# 서버리스 컴퓨팅을 위한 다양한 컨테이너 런타임 환경에서 라이브러리 공유 성능 분석

김세진, 유헌창
고려대학교 컴퓨터학과
{sejjj120, yuhc}@korea.ac.kr

# Performance Analysis of Sharing Library in Various Container Runtime Environments for Serverless Computing

Sejin Kim, Heonchang Yu
Dept. of Computer Science and Engineering, Korea University

**요 약**

서버리스 컴퓨팅에는 가상머신보다 가벼운 장점을 지닌 컨테이너 기술이 많이 사용되었다. 하지만 여러 사용자들의 어플리케이션이 하나의 서버 인스턴스를 공유해서 사용하기 때문에, 취약점으로 인해서 여러 문제점이 생길 수 있다. 이에 서버리스 컴퓨팅 제공자들은 서버리스 컴퓨팅에 적합하며 보안을 강조한 기술들을 발표하고 있다. 대표적으로 구글에서 개발한 샌드박스 형태의 컨테이너 런타임을 제공하는 gVisor 와 오픈스택 재단에서 개발한 Kata Containers 가 있다. 본 논문에서는 미리 준비된 라이브러리를 공유하여 서버리스 컴퓨팅의 콜드 스타트를 완화시키는 관점에서, gVisor 와 Kata Containers 환경에서 라이브러리를 불러올 때 기존의 도커 컨테이너 환경과의 차이를 비교하고 분석한다.

## 1. 서론

서버리스 컴퓨팅이란 개발자가 인프라 등을 고려하지 않고, 어플리케이션의 로직이나 함수의 코드만 작성하여 서비스를 제공하는 클라우드 컴퓨팅 개념이다. 서버리스 컴퓨팅을 사용하면 서버 관리를 할 필요가 없다는 장점이 있지만, 보안성과 콜드 스타트(Cold Start)라는 문제가 발생한다. 서버리스 컴퓨팅에서 사용자의 함수는 어느 인스턴스에서 실행되는 지 알 수 없을 뿐만 아니라 같은 인스턴스 내에서 실행되는 다른 함수에 의해서 영향을 받을 수도 있다. 또한, 사용하려는 코드를 다운 받고 필요한 라이브러리 혹은 모듈을 준비해야 하기 때문에 코드를 실행시킬 때까지 대기해야 하는 콜드 스타트가 발생한다.

서버리스 컴퓨팅에서 취약점을 해결하기 위한 연구로는 AWS 의 Firecracker, 구글의 gVisor 그리고 오픈스택 재단의 Kata Containers 등이 있다. AWS 의 Firecracker 는 리눅스의 KVM 과 함께 사용되며, 가상화 소프트웨어인 QEMU 를 대체하여 경량화 가상머신(MicroVM) 안에서 어플리케이션을 실행시킨다. gVisor 는 기존의 도커에서 사용하고 있는 runC 대신 샌드박스 형태의 runsc 를 제공한다. gVisor 는 runC 에 결함[1]이 있는 것을 고려하여 컨테이너가 사용할 수 있는 시스템 콜을 제한함으로써 컨테이너와 Host OS 를 분리시켰다. Kata Containers 는 오픈스택 재단에서 관리하는 오픈소스 컨테이너 런타임으로, 컨테이너와 같이 동작하지만 가상머신의 격리 수준을 가지는 컨테이너 런타임이다.

현재 연구 중인 대부분의 프로젝트들이 격리 수준을 추가함으로써 취약점을 해결하고 있다. 하지만 기존의 컨테이너 런타임과 비교하였을 때 추가적인 작업이 이루어지기 때문에 성능이 다소 떨어지는 모습[2]을 보임에도 불구하고, 서버리스 컴퓨팅의 관점에서의 분석이 부족한 상황이다. 특히, 서버리스 컴퓨팅에서 라이브러리나 모듈을 빠르게 불러오는 것은 콜드 스타트를 완화시킬 수 있는 중요한 이슈[3]이기 때문에 Host OS 와 컨테이너가 파일을 공유하는 것 에 대한 연구가 필요하다. 본 논문에서는 gVisor 와 Kata Containers 기반의 컨테이너 환경에서 함수가 필요로 하는 라이브러리를 읽어 들이는 시간을 비교하고 분석한다.

본 논문은 2 장에서 서버리스 컴퓨팅과 gVisor, Kata Containers 에 대한 관련 연구를 살펴본다. 3 장에서는 여러 컨테이너 런타임을 활용하여 Node.js 컨테이너에서 모듈을 불러오는 시간을 비교하고 차이점을 분

석한다. 4 장에서는 실험 결과에 대한 평가와 향후 연구 계획을 제시한다.

## 2. 관련 연구

### 2.1 콜드 스타트

서버리스 컴퓨팅은 사용자의 코드나 런타임 환경을 미리 준비하지 않는다. 코드를 동작시킬 트리거에 의해서 요청이 들어오면, 저장소에서 코드를 다운로드 받는다. 그 후에 Node.js, Python, C# 등의 런타임 환경을 제공하는 컨테이너 혹은 가상머신을 준비하고 라이브러리 준비와 같은 추가 작업을 진행한다[4].

