컨테이너 기반의 가상화 기술 활용을 통한 임베디드 시스템 개발

권혁선, 최호성, 유현민, 홍인기

경희대학교

gurtjs0116@khu.ac.kr, to6044@khu.ac.kr, yhm1620@khu.ac.kr, ekhong@khu.ac.kr

Embedded System Development Leveraging Container-based Virtualization Technology

Hyuk-Sun Kwon, Ho-Seong Choi, Hyun-Min Yoo, Een-Kee Hong

요 약

최근 industry 4.0의 스마트팩토리와 같은 생산 환경에서 임베디드 시스템과 Internet Of Things(IoT) 장치 활용을 위한 가상화 기술 연구가 활발히 진행되고 있다. 동시에 기존 생산자 위주에서 고객 요구 맞춤형 유형 생산으로 스마트팩토리의 패러다임이 변화하면서 가상화 기술이 두각을 나타내고 있다. 하지만 초기에 사용되던 가상머신(virtual machine, VM) 가상화 기술은 시스템의 구조적 한계가 인해 적합하지 못했고, 대신 리눅스 커널의 자원 격리 기술 기반인 컨테이너 기술이 떠오르게 되었다. 본 논문에서는 스마트팩토리의 초기 개발 또는 실험적인 목적을 위해 컨테이너 가상화 기술인 docker를 이용하였다. 그리고 docker를 이용해 임베디드 시스템을 구축하고 애플리케이션을 구동하는 과정을 소개함으로써 향후 스마트팩토리 시스템의 확장성과 유연성을 위한 프로토타입을 제시한다.

Ⅰ. 서 론

현시점, 산업 분야는 기존의 정보통신기술을 기반으로 한 3차 산업혁명을 뛰어넘어 4차 산업혁명을 겪고 있다. 제조 및 생산 환경에서의 혁신은 industry 4.0으로 연결되며, 스마트팩토리는 이를 중심으로 중요한 부분을 차지하고 있다. 그리고 다양한 임베디드 시스템과 Internet Of Things(IoT) 장치의 활용이 증가하면서 기존의 하드웨어 종속적인 시스템으로는 스마트팩토리 생산 환경이 요구하는 기술들을 도입하는 것이 제한적이었다. 이러한 변화의 중심에서 빠르게 발전하는 가상화 기술과 5G 네트워크 기술의 혁신적인 활용이 industry 4.0의 핵심으로 자리 잡았다 [1].

초기의 가상화 기술 중 하나인 가상머신(virtual machine, VM)은 다양한 운영체제와 애플리케이션을 실행하는 환경을 제공하는 데 주력해왔다. 그러나 VM은 hypervisor를 기반으로 별도의 게스트 운영체제를 구축해야 하며, 별도로 하드웨어 자원을 배분해야 하므로 불필요한 성능 저하가 나타난다 [2]. 또한 VM은 독립된 운영체제를 가지고 있어 운영체제를 시작 및 중지하는 데 많은 시간이 소요되어 애플리케이션을 배포하고 확장하는 데 불편함이 있다.

이러한 VM 가상화 기술의 한계로부터 비롯된 요구에 따라, 리눅스 커널의 자원 격리 기술을 활용한 컨테이너 기술이 주목받게 되었다. 초창기 컨테이너 기반의 가상화 기술은 구현상의 제약이 있었지만, 컨테이너 기술을 개발한 'dotCloud'라는 회사가 docker로 이름을 변경하며 오픈소스로 공개함으로써 컨테이너 기술이 널리 사용되게 되었다.

컨테이너 가상화 기술은 별도의 게스트 운영체제 없이 바이너리 파일, 라이브러리 등 애플리케이션을 구성하는 데 필요한 요소만을 포함하여 패키지화한다. 이 기술은 자원을 격리하고 동적으로 관리하여 스마트팩토리와 산업 환경에서 널리 사용되고 있다. 특히, 컨테이너 기술을 활용해 엣지 디바이스에서 웹 기반 서비스를 배포하거나 여러 대의 동일 기종 IoT를 사용해 시스템을 구축하는 연구가 진행되었다 [3],[4]. 본 논문은 기존의 동일 기종뿐만 아니라 다양한 하드웨어 플랫폼에서도 호환할 수 있는 컨테이너 개발과정과 이로 인한 확장성과 이식성에 대해 소개한다.

Ⅱ. 본론

먼저 컨테이너 기반의 임베디드 시스템을 개발하기 위한 하드웨어로 라즈베리파이 4, 오렌지파이 제로 그리고 오렌지파이 제로3을 사용하였다. 각 임베디드 시스템들은 armv7 및 arm64의 다양한 하드웨어 플랫폼으로 구성되어 있으며 각기 다른 General-Purpose Input/Output(GPIO) pinout을 가지고 있다. 컨테이너 기반의 임베디드 시스템에서 구현한 것은 GPIO pin 제어와 MQTT 통신 프로토콜을 활용한 애플리케이션이다. 각 IoT (라즈베리파이 4, 오렌지파이 제로) 장치는 온/습도 센서(DHT-11/22)로 읽은 데이터를 중앙 서버인 MQTT broker(오렌지파이 제로3)로 전송하고, broker는 IoT 장치들로부터 받은 데이터를 데이터베이스로 관리한다. 본 논문의 목표는 factory operator의 요구에 맞는 단일의 dockerfile를 작성하고 docker 이미지를 docker 레지스트리에 저장해 다양한 임베디드 시스템 환경에서 컨테이너 형태로 애플리케이션을 구동하는 것이다. Fig. 1은 dockerfile 스크립트부터 각 IoT 기기에서 애플리케이션을 컨테이너로 구동하기까지의 전 과정을 나타낸다.

