J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr
Copyright ⓒ 2019 KSII

A Novel Design of Cloud-based Management Solution for Android Containers☆

Nam Pham Nguyen Xuan[1] Hojin Chun[2] Souhwan Jung[3*]

ABSTRACT

The Android container is used for various purposes such as Bring Your Own Device (BYOD) solution and Android malware analysis. The strong point of android container than other technologies is it can simulate an android device like a real android phone on a hardware layer. Therefore, automatic management solutions for android container are necessary. This paper introduces a new design of cloud-based management solution for android containers. Through the proposal, android containers are managed automatically from a cloud platform – OpenStack with various tasks like: container configuration, deployment, destroy, android version, hardware device. In addition, the system monitoring and system statistics for android containers and hardware devices are also provided.

☞ keyword : Android Container, OpenStack, Container Runtime, HW device, Cloud Computing

1. Introduction

In computing, virtualization technology is to create virtual devices (or software-based) rather than physical devices. Virtualization can be applied to applications, server (or platform), networking, storage and virtualization is the most effective solution to reduce expenses of deployment while boosting efficiency and lightness for all size businesses [1][2]. The benefits of virtualization include reduced capital and operating costs, faster application and resource delivery, minimized or eliminated downtime, and simplified data center management [3]. The traditional platform virtualization refers to create of a virtual instance that runs like a real physical computer with an operating system and an isolated kernel. Software running on a virtual machine is separate from hardware resources and independent of another virtual machine. In traditional virtualization technologies, the hypervisor layer must be installed directly on the physical server device. This layer provides the guest kernel, virtualized hardware for virtual machines run on it. Hypervisor have been being applied widely from open source hypervisor such as: Xen project, KVM to licensed hypervisor: ESXi of VMWare, Hyper-V of Microsoft [4-7].

Except hypervisor technology, Container, a new operating system virtualization technology, has one physical server and one host kernel, which are isolated systems [8]. Containers use the existing features of the Linux kernel when compared to the hypervisor, so the hypervisor does not need to be installed on the host [9]. Containers that run directly in the host kernel are quarantined and deploy many systems, so the use of the underlying hardware is high. In addition, the speed of containers is higher than normal virtual machines run on hypervisor and reducing the cost of management because only host OS needs to be maintained include patches, securities, bugs fixed. Containers are deployed by Container Runtime such as: Linux Container (LXC), Docker or by Container Orchestration Engine (COE) such as: Docker Swarm, Kubernetes, Mesos [10-13].

1 Dept. of Software Convergence, Soongsil University, Seoul, 06978, Seoul Korea.
2 Dept. of Information and Telecommunication Engineering, Soongsil University, Seoul, 06978, Seoul Korea.
3 Dept. of Electronic Engineering, Soongsil University, Seoul, 06978, Seoul Korea.
* Corresponding author (souhwanj@ssu.ac.kr)
[Received 5 September 2018, Reviewed 7 September 2018(R2 22 October 2018, R3 13 November 2018), Accepted 20 November 2018]
☆ This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.2016-0-00078, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning)
☆ Design of Management Solution for Android Containers in OpenStack, June, 2017.
☆ A preliminary version of this paper was presented at APIC-IST 2017 seminar and was selected as an outstanding paper.

Container technology is also used for creating Android container in order to provide Bring Your Own Device (BYOD) solution which separate working environment and user environment [14]. The Android container is configured with the real device and is used to simulate malware analysis [15]. Android container can be deployed on Android operating system or on Linux operating system [16].

After presenting our related work in the Asia Pacific International Conference on Information Science and Technology, we realize that improving the management solutions for android container in cloud environment is important in order to making automatically mechanism for administrator [17][18]. In summary, our work in this paper includes: (1) Develop more management task for android containers, android version templates, hardware devices (2) Provide system monitoring (android container log, hardware logs) (3) system statics for containers and devices.

