

AI Service Infrastructure

터지지 않게 만드는 AI 서비스 엔지니어링 생존 전략

Who is Speaker?



송준호 (PangPang)

AI Engineer

I Braincrew Inc. AI Engineer

프로덕트 개발 및 다양한 서비스 인프라 배포

- LLM Application
- DevOps



github



Linkedin

| PART 01

Opening

01

2026년은 AI의 시대



torvalds committed 3 days ago

Merge branch 'antigravity'

This is Google Antigravity fixing up my visualization tool (which was also generated with help from google, but of the normal kind).

It mostly went smoothly, although I had to figure out what the problem with using the builtin rectangle select was. After telling antigravity to just do a custom RectangleSelector, things went much better.

Is this much better than I could do by hand? Sure is.

근데 왜 배포만 하면..

```

.....
대용량 PDF 비동기 처리 파이프라인
- 2000+ 페이지 PDF를 효율적으로 처리
- 메모리 효율적인 청크 기반 처리
- 병렬 처리로 성능 최적화
.....

import asyncio
import logging
from abc import ABC, abstractmethod
from concurrent.futures import ProcessPoolExecutor, ThreadPoolExecutor
from dataclasses import dataclass, field
from pathlib import Path
from typing import AsyncIterator, Callable

import fitz # PyMuPDF

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

@dataclass
class PageResult:
    """페이지 처리 결과"""
    page_number: int
    text: str
    images: list[bytes] = field(default_factory=list)
    tables: list[dict] = field(default_factory=list)
    metadata: dict = field(default_factory=dict)
    error: str | None = None

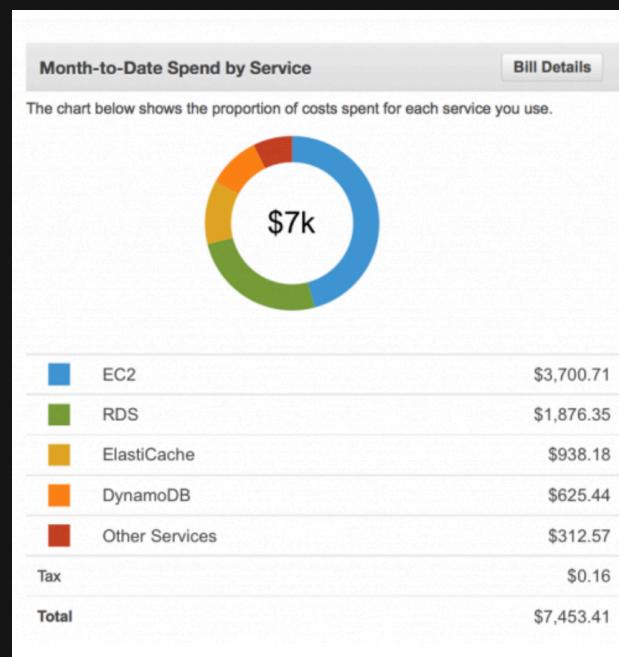
@dataclass
class PipelineConfig:
    """파이프라인 설정"""
    chunk_size: int = 50 # 한 번에 처리할 페이지 수
    max_workers: int = 4 # 병렬 워커 수
    extract_images: bool = True
    extract_tables: bool = False
    timeout_per_page: float = 30.0 # 페이지당 타임아웃 (초)

class PageProcessor(ABC):
    """페이지 처리기 추상 클래스"""
    @abstractmethod
    async def process(self, page_data: bytes, page_number: int) -> PageResult:
        pass

```

만들 때

- 깔끔한 코드
- Works on my Machine
- Localhost



배포 할 때

- 털리는 메모리
- 털리는 지갑
- 털리는 멘탈

AI 서비스의 문제가 뭘까?

- **무겁다**

LLM이 읽기 전에 전처리 단계에서 Out of Memory 발생

- **느리다**

사용자는 빠른 답변을 원한다

- **예측 불가능**

지갑을 털어가는 TPM/RPM

AI 서비스의 문제가 뭘까?

점점 무거워지는 Agent

Agent 고도화
(SubAgent)
(더 많은 tool)
(더 커진 State)



Tool 실행 자원은 어떻게?

실행 시간이 길어지면 State는?

Tool 사용하는데 비용은 얼마나?