서버리스 컴퓨팅에서 발생하는 콜드 스타트에 많은 영향을 주는 것 중 하나가 필요한 라이브러리를 다운로드 받는 것이다. Edward Oakes et al. [3]은 라이브러리를 준비하는 것을 최적화 시키기 위해 여러 개의 Python 모듈을 불러오는 것을 실험했고, 프로비저닝을 강조한 SOCK 이라는 컨테이너 시스템을 개발했다. 도커에서는 Host OS 의 파일에 대한 접근을 제공한다[5]. 도커 전용 저장공간 Volume 을 사용하여 라이브러리를 미리 다운로드 받아 두면, Host OS 의 라이브러리를 여러 컨테이너들이 공유하여 사용할 수 있다.

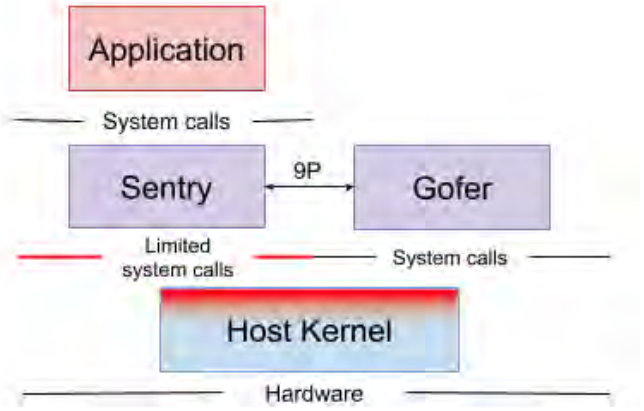
### 2.2 gVisor

gVisor 는 Go 언어로 작성된 어플리케이션 커널이며, 리눅스의 시스템 콜 인터페이스의 일부분을 제공하는 샌드박스이다[6]. gVisor 는 여러 컴포넌트로 구성이 되는데, Sentry 와 Gofer 가 가장 중요하고 대표적인 프로세스이다. Sentry 가 실제로 컨테이너를 실행시키고 시스템 콜을 가로채는 역할을 한다. 보안을 위해서 컨테이너 내부의 어플리케이션은 Sentry 가 정의해 놓은 시스템 콜 만을 사용할 수 있고, 컨테이너에서 호출한 시스템 콜을 Sentry 가 가로채 대신 수행한다. Gofer 는 gVisor 내에서 파일시스템을 담당하는 프로세스이다. 파일시스템에 관련된 시스템 콜은 Gofer 가 담당하며, 파일 입출력의 결과를 9P 프로토콜을 통해서 Sentry 와 통신한다.

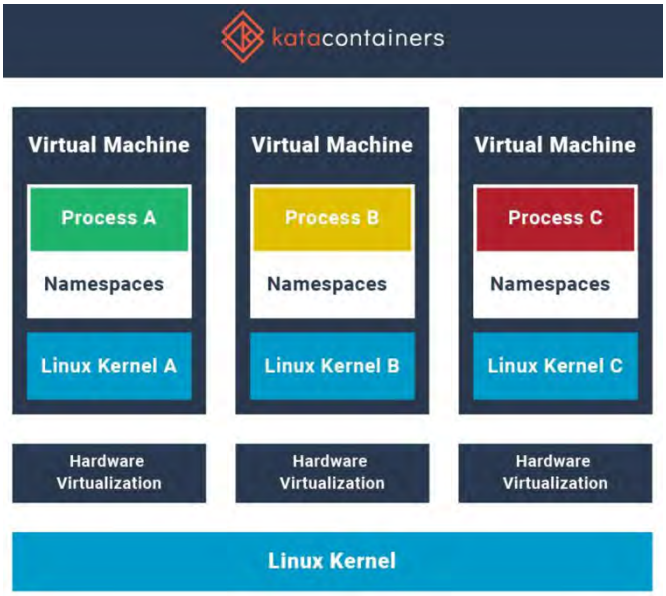
### 2.3 Kata Containers

Kata Containers 는 하드웨어 가상화 기술을 사용해서 가상머신과 컨테이너의 이점을 결합한 컨테이너 런타임이다. Kata Containers 는 경량화된 가상머신을 사용해 컨테이너를 격리시키고 컨테이너가 각각의 리눅스 커널 위에서 동작하도록 한다[7].

Kata Containers 는 QEMU, Firecracker 을 포함한 여러 하이퍼바이저를 지원하며 namespaces 와 cgroups 를 사용하는 runC 와 같은 메커니즘을 가진다.




(그림 1) gVisor 아키텍처[8]




(그림 2) Kata Containers 아키텍처[9]

## 3. 실험 및 평가

공유 디렉토리를 만들기 위해 컨테이너 전용 공간인 Volume 을 생성해 컨테이너 내의 디렉토리에 마운트 시켰다. 그 후 Volume 안에 lodash, express, react 등의 모듈을 미리 다운로드 받았다.

도커 컨테이너, gVisor, Kata Containers 기반 Node.js 런타임 컨테이너에서 Volume 내의 모듈들을 불러오는 시간을 측정하였으며, 동시에 실행되는 컨테이너의 개수를 다르게 하여 파일 동시 접근에 대한 성능도 파악하였다.