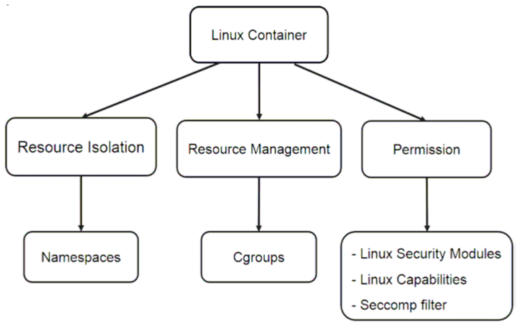
This paper is organized as follows: Section II describes some background information. Related works are explained in Section III. We propose the architecture in section IV and implementation in section V. Performance evaluation is in section VI. Conclusion and limitations will be discussed in section VII.

2. Background

To perform resource isolation, control resource management, and enforce security permissions, the container uses existing Linux kernel features such as: Namespaces, Cgroups, Linux Capabilites, Seccomp Filter, Linux Security Modules. In Figure 1 describes the implementation of Container using the current Linux kernel features. Container applies Namespaces for resource isolation, Cgroups for resource management and Linux Security Modules (LSM), Linux Capabilities, Seccomp Filter for security permission.

2.1 Resource Isolation

In the Linux kernel, Namespaces is a resource isolation feature that added into the kernel from version 3.8. The purpose of Namespaces is to ensure the system resource used by one group of processes does not interfere with other system resource used by other group of processes [19].



(Figure 1) (The Linux kernel features are used by Container)

Therefore, namespaces are essential for container implementations. The Linux kernel uses six namespaces, and each namespace manages elements such as host names, domain names, and networking separately.

Namespaces is implemented by using three namespace-related APIs (or system calls) with the CLONE_NEW* identifier:

- clone(): creating a new namespace and a new process.
- unshare(): calling process from a particular namespace.
- setns(): calling process, creates a new namespace without having to create a new process.

Description of the six types of Namespaces is in Table 1.

(Table 1) (The six types of Namespaces)

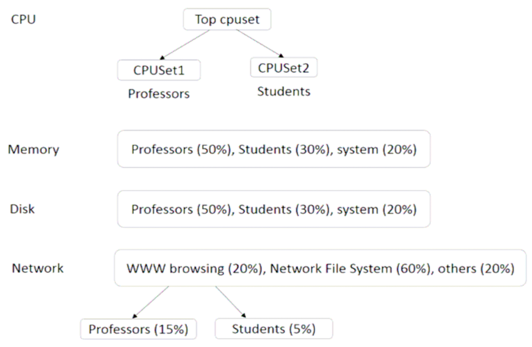
Type of Namespaces	Role
Mount Namespaces (CLONE_NEWNS)	Isolate the set of files system mount points seen by a group of processes.
UTS Namespaces (CLONE_NEWUTS)	Isolate two system identifiers: node name and domain name.
IPC Namespaces (CLONE_NEWIPC)	Isolate certain inter-process communication (IPC) resources.
PID Namespaces (CLONE_NEWPID)	Isolate the process ID number space.
Network Namespaces (CLONE_NEWNET)	Provide isolation of the networking system resources.
User Namespaces (CLONE_NEWUSER)	Isolate and separate the user and group ID number in Operating system.

2.2 Resource Management

Control Cgroups, commonly referred to as Cgroups, are Linux kernel features that provide resource management mechanisms for processes [20]. Cgroups allow processes to be organized into hierarchical groups who usage of different types of resources can then be limited and monitored. Cgroups implemented through the pseudo-file system cgroupfs require hooking the kernel using the cgroup module. In order to use Cgroups the Cgroup modules must be enabled in the Linux kernel. Currently, Cgroups control the following system resources:

- cpu: control cpu usage.
- cpuacct: generate the reports on CPU resources used by processes.
- cpuset: assign CPUs and memory to processes
- memory: control memory usage.
- blkio: control block device usage.
- devices: control device access.
- ns: the namespace subsystem.
- net_cls: control network packets originating from a processe.
- net_prio: set the priority level of network traffic on each network interface.
- freezer: control process status.
- perf_event: make the performance analysis.

In the Figure 2 is the illustration of an example of Cgroups used for the resource division and resource management for professor's processes and student's processes in a system. By using Cgroup, the administrator can calculate and grant the resource usage of each process.


