Implementation of Platform Controller and Process Modules of the Edge Computing for IoT Platform

Hidenobu Watanabe
Information Media Center
Hiroshima University
Higashi-Hiroshima, Japan
h-watanabe@hiroshma-u.ac.jp

Tohru Kondo
Information Media Center
Hiroshima University
Higashi-Hiroshima, Japan
tkondo@hiroshma-u.ac.jp

Toshihiro Ohigashi
School of Information and Telecommunication Engineering
Tokai University
Tokyo, Japan
ohigashi@tsc.u-kokai.ac.jp

***Abstract*—Edge computing requires a flexible choice of data-processing and rapidly computation performed at the edge of networks. We proposed an edge computing platform with container-based virtualization technology. In the platform, data-processing instances are modularized and deployed to edge nodes suitable for user requirements with keeping the data-processing flows within wide area network. This paper reports the platform controller and the process modules implemented to realize the secure and flexible edge computing platform.**

***Index Terms*—edge computing, container technology, virtualization, IoT**

I. Introduction

IoT (Internet of Things) devices have been increasing year by year. In 2020, the 50 billion devices will be connected to the Internet and the volume of digital data around the world will be about 40 zettabytes have been predicted [1], [2]. Edge computing requires the capability for deploying various data-processing demanded as IoT computing appropriately and rapidly.

We proposed the idea of an edge computing platform based on functional modulation architecture [3] in 2017. In this year, we implemented the platform controller (hereinafter called "controller") and process modules (hereinafter called "modules") as a prototype. The platform takes advantage of container-based virtualization technology to allow data-processing instances to be modularized. Each module provides a data-processing service such as filtering, data compression, data storing, security and feedback. The benefit of decomposing data-processing instances into individual service is that it makes data-processing flows easier to provide according to various user requirements. This paper reports details about two components of the controller and modules.

The main contribution of this work is to develop the controller that can be seamlessly worked with the existing orchestration tool for container-based virtualization technology in order to treat an edge processing workflow as a data flow model according to a user requirement.

The rest of this paper is structured as follows. Section II introduces the platform controller. Section III explains about process modules. Section IV indicates the result of simple performance check. Section V shows one of the future evaluation scenarios. Finally, Section VI summarizes the platform and future works.

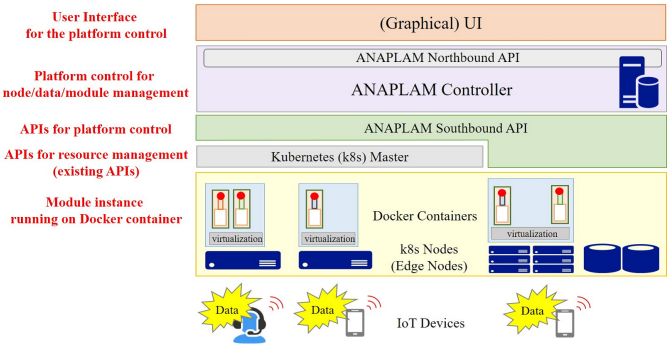

Fig. 1. The component of the platform controller.

II. Platform Controller

A. Controller Component and System Architecture

We implemented the controller which takes advantage of Kubernetes[1] to manage the whole platform. Fig.1 shows the component of the platform controller[2]. Kubernetes (hereinafter called "k8s") is an open source orchestration tool supporting functions such as container grouping, IP address management, load balancing and computer resource monitoring for containers provided by Google Inc. In the platform, the modules are run on the Docker container. Docker is an open source system for operating system-level virtualization and provides lightweight containers which run multiple processes each in their own isolated user-space instance. The platform also provides the northbound API and the southbound API. The northbound API is RESTful API that allows for interaction with the controller from the user interface which can make requests of an end user or system administrator. The southbound API provided by the K8s API is used for automating deployment, scaling and operations of modules across edge nodes located within a wide area network. The novelty of the controller is to manage association among respective modules and conditions for the deployment, which allows modules to be flexibly deployed according to user requirements with maintaining data-processing flows.

