Journal of KIIT. Vol. 24, No. 8, pp. 159-169, Aug. 31, 2026. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2026.24.8.159

쿠버네티스 환경에서 이벤트 기반 오토스케일링 성능 분석

임재연*, 구세완**, 윤동원***

Performance Analysis of Event-Driven Autoscaling in Kubernetes Environment

Jaeyeon Lim*, Sewan Gu**, and Dongweon Yoon***

요 약

본 논문에서는 쿠버네티스 환경에서 서비스 부하 유형에 따른 효율적인 오토스케일링 전략 수립을 위하여 쿠버네티스 이벤트 기반 오토스케일러(KEDA, Kubernetes Event-driven Autoscaling)의 메시지 브로커별 반응 특성과 자원 활용 패턴에 대한 실험을 통해 성능 비교 기준을 제안한다. KEDA의 성능 특성을 정량적으로 비교 분석하기 위하여 KEDA가 지원되는 메시지 브로커인 RabbitMQ, Apache Kafka, Redis Streams, NATS JetStream을 대상으로 급격 부하, 지속 부하, 급격 부하 감소의 세 가지 부하 유형에 대한 오토스케일링 실험을 수행한다. 각 메시지 브로커별 TCP 기반 부하 생성 도구를 활용하여 반응 특성 및 자원 활용 패턴을 측정하고 Scale-In/Out 반응 시간, Overshoot, CPU·메모리 사용률, 메시지 처리량 및 지연 시간을 비교 분석하여, 쿠버네티스 환경에서 서비스 부하 유형에 따른 KEDA와 메시지 브로커의 오토스케일링 성능 특성을 제시한다.

Abstract

This paper quantitatively compares and analyzes the performance characteristics of the Kubernetes Event-driven Autoscaler (KEDA) under various workload conditions to propose performance comparison criteria for establishing efficient autoscaling strategies in Kubernetes environments. Four KEDA-compatible message brokers - RabbitMQ, Apache Kafka, Redis Streams, and NATS JetStream - are evaluated through autoscaling experiments for three workload types: burst workload, sustained workload, and decreasing workload. The experiments are conducted in a controlled Kubernetes environment, where TCP-based workload generators are used to measure autoscaling behavior. The analysis focuses on two major dimensions: reaction characteristics and resource utilization. Based on these experimental results, this paper presents the autoscaling performance characteristics of KEDA with different message brokers under varying workload conditions.

Keywords

kubernetes, event-driven autoscaling, keda, message broker

* 안랩클라우드메이트 솔루션 아키텍트
- ORCID: https://orcid.org/0009-0001-9985-1125
** LG전자 책임연구원(공동1저자)
- ORCID: https://orcid.org/0000-0001-9547-7626
*** 한양대학교 융합전자공학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0001-9631-3500

· Received: Apr. 20, 2026, Revised: Jul. 09, 2026, Accepted: Jul. 12, 2026

· Corresponding Author: Dongweon Yoon
Dept. of Electronic Engineering, Hanyang University,
222 Wangsimni-ro, Seongdong-gu, Seoul 04763, Korea
Tel.: +82-2-2220-0362, Email: dwyoon@hanyang.ac.kr

Ⅰ. 서 론

클라우드 네이티브 기술과 마이크로서비스 아키텍처(MSA, Microservices Architecture)의 확산으로 컨테이너 기반 인프라를 활용한 분산 서비스 환경이 보편화되고 있다[1]. 이러한 환경에서 쿠버네티스는 컨테이너화된 서비스의 배포, 운영 및 확장을 담당하는 핵심 플랫폼으로 활용된다. 쿠버네티스 기반의 MSA 환경에서는 서비스 간 결합도를 낮추고 비동기 데이터 처리를 지원하기 위하여 메시지 브로커를 도입하며, Apache Kafka, RabbitMQ, Redis Streams 등은 서비스 간 이벤트 전달을 담당하는 대표적인 메시지 브로커로 활용되고 있다. 특히 CSP, Cloud Service Provider 등과 같은 주요 클라우드 서비스 제공업체에서도 메시지 브로커들을 관리형 서비스로 제공하고 있다.

이와 같은 구조에서 메시지 브로커는 서비스 간 이벤트를 중개하며, 메시지 큐의 적체 수준이나 메시지 처리량에 따라 특정 서비스의 부하가 동적으로 변화하는 특성을 갖는다. 따라서 메시지 큐 기반 워크로드를 처리하는 파드(Pod)의 수를 부하 변화에 따라 자동으로 조정하는 오토스케일링 메커니즘이 중요하다. 그러나 메시지 브로커 환경에서 발생하는 부하에 즉각적으로 대응하기 위해서는, 쿠버네티스의 HPA, Horizontal Pod Autoscaler와 같은 기존 오토스케일링 메커니즘인 수평적 파드 오토스케일링만으로는 한계가 존재한다. HPA는 CPU·메모리 등 내부 지표 외에 외부 메트릭 확장이 가능하지만, 구성의 복잡성과 수집 주기에 따른 지연으로 인해 신속한 반응성을 확보하기 어렵다[2].

이러한 한계를 극복하기 위해 예측 모델을 활용한 기계학습 기반 스케일링 기법[3][4]과 트래픽 패턴을 분석하여 자동 조정하는 서버리스 플랫폼[5] 등이 대안으로 제안되었다. 그러나, 실제 서비스 환경의 부하 유형(Workload type)을 충분히 반영하지 못하고 시스템 복잡성으로 인해 일반적인 환경에서 적용하기 어렵다는 제약이 존재한다. 특히 메시지 큐의 적체 상태나 메시지 소비 속도(메시지 처리량)와 같은 이벤트 단위의 변화를 오토스케일러가 즉각적으로 감지하고 파드 수 확장에 반영할 수 있는 오토스케일링 기법에 대한 실증적 분석은 여전히 미흡한 상황이다. 기존 연구들은 HPA 개선 및 예측 모델 기반의 오토스케일링 방법론에 국한되어 있다는 제한점이 존재한다. 특히, 이벤트 기반 오토스케일링인 KEDA, Kubernetes Event-driven Autoscaling을 활용하여 다양한 메시지 브로커 환경에서 부하 유형별 성능을 실증적으로 측정하고 비교한 연구는 매우 미흡한 실정이다[6]. 본 논문은 이러한 제한점을 극복하기 위하여, 메시지 브로커 환경에서의 이벤트 기반 오토스케일링 반응 특성과 자원 활용 패턴을 실증적으로 측정하고 분석하여 서비스 부하 유형에 따른 오토스케일링 성능 비교 기준을 제안한다.

