Journal of KIIT. Vol. 23, No. 3, pp. 99-107, Mar. 31, 2025. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2025.23.3.99

무기체계 SW를 위한 컨테이너 가상화 기반 시뮬레이션 아키텍쳐 설계 및 구현

이명호*, 김선진**

Design and Implementation of Container Virtualization-based Simulation Architecture for Weapon System SW

Myoung-Ho Lee*, Sun-Jin Kim**

요 약

본 연구는 무기체계 SW의 시뮬레이션 환경의 효율성과 확장성을 높이기 위해 가상화 기술을 활용하여 아키텍처를 설계하고 구현하였다. 이를 위해 다양한 시뮬레이션 시나리오를 설정하고, 가상화 기술의 장점을 최적화할 수 있는 설계 원칙을 적용하였다. 구현된 아키텍처는 성능 테스트 및 부하 분산 시나리오를 통해 높은 확장성, 유연성, 그리고 자원 효율성을 입증하였다. 특히, 컨테이너 기반 가상화 기술을 적용하여 시뮬레이션 노드의 동적 생성과 자동 확장이 가능해졌으며, 이를 통해 운영 비용 절감과 실행 속도 향상이 동시에 이루어졌다. 본 연구는 시뮬레이션 기반 연구와 시스템 개발 과정에서 가상화 기술을 활용하는 구체적인 방안을 제시하며, 향후 대규모 시뮬레이션 플랫폼 개발에 기초 자료로 활용될 수 있다.

Abstract

This study designed and implemented an architecture using virtualization technology to increase the efficiency and scalability of the simulation environment of inorganic system SW. To this end, various simulation scenarios were set up and design principles that could optimize the advantages of virtualization technology were applied. The implemented architecture demonstrated high scalability, flexibility, and resource efficiency through performance tests and load distribution scenarios. In particular, by applying container-based virtualization technology, it became possible to dynamically create and automatically expand simulation nodes, which reduced operating costs and improved execution speed at the same time. This study presents a concrete plan to use virtualization technology in the simulation-based research and system development process, and can be used as basic data for large-scale simulation platform development in the future.

Keywords

virtualization, M&S, resource efficiency, docker, container

* 한화시스템 SW팀(C4I)(교신저자)
- ORCID: https://orcid.org/0009-0004-3709-2380
** 한화시스템 SW팀(C4I)
- ORCID: https://orcid.org/0009-0007-1329-262X

· Received: Feb. 13, 2025, Revised: Mar. 10, 2025, Accepted: Mar. 13, 2025
· Corresponding Author: Myoungho-Ho Lee
Dept. of SW(C4I), Hanwa System, 188, Pangyoyeok-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, Korea
Tel.: +82-31-80091-7362, Email: myoungho.lee@hanwha.com

Ⅰ. 서 론

무기체계SW 개발에서는 시뮬레이션을 통해 개발된 장비의 기능을 확인하고, 모의훈련을 지원할 수 있도록 한다. 그러나 기존 시뮬레이션 방식에는 한계점이 존재한다. 각 기능 모델 소프트웨어를 개별 단말 장비에 포팅해야 하기 때문에 SW가 운용가능한 OS와 버전을 맞춰야 하고, 의존성 있는 공통 라이브러리들을 설치해야 하기 때문에 기본 환경 구성에 많은 시간이 소요된다. 또한 한 대의 단말에 하나의 역할을 하는 시뮬레이션 SW을 동작시켜야 함으로 단말 자원의 효율성이 저하되는 문제가 발생한다. 더불어서 개별 단말 장치들이 동작함으로 단말 장치 하나에 문제가 발생할 경우 해당 역할을 담당하는 모의 SW가 정보처리를 못하기 때문에, 전체 시뮬레이션에 영향이 발생한다. 이러한 문제를 해결하기 위해 시뮬레이션 운용환경의 가상화 기술을 적용하였다. 가상화 기술을 활용함으로써 하드웨어 의존성을 제거하고, 자원의 효율성을 극대화하며, 시뮬레이션 환경의 구성 및 운영을 보다 유연하고 확장 가능하게 개선할 수 있도록 한다. 또한, 가상화를 통해 다양한 OS 환경을 단일 시스템에서 지원할 수 있어, SW의 배포 및 관리가 간소화되고 장애 발생 시 신속한 복구가 가능해진다.

본 연구에서는 가상화 기술을 적용하여 기존 방식의 단점을 보완하고, 군 체계 개발을 위한 보다 효과적인 시뮬레이션 아키텍처 설계 및 구현 방안을 제시한다. 이를 통해 시뮬레이션 환경의 안정성을 강화하고, 향후 다양한 무기체계 SW 개발 및 운용에 적용할 수 있는 확장 가능한 구조를 마련하고자 한다.