Agent 별 디버깅은 어떻게 하지?

| PART 02

서비스의 배신

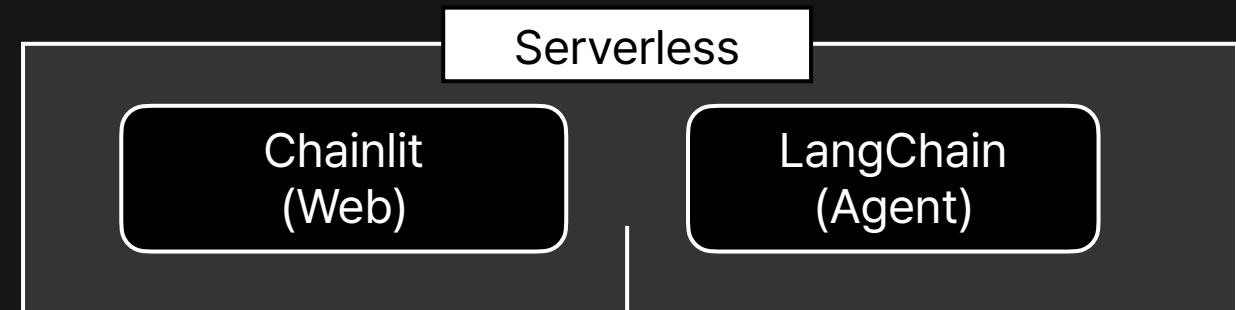
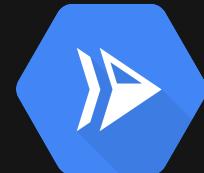
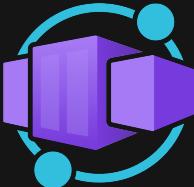
02

서비스야 구해줘

“인프라 고민할 시간에
Agent 하나 더”



AWS Lambda



PoC App

서비스의 배신

서비스를 이용한 LLM App의 단점은?

“인프라 고민할 시간에
Agent 하나 더”



- Cold Start
 - Warm start가 Readiness를 의미하지는 않는다
- Chainlit Stateful vs Ephemeral Serverless
 - 유지 되어야 할 State와 사라지는 Server
- 자원 경합 문제 / Burst 한계
 - 급격한 트래픽 증가 시 후속 Instance 작동 문제 발생
 - 스케일링은 되는데 사용성은 ↓

교훈

- **AI Agent 배포에 서비스는 적합하지 않다**

점유 시간이 길고, 상대적으로 무거운 library들 Initialize

- **서비스는 인스턴스의 확장성만 보장**

각각 App들의 빠른 반응속도를 보장하지는 않는다

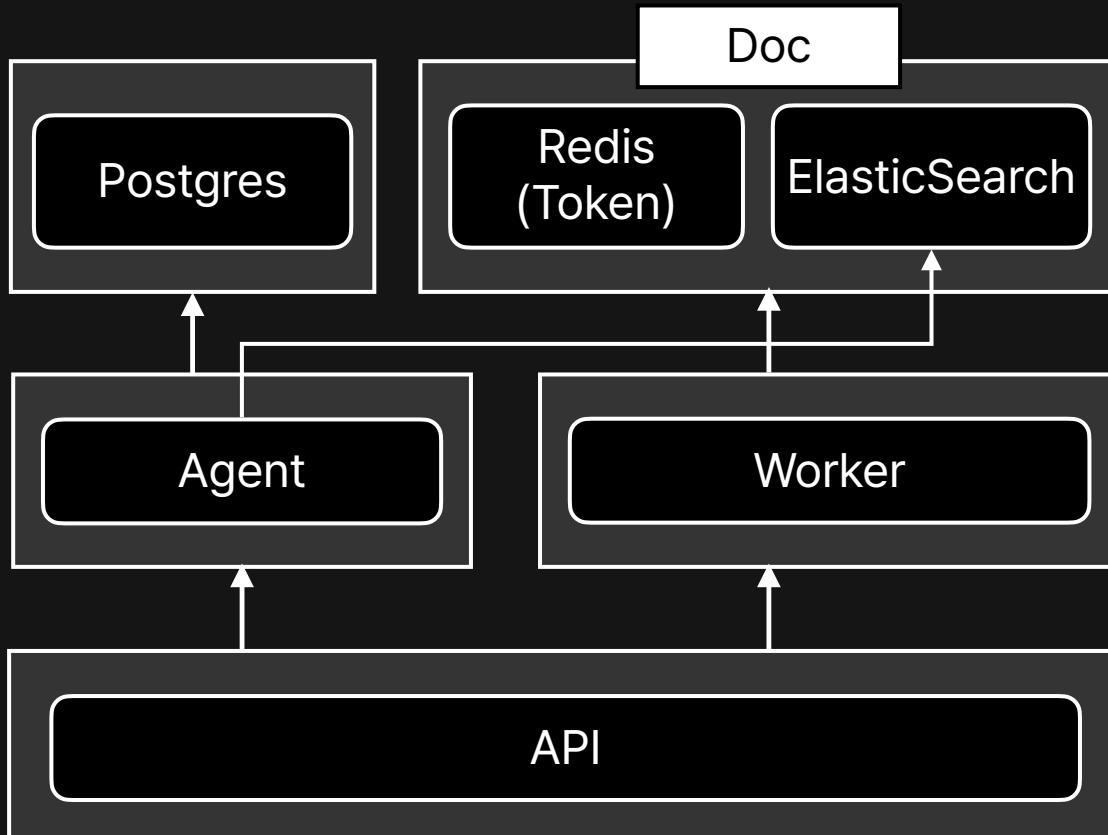
AI App의 사용성은 “빠른 답변”

