https://doi.org/10.7840/kics.2025.50.3.511

클러스터 내 Container Registry 구성을 통한 컨테이너 이미지 배포 효율성 향상

정 진 욱◆, 권 영 우°

Improving the Efficiency of Container Image Deployment through a Cluster Container Registry

Jin-Uk Jung◆, Young-Woo Kwon°

요 약

대규모 머신러닝 모델이나 데이터 분석 애플리케이션에서는 많은 양의 데이터를 처리하기 위해 컨테이너 이미지를 사용하고 있으나 이러한 이미지의 배포 시간이 길어지는 문제가 발생하고 있다. 이미지의 배포 시간이 길어지면 개발 및 배포 주기가 느려져 생산성이 저하되고, 자원 활용이 비효율적으로 이루어지며, 신속한 스케일링이 어려워져 전체 시스템의 성능과 유연성이 떨어질 수 있다. 이러한 문제를 해결하기 위해 컨테이너 이미지 관리 및 배포 효율성을 개선하는 방안을 제시한다. 이미지 배포 시 Cluster Container Registry(CCR)를 활용함으로써 클라우드를 통한 이미지 다운로드에 비해 시간 및 트래픽 소모를 줄일 수 있다. Kubernetes에서 실험하여, 컨테이너 이미지 관리 및 배포 과정의 효율성을 평가하였다. 기존의 방법과 비교하여 각 시나리오에서의 기존의 방법에 대비해 평균적으로 30% 배포 시간이 감소했고 이미지의 크기가 클수록 효과가 좋은 것으로 나타났다.

키워드 : 클러스터, 엣지 컴퓨팅, Container 레지스트리, 컨테이너 이미지, 배포 효율성
Key Words : Cluster, Edge Computing, Container Registry, Container Image, Deployment Efficiency

ABSTRACT

Large-scale machine learning models and data analysis applications use container images to process vast amounts of data, and there is a problem of lengthening the distribution time of such images Longer image distribution time slows down the development and distribution cycle, which lowers productivity and can reduce the performance and flexibility of the entire system. In order to solve this problem, we propose ways to improve container image management and distribution efficiency. We focus on how to reduce time and traffic consumption compared to image downloading through the cloud by utilizing cluster container registry (CCR) when distributing images. Experiments were conducted in a Kubernetes, and the efficiency of container image management and distribution process was explored. Compared to the existing method, the distribution time was reduced by an average of 30% in each scenario, with larger image sizes showing even greater improvements.

※ 이 논문(특허)은 2024년도 정부 (과학기술정보통신부) 및 지자체(대구광역시)의 재원으로 (재)대구디지털혁신진흥원에서 주관하는 지역 디지털혁신거점 조성지원 사업의 지원을 받아 수행된 연구임(No.DBSD1-03).
※ 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2021R1A5A1021944).
◆ First Author : Kyungpook National University Computer Science and Engineering, dsa04156@knu.ac.kr, 학생회원
° Corresponding Author : Kyungpook National University Computer Science and Engineering, ywkwon@knu.ac.kr, 정회원
논문번호 : 202407-164-C-RN, Received July 31, 2024; Revised October 18, 2024; Accepted October 22, 2024

Ⅰ. 서 론

최근 다양한 분야에서 컨테이너 기반 애플리케이션 사용이 증가하고 있다. 특히 대규모 머신러닝 모델이나 데이터 분석 애플리케이션에서는 많은 양의 데이터를 처리하기 위해 컨테이너 이미지를 사용하고 있는데, 이러한 애플리케이션의 배포 시간이 길어지는 문제가 생겨나고 있다. 또한, 멀티미디어 처리, 게임, VR/AR 애플리케이션, 전술 Edge 클라우드 아키텍처[1]과 같은 다양한 플랫폼과 환경에서 애플리케이션을 실행해야 하는 요구로 컨테이너 이미지의 배포는 점차 복잡해지고 있다[2]. 그러므로 서로 다른 환경에서 컨테이너 이미지를 적시에 배포하고 실행하기 위한 효율적이고 신뢰할 수 있는 배포 방법이 필요하다.

이러한 문제를 해결하기 위해 본 논문은 클러스터마다 독립적인 컨테이너 레지스트리 (Container Registry)를 구성하여 이미지 관리와 배포의 효율성을 개선하는 방법을 제안한다. 클러스터 컨테이너 레지스트리 (Cluster Container Registry, CCR)는 각 클러스터 내에 로컬 컨테이너 레지스트리를 두어 클러스터 내의 모든 노드가 중앙 클라우드 서버를 거치지 않고 직접 이미지를 다운로드하고 배포한다. 이를 통해 가까운 위치에 있는 레지스트리에서 이미지를 가져와 네트워크 지연 시간을 줄이고 배포 시간을 단축할 수 있다. 특히 큰 용량의 컨테이너 이미지 배포 시간을 줄여 멀티미디어 처리, 게임, VR/AR 애플리케이션의 성능 저하 문제를 해결할 수 있어, 다양한 플랫폼과 환경에서 컨테이너 이미지를 쉽게 배포하는 방안을 제시한다.

