Proceedings of the APAN – Research Workshop 2019
ISBN 978-4-9905448-9-8

Relocatable Service Composition based on Microservice Architecture for Cloud-Native IoT-Cloud Services

Seunghyung Lee, Jungsu Han, Jincheol Kwon and JongWon Kim*

***Abstract*— With the development of the cloud, the Internet of Things (IoT) and artificial intelligence (AI) technologies, the demand for services that utilize these infrastructure technologies is explosively increasing. In recent years, to develop and verify services quickly and efficiently, traditional monolithic service has evolved to a service composition that is based on a container-based microservice architecture (MSA). In particular, IoT-Cloud services that combine the internet of Things and the cloud are appropriate to make the functions based on cloud-native computing because it requires to connect with IoT, Cloud, and AI functions flexibly. In this paper, we design relocatable service composition to develop and compose intelligent IoT-Cloud service in the cloud-native computing environment based on IoT-Cloud pattern. In addition, to verify the feasibility of the proposed approach of service composition, we apply the specific IoT-Cloud service on multi-site playground to verify mobility and usefulness. We show the proposed approach can be proved that cloud-native based service is partially adaptable to the service environment and can be operated flexibly.**

***Index Terms*—Microservices architecture, cloud-native computing, container orchestration, service composition, IoT, Cloud, multi-site.**

I. Introduction

Over the past few years, the Internet of Things (IoT), which has hit the IT industry as well as entire industries, has had a powerful impact from businesses to everyday life. IoT is a technology that connects real-time data from a sensor attached to an object over a network [1]. Most IoTs use the cloud backbone to link data collection, storage, and utilization. The IoT-Cloud service is a service that provides users with various values by flexibly linking IoT devices and the cloud. In other words, it includes "-" which means inter-connect between the IoT area and the cloud area so that corresponding to the IoT part can be flexibly connected with the cloud part.

In this way, realizing data pipeline that analyzes the data collected from the cloud and produces valuable and meaningful information for an intelligent IoT-Cloud service is becoming an important issue for IoT and Cloud service developers. In addition, as the scale of service becomes more complex and larger, we need to consider MSA-based service composition that develops a small unit of service and links them with each other functions [2]. Service composition refers to the overall process for achieving a set of services such as resource preparation, resource allocation, function distribution, and functional coupling [3]. As the service development approaches shifts from a single monolithic architecture to the microservice architecture, the services are divided into a set of functions, so we need to arrange them in a resource set well and connect them. In addition, existing virtual machine-based cloud computing has evolved into cloud-native computing due to the introduction of container technology and the microservice architecture development method [4]. In other words, the service based on the cloud-native architecture that develops services in small function units and performs in a lightweight virtualized environment, container units, and dynamically manages is becoming more powerful [5].

Thus, in this paper, we present a relocatable service composition for IoT-Cloud service based on cloud-native computing. The rest of this paper is organized as follows. In Section 2, we present IoT-Cloud service Composition in the cloud-native environment. In Section 3, we provide a design of service composition based on IoT-Cloud pattern. We implement the proposed approach in Section 4 and verify the proposed approach by applying the real scenario of IoT-Cloud services in Section 5. Finally, in Section 6, we conclude the paper.

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2015-0-00575, Global SDN/NFV OpenSource Software Core Module/Function Development). This work is also partially supported by the Data-centric IoT-cloud service platform for smart communities (IoTcloudServe@TEIN) project under the WP4 Future Internet of Asi@Connect.

Seunghyung Lee, Jungsu Han, Jincheol Kwon and JongWon Kim are with the school of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), 123 Cheomdangwagi-ro, Buk-gu, Gwangju, 61005, Republic of Korea (e-mail: {shlee, jshan, jckwon, jongwon}@nm.gist.ac.kr).

Correspondence should be addressed to: JongWon Kim

II. IoT-Cloud service composition in the Cloud-native environment

A. Requirements

For IoT-Cloud service composition in the cloud-native environment, we need to understand the requirements at the infrastructure layer that provide computing/network/storage and the software requirements.

