# 컨테이너 기반 마이크로서비스 운영 지원을 위한 검색 증강 생성(RAG) 적용 기법

정수민[1], 조대영[2], 박준석[3], 염근혁[4*]
[1]부산대학교 정보융합공학과 박사과정
[2]부산대학교 정보융합공학과 석사과정
[3]부산대학교 지능물류빅데이터연구소 연구교수
[4]부산대학교 정보컴퓨터공학부 교수
{ sumin2708, jdy08, pjs50, yeom }@ pusan.ac.kr

# Retrieval Augmented Generation(RAG) Method for Supporting Container based Microservice Operations

Sumin Jeong[1], Daeyoung Cho[1], Joonseok Park[2], Keunhyuk Yeom[3]
[1]Dept. of Information Convergence Engineering, Pusan National University
[2]Research Institute of Intelligent Logistics Big Data, Pusan National University
[3]School of Computer Science and Engineering, Pusan National University

요 약

마이크로서비스 아키텍처(MSA, Microservice Architecture)는 독립적인 기능 수행 단위인 소규모 소프트웨어 서비스를 약결합하여 대규모 서비스를 개발하는 것이다. 마이크로서비스 아키텍처의 운영에는 기능 수행 단위로 개발하고 결합하는데 적합한 컨테이너 환경이 주로 활용된다. 그러나, 컨테이너 환경은 컨테이너 환경 리소스를 중심으로 배포와 관리가 수행되어 마이크로서비스 아키텍처의 설계 및 기능 개발이 목적인 마이크로서비스 개발자가 운영하기 위한 기술적 지원이 부족하다. 따라서, 본 논문에서는 검색 증강 생성(RAG, Retrieval Augmented Generation)이 적용된 컨테이너 기반 마이크로서비스의 운영 지원 기법을 제안한다. 제안하는 방법은 컨테이너 기반 마이크로서비스를 소프트웨어 환경과 컨테이너 관리 환경으로 분리하여 분석하고, RAG 파이프라인에 기반한 질의 정제 과정을 나타내었다. 본 논문의 사례연구는 RAG 기반 운영 지원 기법을 통해 사용자 요구사항에 따른 마이크로서비스 애플리케이션을 배포하는 과정을 제시하였다. 또한, 마이크로서비스 애플리케이션 배포 성공률 실험에서 제안하는 방법은 20회 중 17회를 성공적으로 배포하였으며, 이는 요구사항 기반 생성형 AI 추론의 배포 성공 수 0회에 비해 85%의 성능적 개선이 있음을 확인하였다. 제안하는 방법은 컨테이너 환경에서 마이크로서비스 운영을 위한 기반 기술로 활용될 것으로 기대된다.

## 1. 서론

마이크로서비스 아키텍처(MSA, Microservice Architecture)[1]는 독립적인 기능 수행 단위인 소규모 소프트웨어 서비스들을 약결합(Loosely Coupling)하여 대규모 소프트웨어 시스템으로 구축하는 것을 말한다. 마이크로서비스 아키텍처 개발은 약결합 연결이 용이하며 소규모 소프트웨어 서비스 기능 개발에 적합한 컨테이너 환경이 주로 활용된다.

컨테이너 환경에서 운영되는 마이크로서비스(컨테이너 기반 마이크로서비스로 지칭함)는 컨테이너 환경 리소스에 기반하여 배포되고 기능이 수행된다. 컨테이너 환경 리소스에 초점을 둔 배포와 기능 수행은 마이크로서비스 아키텍처 설계 내역 및 기 운영중인 소프트웨어 시스템의 마이크로서비스 전환을 어렵게 만드는 주요 원인이 된다. 특히, 컨테이너 환경에서 특징적으로 나타나는 소프트웨어 기능과 지원을 위한 컨테이너 환경의 분리 배포는 단일 서비스에 대해 다수의 컨테이너 명세들이 요구되어 기능을 배포하고 수행하는 운영적 어려움을 배가시킨다.

따라서, 본 논문에서는 검색 증강 생성(RAG, Retrieval Augmented Generation)을 적용하여 컨테이너 기반 마이크로서비스의 운영을 지원하는 기법을 제안한다. 제안하는 기법은 컨테이너 환경 리소

* 교신저자(Corresponding Author)

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 융합보안핵심인재양성사업의 연구 결과로 수행되었음 (RS-2022-II221201)

스에 맞춰진 운영 정보를 벡터 DB(Database)에 적재하고, RAG 기반 마이크로서비스 명세 생성을 위한 질의 정제 과정을 포함한다.