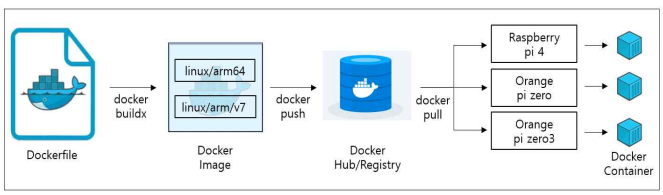

Fig. 1 Docker 컨테이너 기반의 임베디드 시스템 구조

Ⅱ-1. Dockerfile 및 Docker Image

먼저 docker 이미지를 생성하기 위한 dockerfile은 python 3.8 이미지를 base 이미지로 사용하였고, 애플리케이션에 필요한 python 스크립트 및 각종 패키지와 라이브러리들을 레이어별로 작성하였다 (Fig. 2). 그리고 작성된 dockerfile에 buildx 명령어를 적용해 docker 이미지를 생성하였다. Buildx는 docker에서 제공하는 buildkit을 기반으로 하며 platform 옵션을 통해 다양한 하드웨어 플랫폼에 맞게 단일의 docker 이미지를 생성할 수 있다. 본 논문에서는 armv7, arm64 하드웨어 플랫폼에 맞게 생성된

이미지를 docker hub 레지스트리에 저장하였다. 결과적으로 IoT 장치들은 docker hub에 저장된 docker 이미지를 버전 및 하드웨어 플랫폼에 맞게 가져와 컨테이너를 구동할 수 있게 된다.

```
FROM python:3.8-slim

# install necessary dependencies
RUN apt-get update && apt-get install -y \
python3-setuptools \
python3-dev \
gcc \
vim \
net-tools \
sqlite3

RUN pip3 install Adafruit_DHT paho-mqtt

WORKDIR /app

COPY . .

WORKDIR /app/orangepi0/orangepi_zero_gpio

RUN python3 setup.py install

WORKDIR /app/orangepi0

RUN mv -f opi_sensor.py DHT11-DHT22-Python-library-Orange-PI
RUN mv -f opi_mqtt.py DHT11-DHT22-Python-library-Orange-PI

EXPOSE 1883

WORKDIR /app
```

Fig. 2 Dockerfile 스크립트

Ⅱ-2. Docker Container

컨테이너는 docker 이미지에 docker run 명령어를 통해 생성되는데 포트 번호, 환경변수, 스토리지 등 다양한 옵션들을 지정할 수 있다. 본 논문에서는 각 IoT 장치에서 컨테이너를 실행하면서 환경변수 옵션 설정하였다. 예를 들어 로컬 네트워크(local area network, LAN)에서의 MQTT 통신을 위해서는 MQTT broker의 ip 주소를 다른 IoT 장치들이 공유해야 하는데, 이때 환경변수를 통해 쉽게 컨테이너들을 관리할 수 있다. 결과적으로 IoT 장치들은 MQTT broker에 데이터를 전송하고 broker는 데이터를 저장하고 출력한다. Fig. 3은 컨테이너에서 MQTT 통신을 통해 받은 온/습도 데이터들을 broker에서 출력하는 과정을 나타낸다.

```
root@7587493f504c:/app/orangepi03# python3 mqtt_broker.py
Topic=/data/orangepi0, Temp=24.8*C, Humidity=60.8%
Topic=/data/raspberrypi4, Temp=25.0*C, Humidity=66.0%
Topic=/data/orangepi0, Temp=24.8*C, Humidity=60.6%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=66.0%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=66.0%
Topic=/data/orangepi0, Temp=24.8*C, Humidity=60.7%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=66.0%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=66.0%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=67.0%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=66.0%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=66.0%
Topic=/data/orangepi0, Temp=24.8*C, Humidity=60.6%
Topic=/data/raspberrypi4, Temp=24.0*C, Humidity=66.0%
Topic=/data/orangepi0, Temp=24.8*C, Humidity=61.6%
```

Fig. 3 MQTT 통신으로 받은 온/습도 데이터 출력

Ⅱ-3. Docker Volume

이처럼 동일한 docker 호스트에서 생성된 컨테이너들은 서로 독립적으로 격리된 시스템 자원과 네트워크를 사용하는 등 다양한 이점이 있다. 하지만 컨테이너는 오류가 발생하면 종료됨과 동시에 컨테이너 내부 데이터가 삭제되어 중요한 정보를 읽을 수 있는 문제가 발생할 수 있다. 이러한 문제를 해결하기 위해 docker에서는 volume 마운트를 통해 데이터를 보존하고 분리한다 (Fig. 4). 본 논문에서 MQTT broker(오렌지파이 제로3)로 전송된 온/습도 데이터들은 volume 옵션을 통해 로컬에서 데이터베이스 파일로 관리하며 해당 코드는 Python 스크립트로 작성되었다.