본 논문에서는 쿠버네티스 환경에서 서비스 부하 유형에 따른 효율적인 오토스케일링 전략 수립을 위하여 쿠버네티스 이벤트 기반 오토스케일러의 메시지 브로커별 반응 특성과 자원 활용 패턴(Resource utilization patterns)에 대한 실험을 통해 성능 비교 기준을 제안한다.

KEDA의 성능 특성을 정량적으로 비교 분석하기 위하여 KEDA가 지원되는 RabbitMQ, Apache Kafka, Redis Streams, NATS JetStream 등 네 가지 메시지 브로커를 대상으로 급격 부하(Burst workload), 지속 부하(Sustained workload), 부하 감소(Decreasing workload)의 세 가지 부하 유형에 대한 오토스케일링 실험을 수행한다. 각 메시지 브로커별 TCP 기반 부하 생성 도구를 활용하여 반응 특성 및 자원 활용 패턴을 측정하고 Scale-In/Out 반응 시간, Overshoot, CPU·메모리 사용률, 메시지 처리량 및 지연 시간을 비교 분석하여, 쿠버네티스 환경에서 서비스 부하 유형에 따른 KEDA와 메시지 브로커의 오토스케일링 성능 특성을 제시한다.

논문의 구성은 다음과 같다. Ⅱ장에서는 쿠버네티스 오토스케일링 기법의 기본 원리와 기존 연구 동향을 고찰한다. Ⅲ장에서는 KEDA를 활용한 실험 환경과 성능 측정 지표(Metric)를 정의하고, Ⅳ장에서는 부하 유형별 메시지 브로커를 대상으로 수행한 오토스케일링 실험 결과를 분석한다. 마지막으로 Ⅴ장에서는 실험 결과를 종합하여 KEDA의 오토스케일링 특성을 평가하고 비교 기준을 제시한다.

II. 쿠버네티스 오토스케일링

오토스케일링 방법에 대한 연구는 크게 리소스 사용량 기반 확장 개선, 예측 모델 기반 확장, 이벤트 기반 확장의 세 방향으로 발전하고 있다. 특히 HPA의 반응 지연 문제를 개선하기 위하여 임계값 조정이나 기계학습 및 강화학습을 활용하여 부하 패턴을 예측하고 파드 수를 사전에 확장하는 방식이 제안되었다[7]-[9]. 그러나 이러한 예측 기반 접근 방식은 학습 데이터 의존성과 일반화 한계로 인해 다양한 워크로드 환경에서 일관된 성능을 보장하기 어려우며, 모델 구축 및 유지에 따른 복잡성과 초기 비용 부담이 존재한다. 이에 반해 이벤트 기반 오토스케일링은 외부 이벤트를 기반으로 부하 변화를 실시간으로 감지할 수 있다. 본 논문에서는 이러한 특성을 갖는 KEDA를 대상으로 다양한 메시지 브로커 환경에서의 오토스케일링 성능을 분석한다.

본 장에서는 쿠버네티스 환경에서 오토스케일링 기법을 리소스 사용량 기반 방식과 이벤트 기반 방식으로 구분하여 이론적 배경과 구조를 살펴본다. 첫째, 쿠버네티스가 기본적으로 제공하는 리소스 사용량 기반 오토스케일링의 구조와 한계를 고찰한다. 둘째, 메시지와 같은 외부 지표에 기반하여 동작하는 이벤트 기반 오토스케일링의 개념과 특징을 검토한다. 이러한 두 오토스케일링 기법의 비교를 통해 KEDA 중심의 이벤트 기반 오토스케일링 분석의 필요성을 논한다.

2.1 리소스 사용량 기반 오토스케일링

쿠버네티스는 부하 변화에 따라 파드 수나 자원 크기를 자동으로 조정하기 위한 여러 오토스케일러를 제공한다. 대표적으로 수평적 파드 오토스케일링(HPA)은 CPU와 메모리 사용량 같은 리소스 사용량을 기반으로 파드의 수를 자동으로 확장하거나 축소한다[10]. HPA는 메트릭 서버(Metrics server)로부터 일정 주기로 수집된 지표를 바탕으로 대상 디플로이먼트(Deployment)의 복제본 수를 조정하며, 사용자가 정의한 목표 사용률(Target utilization)을 기준으로 파드 수에 대한 확장 임계값을 계산한다. 예를 들어 CPU 사용률이 설정된 목표치를 초과하면 파드 수를 증가시키고, 일정 시간 동안 부하가 감소하면 파드 수를 줄이는 방식으로 동작한다[11]. 이러한 구조는 단순하고 안정적이나, 지표 수집 주기와 통계적 평균값에 의존하기 때문에 급격한 부하 변화에 즉각적으로 반응하기 어렵다는 한계가 있다.

HPA 이외에도 쿠버네티스는 수직적 파드 오토스케일링(VPA, Vertical Pod Autoscaler)과 클러스터 오토스케일링(CA, Cluster Autoscaler)을 지원한다. VPA는 파드의 개수를 변경하지 않고 CPU나 메모리의 리소스 요청(Requests)과 제한(Limits)값을 동적으로 조정하여 개별 파드의 처리 효율을 높이는 방식이다. 반면 CA는 노드 풀(Node pool) 단위로 노드 수를 확장·축소하여 전체 클러스터 용량을 조정하는 역할을 한다[12][13].