CPU
Top cpuset
CPUSet1
Professors
CPUSet2
Students
Memory
Professors (50%), Students (30%), system (20%)
Disk
Professors (50%), Students (30%), system (20%)
Network
WWW browsing (20%), Network File System (60%), others (20%)
Professors (15%)
Students (5%)

(Figure 2) (Cgroup control resource management of system)

2.3 Permission

For the purpose of implementing the permission for each process, Linux Capabilities is implemented from the kernel 2.2 [21]. In general, Linux functions perform per-process authorization checks based on process EUID and EGID. A process can issue system calls to the kernel depending on the functionality it is granted. In Table 2 provides the list of capabilities implemented on Linux, and the operations or behaviors of each capability.

(Table 2) (List of Linux Capabilities)

Types of Linux Capabilities	Purpose of each capability
CAP_SYS_ADMIN	Manipulate a range of system administration tasks.
CAP_AUDIT_CONTROL	Enable and disable kernel auditing
CAP_BLOCK_SUSPEND	Employ features that can block system suspend
CAP_AUDIT_READ	Allow reading the audit log
CAP_AUDIT_WRITE	Write records to kernel auditing log
CAP_CHOWN	Make arbitrary changes to file UIDs and GIDs
CAP_FOWNER	Bypass checking of rights on operations that normally require the file system UID of the process
CAP_IPC_OWNER	Bypass right checks for most of operations on IPC objects
CAP_IPC_LOCK	Lock memory
CAP_KILL	Bypass permission checks for sending signals
CAP_FSETID	Not clear set-user-ID and set-group-ID mode bits when a file is modified
CAP_DAC_OVERRIDE	Bypass file permission checks: read, write, execute.

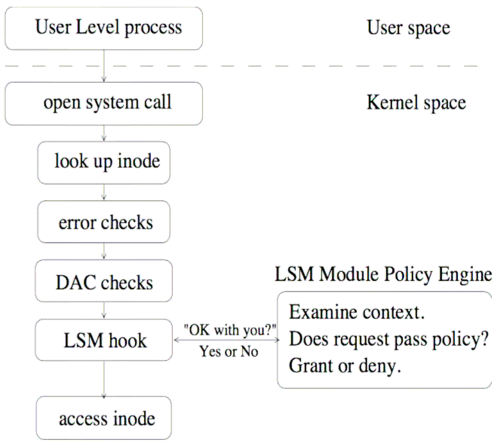
However, Linux features are not strong. For example, if process A has the CAP_SYS_ADMIN capability, process A will be able to execute the system call list from CAP_SYS_ADMIN as follows: setns (), share (), clone (), ptrace (), mount () ... In order to improve the permission of system call for process, Seccomp Filter is applied. Seccomp Filter works as a filter to limit the system call executed by processes [22]. The combination between Linux Capabilities and Seccomp Filter improves the strength on permission for container.

In addition, to improve the security polices for accessing system resource, Container also uses the Linux Security Modules (LSM) [23]. Originally, LSM is a framework that provides hooks into kernel components (such as disk and network services) that can be utilized by security modules such as: SELinux, AppArmor, Smack⋯ to perform access control checks. LSM provides the advanced security policies as kernel modules in Operating system. By providing Linux with a standard APIs for enforcing policies as modules, the LSM project can enable widespread deployment of security hardened systems.

SELinux is one of the most popular projects based on LSM. SELinux has three types of access control:

- Type Enforcement (TE) – primary mechanism
- Role-based Access Control (RBAC)
- Multi-level Security (MLS)

Figure 3 illustrates the architecture of LSM hooks into the Linux kernel. The kernel identifies system resources by looking for inodes whenever a process calls the kernel for system calls and examines the errors encountered. After that kernel carries out the DAC (traditional security policies) checks. And the LSM hook works to check the access control of the process.

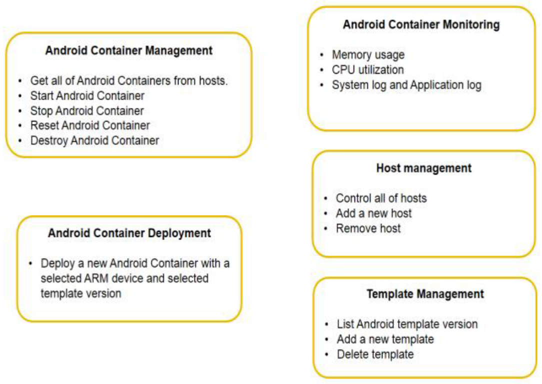


(Figure 3) (The LSM Architecture)

2.4 Android Container

We have already deployed the Android container on the Linux OS. In Linux OS, new Android Containers are deployed by LXC container runtime on Linux OS and physical host is HW device.