Ⅱ-4. Docker Compose

본 논문에서는 각 IoT 장치에서 단일 컨테이너를 실행했지만, 스마트팩토리와 같은 산업현장에서는 여러 개의 컨테이너가 병렬적으로 실행된다. 단일 docker 호스트에서 여러 개의 컨테이너를 연결해 구동해야 하는 시나리오에서는 기존에 설명했던 방식으로도 가능하지만, docker compose를 통해 쉽게 구현할 수도 있다. Docker compose는 yaml 파일을 이용한 다중 컨테이너 실행 방식으로서 IoT 장치에서 구동될 여러 서비스와 옵션들을 조합한 스크립트로 docker compose 명령어를 통해 애플리케이션 스택 전체를 불러온다. 따라서 복잡한 스마트팩토리의 IoT 장치들의 경량화와 효율성을 높이기 위해서는 docker compose를 고려할 수 있다.

```
root@99d94e2cdac9:/app/orangepi03/data# sqlite3 broker.db
sqlite> SELECT * FROM dht;
1|/data/raspberrypi4|25.0|72.0
2|/data/raspberrypi4|23.0|73.0
3|/data/raspberrypi4|23.0|73.0
4|/data/raspberrypi4|23.0|74.0
5|/data/raspberrypi4|23.0|73.0
6|/data/raspberrypi4|23.0|74.0
7|/data/raspberrypi4|23.0|74.0
8|/data/raspberrypi4|23.0|74.0
9|/data/raspberrypi4|23.0|75.0
10|/data/raspberrypi4|23.0|74.0
11|/data/raspberrypi4|23.0|74.0
12|/data/raspberrypi4|23.0|74.0
13|/data/raspberrypi4|23.0|74.0
14|/data/orangepi0|25.7|59.8
15|/data/orangepi0|25.7|59.8
            },
            "Name": "overlay2"
        },
    "Mounts": [
        {
            "Type": "volume",
            "Name": "app-data",
            "Source": "/var/lib/docker/volumes/app-data/_data",
            "Destination": "/app/orangepi03/data",
            "Driver": "local",
            "Mode": "z",
            "RW": true,
            "Propagation": ""
        }
    ],
```

Fig. 4 Docker volume을 활용한 로컬 데이터베이스 마운팅

Ⅲ. 결론

본 논문에서는 온/습도 센서 제어와 MQTT 통신 기반의 애플리케이션을 임베디드 시스템 위에서 컨테이너 형태로 실행하여, 컨테이너 기반의 가상화가 갖는 확장성과 유연성을 확인할 수 있다. 또한 하드웨어 로직이 구현되어 있다는 전제하에 개발자들이 하드웨어 종속성을 벗어나 컨테이너 환경에서 서비스를 개발하고 배포할 수 있도록 하는 방식을 구현하였다. 향후 컨테이너 가상화 기술은 컨테이너 오케스트레이션 기술과 5G 네트워크와 결합함으로써 industry 4.0의 스마트 팩토리 및 다른 분야에서의 중요성을 더욱 높일 것으로 기대된다. 더불어, 엣지 컴퓨팅을 기반으로 한 컨테이너 가상화 기술은 스마트팩토리의 자동화/최적화 및 로드 밸런싱에 대한 연구에 흥미로운 가능성을 제시하며, 5G를 활용한 산업용 임베디드 시스템의 상용화에 기여할 것으로 기대된다 [5].

ACKNOWLEDGMENT

"본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음" (IITP-2021-0-02046*)
“이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 정보통신방송기술국제공동연구사업 지원을 받아 수행된 연구임” (No.RS-2022-00207387, 5G Open RAN 기반 지능형 네트워크 슬라이싱 기술 개발)

참 고 문 헌

[1] 신재승, 김일규, 스마트공장과 5G기반 Industrial IoT, 한국통신학회지(정보와통신), pp. 22-30, Jun 2016.

[2] Vivian Noronha, "Performance Evaluation of Container Based Virtualization on Embedded Microprocessors,"2018 IEEE 30th International Teletraffic Congress (ITC 30), pp. 79-84, Sep. 2018.

[3] Tansangworn, Natapon, "Development of IoT Edge Hub for Wireless Sensor Networks based on Docker Container," 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), Aug. 2020.

[4] Ha, Jihun. "A web-based Service Deployment Method to Edge Devices In Smart Factory Exploiting Docker," ICTC, pp. 708-710, Oct. 2017.

[5] Pankaj Mendki, "Docker container based analytics at IoT edge Video analytics usecase,",2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU), Feb 2018.