Ⅱ. 관련 연구

2.1 운영체제 가상화 동향

가상화는 단일 호스트 시스템에서 여러 개의 운영체제를 동시에 실행할 수 있도록 하는 기술이다. 이를 통해 서로 다른 환경(OS 및 애플리케이션)을 독립적으로 운영할 수 있으며, 물리적 서버의 활용도를 극대화하고 비용 절감, 관리 효율성 향상, 확장성 확보 등의 이점을 제공한다. 가상화 기술은 서버 가상화, 데스크톱 가상화, 네트워크 가상화, 스토리지 가상화 등 다양한 형태로 적용되며, 클라우드 컴퓨팅 및 데이터센터 운영의 핵심 요소로 자리 잡고 있다. 가상화가 상용화되기 이전에는 여러 개의 운영체제를 실행하려면 각각 별도의 물리적 컴퓨터를 사용해야 했으며, 이를 구축하는 데 막대한 초기 비용이 소요되었다. 또한, 각 시스템을 개별적으로 유지·관리해야 했기 때문에 하드웨어 비용뿐만 아니라 운영 및 유지보수 비용도 지속적으로 발생하여 비효율적이고 경제성이 떨어지는 문제점이 있었다. 이러한 한계를 극복하기 위해 가상화 기술이 도입되었으며, 이를 통해 단일 호스트에서 다수의 운영체제를 실행할 수 있게 되면서 비용 절감과 관리 효율성이 크게 향상되었다.

서비스 개발 환경과 실제 운영 환경이 다를 경우, 버전 종속성 문제로 인한 예상치 못한 오류가 발생할 수 있다. 이러한 문제를 해결하기 위해 추가적인 디버깅, 테스트, 환경 조정이 필요하며, 이는 개발자와 운영자의 시간적·인적·물적 비용 증가로 이어진다. 특히, 라이브러리나 의존성이 서로 다른 환경에서 동작하지 않는 경우, 애플리케이션의 안정성 저하와 배포 지연 등의 부작용이 발생할 수 있다. 이를 방지하기 위해 컨테이너화(Containerization) 기술이 도입되어, 개발 및 운영 환경의 일관성을 유지하고 배포 및 관리의 효율성을 향상시키는 데 기여하고 있다. 이러한 문제를 해결하기 위해 운영체제 가상화 기술이 도입되었으며, 이를 통해 단일 운영체제 내에서 가상 환경을 생성하여 다른 운영체제를 실행할 수 있게 되었다. 즉, 하나의 물리적 컴퓨터에서 여러 개의 운영체제를 동시에 실행할 수 있어, 별도의 추가 하드웨어 없이도 다양한 환경을 구축하고 운영할 수 있는 유연성을 제공한다. 이를 통해 시스템 자원의 활용도를 극대화하고, 비용 절감 및 관리 효율성 향상 등의 이점을 얻을 수 있다.

가상화 기술은 하이퍼바이저 기반 가상화와 컨테이너 기반 가상화로 구분되며(그림 1), 각각 고유한 장점과 단점을 지니고 있다[1].

