# A Web-based Service Deployment Method To Edge Devices In Smart Factory Exploiting Docker

Jihun Ha, Jungyong Kim, Heewon Park, Jaehong Lee, Hyuna Jo, Heejung Kim, and Jaeheon Jang
Software R&D Center, Samsung Electronics Co., Ltd., Seoul, South Korea
Email: {jihun.ha, jyong2.kim, h_w.park, jaehong2.lee, hyuna0213.jo, hjnari.kim, jaeheon.jang}@samsung.com

***Abstract*—Recently, a micro-service architecture has been adopted to an edge device in smart factory due to its maintainability and scalability. In the architecture, all services are generally implemented as Docker containers and deployed by Docker APIs which might be harsh to a factory operator. In this paper, we propose an easy-to-use web-based service deployment method for containerized service to in-factory edge device for supporting DevOps to factory operator.**

***Index Terms*—Service deployment, DevOps, Docker, Micro-service, Smart factory**

## I. Introduction

A trend of Industrial IoT technology has been adopted rapidly to various industrial verticals including factories. Such a revolution is driven by 1) **big data** produced by lots of machines at product lines in factory, e.g. assembling parts of a product, 2) **analytics** for predictive maintenance by detecting an abnormal behavior of machines or manufacturing processes using collected big data, and 3) **automation** according to the analytic decision, e.g. activating a recovery process if necessary. To evolve a factory to be such ”smart”, collecting data from each of machines and sensors spatially deployed in a large area of factory are crucial, and a number of certain devices, called “Edge” devices, located in a middle of machines and a server are employed for data acquisition of a set of machines out of entire one.

Additionally, edge devices could perform a bunch of not only predefined pre-processing but also more complex data analytics utilizing collected data because an edge device is able to detect abnormal behaviors or failures of machines and handle them more quickly due to its closeness to machines. In this sense, a software platform for edge device has been required to be composed of of various services and, naturally, a micro-service architecture has been desired and adopted to the edge device. In micro-service architecture, each service is generally implemented as a Docker container [1] which is one of virtualization techniques which can make services isolated from each other, which provides maintainability and scalability to the entire software platform. One of platforms adopting a micro-service architecture for in-factory edge devices is EdgeX Foundry project [2]. In EdgeX project, all services including rule engine for event-driven notification and protocol adapters for collecting data from various machines using different protocols are implemented as Docker containers.

As all (service) Docker containers are ready to be deployed to an edge device, they could be deployed with Docker's own APIs but they might be harsh to factory operator who is not familiar to software deployment. Thus, more well-abstracted APIs especially for a factory operator should be required to deploy a new container and consistently manage it for further update and termination.

In this paper, we propose a web-based service deployment method to provide better DevOps to a factory operator. Our system provides a set of well-abstracted HTTP APIs for deploying, updating, and terminating a container on an edge device. Besides, we also provides a centralized service deployment method coordinating a number of edge devices, which supports concurrent service deployment for a group of edge devices and time-scheduling service deployment.

The remainder of this paper is organized as follows. Section II describes a deployment model for a containerized service. Section III presents a system architecture for our proposed containers for service development. And a challenge for multi-container architecture we have observed is shared in Section IV and finally, Section V concludes this paper.

## II. Service Deployment Model

This section describes our proposed service deployment model in smart factory as shown in Fig.1. The model consists of two actors: application developer and factory operator. First, a software developer is an expert to implement applications to be deployed to an edge device in factory. Note that, a developer can implement a number of containerized *services* composing an integrated one, called *application*, needed to be deployed together. Key components which a developer produces are as follows:

- **Application codes** should be implemented with one of programming languages, e.g., Java, Go, and Python.
- **“Dockerfile”** [3] should be written. This script file includes all essential configurations for building Docker image of implemented application.
- **“Application description”** file should be written and delivered to factory operator. This description includes all essential information required when a developed application is deployed to an edge device. The description could include a set of service descriptions to be deployed in a same time. Note that a schema of the description is identical to one for Docker Compose function [4] for defining and running multiple Docker containers.

Preparing all components mentioned above, a developer can push them to a code repository, e.g. GitHub, maintained for

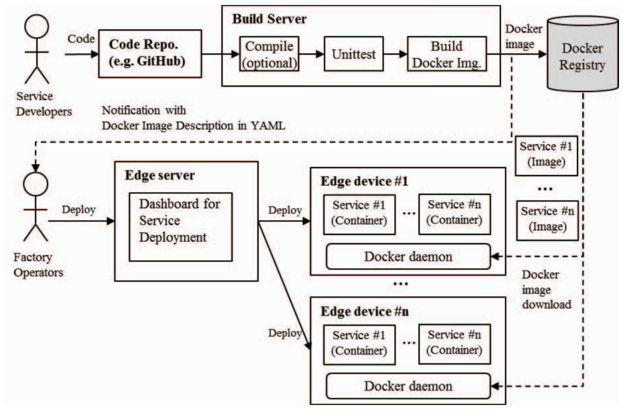

Fig. 1. Service deployment model for smart factory.

softwares for edge devices. Then, implementation codes are compiled at a build server if necessary, packaged to a Docker image containing the compiled codes by exploiting a given Dockerfile. At last, the built image is automatically stored to a Docker registry.