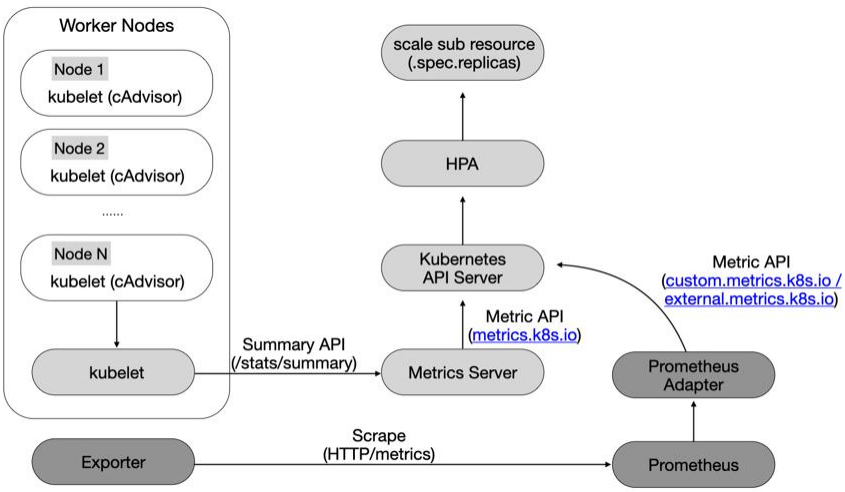


그림 1. HPA와 Prometheus Adapter를 통한 외부 메트릭 연동 구조
Fig. 1. Architecture of HPA with Prometheus Adapter for external-metrics extension

기존 리소스 사용량 기반 오토스케일러들은 CPU나 메모리 사용량과 같은 자원 지표를 기반으로 동작하므로, 비동기 이벤트나 외부 큐 상태와 같은 지표를 직접 활용하기 어렵다. 그림 1은 Prometheus Adapter를 활용하여 외부 메트릭을 HPA에 전달하는 구조로, 외부 메트릭 전달 구조 방식의 복잡성과 메시지 브로커나 비동기 워크로드와 같이 이벤트 발생량이 일정하지 않은 환경에서는 부하 변화를 즉시 반영하기 어렵다는 한계가 존재한다.

2.2 이벤트 기반 오토스케일링

리소스 사용량 기반 오토스케일링의 한계를 보완하기 위하여 이벤트 기반 오토스케일링 기법이 제안되었다. 이벤트 기반 오토스케일링은 CPU·메모리와 같은 내부 리소스 사용률이 아닌 큐 길이, 요청 건수, 스트림 메시지 수 등 외부 이벤트의 발생량을 지표로 활용하여 오토스케일링 동작을 수행한다. 이 방식은 부하가 시스템 내부에 반영되기 전에 외부 이벤트 수준에서 변화를 감지할 수 있기 때문에 빠른 반응성을 확보할 수 있다. 대표적인 구현체로는 쿠버네티스 이벤트 기반 오토스케일러(KEDA)가 있으며, 일부 서버리스 플랫폼에서도 유사한 이벤트 트리거 기반 오토스케일링을 지원한다[14].

KEDA는 쿠버네티스 네이티브 형태로 설계된 이벤트 기반 오토스케일러로 메시지 브로커나 큐 시스템으로부터 메트릭을 수집하여 HPA와 연동한다. KEDA의 핵심 구성 요소는 그림 2와 같이 KEDA Operator, ScaledObject를 포함하는 CRD(Custom Resource Definition), 그리고 HPA에 외부 메트릭을 제공하는 KEDA Metrics API Server로 구분된다[15].

ScaledObject는 오토스케일링이 적용될 디플로이먼트 및 트리거 조건을 정의하며, Trigger는 외부 시스템과의 연결을 통해 부하 수준을 측정한다. Metrics Adapter는 이 측정 결과를 쿠버네티스 메트릭 API 형태로 변환하여 HPA에 전달함으로써, KEDA가 실질적으로 HPA의 동작을 제어하는 구조를 가진다.

이러한 구조를 통해 KEDA는 RabbitMQ, Apache Kafka, Redis Streams 및 NATS JetStream 등 다양한 메시지 브로커를 지원하며, 이벤트의 큐 적체량이나 메시지 소비 속도 등을 실시간으로 감시하여 파드 수를 자동으로 조정한다. 이를 통해 기존 HPA의 주기적 지표 수집 구조보다 빠른 파드 수 확장과 축소 반응을 실현할 수 있다.

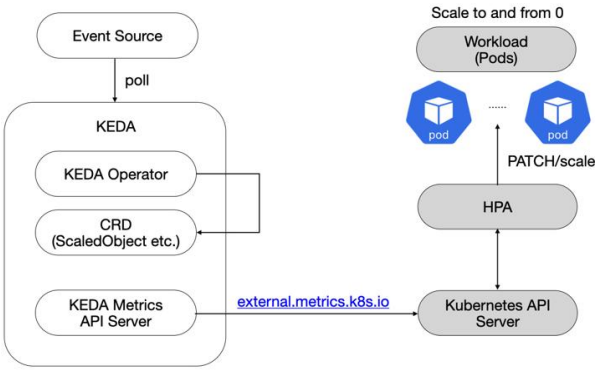


그림 2. KEDA와 HPA의 연동 구조
Fig. 2. Architecture of KEDA and integration with HPA

III. 오토스케일링 실험 환경 설계 및 구현

본 장에서는 쿠버네티스 이벤트 기반 오토스케일러의 성능을 평가하기 위해 구축한 실험 환경의 설계 및 구현 과정을 기술한다. 이를 위하여 먼저 클러스터 및 네트워크 환경을 제시하여 실험 기반 인프라의 특성을 설명하고, 둘째로 KEDA 및 메시지 브로커의 구성을 논한다. 셋째로 부하 유형과 조건(Parameter)을 정의하여 다양한 서비스 상황을 모사하기 위한 실험 조건을 설정하며, 마지막으로 성능 측정 지표를 제시하고 이를 특성별로 분류하여 반응 특성과 자원 활용 패턴의 두 가지 평가 범주(Category)로 구분함으로써, 실험 결과 분석을 위한 기준과 비교 체계를 명확히 한다.