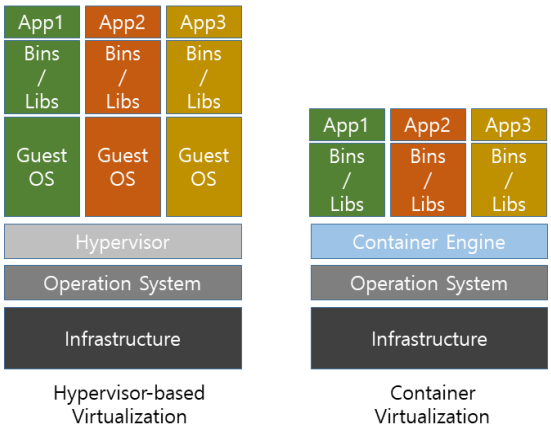

그림 1. 하이퍼바이저 기반 가상화와 컨테이너 기반 가상화

Fig. 1. Hypervisor based virtualization and contaniner virtualization

하이퍼바이저 기반 가상화는 하드웨어 위에 하이퍼바이저를 배치하여 여러 개의 운영체제를 독립적으로 실행할 수 있기 때문에 운영체제 격리성이 우수하고, 다양한 OS를 동시에 운영이 가능한 장점이 있다. 주요 가상화 기술로는 KVM, Xen, VMWare 등이 있으며, 하드웨어 자원을 가상화하는 과정에서 오버헤드가 발생하여 물리시스템 대비 효율성이 떨어지고 실행속도도 상대적으로 느리다는 단점이 있다[2]. 컨테이너 기반 가상화는 단일 운영체제 커널을 공유하면서 각 응용 프로그램을 독립된 환경에서 실행할 수 있도록 한다[3]. 이 방식은 가상 머신보다 경량화되어 실행 속도가 빠르고, 리소스 사용 효율이 높아 물리 시스템에서 직접 응용 프로그램을 실행하는 것과 유사한 성능을 제공할 수 있는 장점이 있다[4]. 주요 기술로는 Docker, Podman등이 있으며, 호스트 커널을 공유하므로 운영체제의 종속성을 가지게 되며 하이퍼바이저 수준만큼의 완전한 하드웨어 격리가 아니기 때문에 호스트 커널 취약점이 발견되면 여러 컨테이너에 영향을 줄 수 있는 단점을 가진다. 하이퍼바이저 가상화와 컨테이너 가상화의 차이점은 표 1과 같다.

그림 2를 보면 동일한 운용 환경에서 가상화를 이용하여 Tomcat 인스턴스를 생성 했을 경우, 컨테이너 가상화 환경에서 생성한 Tomcat 인스턴스(40개)가 하이이퍼바이저 가상화 환경에서 생성한 Tomcat 인스턴스(16개)에 비해 2배 이상 많다[5].

따라서 성능과 자원 효율성 측면에서 비교해보면, 가상 머신을 사용하는 하이퍼바이저 기반 가상화보다 컨테이너 기반 가상화가 소프트웨어 운영 환경을 보다 효과적으로 구성할 수 있다. 또한 컨테이너는 이미지 기반으로 빠르게 생성·종료할 수 있어, CI/CD 파이프라인이나 마이크로서비스 아키텍처(MSA) 환경에서 배포와 확장이 용이하다는 장점도 갖추고 있다. 이러한 특성 덕분에, 최근 많은 조직과 기업에서는 개발 생산성 향상과 운영 효율 극대화를 위해 컨테이너 기반 가상화 기술을 적극 도입하고 있다[6].

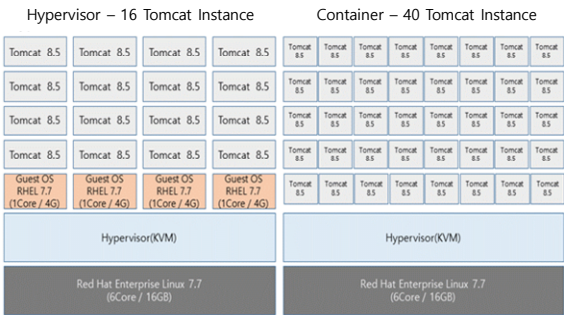

그림 2. 하이퍼바이저 / 컨테이너 집적도 비교

Fig. 2. Hypervisor/container integration comparison

표 1. 하이퍼바이저 가상화와 컨테이너 가상화 비교

Table 1. Hypervisor virtualization versus container virtualization

Category	Hypervisor	Container
Startup time	A few minutes	A few seconds
Image size	From a few GB to several hundred GB	Up to several hundred GB
Guest OS	Any OS can be selected	Same OS as the host
Portability	In most cases, requires converting the virtual image	Use the container image as is
Data management	Stored inside the VM or on attached storage	Data inside the container disappears upon container termination; external storage can be used if persistent data is needed
Relationship with guest OS	The Guest OS recognizes the environment as hardware (virtual hardware)	The host OS is virtualized at the kernel level. In modern usage, it shares resources with the host as needed

2.2 Windows Server 컨테이너 가상화

마이크로소프트(Microsoft)는 Windows Server 2016부터 본격적으로 윈도우 컨테이너(Windows containers) 기능을 도입해왔다. 그 이전까지 리눅스 OS 중심의 컨테이너 기술은 윈도우 환경에서 활용이 제한적이었으나, 윈도우 커널 수준에서 컨테이너를 지원하게 되면서 리눅스 컨테이너와 유사한 운영이 가능해졌다. 이를 통해 기존 .NET Framework 애플리케이션을 비롯하여 윈도우 기반 소프트웨어를 컨테이너 방식으로 패키징하여 배포·운영할 수 있게 되었다[7].

윈도우 서버에서 지원되는 컨테이너는 크게 Windows Server 컨테이너와 Hyper-V 컨테이너로 구분된다.