- **Infrastructure environments such as IoT, IoT-Gateway, Cloud:** From the point of view for edge computing, the cloud-native IoT-Cloud service environment requires a clustering consisted of IoT, IoT-Gateway, and Cloud infrastructure nodes. It also requires software that is deployed on the IoT-Gateway to coordinate IoT and cloud connections.
- **Cloud-native computing based service development with microservice architecture:** To develop and compose the functions of the IoT-Cloud service on the cloud-native environment, we need to develop a service for each small unit function. To follow such a service development method, we should design the service functions based on the microservice architecture and modularize the functions properly. In addition, the functions must be stateless in a container form to ensure high performance, high availability, and sustainability throughout the life cycle.
- **Deploy workloads to the appropriate nodes:** Basically, services in a cloud-native infrastructure environment work by deploying workloads to clustered nodes. However, the workloads of IoT-Cloud services should be run on nodes in different environments. Therefore, we should consider the process of specifying and deploying the actual workload to the appropriate node. We also need to ensure that the workloads do not cause conflicts with each other.
- **Connect service workloads:** To compose IoT-Cloud services in a cloud-native computing environment, the connection of modularized workloads based on the microservice architecture is required. To do this, we need to connect the computing, networking, and storage resources in a programmable, flexible way by specifying interfaces using Container Network Interface (CNI), and Container Storage Interface (CSI).

B. Service composition considering IoT-Cloud pattern

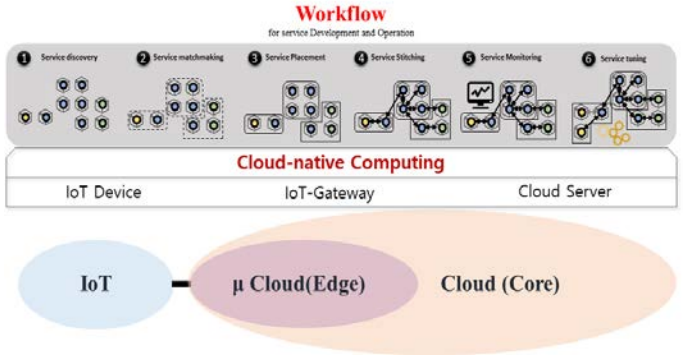

Fig. 1. Cloud-Native IoT-Cloud Service composition

The IoT-Cloud service has a data flow that stores and utilizes data collected from the IoTs connected with the cloud to analyze the stored data and to provide feedback. IoT-Cloud service pattern can be divided into IoT- pattern and -Cloud pattern based on data flow. The IoT- pattern is a part of the IoT-Cloud pattern consisted of functions to collect and forward data from the IoT to the Cloud. The -Cloud pattern means a part of the IoT-Cloud pattern configured by receiving data from the IoT, storing, or utilizing the data.

To apply the service composition to the cloud-native IoT-Cloud services, the services should be designed to consider each pattern of the target IoT-Cloud services since dividing the functions according to the pattern of the IoT-Cloud service can distinguish the workloads with the correct unit Also, to combine the divided functions into a single service, several steps we need to perform. In this paper, we propose the service composition through resource preparation, function creation, function placement, function stitching, and function liaison procedure as shown in Fig. 1.

III. Service composition design based on IoT-Cloud pattern

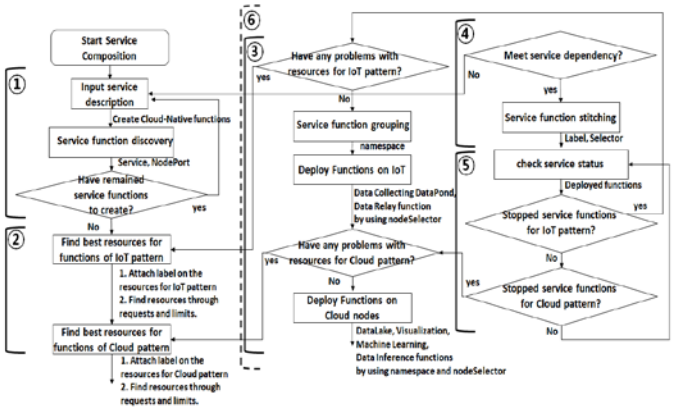

Fig. 2. IoT-Cloud Service composition flowchart

In this section, we designed a service composition process that considers patterns to develop and compose IoT-Cloud services in a cloud-native computing environment. If IoT-Cloud service functions are created based on the patterns and workloads are located in a well-defined sequence, they can be operated well by partially adapting with any other cloud-native service environment.