Time goes by, the number of Android containers on HW devices increase, the challenges we meet that how can manage a group of containers, deploy a new container and monitor the existing containers. Additionally, management of Android version and HW devices is also important. Therefore, we will propose a new solution for managing Android containers. This solution can be integrated with OpenStack [24]. Figure 4 explains all the required functions of the management solution that we provide on OpenStack cloud platform [25].


Android Container Management
Get all of Android Containers from hosts.
Start Android Container
Stop Android Container
Reset Android Container
Destroy Android Container
Android Container Monitoring
Memory usage
CPU utilization
System log and Application log
Host management
Control all of hosts
Add a new host
Remove host
Android Container Deployment
Deploy a new Android Container with a selected ARM device and selected template version
Template Management
List Android template version
Add a new template
Delete template

(Figure 4) (Functions of the proposed management solution)

3. Related Works

There are several projects underway for OpenStack container management. Zun is a Container Management service for OpenStack which aims to provide an OpenStack API for launching and managing containers backed by different container technologies [26]. Originally, Zun project focuses on basic container operations and integration with OpenStack. Zun is developed independently of Nova's API, it is not bounded by Nova's API. In the Docker, Zun uses the container runtime. Container operations provided by Zun can be easily integrated with OpenStack resources such as networking services and image services. Zun works as a mediate layer between OpenStack and Docker container runtime. Containers are deployed on bare metal device by Docker after receiving the requests from OpenStack via Zun APIs. In order to enhance the security for running containers from vector attacks, before creating the containers, Zun will firstly work with the Nova's API to deploy a new Sandbox (an isolated instance) which contains all the containers. When container is destroyed, Zun will destroy the Sandbox automatically.

Also, Magnum works on OpenStack to provide APIs for managing multi-tenant container infrastructure on a virtualization layer [27]. Magnum does not use single Container Runtime like Zun, it works with Container Orchestration Engine such as: Docker Swarm, Kubernetes, Mesos to deploy and manage containers. Magnum works with OpenStack and COE to deploy Containers, and the COE run on virtualization layer (virtual machines).

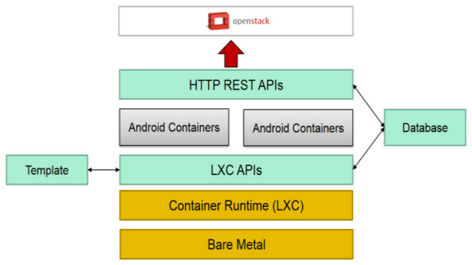
Nevertheless, Magnum and Zun manage only Linux Containers such as Ubuntu Container, Debian Container, and CenOS Container and do not support Android Container. In the other hand, Android Container can not be deployed by Zun or Magnum because Android Container requires a lot of devices for running that are: camera device, SIM device, sound device, WIFI device…

4. Proposed Architecture

This section describes in detail the proposed architecture of management APIs between OpenStack Horizon and Android Containers. As depicted in Figure 5, our architecture consists of six major components: Container Runtime, LXC APIs, HTTP REST APIs, OpenStack Horizon, Template server and Database server.

4.1 LXC Container Runtime

Container Runtime is as a layer interacts directly with the Bare metal (HW device) to provide operating system level virtualization in isolated environments. Users can use the container runtime to take advantage of the API for managing and deploying Android Containers on a single host Container runtime technologies are current being used such as: LXC, Docker, runC, Garden… In our architecture, we use LXC



(Figure 5) (The architecture of management APIs between OpenStack Horizon and Android Containers.)

container runtime to deploy and manage Android Container. Basically, LXC like other container runtime technologies, it only support creating Linux container and in order to deploy Android container, we use our own customized LXC tool. We added and modified the ability to create Android containers in customized LXC.