At this time, a factory operator can deploy a new service to desired edge device(s) with a given application description by utilizing the proposed web-based service deployment APIs. The detailed information for those are described in the next section.

## III. System Architecture

This section describes an overall system architecture of service deployment proposed in this paper. We implement two Docker containers for supporting our proposed service deployment method. First of all, as shown in Fig. 2, every edge device has a Service Deployment Agent container, SDA for short, which handles all requests for service deployment transmitted via HTTP, parses the received request, translates it to corresponding Docker operation, and directly communicate with Docker daemon.

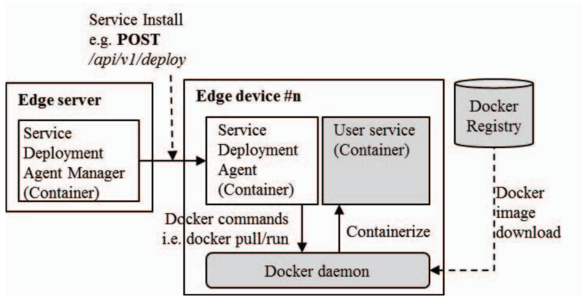

Fig. 2. Overall system architecture

The other shown in Fig. 2, Service Deployment Agent Manager container, SDAM for short, can be shipped on an edge server so as to manage all SDAs running on edge devices to provide a centralized service deployment method. For that, SDAM container can provide HTTP APIs for grouping a subset of managed SDAs and commanding a concurrent service deployment to a group of edge devices.

### *A. Service Deployment Agent in Edge Device*

As briefly mentioned in the above, SDA should be shipped on every edge device to handle proposed web-based service deployment requests. Functional blocks of SDA on an edge device are shown in Fig. 3 and key features of SDA are described as follows:

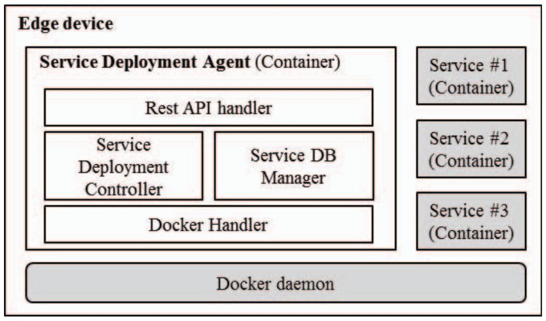

Fig. 3. Function blocks of SDA container in edge device

- **Service deployment for a container or multiple containers** is provided for deploying a container or a set of container concurrently. Supported operations for service deployment is an installation, run, termination, and retrieval of running containers' information. Moreover, for more complex applications, several containers of services are needed to be deployed together to an edge device. Also, an update of all containers has to be performed.
- **Abstracted HTTP APIs** are provided for factory operator who is not familiar to software development and deployment. All deployed containers and deployment operations (e.g., container installation, update, and termination) are mapped to corresponding resource URIs with HTTP methods. Among provided APIs, HTTP APIs with *POST* method provided by SDA container is described in Table. I.

TABLE I
HTTP APIs of SDA with *POST* method

| Method | API | Desc. |
|---|---|---|
| POST | /api/v1/deploy | Install and run app(s). |
| | /api/v1/apps/{app_id}/update | Update app(s). |
| | /api/v1/apps/{app_id}/down | Stop and uninstall app(s). |

### *B. Service Deployment Agent Manager in Edge Server*

In a case where the number of edge devices to collect a data from machines is huge, it would be difficult to manage each of edge device individually. In such an environment, a way for more efficient service deployment to them in a concurrent manner would be require. To aim the goal, we propose a SDAM which can be installed on a edge server, which manages all containers' installation, update, and removal for multiple edge devices concurrently .