- **Windows Server 컨테이너**

리눅스 컨테이너와 유사하게 호스트 OS의 커널을 공유한다. 보다 빠른 생성 속도와 경량화를 제공하지만, 호스트와 커널을 공유하기 때문에 격리 수준이 상대적으로 낮다.

- **Hyper-V 컨테이너**

하이퍼바이저 기반 격리를 적용하여 호스트 커널과 분리된 격리 환경을 제공한다. 완전한 VM 수준은 아니지만, OS 격리가 강화되어 보안성은 높고, 대신 다소 무겁고 실행 속도가 느려질 수 있다.

Windows Server 환경에서 Docker를 설치하면, Windows 커널 내부적으로 컨테이너 기능을 제공하는 레이어가 활성화된다. 사용자는 Docker 엔진을 통해 컨테이너 이미지를 생성·배포·실행할 수 있으며, 선택적으로 Windows Server 컨테이너 또는 Hyper-V 컨테이너를 사용할 수 있다. Windows Server 컨테이너를 활용하면 다음과 같은 이점이 생긴다.

- **.Net Framework 애플리케이션 컨테이너화**

기존에 윈도우 환경에서 운영되던 .NET Framework 기반 애플리케이션을 컨테이너로 전환하여, 배포·스케일링 과정을 자동화할 수 있다.

- **혼합 클러스터 구성**

리눅스와 윈도우 서버가 혼합된 이기종 환경에서도 Docker를 통한 컨테이너 배포가 가능하다. 이는 기업 내 기존 윈도우 애플리케이션과 새로운 리눅스 기반 서비스 간의 통합 운영을 용이하게 해준다.

리눅스에 비해 상대적으로 늦게 도입되었으나, Windows Server 컨테이너 및 Docker 기술이 성숙해지면서 기존 윈도우 기반 애플리케이션도 컨테이너화가 활발히 진행되고 있다. 이를 통해 레거시 .NET 애플리케이션부터 최신 마이크로서비스까지, 윈도우 환경에서의 소프트웨어 개발·배포·운영 생태계가 크게 개선되고 있다.

윈도우 서버에서 Docker를 활용할 때는 보안·업데이트·오케스트레이션 전략을 사전에 수립하고, 애플리케이션 특성에 맞는 컨테이너 방식(Windows Server 컨테이너 또는 Hyper-V 컨테이너)을 선택해야 한다. 향후 마이크로소프트의 지원 확대와 컨테이너 기술 표준화가 더욱 가속화됨에 따라, 윈도우 서버 환경에서도 컨테이너 활용이 보편화될 것으로 전망된다.

III. 컨테이너 기반의 시뮬레이션 실행환경 구축

3.1 실험환경 및 실험 시나리오

실험을 위해 윈도우 서버(2019-ver1809)에 Docker 20.10.21 버전을 설치했다.[8] Dockerfile을 기반으로 이미지를 만들고 해당 이미지를 컨테이너로 만들어 실행한 뒤, 작전 서버의 기능 점검 정상 여부 및 Host Server의 자원 사용률 등을 확인할 것이다. 보유하고 있는 시뮬레이션 SW들이 윈도우 환경의 Visual C++로 구현한 SW이기 때문에 윈도우 컨테이너가 동작할 수 있도록 Host Server는 윈도우 서버 2019를 설치하였고, 컨테이너 생성시 사용한 베이스 이미지도 윈도우 서버를 사용하였다. 테스트에 사용한 Host Server의 제원은 표 2와 같다.

표 2. 가상화 호스트 장비 제원
Table 2. Virtualization host equipment specifications

Type	Specification
CPU	Intel Xeon Gold 6244 (8Core, 3.6Ghz) * 2
RAM	64G
Storage	800G
Network card	1000Base-T

컨테이너 기반의 시뮬레이션 실행환경 구축을 위해 그림 3과 같이 시스템 환경을 설정하였다. 호스트 장비는 윈도우 Server 2019 OS를 기반으로 Docker를 이용하여 컨테이너 기반의 가상환경을 구현하였고, 각 컨테이너들의 이미지는 윈도우 Server Docker 이미지를 베이스로 하여 생성했다.