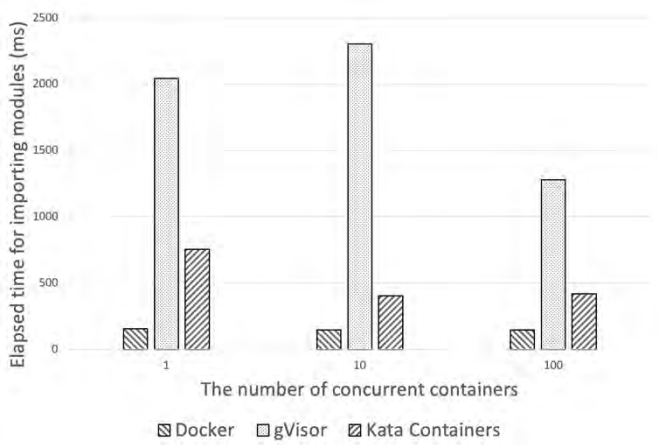
### 3.1 실험 환경

실험은 Ubuntu 기반의 단일 머신에서 진행하였으며, 자세한 소프트웨어 버전 및 하드웨어 사양은 다음과 같다.

| | |
|---|---|
| Host OS | Ubuntu 18.04.5 LTS (Linux 4.15.0) |
| CPU | Intel Core i7-9700 * 8 |
| Memory | 16 GB |
| SSD | 940 GB |
| Docker | Version 19.03.13 |
| gVisor | Release-20200928.0 |
| Kata Containers | Version 1.10.7 |
| node.js | 14.12.0 |

<표 1> 소프트웨어 버전 및 하드웨어 사양

### 3.2 결과 및 분석




(그림 3) 실험 결과

gVisor 와 Kata Containers 환경은 격리 단계를 추가했기 때문에, 파일에 대한 접근이 기존 도커 컨테이너와 비교했을 때 느린 것을 확인할 수 있다. 특히 gVisor 는 도커 컨테이너와 비교했을 때 많게는 약 16 배 차이를 보였다. 또한 gVisor 는 Kata Containers 와 비교했을 때에도 많게는 약 5.7 배에서 적게는 2.7 배 정도 차이가 나는 모습을 볼 수 있다.

gVisor 와 Kata Containers 모두 안전한 서버리스 컴퓨팅을 위한 환경이라는 점에서, 무거운 라이브러리를 사용해야하거나 즉각적인 응답이 필요한 경우에는 Kata Containers 가 이점을 보일 것으로 생각한다.

## 4. 결론 및 향후 계획

서버리스 컴퓨팅의 사용이 늘어나면서 보안에 대한 중요성도 증가하고 있다. 컨테이너 보안을 위한 연구가 진행중이며 AWS 나 GCP 와 같은 클라우드 서비스 제공자들은 이러한 연구 결과를 자신들의 서버리스 컴퓨팅 솔루션에 적용시키고 있다[5].

본 논문에서는 콜드 스타트를 완화하는 방법인 공유 라이브러리 사용의 성능을 파악하기 위해 도커 컨테이너, gVisor 와 Kata Containers 의 모듈을 읽어오는 시간을 비교하였다. 추가적인 격리 수준을 갖고 있는 gVisor 와 Kata Containers 는 기존 도커 컨테이너보다 모듈을 읽어오는데 추가적인 시간이 발생했으며, gVisor 는 Kata Containers 보다 최대 5.7 배의 시간 차이를 보였다.

서버리스 컴퓨팅에서는 보안과 콜드 스타트를 위해 여러 관점에서의 연구와 분석이 필요하다. 향후에는 영상 분석과 같은 특정 서비스를 분류하고 각각의 서비스에 적합한 서버리스 컴퓨팅을 위한 아키텍처를 고안하고 성능 최적화를 위한 연구를 할 예정이다.

## 감사의 글

본 연구는 과학기술정보통신부 및 정보통신기획 평가원의 대학 ICT 연구 센터 지원 사업의 연구결과로 수행되었음. (IITP-2018-0-01405)

## 참고문헌

[1] CVE-2019-5736
[2] Ethan G. Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, “The True Cost of Containing: A gVisor Case Study”, 11th USENIX Workshop on Host Topics in Cloud Computing (HotCloud 19), Renton, WA, USA, 2019
[3] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, “SOCK: Rapid Task Provisioning with Serverless-Optimized Containers”, 2018 USENIX Annual Technical Conference (USENIX ATC 18), Boston, MA, USA, 2018, pages 57-69
[4] Ajay Nair, "AWS re:Invent 2017: Become a Serverless Black Belt: Optimizing Your Serverless Applications"
[5] Docker Documentation, https://docs.docker.com/storage
[6] gVisor GitHub README.md, https://github.com/google/gVisor/README.md
[7] Kata Containers GitHub README.md, https://github.com/kata-containers/kata-containers
[8] gVisor Documentation, https://gvisor.dev/docs
[9] An overview of the Kata Containers project, https://katacontainers.io/learn