1) Service discovery: Service functions created on the cloud-native are dynamically allocated and changed on the network, so a discovery process is required to invoke service functions. First, we should be able to find the service function created through the service description. In Kubernetes, we use the service description to find and access IoT-Cloud functions. First, specify the metadata name of the service in the service description to access the functions created in the clustered node. On the other hand, when the function needs to be accessed from outside the cluster, service discovery is enabled by specifying the Kubernetes NodePort of the Service.

2) Service matchmaking: Each function for service

composition has a different resource requirement and the location of the target resource to be executed. In a cloud-native environment, finding and matching the resources is important to accord with the most suitable requirements of the service function among the various resources. In this process, we find the most appropriate resources to distribute each of the functions of the IoT-Cloud service. Data collecting function, temporary storage called DataPond, and data relay functions should be distributed to resources for IoT pattern. The function corresponding to the IoT pattern preferentially selects and matches a node having a resource capable of performing a function such as a sensor or a camera. In addition, functions such as DataLake, which is the final repository, and analysis and visualization using the accumulated data, are distributed to the nodes in the Cloud pattern. The best resource to consider in service matchmaking is the node with the least amount of CPU and memory usage. Then, the functions are then distributed to the selected nodes in a Round Robin (RR).

3) Service placement: In this stage, The function placing is in progress on selected resources through service matchmaking procedures. If there is a required precedence function in the service, it should be arranged in order. The service distribution should be done after checking the node status of the previously matched functions. At this time, the relation of the separated functions must be specified in order to prevent a crash between the functions. The IoT-Cloud service distinguishes IoT, Gateway, and Cloud nodes through attached labels on the cluster nodes, and distributes the functions by separating them through the namespace in the description of the service to be deployed.

4) Service stitching: In the service stitching procedure, the distributed functions are linked according to the service specification. This linkage enables to transmit and receive data of mutually connected functions through the definition of the connection relationship of functions. Services that need to be linked have dependencies. If the previously distributed functions are essential, the stitching must be done taking into account the order. We should check again to make sure we can re-distribute it in case of not deployed or does not work. When the dependency of the service function is satisfied, we define the connection relationship of the functions by specifying the label and the selector in the service description of Kubernetes.

5) Service monitoring: When the service stitching is completed, the service is operated. However, the service composition should be completed through continuous monitoring and service management. Service monitoring in a cloud-native computing environment should be at the level of modularized service functions. The service is maintained by continuously checking the status of service functions and taking action accordingly.

6) Service tuning: Maintaining continuous service is necessary by taking action if the status of the service functions is abnormal. Service tuning in a cloud-native computing environment fixes services by re-distributing and re-stitching modularized functions. In case of IoT-Cloud service, the service tuning procedure is as follows. First, the functions corresponding to each pattern are classified. Then, after identifying the node where the problem occurred, re-distribution and re-stitching proceed. This results in a relocatable service composition.

IV. Implementation of Selected Cloud-native IoT-Cloud Service

We developed specific IoT-Cloud services called Smart Energy IoT-cloud service to apply the service composition method based on IoT-Cloud pattern. Smart Energy IoT-Cloud service collects temperature, the humidity of the server room, and collects the outdoor weather status data. When the collected data detects an abnormal phenomenon such as high temperature, the server manager can visualize and monitor the situation on the dashboard which can be accessed through a web browser.