## 2. 관련 연구

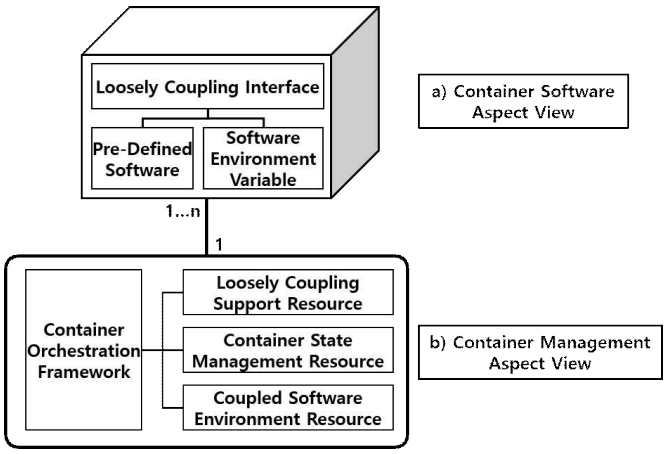
Wan 등[2]은 도커 기반의 마이크로서비스 애플리케이션(다수의 마이크로서비스를 연계하는 환경)을 최적화하여 배포하는 기법을 제시하였다. 해당 논문에서는 비용을 최적화하는 관점에서 마이크로서비스 애플리케이션 구축 시 동일한 소프트웨어 환경을 갖는 마이크로서비스들을 분류한 후 배포하는 방법을 제안하였다.

Kabbay[3]는 컨테이너 환경의 RAG 접목 및 자동화 수행을 위한 방법을 제시하였다. 해당 논문에서는 퍼블릭 클라우드와 오픈소스가 융합된 RAG 파이프라인을 분석하고, 쿠버네티스 중점적인 RAG 기반 컨테이너 애플리케이션 배포 방법을 서술하였다.

반면, 본 논문에서는 컨테이너 환경을 운영하는 관점에서 소프트웨어 환경과 컨테이너 관리 환경을 구분하여 분석하였고, 분석된 요소를 RAG를 통해 반영하는 사용자 맞춤형 컨테이너 기반 마이크로서비스 애플리케이션의 운영 방법을 제안한다.

## 3. 컨테이너 기반 마이크로서비스 운영 지원 기법

컨테이너 기반 마이크로서비스의 운영 요소는 컨테이너에 탑재된 소프트웨어 환경과 컨테이너들이 관리되는 환경으로 구분할 수 있다. (그림 1)은 컨테이너 기반 마이크로서비스의 소프트웨어 환경과 컨테이너 관리 환경을 분석하여 제시한 것이다.




(그림 1) 컨테이너 기반 마이크로서비스 운영 환경
( a) 소프트웨어 환경, b) 컨테이너 관리 환경 )

(그림 1)에 나타낸 바와 같이 컨테이너 기반 마이크로서비스의 컨테이너 소프트웨어 측면은 약결합 인터페이스(Loosely Coupling Interface), 사전 정의 소프트웨어(Pre-Defined Software), 소프트웨어 환경 변수(Software Environment Variable)로 구분된다. 약결합 인터페이스는 소프트웨어에서 기능 수행을 위해 외부 요청을 처리하여 전달하는 역할을 한다. 사전 정의 소프트웨어와 소프트웨어 환경 변수는 컨테이너 내에서 수행할 소프트웨어적 기능과 기능 수행에 필요한 사전 정의 요소를 나타낸다.

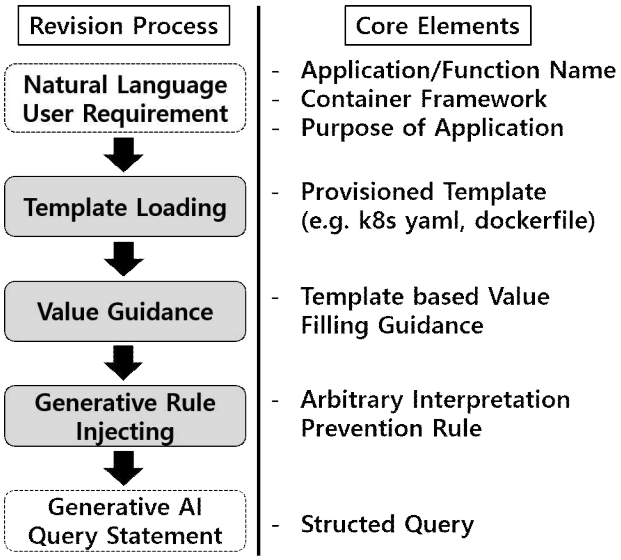
컨테이너 관리 측면에서는 컨테이너 관리 주체인 컨테이너 오케스트레이션 프레임워크(Container Orchestration Framework)가 컨테이너 관리를 위한 리소스들을 제어한다. 특히 컨테이너 기반 마이크로서비스를 배포하고 기능이 수행되는 측면에서 필요한 리소스들은 약결합 지원 리소스(Loosely Coupling Support Resource), 컨테이너 상태 관리 리소스(Container State Management Resource), 결합 소프트웨어 환경 리소스(Coupled Software Environment Resource)로 구분할 수 있다.