Kubernetes 환경에서 CCR을 구현한 후, 컨테이너 이미지 관리 및 배포 과정의 효율성을 평가하였다. 실험 결과, 기존 중앙 집중식 방법과 비교했을 때 배포 시간이 평균적으로 약 30% 단축되는 성과를 확인할 수 있었다. 특히, 이미지의 크기가 클수록 CCR의 성능 향상 효과가 더욱 두드러졌으며, 이는 대규모 애플리케이션 환경에서 이미지 배포의 병목 현상을 효과적으로 줄일 것으로 기대한다.

Ⅱ. 연구 배경

2.1 기존 배포 방법

기존의 컨테이너 이미지 배포 방법은 그림 1과 같이 각 노드가 레지스트리에서 이미지를 가져와 로컬에 저장한 후, 이를 사용하여 애플리케이션을 실행하는 방식이다. Pod에 이미지가 필요할 때, 로컬 저장소에서 이미지를 load 하여 사용하는 방식으로, 이미지 다운로드를

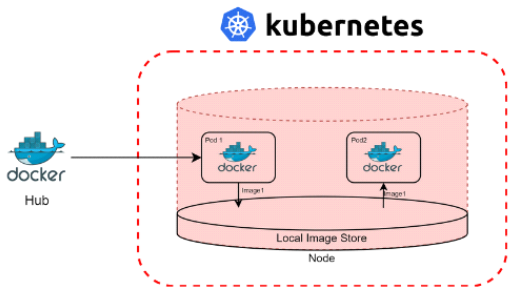

그림 1. 쿠버네티스 IfNotPresent
Fig. 1. Kubernetes IfNotPresent

줄이고 배포를 간소화한다.

그러나 이 방법에는 몇 가지 한계점이 있다. 첫째, 노드 간 이미지 비일관성이 발생할 수 있다. 특정 노드에만 이미지가 캐시 되어 있는 경우, 다른 노드에서 이미지를 요청하면 다시 다운로드해야 하여 시간 지연이 생긴다. 둘째, 사용량이 많은 클러스터에서는 같은 Pod를 같은 노드에 지속해서 배치하기 어려워 이미지가 로컬에 없으면 추가 다운로드가 필요하다. 셋째, 여러 컨테이너를 사용하는 Pod는 각 컨테이너에 필요한 이미지를 순차적으로 가져와야 하므로 배포 속도가 느려진다.

앞선 방법의 문제를 해결하기 위해서 그림 2와 같은 serializeImagePulls 옵션을 false로 설정하여 컨테이너 이미지 풀 작업이 병렬로 수행되도록 한다. 이 설정을 통해 각 컨테이너가 필요한 이미지를 동시에 다운로드할 수 있으며, 배포 속도를 개선한다.

하지만 개선된 배포 방법에도 몇 가지 한계점이 존재한다. 우선, 두 가지 옵션을 사용하여도, 모든 노드가 동시에 컨테이너 레지스트리에서 이미지를 다운로드하는 경우 네트워크 부하가 발생한다. 이로 인해 네트워크 대역폭이 과도하게 소모될 수 있으며, 이는 전체 시스템의 성능에 부정적인 영향을 미칠 수 있다. 또한, 대규모

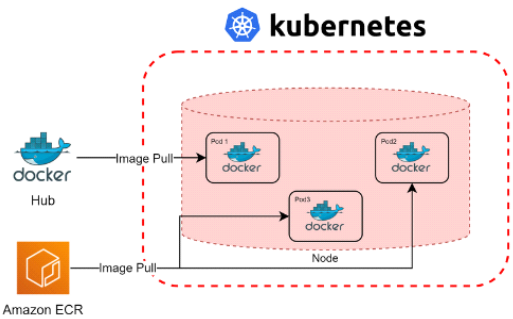

그림 2. 쿠버네티스 serializeImagePulls false
Fig. 2. Kubernetes serializeImagePulls false

배포를 진행할 경우, 네트워크 지연 위험이 커진다. 많은 노드가 동시에 이미지를 다운로드하면서 네트워크의 지연 시간이 증가할 수 있으며, 이는 이미지 배포 시간의 증가로 이어질 수 있다. 이러한 지연은 대규모 IoT 시스템, 멀티미디어 처리, 게임, VR/AR 애플리케이션과 같은 실시간 데이터 처리 환경에서 문제를 일으킬 수 있다.