시뮬레이션 SW는 운용통제기(OCS), 탄도탄 표적 모의기(BMTS), EWR 모의기(EWR), LSAM 모의기(LSAM), MSAM 모의기(MSAM), 패트리어트 모의기(PAT), KDX 모의기(KDX) 이렇게 7종의 SW 모음으로 구성되어 있고, 각 모의기별 각자의 역할에 맞는 기능처리를 수행한다. 운용통제기는 시나리오 작성 및 모의통제, 모의상황 전시를 수행하고, 탄도탄 표적 모의기는 탄도탄 항적을 생성하며 EWR 모의기는 탄도탄 항적을 탐지하여 탐지결과를 작전 Server에 송신한다. LSAM/MSAM/패트리어트/KDX 모의기는 탄도탄 탐지결과 전파 및 탄도탄 요격 임무를 수행한다. 모의기들간에는 DDS 프로토콜 환경에서 모의처리 결과 메시지들을 송수신 했고, 모의기와 작전 Server간에는 TCP/UDP 프로토콜 환경에서 각 모의기별 서버간 정의된 메시지 형식대로 메시지를 송수신 했다. 기존에 운용했던대로 시뮬레이션 환경을 만들었다면 각 모의기별로 단말장비를 구성하여 운용했을 것이다. 호스트 서버는 윈도우 서버이며 각 모의기들은 Docker를 이용하여 컨테이너 가상화를 수행했다.

각 모의기들의 컨테이너 이미지는 다음과 같이 생성하였다. Dockerfile이라는 이미지 빌드용 DSL(Domain Specific Language) 파일을 사용하여 그림 4와 같이 윈도우 서버 베이스 이미지에 Visual C++ 라이브러리를 설치하고, 각 모의기 종류별 SW를 추가하여 이미지를 생성하였다.

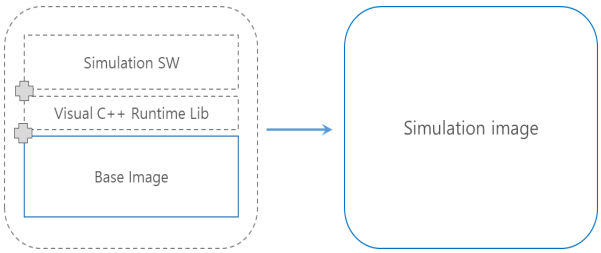

그림 4. 시뮬레이션 SW 이미지 파일 생성
Fig. 4. Create simulation SW image file

베이스 이미지는 abrarov/windows-dev:2.16.0이며 [9], visual C++를 실행하기 위한 라이브러리는 깃허브(Microsoft/vswhere)에서 받아 추가 할 수 있도록 했다.[10] 시뮬레이션 SW 이미지 파일 생성을 위한 Dockerfile은 표 3과 같다.

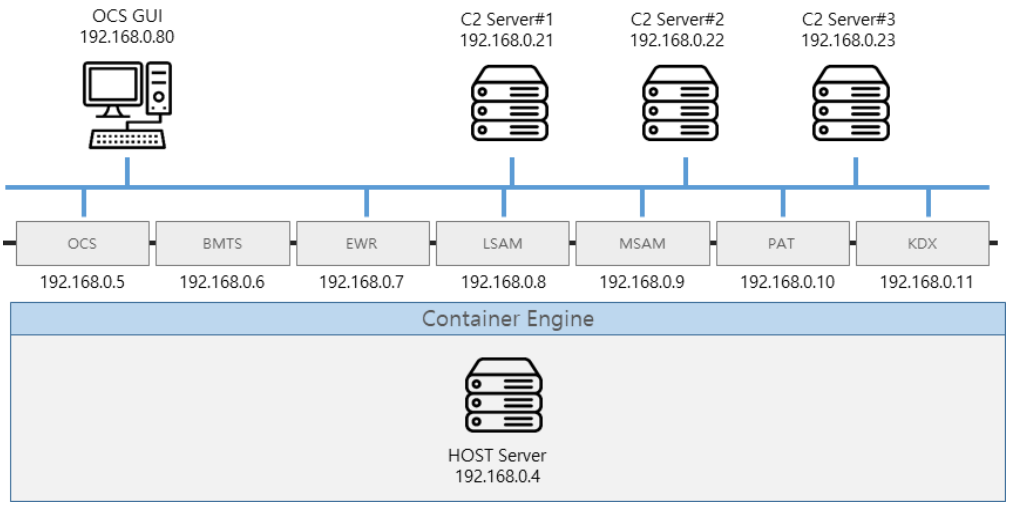

그림 3. 컨테이너 기반의 시뮬레이션 실행환경
Fig. 3. Container-based simulation execution environment

