클라우드 기반 이미지 처리 파이프라인 구축

조민준, 김종원*
광주과학기술원 지스트 대학 전기전자컴퓨터공학부, *광주과학기술원 AI 대학원

minjun_jo@gm.gist.ac.kr, *jongwon@smartx.kr

A cloud-based image processing pipline

Minjun Cho and JongWon Kim
GIST (Gwangju Institute of Science & Technology)

요 약

본 논문은 클라우드 네이티브 환경에서 복수의 컨테이너를 효율적으로 배포하고 관리할 수 있는 오픈소스 컨테이너 오케스트레이션 플랫폼인 쿠버네티스를 활용하여 카메라(CCTV) 이미지 데이터 AI 분석을 위한 파이프 라인을 구축하고 상황에 따른 자유로운 AI 모델 적용과 AI 모델 학습을 위한 환경 구성을 구현해보고자 하였다. 결론적으로 Message queue system NATs, Object Storage MinIO 를 통해 데이터 저장의 파이프라인을 구축하였고, Hugging Face 로 내려받은 pre-trained model 이 최소한의 수정으로 적용 가능한 시스템을 구현하였다.

Ⅰ. 서 론

AI 에 대한 관심이 증가함에 따라 훈련된 AI 모델을 운영하기 위한 하드웨어 시설과 소프트웨어적 운용에 대한 관심도 함께 증가하고있다. 이 중 소프트웨어 운영 측면에서 복수의 AI 모델을 효율적으로 운영하기 위해 Cloud Native 가 활용되고 있다. Cloud Native 란 Container, Microservice, Immutable infrastructure, Declarative APIs 접근 방식으로 조직이 공개적 혹은 비공개적 또는 하이브리드 클라우드 상황에서 현대적이고 동적인 환경에서 확장가능한 애플레이션을 개발하고 실행할 수 있도록 하는 것으로 Cloud Native 기술을 통한 탄력적인 관리와 자동화 기능으로 최소한의 노력으로 자주 그리고 예측가능한 관리가 가능하다.[1]

본 논문에서는 일반적인 로컬 환경에 비해서 Cloud Native 환경이 가지는 강점과 Cloud Native 환경 구성에 도움을 주는 Kubernetes 에 대해 이론적으로 살펴보고자 한다. 이후 Cloud Native 환경에서 Kubernetes 를 활용해 이미지 분석을 위한 간단한 시스템 구현을 통해 무중단 배포와 유연한 배포가 가능함을 보이고자 한다.

Ⅱ. Cloud Native 환경의 강점 : Micro Service Architecture(MSA)

기존의 Monolithic Architecture(MS) 방식과 Cloud Native 의 예시인 Micro Service Architecture(MSA)를 운영적인 관점에서 비교해보고자 한다.

Monolithic Architecture(MS)는 Software engineering 에서 소프트웨어의 모든 구성요소가 하나의 프로젝트로 담겨있는 형태이다. 웹 서비스를 예시로 각 기능을 담당하는 로직들을 모듈별로 개발하고 하나로 통합시켜 운영하는 방식이다. MS 의 가장 큰 장점은 단순함으로 인한 낮은 운영 복잡도에 있다. 개발과 배포에 있어 테스트가 용이하고 하나의 프로젝트만 배포하면 된다는 점으로 운영 복잡도를 낮출 수 있지만 Scale-Out 의 문제와 업데이트 시 서버 재배포로 인한 지속적인 운영 불가 그리고 프로젝트의 규모가 커질수록 전체 시스템의 구조파악과 코드 복잡도가 기하급수적으로 높아지는 문제가 있다. 이러한 분명한 한계점과 달리 MSA 는 MS 가 가지는 구조적인 한계를 극복하였다.