2.2 관련 연구

컨테이너 이미지 배포 효율성 향상을 위한 연구가 많이 시도되었다[3,4]. 에지 서버의 효율적 컨테이너 배포를 위한 시스템 모델과 알고리즘[5], 전술 엣지 클라우드에서의 컨테이너 이미지 및 상태[1], 스마트 홈 환경에서의 컨테이너화된 딥 러닝 모델 배포가 있다[6].

[5]연구는 에지 서버에서의 효율적인 컨테이너 배포를 위해 시스템 모델과 다중 k 레지스트리 배치 알고리즘(MKRP)을 제안한다. 이 연구는 네트워크 오버헤드와 서버 간의 거리 문제를 고려하여 여러 레지스트리 서버를 배치하여 컨테이너 이미지 배포를 최적화한다. 연구 결과는 이러한 다중 레지스트리 접근 방식이 이미지 배포의 효율성을 개선할 수 있음을 입증한다.

그러나 이 연구는 리더로 선출된 레지스트리 서버가 과부하 상태가 되면 전체 시스템 성능이 저하될 수 있음을 지적한다. 특정 레지스트리 서버가 과중한 부하를 처리하게 되면 전체 시스템의 성능에 부정적인 영향을 미칠 수 있다. 이러한 한계점은 다중 레지스트리 배치의 효과를 극대화하기 위해 고려해야 할 중요한 요소다.

[1]연구는 P2P 분산 시스템을 적용하여 인접 노드에서 필요한 컨테이너 이미지와 상태 정보를 가져오는 방법을 제시하였다. 이를 통해 지연 시간과 저장 용량을 감소시킬 수 있었다. 연구 결과, 제안된 기법은 기존 Kubernetes 기반의 중앙 집중식 이미지 저장 방식보다 다운로드 속도와 저장 용량 면에서 우수한 성능을 보였다.

그러나 연구에는 몇 가지 아쉬운 점이 있다. 제안된 기법의 속도는 테스트 환경에서 비교적 양호하였지만, 실제 전술 환경에서는 더 복잡한 변수들이 작용할 수 있으며, 이에 대한 추가적인 테스트가 필요하다. 또한, 테스트 과정에서 다양한 네트워크 조건과 시나리오를 고려하지 않았으므로, 실제 상황에서의 성능 변화를 확인하기 위한 보다 폭넓은 테스트가 필요하다. 그리고 P2P 방식의 적용 시 네트워크 상태가 불안정할 경우 성능이 저하될 수 있으며, 이러한 상황에서의 최적화 방안도 추가 연구가 필요하다.

[6]연구는 스마트 홈 환경에서 딥 러닝 모델의 효율적 배포를 위해 컨테이너화 기술을 활용한 아키텍처를 제안한다. 이 연구는 Edge 컴퓨팅을 통해 클라우드 의존성을 줄이고, 네트워크 지연을 개선하며, 데이터 프라이버시를 보장하는 방법을 제시한다. 제안된 아키텍처는 Edge 장치와 클라이언트 장치를 구분하여, 모델의 배포와 관리를 최적화하고, 각 모듈이 독립적으로 작동할 수 있도록 설계된다.

이 연구는 Edge 환경에서의 딥 러닝 모델 배포를 위한 컨테이너화의 이점을 보여준다. 특히, Container를 활용하여 모델의 이식성과 독립성을 보장하며, 다양한 하드웨어 환경에서도 일관된 성능을 유지할 수 있음을 설명한다. 그러나 연구는 Container 컨테이너가 모델의 성능에 미치는 영향을 분석한 결과, 모델 실행 시간에 거의 영향을 미치지 않는다는 점을 지적한다. 이는 Container를 사용하는 것이 시스템 성능에 거의 오버헤드를 추가하지 않으면서도, 모델 배포 및 관리의 효율성을 크게 향상시킬 수 있음을 보여준다.

다만, 연구는 Edge 장치의 자원 제약으로 모델의 실행과 관리에서 발생할 수 있는 문제를 고려해야 한다고 강조한다. 특히, 자원 제약으로 인해 모델을 자주 교체하거나 업데이트할 때 성능 저하가 발생할 수 있으며, 이를 효과적으로 해결하기 위한 추가적인 연구가 필요함을 시사한다.

Ⅲ. 연구 내용

이 연구에서는 Edge 컴퓨팅 환경에서 이미지 배포의 효율성을 높이기 위한 두 가지 주요 방식을 탐구한다. 첫 번째로, 비활성 노드를 활용한 이미지 배포 최적화 방법을 제안한다. 일반적으로 Edge 컴퓨팅 환경에서는 활성화된 노드만을 이용하여 이미지를 배포하는 방식이 주로 사용되는데, 이는 노드 간 부하 분산이 제대로 이루어지지 않아 배포 시간이 길어지는 문제를 초래할 수 있다. 이를 해결하기 위해, 제안된 방법은 필요할 때 비활성화된 노드를 일시적으로 활성화하여 이미지 배포에 참여시키고, 배포가 완료되면 다시 비활성화하는 방식으로 배포 속도를 최적화한다.