[1]https://kubernetes.io/
[2]ANAPLAM is our project codename

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

In the platform, configuration files are provided as a simple user interface. The configuration files consist of a flow configuration file and a module configuration file. They are posted by the northbound API to the controller using HTTPS request, which its results are replied by HTTPS response. For the details about the configuration files and the APIs, please refer to II-B. The k8s follows the master-slave architecture. The k8s master is the main controlling unit of modules, managing its workload and directing communication across the k8s nodes. The k8s node is an edge node where containers are deployed. Every node runs a container runtime of Docker for communication with the k8s master over HTTPS. For smooth control with consistency among the controller and each k8s node, the controller uses components of k8s control plane which has a distributed sharing service such as etcd or a Docker repository. The data-processing flow is a simple mechanism that is to associate the respective modules in a way that one module pushes data to the next over HTTPS as the data pipeline like. In the platform, it assumes that data is transmitted from IoT devices to modules using JSON over HTTPS.

B. Configuration Files and APIs

The configuration files are YAML files. Fig.2 shows example of configuration files. The flow configuration file is described by users to define data-processing flow. For example, a user can define flow information such as a module type, source IP address of a k8s node that runs the module, a name of next module, URI for module access and conditions for the deployment. The module configuration file is used to register a module with the controller, which allows users to define module information such as file path of module image or access control for the module.

Table I shows the API list. The controller provides three type APIs for k8s node management, module control and flow control. The API for k8s node management has a ”list” method which is used to get system information of a k8s node. APIs for module control have ”get”, ”define”, ”undef”, ”list” and ”update” methods which are used for obtaining of module configuration information, definition of a module, deletion of a module or updating of a module configuration etc. APIs for flow control have ”get”, ”create”, ”delete”, ”list” and ”update” methods which are used for flow creation or flow deletion etc.

```
metadata:
  name: hogeflow          # name of a flow configuration file
  description:
  labels: streamingapps                                  # label name
  uuid:  d251bf51-59a6-40ee-965a-9274ecbadc01   # (return value)
  creationTimestamp: 2017-01-01T02:00:00Z        # (return value)
  lastmodificationTimestamp:                             # (return value)
flows:
  endpoint:
   name: th1            # name of first module instance
     module: ifthen             # module type (ex. If-then module)
     location: nodeA            # k8s node to run first module instance
     url: 192.0.2.100/input     # defined URI (return value)
     debug: on
     config:                    # configuration of first module
       source:
         - addr1                        #  source IP address of k8s node
         - addr2
       destination:
         - (name)                       # name of second module instance
       params:                  # details of parameter
         key1:  “value”
         key2:  “value”
   name: ibe1               # name of second module instance
     module: ibe
     location: nodeB
     url: 192.0.2.101/input
      …
```

```
metadata:
  name: ifthen              # name of a module configuration file
  description:
  labels:                                                  # label name
    app: data
  uuid:  949 15955-2803-4898-BD55-2EA0675D981C  # (return value)
  creationTimestamp: 2017-01-01T01:59:59Z               # (return value)
  lastmodificationTimestamp:                               # (return value)
modules:
  image: anaplam/theshold.img               # file path of module image
  endopoint: input
  source:
    protocol: tcp
    port: 80
    deny:
  deny:
    source:                 # modules to deny connection from source
      - comp
    destination:            # modules to deny connection to destination
      - comp
      - datastore
```

Fig. 2. example of a flow configuration file (above) and a module configuration file (below).

C. Procedure

Fig.3 shows procedure of the controller. It is a premise that module images are stored in a private repository of the controller in advance. When the controller receives HTTPS request posted from the user interface, JSON file is created for a request to create pod and service which are technical terms of k8s. The pod consists of one or more containers that are guaranteed to be co-located on the host machine and can share resources. In our platform, notice that one pod is one container. The service is an abstraction function which defines a logical set of pods and a policy by which to access them, which provides transparent access without conscious of the number of pods and location of each k8s node. After receiving the response from a k8s master node that created pod and service, the controller setups and runs modules using the configuration files. The platform takes advantage of container-based virtualization technology to control logical set of modules easily and simultaneously in addition to maintaining consistent access from IoT devices to the module even if next module is rearranged halfway through the flow.

TABLE I
THE API LIST.