4.2 LXC APIs – Template – Database

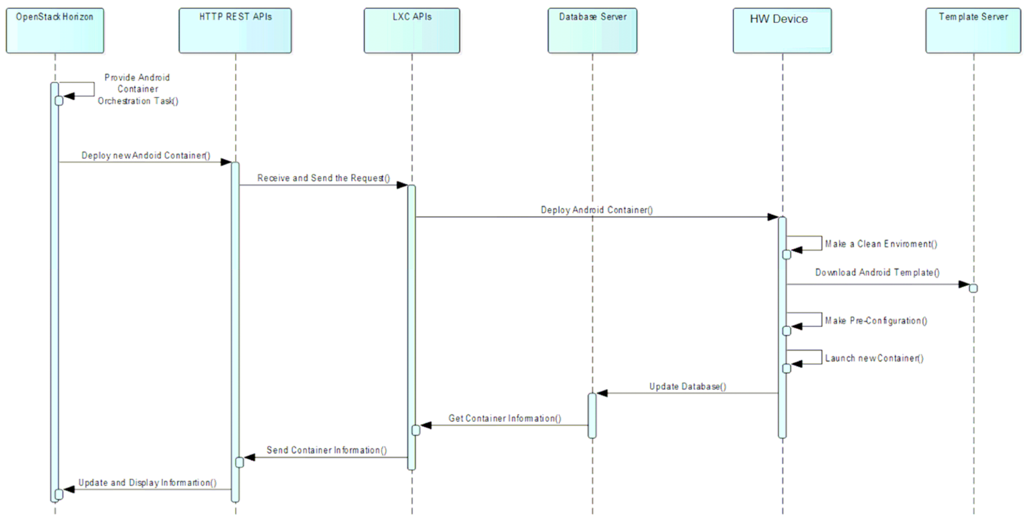
In originally, LXC only run on a single HW device; in order to manage a group of HW devices and Android containers in the cloud platform, we develop the LXC APIs written by Python language to provide remote functions: deploy, list, start, stop, reboot, monitor Android containers.

Template server is a repository which stores and manages different template versions for Android container. Each template includes an Android OS root file system and other configurations. Each time LXC deploys a new container, it requests template server to download suitable version. Currently, we are supporting three Android template versions: KitKat, Lollipop and Marshmallow.

Android container information such as IP address, granted resource, container name, and created time is stored on the database server. Database server not only stores Android container information, but also stores user authentication, Android template version, HW device information and monitoring logs.

4.3 HTTP REST APIs

HTTP REST APIs (HRA) works as a mediate layer between OpenStack Horizon and LXC APIs. HRA is written


OpenStack Horizon
HTTP REST APIs
LXC APIs
Database Server
HW Device
Template Server
Provide Android Container Orchestration Task()
Deploy new Andoid Container()
Receive and Send the Request()
Deploy Android Container()
Make a Clean Enviroment()
Download Android Template()
Make Pre-Configuration()
Launch new Container()
Update Database()
Get Container Information()
Send Container Information()
Update and Display Informartion()

(Figure 6) (The architecture of management APIs between OpenStack Horizon and Android Containers.)

by PHP language, it receives requests from OpenStack Horizon and then transfers the requests to LXC APIs. HRA also connect with database server for getting information and send to OpenStack Horizon. HRA works as a flexible module to provide interfaces for other applications not only OpenStack Horizon.

4.4 OpenStack Horizon

OpenStack Horizon is an OpenStack dashboard project for managing different type of instances such as: virtual machine (KVM, Xen) and physical machine (Ironic). OpenStack Horizon is a project used to integrate systems and is used by many organizations. We develop an Android container management module on OpenStack Horizon for administrator carry outs orchestration tasks such as: manage all of Android version, HW devices and make the system statistic. Our module on OpenStack Horizon communicates directly with HRA to send the requests, receive the result and display on Horizon.