두 번째로, 클러스터 내 컨테이너 레지스트리 구축을 통한 이미지 배포 효율성 향상 방안을 제시한다. 클러스터 내부에 자체적인 이미지 저장소를 구축하고, 클라우드 이미지 저장소로부터 이미지를 한 번만 가져온 후 클러스터 내 각 노드에 이를 분배하는 방법을 제안한다.

3.1 비활성 노드의 이미지 배포

그림 3은 제안하는 첫 번째 방법을 보여주며 비활성

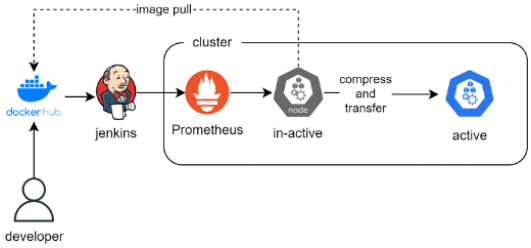

그림 3. 비활성 노드 배포 아키텍처
Fig. 3. Inactive Node Deployment Architecture

노드에 이미지를 배포하여 Edge 컴퓨팅 환경에서 이미지 관리의 효율성을 높이기 위한 방식이다. 이 방법은 비활성 노드를 모니터링하고, 비활성 노드에서 활성 노드로 이미지를 전달하는 절차로 구성된다.

그림 3에서 볼 수 있듯이 Prometheus를 활용하여 노드 상태를 실시간으로 모니터링하고, 비활성 노드를 자동으로 감지한다[9]. 감지된 비활성 노드는 Jenkins를 통해 컨테이너 레지스트리로부터 필요한 이미지를 다운로드한다. 다운로드 된 이미지는 tar 파일 형식으로 저장된다. 이후, tar 파일은 비활성 노드에서 활성 노드로 전송되며, 활성 노드는 이 파일을 수신한 후 이를 다시 Docker 이미지로 로드하여 컨테이너를 배포할 수 있다. 하지만 이 과정에서 이미지 파일을 다른 노드로 전송하기 위해 tar 파일로 변환하고 tar 파일을 다시 Container 이미지로 load 하는 과정에서 추가적인 시간이 소모되어 효율성이 저하되는 단점이 있다.

3.2 컨테이너 레지스트리 (CCR)

3.2.1 CCR 구성

그림 4는 클러스터 컨테이너 레지스트리 (CCR)의 방법을 보여준다. 클러스터 내에 로컬 컨테이너 레지스트리를 설치하여 클라우드 상의 Public Registry으로부터 이미지를 직접 받아 클러스터 내 노드들에 배포하는 방법이다. 이 방법은 물리적으로 가까운 위치에 있는 Registry에서 이미지를 가져와 네트워크 지연 시간을 줄이고, 기존 배포 방식인 모든 노드의 개수에서 클러스터의 개수로 클라우드 이미지 저장소로 요청이 줄어들어 전체적인 배포시간을 줄인다. 또한, 클라우드 저장소와의 중복된 네트워크 트래픽을 최소화하여 시스템 자원의 효율적인 사용을 가능하게 한다.

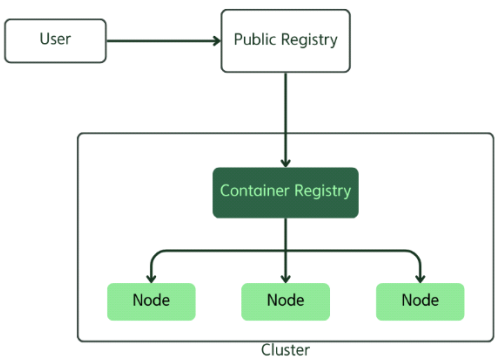

그림 4. Cluster Container Registry
Fig. 4. Cluster Container Registry

3.2.2 Registry Horizontal Pod Autoscaler

CCR을 효과적으로 구현하기 위해, 클러스터 내 Registry Pod의 수를 자동으로 조절할 수 있는 Horizontal Pod Autoscaler (HPA)를 도입한다[11]. HPA는 그림 5와 같이 CPU와 메모리 사용률을 기반으로 Registry Pod의 수를 동적으로 조정하여 부하에 따라 Pod를 확장하거나 축소한다.

Metric 서버를 통해 Registry Pod의 리소스 사용량을 실시간으로 모니터링하고, 이 데이터를 바탕으로 필요한 Replica 수를 계산한다. 이후, 계산된 Replica 수에 따라 Pod의 수를 자동으로 조절하여 시스템의 성능과 자원 효율성을 최적화한다.