Method	HTTP Request	Argument	Return Value (JSON)	Description
API for management of k8s node				
list	GET /nodes	none	result by command "kubectl get nodes -o json"	Get system information of k8s node
API for module control				
get	GET /modules/moduleId	module ID	module configuration	Get module configuration information
define	POST /modules	module configuration	module configuration	Define module
undef	DELETE /modules/moduleId	module ID	success or failure	Delete module
list	GET /modules	none	module list	Get module list
update	POST /modules/moduleId	module ID	module configuration	Update module configuration
API for flow control				
get	GET /flows/flowId	flow ID	flow configuration	Get flow configuration information
create	POST /flows	flow configuration	flow configuration	Create flow
delete	DELETE /flows/flowId	flow ID	success or failure	Delete flow
list	GET /flows	none	flow list	Get flow list
update	POST /flows/flowId	flow ID	flow configuration	Update flow configuration

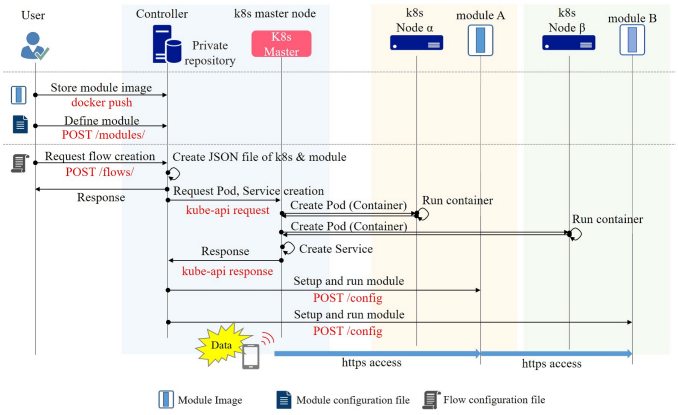

Fig. 3. procedure of the controller.

TABLE II
THE MODULE LIST.

Uses	Module name	function
Data processing	threshold	Filtering and data transfer based on threshold (if-then type condition statement).
	comp	Data compression by requested algorithm and data transfer using HPACK as HTTP-based proxy-like.
	datastore	Data storing to object storage compatible with Amazon S3.
Security	ibe	Data encryption and decryption by Identity-based encryption in addition to key generation and key issue.
	pre	Data encryption and decryption by Proxy re-encryption
Feedback	slacknotify	Notification using Slack API.

III. PROCESS MODULE

We implemented process modules for data processing, feedback and security. Table II shows the module list. Each module meets the API specification described in Table I and the return value in each module obeys HTTP response code. A user can describe the particular processing in each module to the params field of the flow configuration file. In this section, we explain about *the ibe module* and *the pre module* for security which is characteristic of modules provided by our platform.

The identity-based encryption (IBE) enables to encrypt data using ID as a public key in contrast to encryption system based on standard PKI. If there is ID in advance, IBE can start to encrypt. Thus, in the platform which executes to run and to stop modules frequently, we consider that IBE is one of effective security module because that it does not require to verify a public key using a certificate in a typical after running a module. We modularized IBE processing such as encryption, communication and KGC based on encryption system proposed by Boneh et al. [5]. In an implementation, we also defined methods for creation and acquirement of a private key in addition to an internal and external function for encryption.

The proxy re-encryption (PRE) is a crypto-system which allows another user to decrypt a ciphertext which has been encrypted for one user. In advance, a user is able to create the re-encryption key using own private key. If a user increased the number of modules for distributed computing with the purpose of more high performance, PRE allows the module to be accessed to previously encrypted data. In the also case that a user collects modules to one edge node, PRE enables the authority of decryption for previously encrypted data to be delegated to another module. Thus, we consider that PRE is one of an effective security module as same as IBE in our platform. We modularized the PRE processing such as encryption, communication and a public key server based on an encryption system proposed by Ateniese et al. [6]. In an implementation, we defined parameters for the private key, public key and re-encryption in addition to an internal and external function for encryption. The methods for creation, acquirement and registration of a re-encryption key were also implemented.

TABLE III
AVERAGE PROCESSING TIME IN EACH API METHOD.

API for module			API for flow	
define	undef	list	create	list
0.26s	0.28s	0.27s	10.23s	0.39s

IV. SIMPLE PERFORMANCE CHECK

As simple performance check, we confirmed processing time elapsing module deployment controlled through the controller and APIs. We constructed an experimental environment that has the controller and a client on one VM which has 8 core vCPU, 128GB RAM and 100GB SSD. OS is CentOS 7. In this time, we chose the threshold module from Table II for the check. As APIs for evaluation, we chose APIs for module control and flow control from Table I. Notice that we focused on ”define”, ”undef” and ”list” methods in API for module control, then ”create” and ”list” methods in API for flow control. We confirmed the processing time five times in each API method using ”curl” and ”time” of Linux command.