MSA 란 애플리케이션이 서비스 모음으로 개발되는 애플리케이션 아키텍처의 한 유형이다.[2] MSA 는 서비스를 작은 단위의 기능으로 분리하여 운영함으로써 독립적인 운영이 가능하도록 한다. 이를 통해 코드 복잡도를 낮추고, 독립적 서비스에서 유연한 확장성과 코드 재사용, 문제 해결에 가장 적합한 도구 사용 가능한 기술적 자유에서 MS 에 비해 다양한 강점을 가지고 있다.

본 논문의 구현에서 MSA 방식의 채택으로 서비스의 확장과 다른 서비스에 간섭을 최소한 제한적 수정이 가능하다. 이를 통해 궁극적으로 하나의 기본 시스템에서 간단한 수정으로 유저의 사용 목적에 따른 커스터마이징이 가능할 것 이다.

Ⅲ. Kubernetes 의 장점 : Automated rollouts

본 논문에서는 MSA 를 적용하기 위해 Kubernetes 를 이용한다. Kubernetes 는 Cloud Native 환경에서 컨테이너를 효율적으로 배포하고 관리할 수 있는 오픈소스 컨테이너 오케스트레이션 플랫폼이다.

본 논문에서 가장 대두되는 Kubernetes 의 장점은 배포자동화에 있다. 기존에 서비스를 배포하기 위해서는 기존에 실행되고 있는 서버를 직접 내리고 새로운 서비스를 다시 올려 서버의 공백이 발생하게 되고 배포환경 등 다양한 변수들로 인한 예기치 못 한 문제가 발생할 수 있다. Kubernetes 는 모든 서버(워커 노드)를 하나의 환경으로 제공하고 Kubernertes 를 통해 배포함으로써 간단한 배포가 가능하다. 또한 각 노드에 대한 상태확인과 Sacle out, Self-healing 을 통해 끊임 없는 서비스 운영이 가능하고, MSA 의 독립적인 최소 기능 서비스 운영의 장점을 살릴 수 있다

따라서 본 논문에서는 MSA 와 Kubernetes 를 이용하여 간단한 이미지 분석 파이프라인을 구현하고자한다..

Ⅳ. Cloud Native 환경에서 Kubernetes 를 이용한 Hugging Face 이미지 분석 시스템 설계 및 구현

Ⅱ, Ⅲ에 언급한 부분을 구현에 적용하기 위해 NATs 와 MinIO 를 이용해 설계하였다. Cloud Native 환경에서 분석을 위한 이미지 데이터 수집을 위해 Message queue system 인 NATs, Object Storage 인 MinIO 를 이용하여 카메라 이미지 데이터를 NATs 를 통해 한 곳으로 받아 MinIO 가 DataLake 를 수행할 수 있도록 설계하였다.

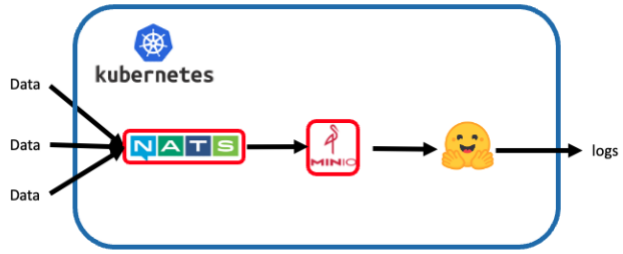

그림 1 Cloud Native 환경에서의 이미지 분석 환경

Inputs 데이터들을 보장하고 DataLake 담당인 MinIO 로 바로 흘러가는 것을 막기 위해 NATs 를 이용하여 데이터를 MinIO 를 외부에서 숨기고 일정한 간격으로 데이터를 저장하도록 설계하였다. MinIO 를 직접 내보이지 않음으로서 악성 공격을 1 차적으로 방어할 수 있게 된다.

이후 직접적으로 이미지 분석을 담당할 서비스는 Python 환경에서 transformers 패키지를 통해 Hugging Face 의 모델을 받아 적용한다.