이 방식은 컨테이너 레지스트리의 리소스 수요에 따라 Pod 수를 자동으로 조절함으로써 이미지 배포와 관리의 효율성을 높이고, 시스템의 부하를 효과적으로 분산시킨다. CCR은 클라우드 서버에 대한 의존도를 줄이고, 엣지 노드에서 직접 이미지를 다운로드할 수 있는 장점을 제공하며, 특히 큰 용량의 애플리케이션 컨테이너 이미지의 배포를 가속하는 데 유용하다.

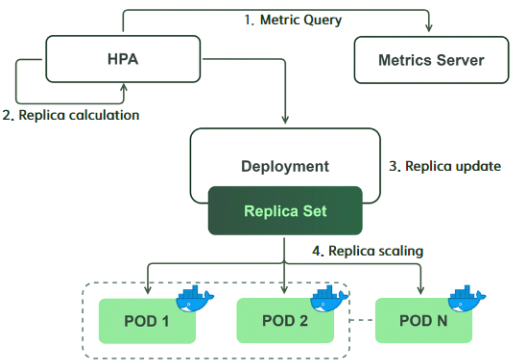

그림 5. Horizontal Pod AutoScaler
Fig. 5. Horizontal Pod AutoScaler

Ⅳ. 실 험

본 실험에서는 다양한 크기의 컨테이너 이미지를 사용하여 배포 효율성을 측정하였다. 다양한 크기의 이미지와 실제 애플리케이션 배포 시나리오를 바탕으로 실험을 진행하여 배포 시간을 측정하였다.

4.1 실험 환경 구축

본 연구에서는 종합적인 CI/CD(지속적 통합/지속적 배포) 파이프라인을 추가하였다[7]. 그림 6과 같은 시스템을 구성하였고, 이 시스템은 코드의 변경을 감지하고, 새로운 버전의 컨테이너 이미지를 자동으로 빌드하고 배포하며, 이를 통해 실시간으로 안정성과 성능을 검증할 수 있는 환경을 제공한다.

구체적으로, 개발자가 GitHub에 코드를 올리면, GitHub에서 Jenkins로 web hook을 보내 새로운 컨테이너 이미지를 빌드하고, 이 이미지를 Harbor에 저장한다. Harbor는 이미지를 안전하게 저장하고 관리할 수 있는 기능을 제공한다.

다음 단계로, Argo CD를 사용하여 Kubernetes 클러스터에 새로운 이미지를 배포한다. Argo CD는 GitOps 원칙을 따르며, 코드 저장소에 정의된 설정을 기반으로 자동으로 배포를 수행하고, 클러스터의 상태를 지속해서 모니터링한다[8]. 이를 통해 최신 이미지를 클러스터에 신속하고 안정적으로 배포할 수 있다.

또한, 시스템의 성능과 안정성을 모니터링하기 위해 Prometheus와 Grafana를 사용한다. Prometheus는 시스템의 Metric을 수집하고 저장하며, Grafana는 이러한 Metric을 시각화하여 대시보드에서 모니터링 할 수 있게 한다. 이를 통해 시스템의 성능 지표를 신속하게 분석하고, 문제 발생 시 빠르게 대응할 수 있다.

로그 수집 및 분석을 위해 EFK(Elasticsearch, Fluentd, Kibana) 스택을 도입하였다[10]. Fluentd는 로그 데이터를 수집하여 Elasticsearch에 전송하며, Elasticsearch는 로그 데이터를 저장하고 인덱싱한다. Kibana는 저장된 로그 데이터를 시각화하여 분석할 수 있는 인터페이스를 제공한다. 이 스택은 로그 데이터의 검색과 분석을 용이하게 하여, 문제의 원인을 신속하게 파악하고 시스템의 상태를 지속해서 추적할 수 있도록 한다.

이와 같은 종합적인 CI/CD 파이프라인은 실제 운영 환경과 유사한 조건에서의 실험과 검증을 가능하게 하며, 시스템의 배포 과정에서 발생할 수 있는 문제를 조기에 발견하고 해결할 수 있는 기반을 마련한다.