Table III shows the average time of five times in each API method. The average process time except ”create” method was less than 0.5 second. On the other hand, the time of ”create” method was over 10 seconds. We consider that a download time of a module image is a factor in many time were spent because that the client needed to download the image which is hundreds MB size from a private repository of the controller. If there is the HTTP cache, download time is reduced. Therefore, we consider that loading the image onto an edge node in advance will be one of an effective solution to improve the time.

V. FUTURE EVALUATION SCENARIO

We are preparing to establish the experiment environment assuming some evaluation scenarios. Fig.4 shows one of future evaluation scenarios. As a general scenario, we assume that the edge node stores sensor data to cloud storage according to user requirement by if-then module. If the controller monitoring network condition between the edge node and cloud detects an increase of network traffic, it changes to data-processing flows that added the comp module which can shrink data size to reduce network traffic. If user hopes to ensure confidentiality, the controller will change to the data-processing flows that added further the ibe modules. We plan to quantitative evaluation of the performance such as processing time, a load of CPU, memory and HDD by changing of parameters such as data size, a frequency of data transmission or location of module deployment based on the scenario.

VI. CONCLUSION

We implemented the platform controller and the process modules as a prototype. The platform using container-based virtualization technology modularizes each data-processing instances and provides various data-processing services such as threshold judgment, data compression, data storing, notification, IBE and PRE. The modules are associated as a data-processing flow according to user requirement by the configuration files, then are deployed to an edge node within the wide area network by the controller with keeping the flow. As a future work, we plan to establish the experiment environment assuming evaluation scenarios to the wide area network and quantitative evaluate the performance. As another work, we will improve the controller to dynamic rearrange modules in cooperation with the controller and edge nodes.

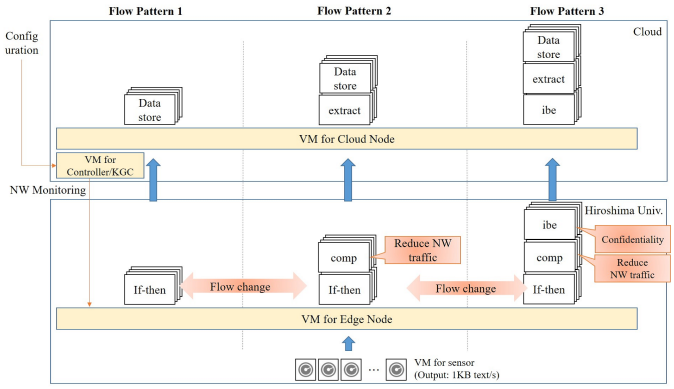

Fig. 4. The future evaluation scenario.

ACKNOWLEDGMENT

This research and development work was supported by the MIC/SCOPE #162108102 and JSPS KAKENHI Grant Number 15K00130, 15K00185, 16H02808.

REFERENCES

[1] Ministry of Internal Affairs and Communications of Japan (MIC), “White Paper 2014 on Information and Communications in Japan,” 2014, http://www.soumu.go.jp/johotsusintokei/whitepaper/eng/WP2014/2014-index.html.
[2] Ministry of Internal Affairs and Communications of Japan (MIC), “White Paper 2015 on Information and Communications in Japan,” 2015, http://www.soumu.go.jp/johotsusintokei/whitepaper/eng/ WP2015/2015-index.html.
[3] T. Kondo, H. Watanabe, and T. Ohigashi, “Development of the edge computing platform based on functional modulation architecture,” 41st IEEE Annual Computer Software and Applications Conference, Turin, Italy, July 4-8, 2017. vol. 2, pp.284-285, 2017.
[4] B. I. Ismail, E. M. Goortani and M. A. Karim, “Evaluation of Docker as Edge computing platform,” Proc. IEEE Conference on Open Systems, 2015.
[5] D. Boneh and M.K. Franklin, “Identity-based encryption from the weil pairing, ” SIAM J. Comput., vol.32, no.3, pp.586-615, 2003.
[6] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, “Improved proxy re-encryption schemes with applications to secure distributed storage, ” ACM Trans. Inf. Syst. Secur., vol.9, no.1, pp.1-30, 2006.