4.5 Running Process

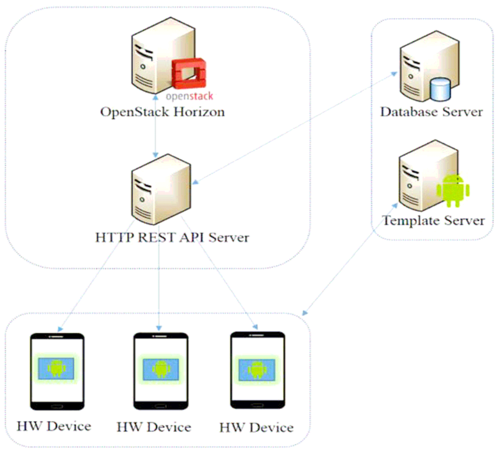
Figure 6 illustrates the detail steps for deploying process of a new android container is our model. On OpenStack Horizon, administrator sends a request to deploy a new Android Container. HTTP REST APIs receives the request from OpenStack Horizon as HTTP request format and sends the request to LXC APIs module. On each HW devices, LXC APIs module runs the function deploy android container including follow steps: making a clean environment, download android template, make some pre-configurations and launch new container instance. After finish deploying new android container, LXC APIs update container information such as: hostname, IP address, android version… to database and sends the container information to LXC APIs. LXC APIs module receives container information and send it back to OpenStack Horizon to display on the GUI.

5. Implementation

This section provides readers with a detailed of Management APIs implementation. Our implementa -tion topology showed in Figure 7 containers in our platform, deploy a new container, monitor all existing containers, manage template includes an OpenStack Horizon server, HTTP REST APIs server, template server, database server and HW devices which used for deploying android containers.

5.1 System Preparation

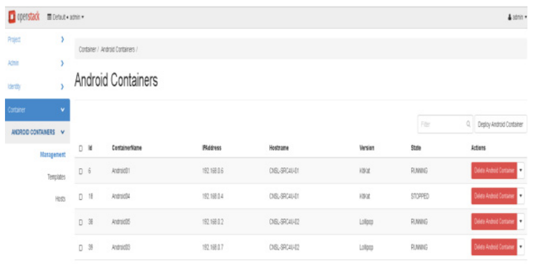
We implement the platform on the Linux operating system. And we uses the Newton version of OpenStack Horizon. The Android container is deployed on HW devices and MySQL is a database server. Programming languages are Python and PHP



(Figure 7) (The implementation of management APIs between OpenStack Horizon and Android Containers)

5.2 Implementation

As in Figure 8 of the implementation task, we develop Android container management functions on OpenStack Horizon such as: get all existing containers, deploy a new container, stop, reboot or destroy. Other functions for managing Android template version and HW devices are also provided on the dashboard.

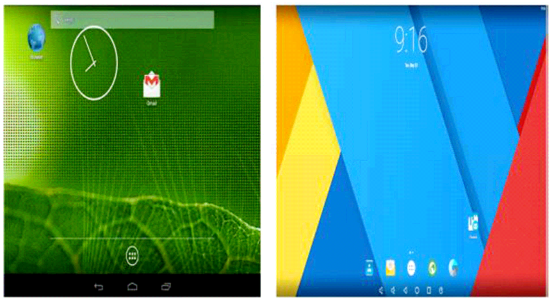


Android Containers

Id	ContainerName	IPAddress	Hostname	Version	State	Actions
6	Android01	192.168.0.6	CNSL-SRCAU-01	Kitkat	RUNNING	Delete Android Container
18	Android04	192.168.0.4	CNSL-SRCAU-01	Kitkat	STOPPED	Delete Android Container
38	Android05	192.168.0.2	CNSL-SRCAU-02	Lollipop	RUNNING	Delete Android Container
39	Android03	192.168.0.7	CNSL-SRCAU-02	Lollipop	RUNNING	Delete Android Container

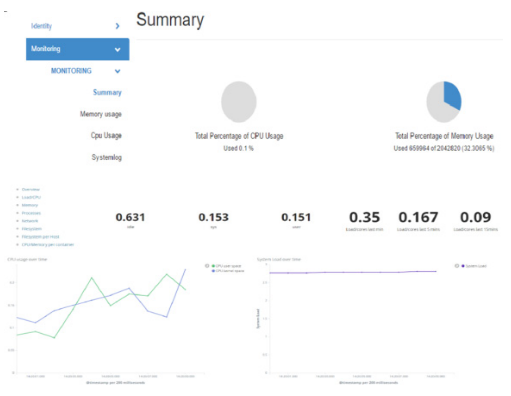
(Figure 8) (Management task are provided on OpenStack Horizon.)