(그림 1)에 나타낸 컨테이너 기반 마이크로서비스 운영 요소들은 필요한 정보를 컨테이너 중점적으로 명세한다. 컨테이너 중점적인 명세를 마이크로서비스의 배포와 기능 수행을 지원하기 위한 정보로 분석한 결과가 <표 1>과 같다.

<표 1> 컨테이너 기반 마이크로서비스 운영 요소 분석 표

| 운영 대상 분류 | 정보명 | 설명 |
|---|---|---|
| 컨테이너 소프트웨어 측면 | 소프트웨어 노출 포트 | 소프트웨어의 약결합 수행을 위한 진입점 |
| | 소프트웨어 이미지 | 소프트웨어 기능 |
| | 환경 변수 | 소프트웨어 기능 수행을 위해 사전 정의가 필요한 정보 |
| | 저장소 볼륨 | 소프트웨어의 저장 공간 |
| 컨테이너 관리 측면 | 컨테이너 요청 처리 방법 | 약결합 요청 시 요청 처리 수행 방법 |
| | 컨테이너 인스턴스 수 | 컨테이너 유지 성능을 보장하기 위한 복제 수량 |
| | 리소스 제약사항 | 컨테이너에 할당 가능한 최대 하드웨어 리소스 사양 |
| | 컨테이너 레이블 | 컨테이너 구분자 |
| | 컨테이너 노출 포트 | 컨테이너 환경 외부에서 컨테이너로 기능 요청하기 위한 진입점 |

<표 1>에 분석된 정보는 유사도 기반 검색을 수행하는 벡터 DB에 저장되어 RAG 프로세스에서 생성형 AI의 레퍼런스로 활용된다. 본 논문에서는 생성형 AI가 <표 1>에 분석된 정보를 활용하고 자의적 해석에 의한 환각 현상(Hallucination)을 최소화하기 위해 (그림 2)와 같은 프롬프트 정제 프로세스를 정의하였다.

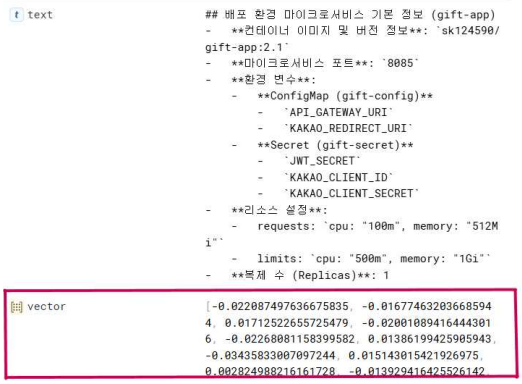



(그림 2) 프롬프트 정제 프로세스

(그림 2)에 나타낸 바와 같이 본 논문에서 활용한 프롬프트 정제 프로세스는 사용자의 기능적/비기능적 요구사항이 담긴 컨테이너 기반 마이크로서비스 애플리케이션 요구사항(Natural Language User Requirement)을 기본 형태로 한다. 전달된 기본 형태에서 컨테이너 환경에 대한 템플릿 인출(Template Loading), <표 1>의 운영 요소를 반영하는 적재 값 지도(Value Guidance), 자의적 해석 방지를 위한 생성 규칙 주입(Generative Rule Injecting)의 과정을 거쳐 생성형 AI에 전달할 구조화된 질의문(Generative AI Query Statement)을 생성하게 된다.

## 4. 사례연구 및 평가

본 논문의 사례연구는 사용자 요구사항이 반영된 컨테이너 기반 마이크로서비스 애플리케이션을 배포하는 과정을 나타낸다. 사례연구를 위해 쿠버네티스 컨테이너 오케스트레이터, RAG 파이프라인 구축을 위한 오픈소스인 LangChain, 벡터 DB 구축을 위한 ElasticSearch를 활용하였다. (그림 3)은 벡터 DB 저장 데이터의 일부를 나타낸다.