3.1 물리 테스트 베드 및 네트워크 환경

실험 클러스터는 동일 사양의 물리 서버 3대로 마스터 역할 서버 1대와 워커 역할 서버 2대로 구성된다. 각 노드는 그림 3에 나타낸 것과 같이 10Gbps DAC(Direct Attach Copper) 케이블을 이용하여 고속 내부망으로 연결되었으며, 내부 네트워크는 Linux Bridge(br0)를 기반으로 구성한다. 외부망은 관리 및 접근용 네트워크로, 내부망은 데이터 교환용 네트워크로 분리한다.

모든 노드는 표 1과 동일한 CPU, 메모리 사양을 갖추어 하드웨어적 편차에 따른 실험 오차를 최소화한다. 클러스터의 CNI(Container Network Interface)는 Calico의 IP-in-IP 방식을 사용하며[16], MTU(Maximum Transmission Unit)는 9000으로 설정하여 패킷 전송 효율을 높인다.

표 1. 물리 노드 하드웨어 및 네트워크 사양
Table 1. Hardware and network specification of physical nodes

No.	Item	Specification
1	CPU	AMD Ryzen 550GT
2	Memory	16GB
3	Simultaneous Multithreading (SMT)	Disabled
4	Operating system	Ubuntu 24.04.3 LTS
5	Kubernetes version	1.31.12
6	Network interface card (Internal)	Realtek RTL8111/8168/8211/8411
7	Network interface card (External)	Intel X710-DA4-FH
8	Interface MTU	9000

3.2 KEDA 및 메시지 브로커 구성

그림 4에는 KEDA 기반의 오토스케일링 배포 구조를 나타내었다. 생산자 파드(Producer pod)는 메시지를 브로커로 발행하고, 소비자 파드(Consumer pod)는 KEDA 트리거에 의해 확장과 축소를 동작한다. 실험 환경에서는 역할별 파드 간 자원 간섭을 최소화하고, 서비스 간 통신을 독립적으로 관찰할 수 있도록 생산자 파드는 워커 노드 01(Worker Node 01)에 소비자 파드는 워커 노드 02(Worker Node 02)에 우선 배치되도록 설정한다. 이 구성은 쿠버네티스 환경에서 리소스 간섭을 줄이고 성능을 독립적으로 검증할 수 있도록 한다[17][18].

KEDA는 브로커의 대기 메시지 수, 스트림 길이, 또는 소비 지연(Consumer lag)과 같은 값을 주기적으로 감시하여, 트리거(Trigger) 조건이 만족될 경우 파드 수를 조정한다. 이 구조를 통해 부하 변화에 따라 소비자 파드의 개수를 자동으로 관리할 수 있다.

실험에 사용된 메시지 브로커와 트리거 조건은 표 2와 같다. RabbitMQ는 큐 길이(Queue length), Redis Streams는 스트림 길이(Stream length), Apache Kafka와 NATS JetStream은 소비 지연(Consumer lag)를 트리거로 사용한다.

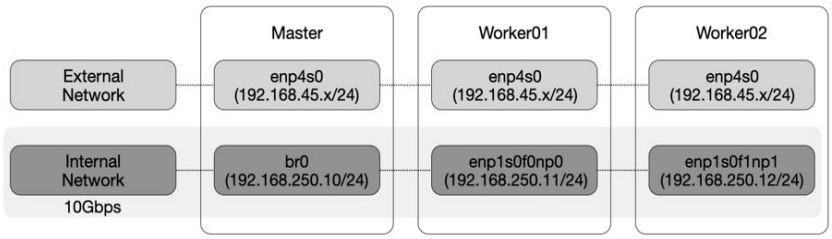


그림 3. 물리 테스트 베드 네트워크 구성
Fig. 3. Network configuration of the physical test bed

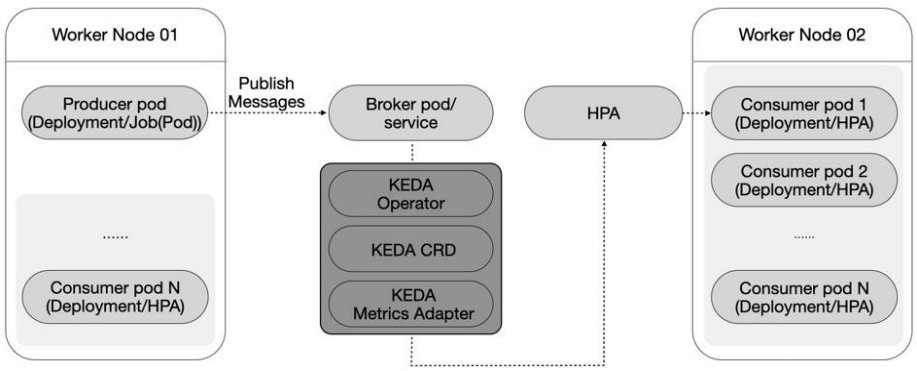


그림 4. KEDA 기반 생산자/소비자 파드 배포 구조
Fig. 4. Architecture of KEDA-based producer/consumer deployment

표 2. 메시지 브로커별 트리거 지표
Table 2. Trigger metrics by message broker

No.	Message broker	Scaling trigger
1	RabbitMQ	Queue length (ready messages)
2	Redis Streams	Stream length (XLEN)
3	Apache Kafka	Consumer lag (offset lag)
4	NATS JetStream	Consumer lag (ack pending messages)

3.3 부하 유형 및 조건 정의

이벤트 기반 오토스케일러의 반응 특성을 정량적으로 평가하기 위하여 세 가지 부하 유형을 표 3과 같이 정의한다. 각 부하 유형은 메시지 생성 속도(Message generation rate), 파드당 처리 가능 메시지 용량(Per-Pod processing capacity) 그리고 부하 유지 시간(Workload duration)의 세 가지 변수를 조합하여 구성한다.