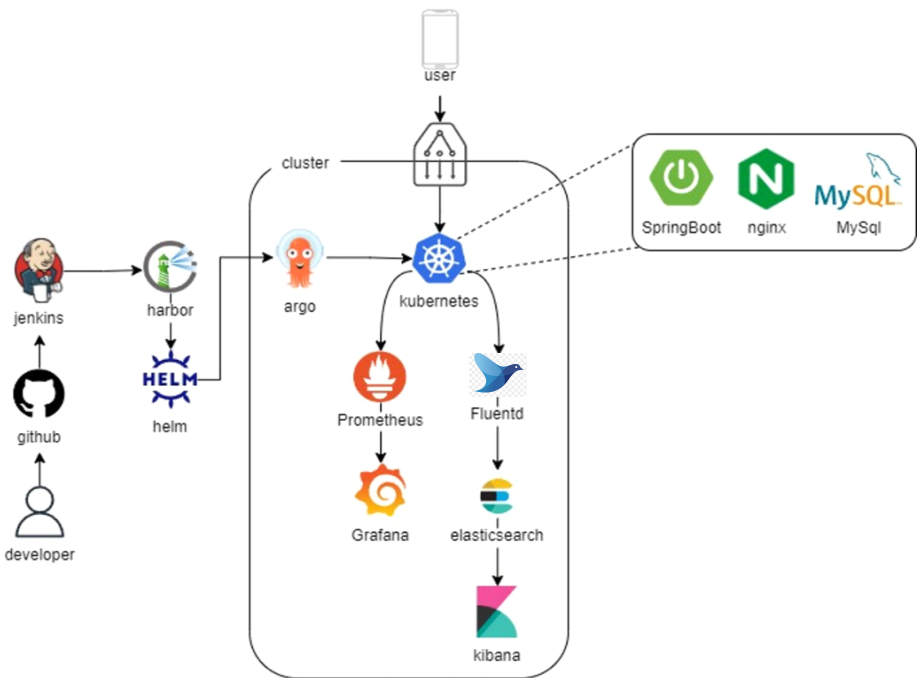

그림 6. Cluster Docker Registry 아키텍처
Fig. 6. Cluster Docker Registry architecture

4.2 실험

Google Cloud Platform 환경에서의 실험은 실제 시스템과 유사한 환경에서 CCR의 성능을 평가하기 위해 구성되었다. 이 실험을 위해 세 개의 노드로 구성된 Kubernetes 클러스터를 구성했다.

평균 Round Trip Time(RTT)은 북미가 0.11s, 도쿄가 0.025ms, 서울이 0.01ms로 측정되었다. 이 실험은 기존 방법과 제안된 방법들의 성능을 비교하고, 이미지 크기에 따른 배포 시간을 분석하여 CCR 방식이 네트워크 지연을 최소화하고 배포 효율성을 향상하는지를 평가한다.

Jenkins 서버를 도쿄 지역에 배치하여 CI/CD 파이프라인의 빌드 및 배포 과정을 관리하도록 했다. Container Registry는 북미 지역에 배치하여 기본 이미지 저장소 역할을 하고, 클러스터는 서울 지역에 배치하여 실제 애플리케이션이 실행되는 환경을 제공한다. 또한, CCR도 서울 지역에 배치하여 클러스터 내부에서의 효율적인 이미지 배포를 지원하도록 한다.

4.3 실험 결과

Nginx 이미지의 크기를 변경하여 이미지 크기에 따른 배포 시간을 분석하고 인공지능 기반 맞춤형 다채널 영상 분석 애플리케이션을 사용하여 실험을 진행하였다.

4.3.1 성능 벤치마크

그림 7은 다양한 크기의 컨테이너 이미지 배포에 있어 기존 방법, 비활성 노드의 이미지 배포(Method 1), 그리고 클러스터 내 컨테이너 레지스트리를 활용한 배포(Method 2)의 성능을 비교한 결과를 보여준다.

133MB 이미지의 경우, 기존 방법은 66초가 소요되었고, Method 1은 46초, Method 2는 48초가 걸렸다. 670MB 이미지에 대해서는 기존 방법이 87초 소요됐지만, Method 1은 73초, Method 2는 65초가 걸렸다. 1210MB 이미지의 경우, 기존 방법은 140초가 소요되었고, Method 1은 94초, Method 2는 89초가 소요되었다. 마지막으로, 2230MB 이미지에서는 기존 방법이 250초 소요됐지만, Method 1은 183초, Method 2는 136초가 걸렸다.