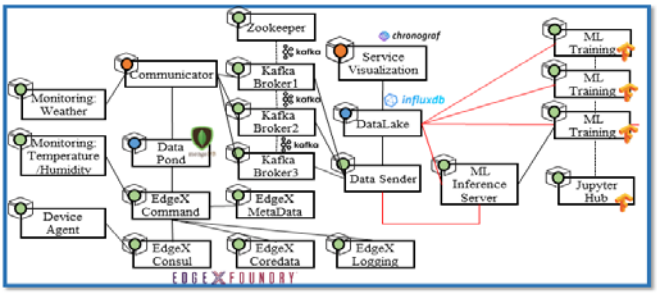

Fig. 3. Implemented Smart Energy IoT-Cloud Service function diagram

First, the functions of Smart Energy IoT-Cloud service in Fig. 3 are Python programs developed with a microservice architecture. The implementation of data collecting function utilizes the DHT22 sensor connected to Raspberry Pi 2. Adafruit Python package was installed and utilized to measure the temperature and humidity of the server room using connected sensors. We also used the open weather map, an open-source API, to collect weather data such as external temperature, external humidity, and external weather data. We also leveraged the open source framework EdgeX Foundry for Edge computing to implement data relaying. The DataPond function used MongoDB and created Collections for data storage. The functions of EdgeX service connected other EdgeX functions through the port inside the container. The EdgeX Metadata function registers the IoT device and data format. This allows EdgeX Coredata to transmit the data collected by IoT to the DataPond based on the RESTful API.

We also implemented the relay function utilizing Kafka to cope with large data efficiently collected from IoT devices. By using Kafka, we can deliver the data to the Data storage in the cloud reliably. In this implementation, Kafka was composed of three brokers and one zookeeper. To transfer data from DataPond to Kafka, we implemented an API server called Communicator using Flask. When transferring data from the IoT device to the EdgeX service, the API request is sent simultaneously to transfer the data from the DataPond to the Kafka broker. The final data repository for Smart Energy IoT-Cloud service was built using InfluxDB, an open-source

time series database, and it was defined as DataLake. The data stored in Kafka is transferred to the DataLake through Data Sender as Kafka consumer. And the Smart Energy IoT-Cloud service used the Influx database to store and monitor the status of the service. We implemented chart and graph visualization function using Chronograf visualization tool.

Data analysis function is implemented as the machine learning function that considers energy efficiency. First, machine learning training is performed by ingesting data of Smart Energy IoT-Cloud service by using Kubeflow which is an open source project for machine learning workflows on Kubernetes. One record of the training data consists of the time stamp, air conditioner current temperature, outdoor humidity, outdoor temperature, server room temperature, server room humidity, server temperature, weather. This data is extracted as a CSV file for machine learning and delivered to the machine learning training function. The machine learning training server uses the received CSV file to perform learning and generate a trained model. After the trained model is deployed in an inference server, the inference server is ready to offer the inferred temperature using the RESTful API.

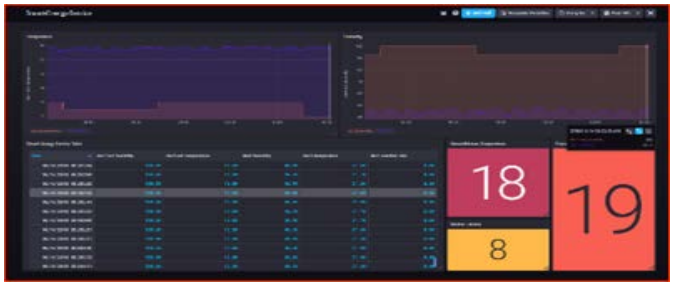

Fig. 4. UI for Smart Energy IoT-Cloud Service monitoring

Fig. 4 shows the result of the visualization of the implemented Smart Energy IoT-Cloud service. The graph on the upper left of the user interface shows room temperature and outdoor temperature, and the graph on the upper right illustrates indoor humidity and outdoor humidity. This UI enables monitoring of temperature and humidity and detailed data accumulation results in the form of a table.

V. Validation of Selected IoT-Cloud Service

A. *Target environment for service verification*

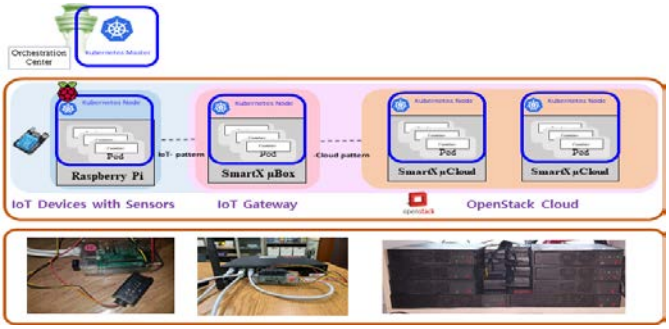

Fig. 5. Local-site verification environment

Fig. 5 shows the local-site verification playground for cloud-native infrastructure environment. IoT device consisting of a Raspberry Pi, physical IoT-Gateway micro box, and two logical VMs created in OpenStack Cloud. Each component is a Kubernetes worker node and consists of one cluster, including the Kubernetes master at the Orchestration Center. The local-site playground utilizes the Weave network add-on for networking of functions.