| PART 03

Virtual Machine

03

VM으로 돌아와서..



- **Serverless의 떨어지는 사용성 때문에**
새로운 서버 배포 할 때는 24시간 자원 고정 할당된 VM으로 하자!

- **Document Worker/ Vectorstore / API / Web**
분리하여 각 자원의 부하가 간섭하지 않게 격리

→ 서비스 자원 사용을 “계산 가능하도록” 만들자

VM으로 돌아와서..

- 문서 처리 시 병목

단순 Queue로 하면, 한명은 2,000p짜리 넣고 한명은 1p짜리 넣어도 순차 처리
1p짜리 문서 넣은 사람은 사용성 매우 떨어짐

Parser API RPM(Rate per Minute)

처리할 수 있는 분당 용량에 한계 존재

VM으로 돌아와서..

문서 처리 병목 해결

문제: 2,000page pdf 넣은 유저 → 1장 넣은 유저(라이트)가 20분 대기 고르지 않은 트래픽 발생

- **해결:** 트래픽 고르게 발생하도록 나눔

Heavy Lane (1차선): 200장 이상의 대형 작업 전용 워커 배정

Light Lane (2차선): 1~100장 이내의 가벼운 작업 전용 워커 상시 대기

문제: 무작정 쏘면 Parser Rate Limit, 털리는 지갑

- **해결:** Redis 기반 **Token Bucket** 구현
 - 분당 처리 가능한 토큰량을 계산하여 워커 물리적으로 제어.
- **결과:**
 - 예측 가능한 비용 범위 내에서 최대의 처리량(Throughput) 유지

교훈

VM으로 하나하나 설정해보면서 RAG 서비스에서 컴포넌트별 성능 최적화

- 기본적인 네트워크 개념 및 서비스 배포에 대한 구체화
- 급격한 트래픽 증가에 따른 대응 어려움 (Autoscale 부재)
- 세밀한 자원 할당이 어려움

현재 서비스는 사용자 베이스가 적어서 자원 고정해서 사용 가능했음

- 다음에 구축하는 서비스는 사용자 규모가 유동적이고, 급격한 트래픽 발생 가능성 존재

→ 좀 더 상세한 자원 처리 및 서비스 격리가 필요

| PART 04

Kubernetes

04

K8S 사용기

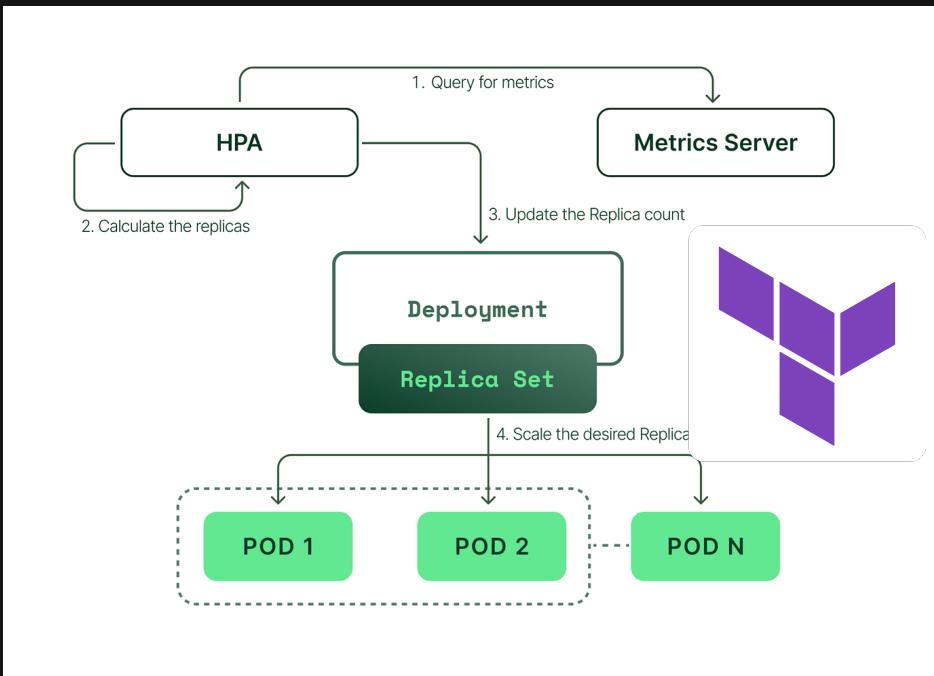
“K8S는 오버 엔지니어링 아니에요?”