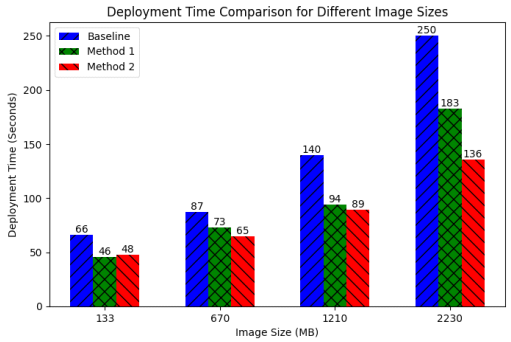

그림 7. 이미지 크기와 방법에 따른 소요 시간
Fig. 7. Time required by image size and method

모든 이미지 크기에서 Method 1와 Method 2가 기존 방법보다 빠른 배포 시간을 기록했다. 특히, 이미지 크기가 커질수록 Method 1와 Method 2의 배포 효율성이 더욱 커지며, 배포 시간이 단축되었다. 두 방법 중에서는 Method 2가 전반적으로 가장 짧은 배포 시간을 기록하고 있으며, 이는 클러스터 내 Container Registry를 활용한 배포가 배포 효율성을 극대화하는 데 더욱 효과적임을 시사한다. 이는 클러스터 내에 설치된 Container Registry가 클라우드 상의 Container Registry로부터 직접 이미지를 받아 같은 클러스터에 속한 노드에 배포하기 때문이다. 그리고 물리적으로 가까운 위치의 Registry에서 이미지를 가져오므로 네트워크 지연 시간이 크게 줄어들었다. 이는 클라우드의 Container Registry와 클러스터의 Registry 간의 거리가 멀어질수록 배포 시간의 감소율이 더욱 높아질 것으로 예상한다.

4.3.2 사례 연구

제안 기법의 효과를 측정하기 위하여 CCTV 기반의 안전 진단 시스템에 적용하였다. 그림 8은 구축된 시스템을 보여주고 있다.

그림 9는 노드 수가 4개이고 클러스터가 2개인 환경에서 이미지 배포 시간을 비교한 결과를 보여준다. 제안한 방법의 배포 시간 측정은 클라우드 이미지 저장소에서 클러스터 내 이미지 저장소로 가져온 후, 각 노드로의 배포 시간을 합쳐서 평균 5분 30초로 기록되었다. 이는 각 노드가 같은 클러스터 내의 컨테이너 레지스트리에 요청을 보내기 때문에 효율성이 향상되었음을 나타낸다.

반면 기존 방법으로 진행했을 경우, 전체 배포 시간은 평균 7분 45초로 나타났으며, 이는 각 노드에서 하나

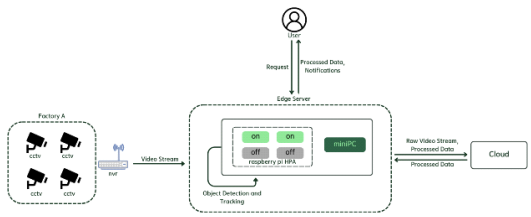

그림 8. 애플리케이션 시스템 구성도
Fig. 8. Application system architecture

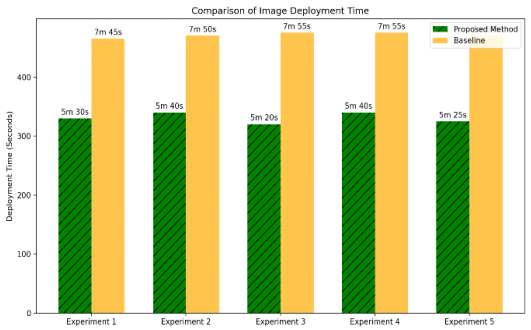

그림 9. 방법에 따른 소요 시간
Fig. 9. Time required by method

의 이미지 저장소에 요청을 보내 부하가 생겨 늦춰진 것으로 보인다. 이러한 결과는 제안한 방법이 멀티 노드 환경에서의 이미지 배포에 있어 시간 효율성을 높일 수 있음을 나타내며, 특히 클라우드 이미지 저장소에서 클러스터 내 이미지 저장소로의 전송 과정에서의 성능 개선이 두드러진다. 따라서 제안한 방법은 CCTV 기반의 안전 진단 시스템과 같은 대규모 애플리케이션의 배포 환경에서 더 효과적이고 효율적인 솔루션임을 확인할 수 있다.

V. 결 론

본 연구는 컨테이너 이미지 배포 방식의 효율성을 비교 분석하여, 클러스터 환경에서의 이미지 배포 최적화 전략을 제시한다. 연구의 초점은 기존 배포 방식과, 클러스터 내에 컨테이너 레지스트리를 구축하여 사용하는 방식인 CCR 방식의 이미지 배포 속도를 비교한다.

실험 결과, CCR 방식을 사용하는 것이 클라우드 환경에서의 네트워크 트래픽을 효과적으로 줄이고, 이미지 배포 시간을 크게 단축한다는 것을 보여준다. 이는 특히 대규모 클러스터 환경에서 큰 이미지 파일을 자주 배포해야 하는 상황에서 중요한 의미가 있다. CCR 방식은 이미지가 물리적으로 가까운 위치에서 제공됨으로써, 클라우드 상의 중앙 서버로부터 이미지를 다운로드할 때 발생할 수 있는 네트워크 지연을 최소화할 수 있다. 이러한 접근 방식은 클라우드 자원 사용의 효율성을 크게 증가시키며, 결과적으로 비용 절감에도 기여한다.