표 3. Dockerfile (시뮬레이션 SW 컨테이너 이미지)
Table 3. Dockerfile (Simulation SW container image)

```
FROM abrarov/windows-dev:2.16.0
ENV MSVS_VERSION="16" \
    MSVS_URL="https://aka.ms/vs" \
    MSVS_DIST_NAME="vs_community.exe" \

VSWHERE_URL="https://github.com/Microsoft/vswhere/releases/download" \
    VSWHERE_VERSION="3.1.7" \
    VSWHERE_DIST_NAME="vswhere.exe"
# Place the current sim folder in c:\sim

ADD ["sim", "C:/sim/"]
ADD ["app", "C:/app/"]
RUN powershell -ExecutionPolicy Bypass -File "C:\app\install.ps1" && \
    powershell "Remove-Item -Path 'C:\app' -Recurse -Force"
ARG image_version=""
ARG image_revision=""
LABEL name="abrarov/msvc-2019" \
    version="${image_version}" \
    revision="${image_revision}" \
    description="Microsoft Visual C++ as part of Microsoft Visual Studio 2019 Community"
```

각 모의기 SW들은 모의기 컨테이너 상에서 C:\sim 폴더의 서브 폴더로 존재할 수 있도록 하였고, 환경 설정파일은 상황에 따라 변경할 수 있도록 호스트에 있는 파일을 불러 올 수 있도록 하였다.

컨테이너는 데스크톱 환경 즉 GUI SW가 동작하지 않기 때문에 각 모의기들은 그림 5와 같이 제어부와 전시부로 나누었고, 각 전시부들은 외부 단말에 설치하여 제어부와 UDP 통신을 통해 운용자와의 인터페이스를 제공할 수 있도록 하였다.

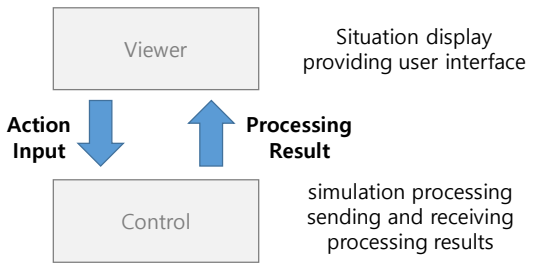

그림 5. 시뮬레이션 SW 전시부/제어부 관계
Fig. 5. Simulation SW exhibition/control department relationship

각 컨테이너마다 표 4와 같이 시뮬레이션 SW 동작하여 시나리오를 기반으로 모의 처리를 진행하여 결과를 다른 시뮬레이션 SW 및 작전 Server, 전시부 SW에 송신한다.

표 4. 시뮬레이션 SW별 입출력 메시지
Table 4. Input/output messages by simulation SW

Simulation SW	INPUT/OUTPUT Message
OCS	In: BM / IM postion
	Out: SimControl / Scenario
BMTS	In: Scenario
	Out: Current position of each BM
EWR	In: BM position
	Out: Bm search Information
LSAM	In: BM postion
	Out: Bm search Information/ Engagement Result
MSAM	In: BM postion
	Out: Bm search Information/ Engagement Result
PAT	In: BM postion
	Out: Bm search Information/ Engagement Result
KDX	In: BM postion
	Out: Bm search Information/ Engagement Result

3.2 실험내용 및 실험결과

컨테이너로 시뮬레이션 SW 운용환경을 제공하여 기존의 단말별로 관리해야 하는 운용환경이 하나의 호스트 서버만 관리하도록 간추려 졌다. 더불어서 호스트만 부팅이 되면 각 컨테이너들은 부팅이 필요없기 때문에 시뮬레이션 환경을 위한 실행 준비 시간이 줄어 들었다.

모의 시나리오별로 생성되는 개체 수가 다르기 때문에 HOST Server의 자원소모량이 다르게 나왔지만 평균적으로 CPU 자원 점유율(18%~22%), 메모리 자원 점유율(30%~35%) 안팎으로 나타났다. 가용자원이 있기 때문에 Docker Swarm을 활용하여 멀티 테스트 환경을 구축할 수 있다. Docker Swarm은 노드들의 집합에 서비스형태로 Docker 컨테이너를 배포하고 관리할 수 있게 하는 기술이다. 즉 클러스터용 컨테이너 서비스 관리 도구를 말한다. Docker Swarm은 그림 6과 같은 구조로 되어 있다[11]. 스웜 매니저를 통해 각 컨테이너들을 그룹단위로 만들어서 서비스를 할 수 있다.