기존에는 VM 사용 → Chat 서비스 특성 상 극단적인 트래픽 존재

- 유저가 언제 몰릴 지 알기 힘들다 (조사 목적으로 트래픽 변동성 매우 큼)
- CPU / MEM 사용량에 따라 컨테이너를 실시간으로 늘이고 줄이는 탄력성 필요

K8S 사용기



다양한 Agent 및 여러 State, 장기간 점유하는 Workflow 존재

- 동적으로 State 주입받아서 실시간 chat에 System Prompt 주입
- 복잡해지는 Agent 환경의 빠른 복제를 위해 K8S 도입

복잡한 .yaml config

- 더불어 늘어나는 config 관리를 위해 Terraform(IaC) 도입
- Human Error 최소화 및 운영 효율 극대화

교훈

- '잘 구축'한 k8s의 가능성
 '한 번만 고생하자'
- 모니터링의 중요성
 처음 마주한 모니터링 애로사항 (여러 Agent)

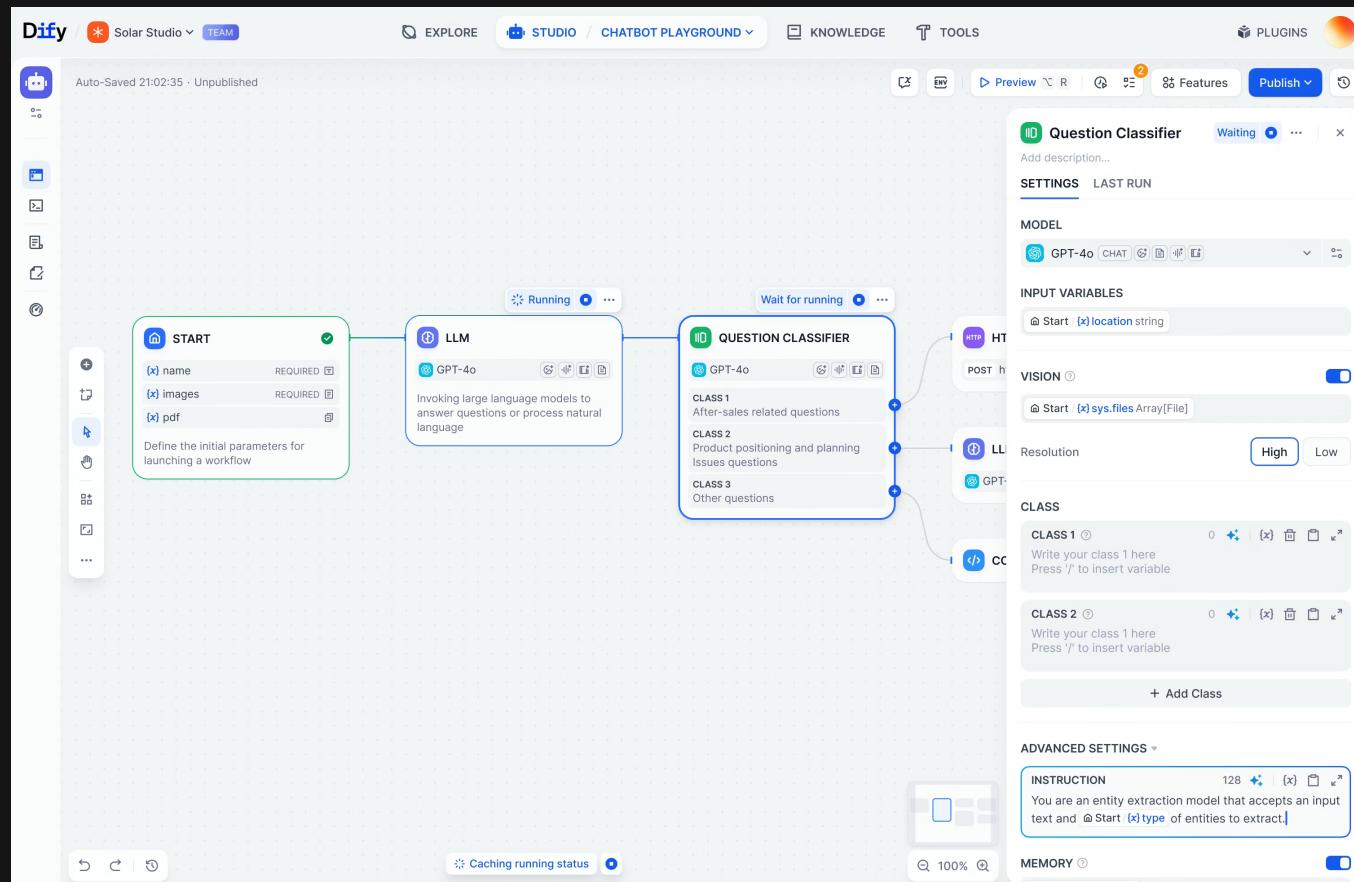
| PART 05

ECS with Dify

05

ECS with Dify

dify 마이그레이션 해보기



AI App Nocode Opensource Platform

- Version 업데이트
- DB 마이그레이션
- 서비스 파악 및 병목 해결

ECS with Dify

dify 이용해보기

K8S 쓰다가 갑자기 왜?

- Managed Service
- 10개 넘는 컴포넌트, DB 마이그레이션