본 연구의 결과는 클러스터 환경에서 컨테이너 이미지를 효율적으로 배포하기 위한 전략 수립에 중요한 기초 자료를 제공한다. 특히, CCR 방식은 이미지 배포의 속도와 효율성을 개선하여, 실시간 데이터 처리와 빠른 배포가 중요한 대규모 시스템에서 그 효과를 극대화할 수 있음을 입증한다.

향후 연구에서는 다양한 클러스터 환경과 이미지 크기에 따른 세부적인 분석을 통해, 보다 구체적이고 실용적인 가이드라인을 제시할 수 있을 것으로 기대된다. 이러한 추가 연구는 다양한 환경과 요 중요한 방향성을 제시할 것이며, Edge Computing 환경에서의 컨테이너 기반 애플리케이션 배포와 관리에 기여할 것이다.

References

[1] H. Lim and Y. Kim, “Container image and state management in the tactical edge cloud environment,” *J. KICS*, vol. 47, no. 7, pp. 1010-1016, Jul. 2022.

[2] A. Makris, E. Psomakelis, I. Korontanis, T. Theodoropoulos1, A. Protopsaltis, M. Pateraki, Z. Ledwon, C. Diou, D. Anagnostopoulos, and K. Tserpes, “Streamlining XR application deployment with a localized container registry at the edge,” *ESOCC 2023*, pp. 188-202, Larnaca, Cyprus, Oct. 2023.

[3] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, and S. Cretti, “Foggy: A platform for workload orchestration in a fog computing environment,” *2017 IEEE Int. Conf. CloudCom*, pp. 231-234, Hong Kong, China, 2017. (https://doi.org/10.1109/CloudCom.2017.62)

[4] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, “Extend cloud to edge with KubeEdge,” *2018 IEEE/ACM SEC*, pp. 373-377, Seattle, WA, USA, 2018. (https://doi.org/10.1109/SEC.2018.00048)

[5] C. G Song, H. Yu, and J.-M. Gil, “MkRP: Multiple k registry placement for fast container deployment on edge computing,” 2024. (https://doi.org/10.20944/preprints202402.0064.v1)

[6] N. Gupta, K. Anantharaj, and K. Subramani, “Containerized architecture for edge computing in smart home: A consistent architecture for model deployment,” *ICCCI*, pp. 1-8, Coimbatore, India, 2020.

(https://doi.org/10.1109/ICCCI48352.2020.9104073)

[7] M. K. Abhishek, D. R. Rao, and K. Subrahmanyam, "Framework to deploy containers using kubernetes and CI/CD pipeline," *Int. J. Advanced Comput. Sci. and Appl.*, 2022. (https://doi.org/10.14569/IJACSA.2022.0130460)

[8] R. López-Viana, J. Díaz, and J. E. Pérez, "Continuous deployment in IoT edge computing: A GitOps implementation," *17th Iberian CISTI*, pp. 1-6, Madrid, Spain, 2022. (https://doi.org/10.23919/CISTI54924.2022.9820108)

[9] V. Sharma, "Managing multi-cloud deployments on kubernetes with istio, prometheus and grafana," *2022 8th ICACCS*, pp. 525-529, Coimbatore, India, 2022. (http://do.org/10.1109/ICACCS54159.2022.9785124)

[10] F. Ahmed, U. Jahangir, H. Rahim, K. Ali, and D.-e.-S. Agha, "Centralized log management using elasticsearch, logstash and kibana," *2020 ICISCT*, pp. 1-7, Karachi, Pakistan, 2020. (http://doi.org/10.1109/ICISCT49550.2020.9080053)

[11] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D. H. Park, and S. H. Kim, "Horizontal pod autoscaling in kubernetes for elastic container orchestration," *Sensors*, Retrieved Aug. 20, 2020, from https://www.mdpi.com/1424-8220/20/16/4621. (https://doi.org/10.3390/s20164621)

정 진 욱 (Jin-Uk Jung)

2021년 8월 : 계명대학교 컴퓨터 공학과 졸업

2024년 3월~현재 : 경북대학교 컴퓨터학부 석사과정

<관심분야> 클라우드, 엣지 컴퓨팅, 네트워크 보안

[ORCID:0009-0005-3362-4201]

권 영 우 (Young-Woo Kwon)

2003년 2월 : 경북대학교 컴퓨터과학과 졸업

2005년 2월 : 광주과학기술원 정보통신공학과 석사

2014년 7월 : Virginia Tech, 컴퓨터과학과 박사

2014년 8월~2017년 6월 : Utah State University, 교수

2017년 8월~현재 : 경북대학교, 컴퓨터학부 교수

<관심분야> 분산시스템, 빅데이터, 인공지능, IoT, 지진조기경보, 재난 ICT

[ORCID:0000-0003-0625-8232]