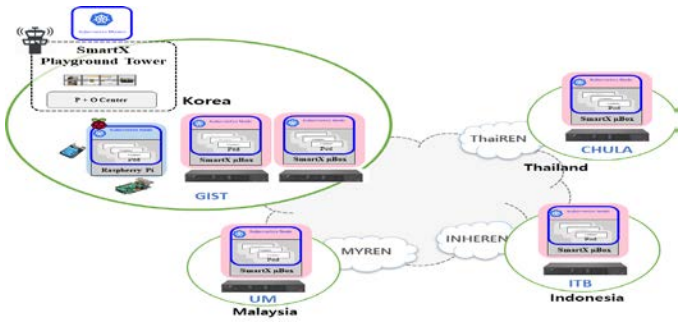

Fig. 6. Multi-site verification environment

Fig. 6 shows a multi-site verification playground with multi-site distributed small-size boxes so called microboxes. Unlike the local-site verification playground, multi-site verification playground is provisioned through physical nodes that are not logical VM nodes through the cloud. First, two microboxes and one Raspberry Pi are working as the Kubernetes worker nodes and P+O Center of SmartX Playground Tower is used as the master node in GIST (Gwangju Institute of Science and Technology) in Korea. In addition, a multi-site verification playground was composed through microboxes located in CHULA (Chulalongkorn University) in Thailand, ITB (Institute Teknologi: Bandung) in Indonesia, and UM (University of Malaya) in Malaysia. They are Belonging to the OF@TEIN project for international collaboration [6]. For the container networking between distributed boxes, we use Calico network plugin on Kubernetes.

B. *Smart Energy IoT-Cloud Service composition*

1) Service discovery: The functions of Smart energy service are divided into internal discovery and external discovery according to the access policy. Smart Energy service has Kafka, DataLake and Visualization functions that service discovery is done according to the internal access policy by describing metadata of service. In contrast, the Communicator, EdgeX service, DataPond, and ML functions, which are external discovery approaches, utilize NodePort on Kubernetes.

2) Service matchmaking: In Smart Energy Service, Kubernetes deployment description is used to make requests and limits on required resources by specifying them in the YAML file. For example, the Communicator function matches the cloud node resources with 200 Milli-cores CPU and 64 Milli-byte memory requirements. Through this process, appropriate nodes can be matched with functions by considering requirements.

3) Service placement: First, IoT, Edge, and Cloud are

assigned to nodes in the cluster, respectively through label. We use the label to specify the Kubernetes NodeSelector to distribute service functions to the right node. In smart energy IoT-Cloud service, The IoT node has a data collecting function and the Edge node has a data relay function. We also deploy DataLake and visualization functions on Cloud nodes, and distributed machine learning server and inference server. We also proceed through the grouping of dependent functions. The grouping is done through the namespace, which was intended to prevent conflicts between the distributed functions.

4) Service stitching: The connection between the data collection function and the relay function is defined using the Kubernetes Selector in the YAML file. In addition, we use the Kubernetes NodePort to forward ports from the actual ports to the Kubernetes ports. For example, the inference server function uses NodePort through service discovery to support inference requests based on RESTful API. It sends six consecutive datasets with five fields, such as external temperature, external humidity, server room temperature, server room humidity, and weather information, to the connected port in JSON format.

5) Service monitoring: In the service monitoring phase, the connection relation and the operation state of the distributed functions are divided into nodes and namespaces. Deployment proceeds to other nodes, and monitoring is discussed in more detail in the next section, along with the results of using the visualization tools.

6) Service tuning: In Smart Energy Services, service tuning identifies problematic functions through monitoring, re-distributes them, and re-stitching them. If there are problems among the inter-connected functions, the functions are re-distributed six times to the same node. If there are persistent problems after 6 re-distributions, try again to another node with the same label via NodeSelector.