(그림 3) LangChain 벡터 DB 저장 데이터

(그림 3)에 나타낸 바와 같이, <표 1>에서 분석한 마이크로서비스 운영 요소는 LangChain 활용을 위해 벡터화 되어 저장된다. (그림 3)에서 컨테이너 이미지 및 버전 정보, 마이크로서비스 포트(소프트웨어 노출 포트), 환경 변수와 같은 컨테이너 소프트웨어 측면 운영 요소와 리소스 설정, 복제 수(컨테이너 인스턴스 수)와 같은 쿠버네티스의 컨테이너 관리 측면 정보가 저장된 것을 확인할 수 있다. 위와 같이 저장된 마이크로서비스 운영 요소들을 바탕으로 사례연구에서 배포할 마이크로서비스 애플리케이션의 자연어 사용자 요구사항이 (그림 4)와 같다.

"선물하기 서비스를 구축하려고 하거든? 실제 배포가 가능해야 할텐데 유저 등록, 관리도 가능해야 하고 상품 조회, 사용자 위시리스트 조회랑 주문하기, 주문하기 시 카카오 메시지로 메시지 보내는 것까지 기능적으로 가능해야 해. 이걸 구현하려면 user-service, product-service, wish-service, order-service, gift-service, api-gateway가 모두 필요해. 그러려면 배포 파일을 어떻게 구성해야 할까?"

(그림 4) 자연어 사용자 요구사항

자연어 사용자 요구사항에서 작성된 내역을 RAG 기반으로 동작시키기 위해 (그림 2)의 프롬프트 정제 프로세스를 적용하면 (그림 5)-(그림 7)과 같다.

Kubernetes YAML Template Loading

```
- 전체 결과를 **YAML 형식**으로 출력하세요.
- 각 필드의 역할을 **간단한 주석** 으로 설명하세요.
- 기본적인 리소스 설정을 포함하세요:
- `replicas`, `containerPort`, `resources.limits/requests` 등
- 필요한 경우 다음과 같은 리소스도 함께 생성하세요:
- `Service`, `Deployment`, `ConfigMap`, `Ingress`, `PersistentVolumeClaim`
- 생성된 YAML은 실제 배포 가능한 형식이어야 합니다.
- 하나의 YAML 안에 여러 리소스가 필요한 경우 `---` 로 구분해서 함께 정의하세요.
```

(그림 5) 쿠버네티스 환경 기반 Template Loading

Value Guidance

```
'문맥'의 내용을 반드시 준수해서 사용자 질의에 맞는 Kubernetes YAML 명세와 그 출처를 JSON 형식으로 생성하세요.

### 절대 규칙 ###
1. '문맥'은 유일한 신뢰 가능한 정보 출처입니다. 사전 지식과 문맥이 충돌하는 경우, 반드시 문맥의 내용을 우선해야 합니다.
2. '문맥'에 명시된 리소스 종류, 포트, 이미지 버전, 레이블 규칙, 리소스 할당량 등 모든 세부 사항을 정확하게 반영해야 합니다.
3. '문맥'에 설명된 민감 정보를 바탕으로 단일 `apiVersion: v1, kind: Secret` 리소스를 먼저 생성해야 합니다.
4. 이 민감 정보가 필요한 다른 모든 리소스(예: Deployment)는 생성된 Secret을 `secretKeyRef` 또는 `envFrom`을 사용하여 참조해야 합니다. Deployment에 Secret 값을 직접 포함하지 마세요.
5. '문맥'에 언급된 모든 마이크로서비스(예: api-gateway, user-service, product-service 등)에 대해 Deployment와 Service를 반드시 생성해야 합니다. 하나도 빠뜨리면 안 됩니다.
```

(그림 6) 쿠버네티스 환경의 Value Guidance

Generative Rule Injecting

```
### YAML 구조 절대 규칙 ###
1. `ConfigMap`과 `Secret`의 `data` 필드는 절대로 `metadata` 필드의 하위에 위치해서는 안 됩니다. `apiVersion`, `kind`, `metadata`, `data`는 모두 동일한 레벨의 필드여야 합니다. 이 규칙은 쿠버네티스 문법의 기본이며 `metadata.data` 형태는 문법 오류입니다.
### 컨텍스트 정책 준수 규칙 ###
1. '문맥' 내의 '조직 내부 정책'에 명시된 규칙을 **반드시** 준수해야 합니다.
2. 예를 들어, 'Secret 관리' 정책에서 `data` 필드를 사용하고 Base64 인코딩을 요구했다면, `절대로 `stringData`를 사용하거나 평문 텍스트를 사용해서는 안 됩니다.`
```