최종적으로 시스템은 사용자의 목적에 따라 다른 이미지 모델을 사용하는 여러 개의 Hugging Face 모델 서비스를 구동 시켜 같은 데이터에 대해 다른 모델의 Outputs logs 를 얻는 것이 가능하다.

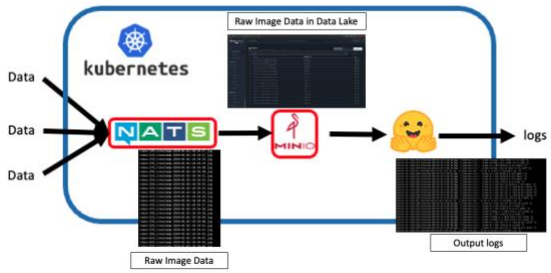

그림 2 Cloud Native 환경에서의 Hugging Face 이미지 분석 시스템 구현 결과

Kubernetes 1.26 버전에서 그림 1 과 같이 Cloud Native 하게 구현을 진행하였다. 먼저 데이터를 생산하는 쪽에서는 MinIO 로 데이터를 전달할 수 있도록 NATs 를 통해 데이터를 잇고 보내고 있는 로그를 출력하여 확인이 가능하도록 구현하였다. 이후 Hugging Face 에서 MinIO 를 받아 이미지 분석을 시작한다. 그림 2 는 해당 과정의 구현 결과를 그림 1 에 로그 및 GUI 화면을 추가하여 나타낸 그림이다.

Ⅴ. 결론

본 논문에서 제시한 설계를 기반으로 구현한 결과 카메라로부터 이미지 데이터를 NATs 로 넘겨 받아 MinIO DataLake 로 저장하고, Python 환경의 Hugging Face 로 내려받은 AI 모델에서 DataLake 로 부터 최신 데이터를 분석하여 로그를 출력하는 파이프라인을 구축하였다. 3 대의 카메라에서 영상 데이터를 1 초에 약 30 장씩 NATs 로 수신하여 MinIO 에 안정적으로 저장되고 있는 모습을 확인하였고, AI 모델에서 MinIO 로부터 데이터를 받아 정상적으로 로그를 출력하는 모습을 확인하였다.

해당 파이프라인 시스템은 기존에 특정 목적을 대상으로 만들어진 static 한 시스템에 비하여 이미지 데이터를 입력받는 부분과 AI 모델의 분석 결과 로그를 출력하는 부분을 static 하게 고정하고 내부에서 간단한 AI 모델 서비스 포드 배포 코드 수정으로 원하는 Hugging Face 모델명을 기입하면 사용 목적에 따른 다양한 모델 적용이 가능하다. 또한 활용하고자하는 모델의 수에 따라 포드의 개수를 조절하여 elastic 한 시스템 운용이 가능하다.

해당 시스템을 조금 더 고도화하여 Front ui 를 제작하여 원하는 모델의 명과 개수를 입력 받아 모델을 내려받는 일시적인 포드를 생성하여 사용자 지정 모델을 받고 이를 사용하는 포드를 실행하는 로직을 추가한다면 사용자는 내부에 관여할 필요없이 사용자의 목적에 맞춘 간단하면서 탄력적인 시스템으로 발전할 수 있을 것이라 기대한다.

ACKNOWLEDGMENT

본 논문은 2023 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No. 2019-0-01842, 인공지능대학원지원(광주과학기술원)).

이 논문은 과학기술원정보통신부 및 정보통신기획평가원의 대학 ICT 연구센터지원사업의 연구결과로 수행되었음 (IITP-2023-2021-0-01835).

참 고 문 헌

[1] Cloud Native Computing Foundation(CNCF), about, who we are, Cloud Native Definition, (https://www.cncf.io/about/who-we-are/).

[2] Google Cloud, learn, What is microservices architecture?, (https://cloud.google.com/learn/what-is-microservices-architecture?hl=en).