C. Verification of Smart Energy IoT-Cloud service using visualization tool

Finally, we check the status of the container-based service through the service composition using the basic open-source visualization tool. We confirm that Smart Energy IoT-Cloud service using a cloud-native environment not only local-site but also multi-site can operate well.

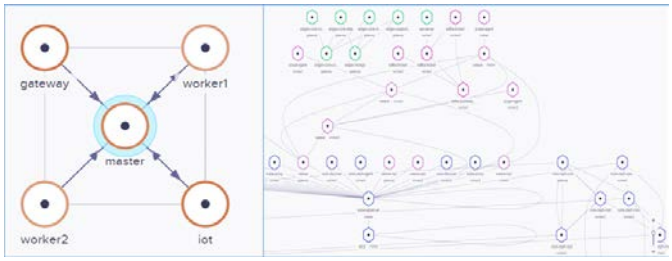

Fig. 7. Local-site workload layer visualization

Fig. 7 shows the state of the workload deployed in the local-site verification playground. We can see container-based workloads deployed in IoT, IoT-Gateway, and Cloud nodes. We can also see the service functions completed through the service composition process of the five steps are deployed to the appropriate nodes and inter-connected and running.

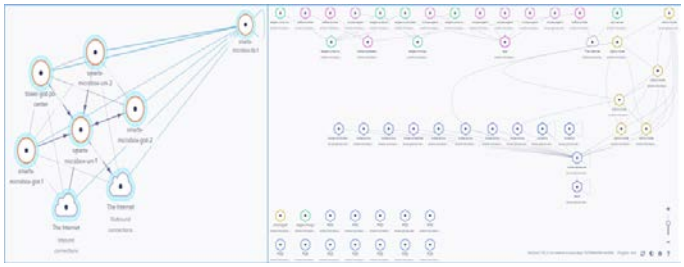

Fig. 8. Multi-site workload layer visualization

Fig. 8 shows the container-based workload status deployed in a multi-site verification playground. The left side demonstrates the connection status of the nodes spread on the mult-site, and the right side shows the status of the workloads distributed on the nodes. In a multi-site verification playground where there is no IoT-Gateway node, the IoT-Gateway functions spread to the smartx-microbox-gist-1 node. And the microbox nodes spread to other sites are used as a cloud. Also, it illustrates that the functions of -Cloud pattern are deployed and running on the microbox which is spread in UM, ITB, and CHULA. The functions of different patterns are deployed in the isolated namespace as pod units and operate normally.

VI. CONCLUSION

In this paper, we propose a relocatable service composition based on microservice architecture according to IoT-Cloud pattern. In addition, specific IoT-Cloud service that we implement is applied to two different cloud-native computing environments and verify the feasibility of the proposed approach by using visualization tools. Thus, we confirmed that the cloud-native service has excellent portability. In conclusion, like the approach proposed in this paper, we can develop services through service composition in response to future-oriented and practical cloud-native services.

REFERENCES

[1] Madakam, Somayya, R. Ramaswamy, and Siddharth Tripathi. (2015, Jan.). Internet of Things (IoT): A literature review. *Journal of Computer and Communications. 3(5),* pp. 164–164.

[2] Tang, Bo, Ravi Sandhu, and Qi Li. (2015, Nov.). Multi-tenancy authorization models for collaborative cloud services. *Concurrency and Computation: Practice and Experience. 27(11),* pp. 2851–2868.

[3] H. Mansoo, L.Kwanwoo, Y. Seonghye. (2014, Feb.). Software Development Methodology for SaaS Cloud Service. *The Institute of Internet, Broadcasting and Communication (IIBC). 14(1),* pp. 61–67.

[4] Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi, "Migrating to cloud-native architectures using microservices: an experience report," European Conference on Service-Oriented and Cloud Computing, 2015, pp. 201-215.

[5] Cloud Native Computing Foundation (CNCF) [Online]. Available: https://www.cncf.io

[6] Kim, J, et al. "OF@ TEIN: An OpenFlow-enabled SDN testbed over international SmartX Rack sites," Proceedings of the Asia-Pacific Advanced Network 36, 2013, pp. 17-22.