SDAM for the centralized service deployment provides the following features:

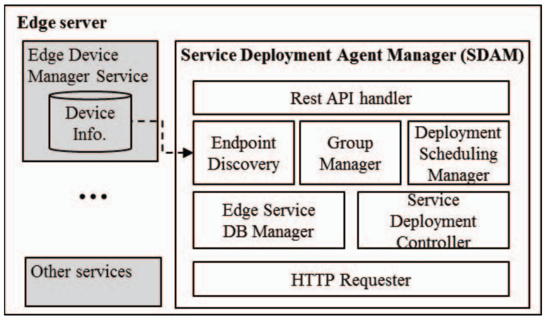
Edge server
Edge Device Manager Service
Device Info.
...
Other services
Service Deployment Agent Manager (SDAM)
Rest API handler
Endpoint Discovery
Group Manager
Deployment Scheduling Manager
Edge Service DB Manager
Service Deployment Controller
HTTP Requester

Fig. 4. Functional blocks of SDAM and other components in edge server

- **Group management** is provided to create/remove a group of edge devices and join/leave a edge device to/from a group.
- **Concurrent service deployment for multiple edge devices** is provided to install, update, and remove a service to/from a group of edge devices in a concurrent manner.
- **Deployment scheduling management** is provided to scheduled service deployment. This feature could be utilized in an update of applications to the latest version periodically(e.g. once a week) or in a desired time (e.g. night time).
- **Abstracted HTTP APIs** are provided for factory operator. All managed edge devices, their created groups, their applications, and deployment operations are mapped to corresponding resource URIs with HTTP methods. Among provided APIs, HTTP APIs with *POST* method provided by SDAM is described in Table. II.

To accomplish the above features, SDAM is implemented with the architecture as shown in Fig. 4. Note that managing all edge devices and their addresses is responsible to other dedicated container like Edge Device Manager Service container as shown in Fig. 4 and SDAM simply acquires the addresses from the container.

TABLE II
HTTP APIs OF SDAM WITH *POST* METHOD

| Method | API | Desc. |
|---|---|---|
| POST | /api/v1/group/create | Create a group with empty. |
| | /api/v1/group/{gid}/join | Join an edge to a group. |
| | /api/v1/group/{gid}/leave | Leave an edge from a group. |
| | /api/v1/group/{gid}/deploy | Install and run app(s) to all edges in a group. |
| | /api/v1/group/{gid}/update | Update app(s) of all edges in a group. |
| | /api/v1/group/{gid}/down | Stop and uninstall app(s) of all edges in a group. |

## IV. Challenge For Multi-Container Architecture

During investigating and testing a multi-container architecture, we have observed a strange degradation of network performance phenomenon when the number of containers reaches to a certain number. To look into this phenomenon more detailedly, we conduct an experiment with iPerf benchmark tool [5] for measuring network bandwidth between containers. The experiment is performed on two different machines: Ubuntu desktop PC and a generic Raspberry PI 3. And there are a number of containers running iPerf client to transmit TCP packets to the server container at maximum rates and a single container running iPerf server to measure a total network bandwidth.

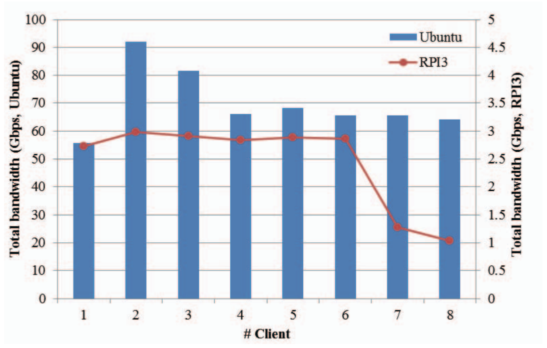

Fig. 5. Comparison of total network bandwidth by varying the number of containers.

The experiment result is shown in Fig. 5. Note that, back-to-back TCP transmission can not saturate a maximum TCP bandwidth in case of 1 client running on Ubuntu, so the result of it shows a low performance like Fig. 5. For other cases, as easily detected, once the number of iPerf client containers exceeds to a certain number(3 and 6 clients for Ubuntu and RPI3 cases, respectively), a network performance is steeply dropped. Since communications between different containers in a same device are generally performed through network interface, such unexpected performance degradation would be a crucial factor to realize the multi-container software platform. We suspect that a bridge network implemented in Linux kernel would cause this problem as known in Docker community, and investigating it to resolve the problem more detailedly as a future work.

## V. Conclusion

In this paper, we propose a web-based service deployment method for edge devices, which helps a factory operator to manage all services on edge devices in easier way. SDA provides a set of web APIs to install, run, update, and terminate a service container to a specific edge device. And SDAM cooperated with multiple SDAs provides centralized functionalities of all service deployment operations per a group of multiple edge devices.

As a future work, we will optimize Docker platform and Linux kernel for removing an overhead of bridge network to realize multi-container software platform especially for an edge device in a smart factory environment.

## References

[1] Docker open-source project. Available: https://www.docker.com/
[2] EdgeX Foundry open-source project. Available: https://www.edgexfoundry.org/.
[3] Dockerfile reference. Available: https://docs.docker.com/engine/reference/builder/.
[4] Docker Compose. Available: https://docs.docker.com/compose/.
[5] iPerf. Available: https://iperf.fr/.