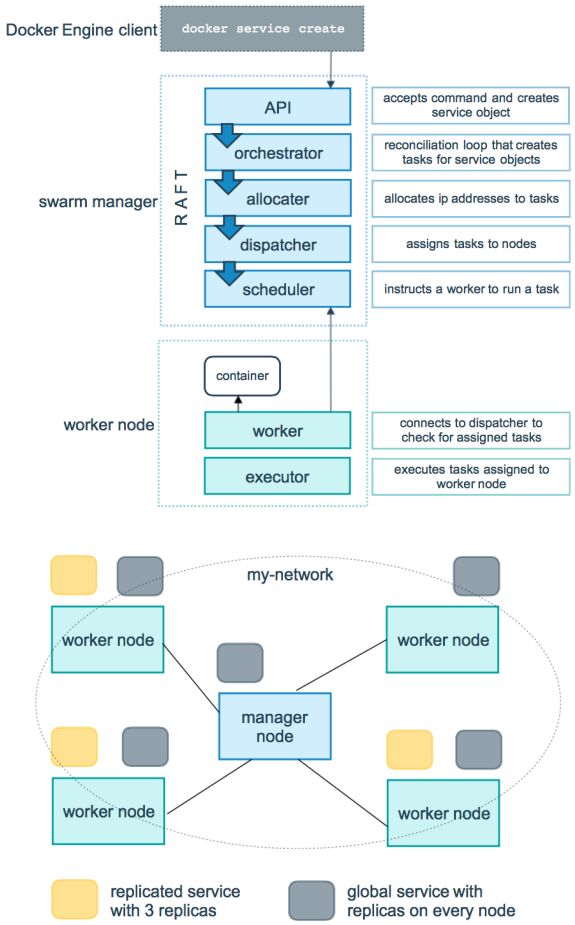

그림 6. Docker Swarm 구조
Fig. 6. Docker swarm structure

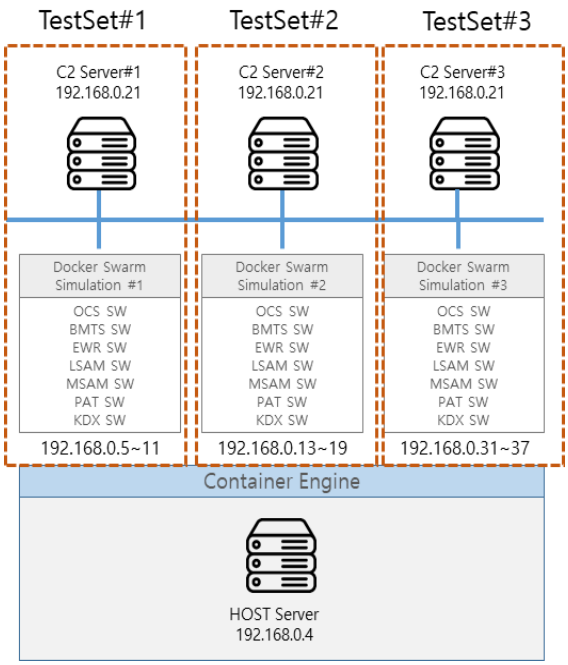

그림 7. Docker Swarm을 활용한 멀티 테스트 환경
Fig. 7. Multi-testing environment with Docker swarm

Docker Swarm을 이용하여 그림 7과 같이 시뮬레이션 SW 컨테이너들을 그룹핑 하여 생성 후 같은 환경을 2세트 더 생성해 봤다. 각 서버 장비별로 각각 다른 상황모의를 통해 기능 점검이 수행되었고, 각 시뮬레이션 Set 간 독립적으로 모의처리를 수행하기 때문에 서버들의 기능 점검 간 간섭은 없었다.

Docker Swarm을 활용한 테스트 환경을 구축한 결과 Host Server의 자원 점유율을 확인해 보니 표 5와 같이 나왔다. TestSet의 개수를 늘리면 자원 점유율은 올라가지 비례하여 올라가지는 않는다. 3개를 동시에 실행해도 여유 자원이 있는 것이 확인된다.

표 5. TestSet 별 자원 점유율
Table 5. Resource share by TestSet

Number of TestSet	Resource share
1	CPU : 20% ~ 24% MEMORY : 31% ~ 33%
2	CPU : 32% ~ 40% MEMORY : 50% ~ 57%
3	CPU : 48% ~ 60% MEMORY : 65% ~ 77%

Docker Swarm을 활용하여 HOST Server의 여유분의 자원을 사용하여 하나의 호스트 서버에서 제공 가능한 모의 환경을 확장할 수 있었다.

Ⅳ. 결론 및 향후 과제

본 연구는 기존 무기체계 소프트웨어(SW) 지원용 시뮬레이션 SW를 가상화하여, 단말장비 형태로 구성되던 시뮬레이션 환경을 한 대의 서버에서 운영할 수 있도록 개선함으로써 자원 활용 효율을 높이고 유지보수 부담을 낮추는 방안을 제시하였다. 또한 Docker Swarm을 활용해 시뮬레이션 SW를 클러스터링함으로써 더 많은 테스트 환경을 지원할 수 있었다.