- 모든 로직을 파악하고 있지는 않기 때문에 전략적인 선택 필요

ECS with Dify

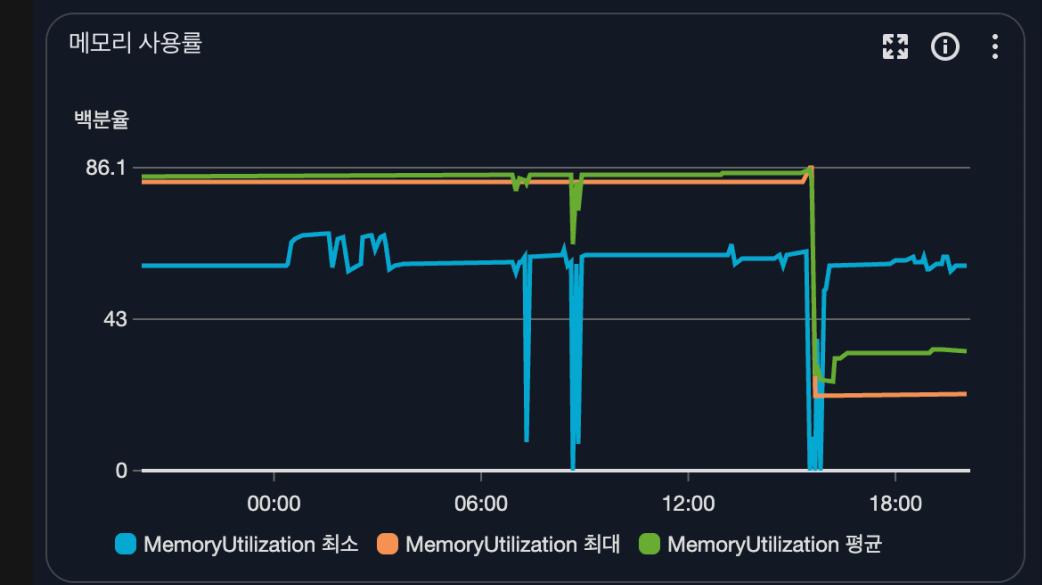
관리형 서비스의 편리함

CloudWatch



Amazon
CloudWatch

자원 누수 해결 사례



교훈

- **Managed Service의 편리함**
모니터링, 로깅
- **App 특성 및 내부 로직 파악 필요성**

PART 06

결론

06

최적의 인프라는 있나?

- **다양한 서비스를 구축해보면서 느낀 경험 공유**

직접 해보면서 느껴야 한다

- **서비스 특성별 최적의 인프라는 있는가?**

내가 배포하고 싶은 서비스를 잘 이해하는 것!

Q&A

Thank You

The background features a dark, solid black area on the left side. On the right side, there are several large, semi-transparent, organic-shaped overlays in various colors. These shapes include shades of orange, yellow, red, pink, purple, blue, and teal. They overlap each other, creating a sense of depth and motion. The overall aesthetic is modern and minimalist.