표 3. 부하 유형 파라미터
Table 3. Parameters of workload patterns

No.	Workload type	Parameter	Value
1	Burst workload	Message generation rate (msgs/s)	8000
		Per-Pod processing capacity (msgs/s/pod)	1500
		Workload duration (s)	90
2	Sustained workload	Message generation rate (msgs/s)	3000
		Per-Pod processing capacity (msgs/s/pod)	1000
		Workload duration (s)	600
3	Decreasing workload	Message generation rate (msgs/s)	6000
		Per-Pod processing capacity (msgs/s/pod)	1200
		Workload duration (s)	60

첫째, 급격 부하는 단기간(90s)동안 대량 메시지(8,000msgs/s)를 큐에 삽입하여 순간 트래픽 급증 상황을 모의한다. 둘째, 지속 부하는 메시지 생성 속도(3,000msgs/s)와 파드당 처리 용량(1,000msgs/s)을 설정한 상태에서 전체 부하 지속 시간(600s)을 유지하여 안정적 확장 동작을 관찰한다. 셋째, 급격 부하 감소는 트래픽(6,000msgs/s)이 특정 시간(60s) 이후 갑자기 종료되는 상황을 모의하여 축소 반응성을 검증한다.

3.4 성능 측정 지표 및 평가 범주

성능 측정 지표는 반응 특성과 자원 활용 패턴 두 범주로 구분한다. 반응 특성은 부하 변화에 따라 KEDA에 의한 파드 확장 반응 시간(Scale-Out), 파드 축소 반응 시간(Scale-In) 및 오버슈트(Overshoot)를 기준으로 파드 확장 안정성을 평가한다. 자원 활용 패턴은 변화하는 부하 조건에 따라 KEDA에 의해 확장된 파드의 CPU 사용률(CPU utilization), 메모리 사용률(Memory utilization), 메시지 처리량(Message throughput), 메시지 지연 시간(Message latency)을 기준으로 CPU 및 메모리 자원 사용 평가, 메시지 처리 성능 평가, 메시지 처리 서비스 응답 안정성을 평가한다. 각 성능 측정 지표의 정의와 측정 목적(Evaluation objective)은 표 4와 같다.

표 4. 성능 측정 지표
Table 4. Performance evaluation metrics

No.	Category	Metric	Definition	Evaluation objective
1	Reaction characteristics	Scale-Out (ms)	Time to first scale-out triggered by KEDA	Scale-Out responsiveness
		Scale-In (ms)	Time to first scale-in triggered by KEDA	Scale-In responsiveness
		Overshoot (pod)	Number of excess pods beyond required capacity	Scaling stability
2	Resource utilization patterns	CPU utilization (%)	Average CPU utilization of scaled pods	CPU efficiency
		Memory utilization (%)	Average Memory utilization of scaled pods	Memory efficiency
		Message throughput (msgs/s)	Message processed per second	Processing performance
		Message latency (ms)	Average message processing delay	Service stability

모든 성능 측정 지표는 반응 특성과 자원 활용 패턴의 두 범주에 따라 구분하여 해석함으로써, 부하 유형에 따른 오토스케일러의 동작 성능 차이를 체계적으로 비교할 수 있도록 한다. 각 성능 측정 지표의 값은 파드 단위 평균값을 기준으로 분석하며, 메시지 처리량과 메시지 지연 시간은 각 메시지 브로커의 부하 도구 결과 로그를 기반으로 산출한다.

Ⅳ. 오토스케일링 실험 및 성능 분석

본 장에서는 Ⅲ장에서 구축된 실험 환경을 기반으로 메시지 브로커별 부하 유형에 따른 KEDA의 오토스케일링 실험을 수행하고, 메시지 브로커와 부하 유형별 결과를 정량적으로 분석한다. 각 실험은 동일한 부하 조건에서 3회 반복 측정하여 평균값을 도출하였으며, 정의된 성능 측정 지표를 통해 오토스케일러의 반응성, 안정성, 자원 활용률을 종합적으로 분석한다.

4.1 급격 부하 결과

급격 부하 환경에서 각 메시지 브로커의 평균 성능을 비교한 결과를 표 5에 나타내었다.

표 5. 급격 부하 결과 평균
Table 5. Average performance under burst workload

No.	Metric	Rabbit MQ	Redis streams	Apache kafka	NATS JetStream
1	Scale-Out (ms)	4000.0±43.6	5000.0±45.8	158.7 ±1.5	3656.7±41.6
2	Scale-In (ms)	38333.3± 107.9	90270.0± 701.9	N/A	317989.3 ±2653.9
3	Overshoot (pod)	0.0±0.0	24.0±0.0	24±0.0	6.0±0.0
4	CPU utilization (%)	2.3±0.1	2.9±0.0	2.5±0.1	6.5±0.3
5	Memory utilization (%)	0.4±0.0	0.3±0.0	1.2±0.0	0.2±0.0
6	Message throughput (msgs/s)	16216.7± 104.1	8000.0±86.6	8000.0±43.6	554.3 ±3.8
7	Message latency (ms)	0.6±0.0	0.0±0.0	1.0±0.0	0.0±0.0

CPU 및 메모리 사용률의 표준편차는 소수점 첫째자리 기준으로 나타내었으며, 이는 측정값의 절대적 스케일이 작은데에 따른 것으로 회차 간 편차가 매우 낮은 수준임을 의미한다.

급격 부하는 단기간 내 트래픽이 집중되는 특성상 확장 민첩성을 나타내는 Scale-Out 반응 시간과 메시지 처리량이 핵심 성능 지표로 작용한다. RabbitMQ는 안정성과 일관성 측면에서 우수한 성능을 보였으며, Redis Streams는 Overshoot가 24개 발생하여 확장 민첩성은 낮았으나 개별 파드의 CPU 및 메모리 사용률이 비교적 낮게 유지되어 자원 효율 측면에서 강점을 나타냈다. NATS JetStream은 반응 속도는 빠르지만 메시지 처리량이 낮아 처리량 측면의 성능은 상대적으로 제한되었다. 반면, Apache Kafka는 평균 158ms(표준편차 1.5ms)의 Scale-Out 반응 시간을 기록하였으며, Consumer Lag 기반 트리거를 통해 메시지 적체를 즉시 감지함으로써 반응 속도와 처리량 측면에서 우세한 성능을 보였다. 다만 Scale-In 반응 시간이 측정되지 않은 이유는 잔존 로그로 인한 측정 지연이 발생한 것으로 해석된다. 이와 같이 급격 부하 환경의 핵심 성능 측정 지표인 Scale-Out 반응 시간과 메시지 처리량 측면에서 Apache Kafka가 가장 우수한 성능을 나타내었다. 특히 Apache Kafka의 Scale-Out 반응 시간은 표준편차가 1.5ms로 작아 반복 측정 간 편차가 거의 없는 것으로 확인되었다.