특히, 본 연구에서 적용한 컨테이너 기반 가상화 기술은 기존 하이퍼바이저 기반 가상화에 비해 경량화된 구조와 자원 활용 효율성의 장점을 보였다. 시뮬레이션 SW의 클러스터링을 Docker Swarm을 통해 수행함으로써, 단일 서버 내에서 여러 개의 시뮬

레이션 환경을 운영할 수 있었으며, 이를 통해 단말 장비의 물리적 한계를 극복하고 더 많은 테스트 환경을 지원할 수 있었다. 또한, 하드웨어 가속 및 최적화 기술과 결합하여 시뮬레이션 수행 속도를 향상시킬 가능성을 확인하였으며, 이는 향후 클라우드 환경과 연계하여 보다 확장된 시뮬레이션 환경을 구축하는 데 기여할 수 있을 것으로 기대된다.[12] 따라서 컨테이너 기반 가상화는 무기체계 시뮬레이션 환경의 효율성을 극대화하는 핵심 기술로 자리잡을 가능성이 높으며, 이를 활용한 다양한 확장 방안을 추가적으로 연구할 필요가 있다.

향후에는 컨테이너 병렬 확장 클러스터링을 적용하여 여러 대의 시뮬레이션 호스트 서버를 하나의 가상화 호스트 서버처럼 운용하고, 무기체계 SW 역시 컨테이너 가상화를 통해 자원 활용도와 SW 생존성을 높이는 방안을 모색할 계획이다.

References

[1] Y. Heo, C. Kim, and C. Cho, "Automation of Resource Release and Error Recovery Method for Improving Stability of Container Management System", Proceedings of KIIT Conference, Jeju Korea, pp. 1324-1326, Sep. 2024.

[2] J.-Y. Hwang and H.-Y. Ryu, "Performance Comparison and Forecast Analysis between KVM and Docker", Journal of KIIT, pp. 127-136, Sep. 2015. https://doi.org/10.14801/jkiit.2015.13.11.127.

[3] V. G. Silva, M. Kirikova, and G. Alksnis, "Containers for Virtualization: An Overview", Applied Computer Systems, Vol. 23, No. 1, pp. 21-27, May 2018. https://doi.org/10.2478/acss-2018-0003.

[4] V. M. Antonova, , M. A. Egorov, V. P. Blinov, E. E. Malikova, and A.Y. Malikov, "Studying the Principles of Infocommunication Network Virtualisation Using the Docker Platform", 2024 Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russian Federation, pp. 1-5, Oct. 2022. https://doi.org/10.1109/IEEECONF60226.2024.10496739.

[5] Openmaru, https://www.openmaru.io/. [accessed: Feb. 15, 2025]

[6] R. Queiroz, T. Cruz, J. Mendes, P. Sousa, and P. Simões, "Container-based Virtualization for Real-Time Industrial Systems—A Systematic Review", ACM Computing Surveys, Vol. 56, No. 3, pp. 1-38, Oct. 2023. https://doi.org/10.1145/3617591.

[7] R. Lazauskas, "Nano Server and Containers in Windows Server 2016", South-Eastern Finland University of Applied Sciences, pp. 1-45, Apr. 2018.

[8] Docker, https://www.docker.com/. [accessed: Jan. 18, 2025]

[9] Docker Hub, https://hub.docker.com/. [accessed: Jan. 18, 2025]

[10] Git Hub - microsoft / vswhere, https://github.com/Microsoft/vswhere/releases/. [accessed: Jan. 18, 2025]

[11] M. Ilean, M. I. Oproiu, and C. V. Marian, "Using Docker Swarm to Improve Performance in Distributed Web Systems", 2024 International Conference on Development and Application Systems (DAS), Suceava, Romania, pp. 1-6, May 2024. https://doi.org/10.1109/DAS61944.2024.10541234.

[12] S.-Y. Bae, "A Comparative Analysis of Domestic and Foreign Docker Container-Based Research Trends", The Journal of the Korea Contents Association, Vol. 22, No. 10, pp. 742-753, Oct. 2022. https://doi.org/10.5392/JKCA.2022.22.10.742.

저자소개

이 명 호 (Myoung-Ho Lee)

2009년 2월 : 서강대학교
정보처리과(공학석사)
2019년 12월 ~ 현재 :
(주)한화시스템 연구원
관심분야 : 가상화, AI, .M&S

김 선 진 (Sun-Jin Kim)

2011년 2월 : 조선대학교
컴퓨터공학과(공학사)
2022년 5월 ~ 현재 :
(주)한화시스템 연구원
관심분야 : 가상화, AI, .M&S