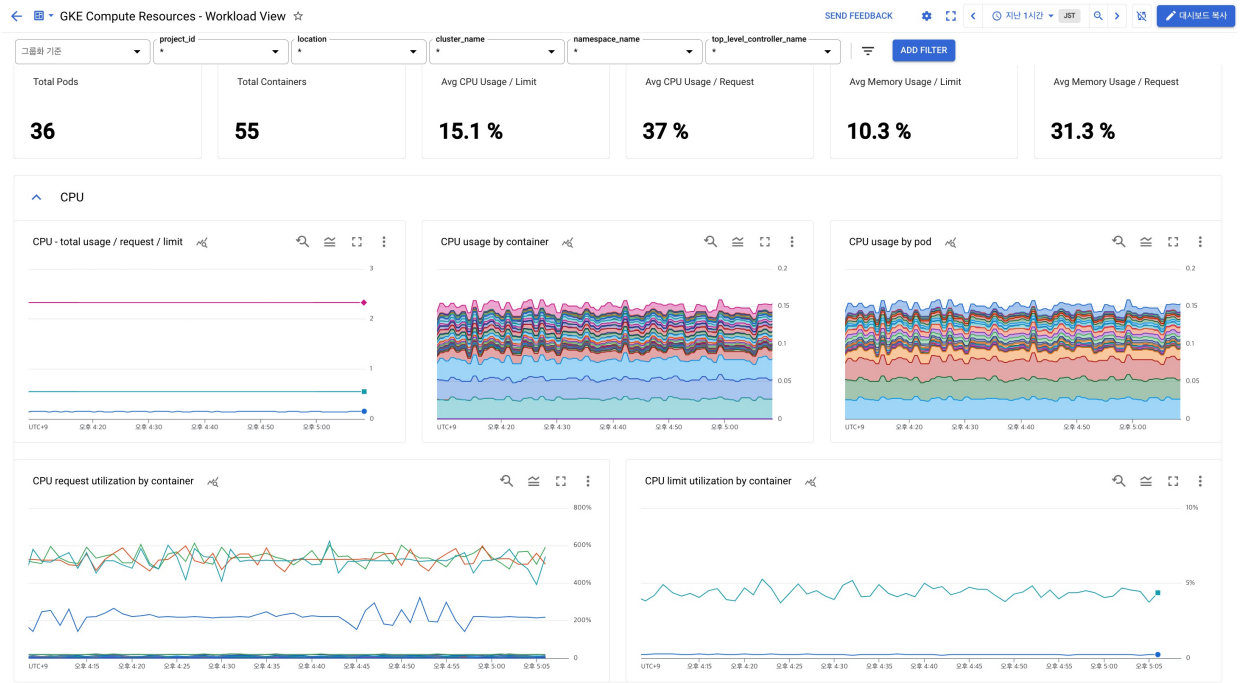
모니터링 -> 대시보드

Custom Dashboard

다른 모니터링 템플릿을 쉽게 추가 할 수 있으며 만약 Prometheus & Grafana와 같은 서드파티를 사용하고자 한다면, helm을 통해 설치 또는 GKE의 Marketplace에서 간단하게 추가 할 수 있습니다.



Metrics Dashboard



이 대시보드는 GKE Cluster의 Metrics를 확인 할 수 있습니다.
또한 Node, POD 등의 Metrics를 확인 할 수 있는 템플릿도 따로 있습니다

GCP의 UI/UX가 상당히 잘 만들어졌으며, 검색을 많이 안하고도 GKE를 사용했습니다. 또한 따로 SSH 접속 할 필요 없이 Web SSH로도 설정이 가능했던 점이 굉장히 편했습니다.

Gitlab의 CI/CD를 구축 했을 때 Gitlab의 기능들을 눌러보며 연구해봤으며 그 중에서도 직접 쿠버네티스, GCP에 연동하기 쉽게 한 것도 엄청난 장점이라고 생각했습니다.

ArgoCD를 설정했을 때 “진짜 쿠버네티스를 위해서 만든건가 보네” 라는 생각을 했습니다. Manifest를 Update만 하면 GKE의 POD들이 교체가 되는 것을 보고 정말 짜릿했습니다.