4.2 지속 부하 결과

일정 부하가 장시간 유지되는 환경에서의 실험 결과는 표 6에 나타내었다. 지속 부하 조건에서는 장기간 안정적인 처리가 요구되는 특성상 CPU·Memory 사용률과 Scale-Out 반응 속도가 핵심 성능 지표로 작용한다.

RabbitMQ는 CPU 1.3%(표준편차 0.0%)와 메모리 0.2%(표준편차 0.0%)로 회차 간 편차가 거의 없는 일관된 자원 활용률을 유지하였다. Redis Streams는 Overshoot가 27개 발생하였으나 전체 메시지 처리량의 편차가 작아 처리 안정성을 확보하였다. Redis Streams의 Scale-In이 측정되지 않은 원인은 트리거 지표 특성상 메시지 소비 이후에도 스트림 엔트리가 자동 삭제되지 않아 KEDA가 지속적으로 처리

대상 메시지가 존재한다고 판단하였기 때문이며, 이로 인해 Scale-In 조건이 충족되지 않아 축소 반응 시간을 측정할 수 없었다. Redis Streams에서 측정된 308,000ms의 높은 지연 시간은 트리거 지표 기반 잔류 메시지(backlog) 값이 지속적으로 유지되는 특성으로 인해 KEDA가 실제 처리량 감소 이후에도 높은 부하 상태로 판단하여 불필요한 Scale-Out이 반복적으로 발생하였고, 소비자 간 처리 경쟁 및 메시지 대기 시간이 증가한 것으로 분석되었다. 또한 Redis Streams와 Apache Kafka에서 관측된 높은 Overshoot는 잔류 메시지 및 트리거 지표 정보가 즉시 감소하지 않는 구조적 특성에 기인하여 KEDA가 반복적으로 Scale-Out을 수행한 것으로 확인되었다. NATS JetStream은 메시지 처리 지연 시간이 가장 낮아 지연 측면에서 강점을 보였다. 반면 Apache Kafka는 평균 240.0ms(표준편차 5.0ms)의 짧은 Scale-Out 반응 시간을 기록하여 빠른 확장 응답성을 나타내었으며, 메시지 처리량은 Redis Streams와 동일한 수준을 유지하였다. 이와 같이 지속 부하 환경의 핵심 성능 측정 지표인 CPU·Memory 사용률과 Scale-Out 반응 속도 측면에서 Apache Kafka가 가장 안정적인 특성을 나타내었으며, 이는 파티션 기반 병렬 처리 구조가 KEDA의 이벤트 트리거 인식 및 확장 동작에 반영된 결과로 판단된다.

표 6. 지속 부하 결과 평균
Table 6. Average performance under sustained workload

No.	Metric	Rabbit MQ	Redis streams	Apache kafka	NATS JetStream
1	Scale-Out (ms)	4666.7±57.7	4333.3±28.9	240.0±5.0	2476.7±23.1
2	Scale-In (ms)	36700.0±86.8	N/A	N/A	313933.3±1101.5
3	Overshoot (pod)	0.0±0.0	27.0±0.0	27±0.0	17.0±0.0
4	CPU utilization (%)	1.3±0.0	0.8±0.0	1.8±0.0	7.1±0.1
5	Memory utilization (%)	0.2±0.0	0.0±0.0	0.5±0.0	0.5±0.0
6	Message throughput (msgs/s)	4810.0±10.0	3000.0±17.3	5006.7±15.3	163.7±1.5
7	Message latency (ms)	1.0±0.0	308000.0±500.0	2496.7±25.2	0.0±0.0

4.3 부하 감소 결과

부하가 급격히 감소하는 시점에서의 이벤트 기반 오토스케일러 반응 결과를 표 7에 제시하였다. 부하 감소 환경에서는 트래픽 급감 후의 불필요한 자원 점유를 방지하기 위한 Scale-In 반응 속도와 메시지 지연 시간이 핵심 성능 측정 지표로 작용한다.

표 7. 부하 감소 결과 평균
Table 7. Average performance under decreasing workload

No.	Metric	Rabbit MQ	Redis streams	Apache kafka	NATS JetStream
1	Scale-Out (ms)	5333.3±53.5	3000.0±30.0	920.0±9.0	4947.0±49.0
2	Scale-In (ms)	40000.0±400.0	360000.0±3600.0	N/A	21107.0±211.0
3	Overshoot (pod)	0.0±0.0	25.0±0.0	25±0.0	3.0±0.0
4	CPU utilization (%)	0.6.±0.0	0.8±0.0	1.9±0.0	6.4±0.1
5	Memory utilization (%)	0.3±0.0	0.0±0.0	0.3±0.0	0.0±0.0
6	Message throughput (msgs/s)	9664.0±97.0	6000.0±60.0	6000±60.0	265.0±3.0
7	Message latency (ms)	0.8±0.0	54700.0±547.0	54667.0±547.0	0.0±0.0

RabbitMQ는 안정적인 동작을 보였으나 축소 반응 속도는 상대적으로 느린 것으로 나타났으며, Redis Streams는 Overshoot가 25개 발생하여 과잉 확장 이후 축소가 지연되는 현상이 관찰되었다. Redis Streams와 Apache Kafka에서는 각각 54,700ms와 54,667ms의 높은 지연 시간이 측정되었다. 이는 잔류 메시지 및 트리거 지표 정보가 즉시 감소하지 않는 구조적 특성으로 인해 KEDA가 실제 부하 감소 이후에도 지속적으로 높은 부하 상태로 판단하여 불필요한 Scale-Out이 반복 수행되었고, 이로 인해 메시지 처리 대기 시간이 증가한 것으로 분석되었다. Apache Kafka는 잔류 메시지로 인해 축소 트리거가 발생하지 않아 Scale-In이 불가능하였다. 이 결과는 이벤트 기반 오토스케일러의 축소 효율이 트리거 반응 시간뿐 아니라, 메시지 소비 구조의 특성과 밀접히 연관되어 있음을 보여주었다. 반면,

NATS JetStream은 메시지 확인(Ack) 기반 구조로 인해 불필요한 파드 잔류가 방지되어 평균 21초 내외의 Scale-In 반응 시간을 기록하였다. 이와 같이 부하 감소 환경의 핵심 성능 측정 지표인 Scale-In 반응 속도와 메시지 지연 시간 측면에서 NATS JetStream이 평균 21107.0ms(표준편차 211.0ms)의 반응 시간으로 우수한 성능을 나타내었으며, 반복 측정 간 편차가 작아 효율적인 자원 회수 모델을 구현하고 있음을 확인하였다.