(그림 7) 쿠버네티스 환경의 Generative Rule Injecting

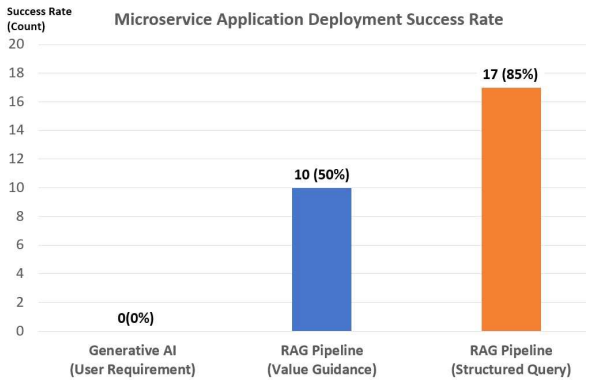
위와 같은 프롬프트 정제 프로세스를 수행하여 정상적으로 배포된 컨테이너 환경의 결과가 (그림 8)과 같다.

```
<-deploy-config-14$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                READY   STATUS    RES
TARTS   AGE
default       api-gateway-68d7979749-7frht        1/1     Running   0
        9s
default       gift-app-68c4f76764-brptk           1/1     Running   0
        9s
default       order-service-55d74d8ff6-8lglc      1/1     Running   0
        9s
default       product-service-547464f5c9-qvtpb    1/1     Running   0
        9s
default       user-service-74d6d9f8d7-vxmhc       1/1     Running   0
        9s
default       wish-service-5997f45d9-grvqh        1/1     Running   0
        9s
```

(그림 8) 컨테이너 기반 마이크로서비스 배포 결과

(그림 8)에 나타낸 바와 같이 배포된 컨테이너들이 요구사항에 맞춰(gift-app, order-service, product-service, user-service, wish-service) 배포되었으며, 정상 동작하는 것을 확인할 수 있다.

본 논문의 평가는 사례연구에서 수행한 마이크로서비스 애플리케이션 배포를 생성형 AI와 RAG 파이프라인에서 수행한 후 비교하였다. (그림 9)는 평가 실험의 성공률 비교 결과를 나타낸다.




(그림 9) 비교군 별 성공률

(그림 9)에 나타낸 바와 같이 마이크로서비스 애플리케이션 배포 성공률은 제안하는 방법(RAG Pipeline(Structured Query)), 벡터 DB 기반 데이터 명세 방법(RAG Pipeline(Value Guidance)), 단순 생성형 AI 요구사항 질의(Generative AI(User Requirement)) 순으로 높았다. 이는, 제안하는 방법의 컨테이너 기반 마이크로서비스의 분석이 컨테이너 기반 마이크로서비스의 운영 환경에 적합함을 나타낸다. 또한, 벡터 DB에 참고용 데이터를 제공하는 것에 더하여 질의를 위한 프롬프트 정제 프로세스를 수행하는 것이 마이크로서비스 애플리케이션의 정상적인 운영을 지원하는 방법임을 확인할 수 있다.

## 5. 결론

본 논문에서는 컨테이너 기반 마이크로서비스의 운영을 지원하기 위해 컨테이너 기반 마이크로서비스를 내부 소프트웨어와 컨테이너 관리 환경으로 구분하여 분석하였다. 분석된 요소들을 RAG에 적용하고, RAG 파이프라인의 결과물인 배포 명세 생성 정확도를 높이기 위해 프롬프트 정제 프로세스를 제시하였다. 또한, 제안하는 방법의 실증을 위해 자연어 기반의 사용자 요구사항을 바탕으로 마이크로서비스 애플리케이션을 배포하는 벡터 DB 적재, 프롬프트 정제 과정과 과정에 따른 배포 결과를 나타내었다. 배포 성공률 측면에서 생성형 AI, 프롬프트 정제 미실시, 제안하는 방법을 비교하였으며, 각각 0회, 10회, 17회의 배포가 성공하였고 제안하는 방법이 가장 배포 성공이 많음을 확인하였다. 향후 본 논문에서 분석한 운영 요소는 컨테이너 기반 마이크로서비스의 정적/동적 측면 분석에 대한 토대가 될 수 있을 것으로 기대된다.

## 참고문헌

[1] N. Alshuqayran, N. Ali and R. Evans "A Systematic Mapping Study in Microservice Architecture," 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 2016, pp. 44-51.

[2] X. Wan, X. Guan, T. Wang, G. Bai and B. Y. Choi, "Application deployment using Microservice and Docker containers: Framework and optimization," Journal of Network and Computer Applications, Vol. 119, pp. 97-109, 2018.

[3] H. S. Kabbay, "Streamlining AI Application: MLOps Best Practices and Platform Automation Illustrated through an Advanced RAG based Chatbot," 2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS), Coimbatore, India, 2024, pp. 1304-1313.