본 장에서 제시한 메시지 브로커 간 성능 비교는 각 부하 유형 조건별 3회 반복 측정의 평균과 표준편차에 기반하였으며, 대부분의 지표에서 변동계수가 5% 이내로 나타나 측정값의 재현성이 높은 것으로 확인되었다. 결과적으로 KEDA는 단일 환경에서 절대적인 성능 우위를 나타내는 것이 아니라, 부하 유형과 메시지 브로커 구조 간의 상호작용에 따라 성능 특성이 상대적으로 달라지는 특성을 갖는다. 따라서 실제 서비스 환경에서는 부하 유형과 메시지 처리 모델에 따라 적절한 메시지 브로커와 오토스케일러 조합을 선택해야 하며, 이를 통해 최적의 오토스케일링 성능을 확보할 수 있을 것이다.

Ⅴ. 결 론

클라우드 네이티브 기술과 마이크로서비스 아키텍처의 확산으로 분산 서비스 간 비동기 통신을 담당하는 메시지 브로커의 중요성이 점차 부각되어 서비스 간 데이터 전달을 지원하는 핵심 인프라로 자리 잡았다. 그러나 쿠버네티스의 기존 수평적 파드 오토스케일링만으로는 이벤트 기반 메시지 브로커 환경에서 발생하는 부하에 즉각적으로 대응하는데 한계가 존재하며, 메시지 큐 적체 및 소비 속도와 같은 이벤트 단위의 변화를 즉각적으로 감지하고 확장에 반영할 수 있는 오토스케일링 기법에 대한 실증적 분석이 미흡한 실정이다.

본 논문은 이러한 한계를 보완하기 위해 쿠버네티스 환경에서 이벤트 기반 오토스케일러인 KEDA의 특성을 메시지 브로커별 부하 유형에 따라 실험적으로 분석하고, 실험 결과를 바탕으로 부하 유형별 대표 성능 측정 지표와 KEDA에 적합한 메시지 브로커를 제시하였다. 첫째, 급격 부하 환경에서 확장 민첩성을 평가하기 위하여 Scale-Out 반응 속도와 메시지 처리량 분석한 결과, Apache Kafka가 평균 158.7ms(표준편차 1.5ms)로 가장 빠른 반응속도를 나타냈다. 둘째, 지속 부하 환경에서 장기적 안정성 검증을 위해 CPU·메모리 사용률과 Scale-Out 반응 속도를 분석한 결과, 이 영역에서도 Apache Kafka가 CPU 1.8%, 메모리 0.5% 수준의 낮은 자원 사용률과 평균 240.0ms(표준편차 5.0ms)의 짧은 Scale-Out 반응 시간을 기록하여 상대적으로 높은 안정성을 보였다. 셋째, 부하 감소 환경에서 축소 반응 특성을 평가하기 위해 Scale-In 반응 속도와 메시지 처리 지연 시간을 분석하였는데 NATS JetStream 환경에서 평균 21107.0ms(표준편차 211.0ms)로 KEDA를 통한 가장 빠른 축소 반응 특성이 나타났다. 따라서 본 실험 결과를 바탕으로 KEDA와의 연동 관점에서 워크로드 유형별 메시지 브로커 선택 시 고려할 수 있는 정량적 기준을 다음과 같이 제안할 수 있다. 급격 부하 환경에서는 평균 158.7ms(표준편차 1.5ms)의 Scale-Out 반응 시간과 평균 8000.0msgs/s(표준편차 43.6)의 메시지 처리량을 보인 Apache Kafka가 반복 측정 전반에서 상대적으로 안정적인 것으로 분석되었다. 지속 부하 환경에서는 평균 240.0ms(표준편차 5.0ms)의 짧은 Scale-Out 반응 시간과 CPU 1.8%와 메모리 0.5%의 낮은 자원 사용률을 기록한 Apache Kafka가 측정값의 변동폭이 작아 일관된 자원 효율성을 나타내었으며 장기적 안정성 측면에서 상대적으로 적합한 것으로 분석되었다. 부하 감소 환경에서는 평균 21,107ms(표준편차 211.0ms)의 빠른 Scale-In 반응 시간을 기록한 NATS JetStream이 효율적인 자원 회수 측면에서 강점을 나타내었다. 이러한 결과는 향후 메시지 브로커 기반 서비스 환경에서 확장성과 자원 활용 패턴을 동시에 확보하기 위한 오토스케일링 전략 수립에 있어 정량적 비교 근거로 활용될 수 있을 것이다.

References

[1] O. C. Oyeniran, A. O. Adewusi, G. Adeleke, L. Akwawa, and C. F. Azubuko, "Microservices architecture in cloud-native applications: Design

patterns and scalability", Computer Science & IT Research Journal, Vol. 5, No. 9, pp. 2107-2124, Sep. 2024. https://doi.org/10.51594/csitrj.v5i9.1554.

[2] D. R. Augustyn, Ł. Wyciślik, and M. Sojka, "Tuning a Kubernetes Horizontal Pod Autoscaler for Meeting Performance and Load Demands in Cloud Deployments", Applied Sciences, Vol. 14, No. 2, Art. no. 646, Jan. 2024. https://doi.org/10.3390/app14020646.

[3] Y. H. Jang, H. Yu, and S. S. Kim, "Transfer Learning Technique for Accelerating Learning of Reinforcement Learning-Based Horizontal Pod Autoscaling Policy", KIPS Transactions on Computer and Communication Systems, Vol. 11, No. 4, pp. 105-112, Aug. 2022. https://doi.org/10.3745/KTCCS.2022.11.4.105.

[4] Y. Jang, J. Jeong, J. Lee, H. Yu, and E. Lee, "Modelling the Kubernetes Horizontal Pod Autoscaler based on number of requests for comparing the autoscaling algorithm", Proc. of Korean Institute of Information Scientists and Engineers, Korea Computer Congress, Jeju, Korea, pp. 1385-1387, Jun. 2021.

[5] O. Kyrychenko, S. Ostapov, and O. L. Kyrychenko, "Predictive autoscaling in AWS Serverless by means of machine learning and SQS metrics", CEUR Workshop Proceedings, Khmelnytskyi, Ukraine, Vol. 3963, pp. 77-87, Apr. 2025.

[6] X. Pilyai, R. N. Irsyad, I. N. Zaini, R. M. Negara, S. N. Hertiana, and R. Tulloh, "Implementation and Benchmarking of Kubernetes Horizontal Pod Autoscaling Method to Event-Driven Messaging System", Advances on Broad-Band and Wireless Computing, Communication and Applications (BWCCA 2023), Lect. Notes Data Eng. Commun. Technol., Vol. 186, pp. 45-56, Nov. 2023. https://doi.org/10.1007/978-3-031-46784-4_5.

[7] A. Rubak and J. Taheri, "Machine Learning for Predictive Resource Scaling of Microservices on Kubernetes Platforms", 2023 IEEE/ACM 16th International Conference on Utility and Cloud Computing, Taormina, Italy, pp. 1-8, Dec. 2023. https://doi.org/10.1145/3603166.3632165.

[8] P. B. Guruge and Y. H. P. Priyadarshana, "Time Series Forecasting-Based Kubernetes Autoscaling using Facebook Prophet and Long Short-Term Memory", Frontiers in Computer Science, Vol. 7, Art. no. 1509165, Feb. 2025. https://doi.org/10.3389/fcomp.2025.1509165.

[9] Y. H. Kim and Y. H. Kim, "Improvement of Kubernetes Auto-Scaling Based on Multivariate Time Series Analysis", KIPS Transactions on Computer and Communication Systems, Vol. 11, No. 3, pp. 73-82, Mar. 2022. https://doi.org/10.3745/KTCCS.2022.11.3.73.

[10] Horizontal Pod Autoscaler. Kubernetes.Io, https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. [accessed: Nov. 01, 2025]

[11] Autoscaling Concepts. Kubernetes.Io, https://kubernetes.io/docs/concepts/workloads/autoscaling/. [accessed: Nov. 01, 2025]

[12] Vertical Pod Autoscaler. Kubernetes.Io,https://kubernetes.io/docs/concepts/workloads/autoscaling/vertical-pod-autoscaler/. [accessed: Nov. 03, 2025]

[13] Cluster Autoscaler. Kubernetes.Io, https://kubernetes.io/docs/concepts/cluster-administration/cluster-autoscaler/. [accessed: Nov. 13, 2025]

[14] Kubernetes Event-driven Autoscaling (KEDA). KEDA, https://keda.sh/. [accessed: Nov. 04, 2025]

[15] KEDA Concepts. KEDA.Sh, https://keda.sh/docs/2.17/concepts/. [accessed: Nov. 04, 2025]

[16] S. H. Paek, S. W. Gu, and D. W. Yoon, "Performance Analysis of Container Network Interface for Kubernetes Networking", Journal of Korean Institute of Information Technology, Vol. 23, No. 5, pp. 117-128, May 2025. http://dx.doi.org/10.14801/jkiit.2025.23.5.117.

[17] A. Marchese and O. Tomarchio, "Evaluating Microservices Communication Relationships for Scheduling Containers on Kubernetes Clusters", Communications in Computer and Information Science, Vol. 1845, pp. 45-65, Aug. 2024. https://doi.org/10.1007/978-3-031-68165-3_3.

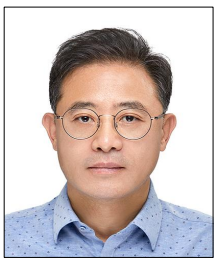
[18] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga, "Practical Service Placement Approach for Microservices Architecture", Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Madrid, Spain, pp. 401-410, May 2017. https://doi.org/10.1109/CCGrid.2017.28.

저자소개

임 재 연 (Jaeyeon Lim)

2019년 8월 : 단국대학교 영미인문학과·소프트웨어학과(학사)
2026년 2월 : 한양대학교 통신정보공학과(공학석사)
2021년 7월 ~ 2022년 7월 : 삼양데이타시스템 클라우드 엔지니어
2022년 9월 ~ 2024년 7월 : 안랩 대리
2024년 7월 ~ 현재 : 안랩클라우드메이트 솔루션 아키텍트
관심분야 : 클라우드 플랫폼, 쿠버네티스

구 세 완 (Sewan Gu)



1994년 2월 : 한양대학교 전자통신공학과(공학사)
1996년 2월 : 한양대학교 전자통신공학과(공학석사)
2023년 8월 : 한양대학교 전자컴퓨터통신공학과(공학박사)
1997년 3월 ~ 2000년 3월 : 한국철도기술연구원
2000년 3월 ~ 현재 : LG전자 책임연구원
관심분야 : 클라우드 로보틱스, 네트워크 제어 시스템, 지능형 로봇시스템,

윤 동 원 (Dongweon Yoon)

1989년 2월 : 한양대학교 전자통신공학과(공학사)
1992년 2월 : 한양대학교 전자통신공학과(공학석사)
1995년 8월 : 한양대학교 전자통신공학과(공학박사)
1995년 3월 ~ 1997년 8월 : 동서대학교 정보통신공학과 조교수
1997년 9월 ~ 2004년 2월 : 대전대학교 정보통신공학과 부교수
2004년 3월 ~ 현재 : 한양대학교 융합전자공학부 석학교수
관심분야 : 통신이론, 위성 및 우주통신, 추정 및 검출