Figure 9 indicates two Android Containers with version: KitKat (4.0) and Lollipop (5.0) are deployed on HW device after receiving the request from OpenStack Horizon.



(Figure 9) (Android Container is deployed)

It also provides the ability to run Android containers and calculate system information for HW devices such as memory utilization, CPU utilization, system log monitoring, and the number of execution processes. Based on these information, administrator can detect the system usage in order to balance the resource. Figure 10 describe the summary of system information on our system.



(Figure 10) (System statistic)

6. Conclusion and Limitations

In this paper, we proposed the design and implementation of the management API for Android containers and OpenStack Horizon. Our proposal can help the operator to deploy and manage Android containers from many HW devices on OpenStack Horizon. On the other hand, by integrating in OpenStack Horizon, the administrator not only manages Android containers but also manages other types of OpenStack instances such as: Nova virtual machine, physical machine. We also improved the deployment time of Android Container, statistic of system information for Android Containers, monitoring Android Containers.

Currently, in our proposed model, the deployment time of a new android container is still quite long, around two minutes. Additional work is planned for platform extensions such as the ability to automatically deploy HW devices, load balancing, and failover between Android containers and HW devices.

Reference

[1] Alouane, Meryeme, and Hanan El Bakkali, "Virtualization in Cloud Computing: Existing solutions and new approach," *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference on.* IEEE, 116-123, 2016. https://doi.org/10.1109/cloudtech.2016.7847687

[2] Mateo, John Cristopher A., and Jaewan Lee, "A Dynamic Task Distribution approach using Clustering of Data Centers and Virtual Machine Migration in Mobile Cloud Computing," *Journal of Internet Computing and Services*, 17(6), 103-111, 2016. https://doi.org/10.7472/jksii.2016.17.6.103

[3] Park, Min Gyun, et al, "Pratical Offloading Methods and Cost Models for Mobile Cloud Computing," *Journal of Internet Computing and Services*, 14(2), 73-85, 2013. https://doi.org/10.7472/jksii.2013.14.2.73

[4] "Xen open source virtualization platform", Accessed 2016 [Online]. Available from: https://www.xenproject.org/

[5] "Kernel-based Virtual Machine", Accessed 2016 [Online]. Available from: https://www.linux-kvm.org.

[6] "VMWare virtualization platform", Accessed 2016 [Online]. Available from: http://www.vmware.com/products/vsphere-hypervisor.html

[7] "Microsoft virtualization platform", Accessed 2016 [Online]. Available from: https://www.microsoft.com/en-us/cloud-platform/server-virtualization.

[8] Li, Zheng, et al, "Performance overhead comparison between hypervisor and container based virtualization," *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on.* IEEE, 955-962, 2017.

[9] Dua, Rajdeep, A. Reddy Raja, and Dharmesh Kakadia, "Virtualization vs containerization to support paas," *Cloud Engineering (IC2E), 2014 IEEE International Conference on.* IEEE, 610-614, 2014.

[10] "Linux Container Runtime", Accessed 2016 [Online]. Available from: https://linuxcontainers.org/

[11] "Application Containers", Accessed 2016 [Online]. Available from: https://www.docker.com/

[12] "Google COE", Accessed 2016 [Online]. Available from: https://kubernetes.io/

[13] "A distributed systems kernel", Accessed 2016 [Online]. Available from: http://mesos.apache.org/

[14] Shumate, Thomas, and Mohammed Ketel, "Bring your own device: benefits, risks and control techniques," *SOUTHEASTCON 2014*, IEEE, 1-6, 2014. https://doi.org/10.1109/secon.2014.6950718

[15] Chen, Wenzhi, et al, "A lightweight virtualization solution for Android devices," *IEEE Transactions on Computers,* 64(10), 2741-2751, 2015.

[16] Kanonov, Uri, and Avishai Wool, "Secure containers in Android: the Samsung KNOX case study," *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices,* ACM, 3-12, 2016. https://doi.org/10.1145/2994459.2994470

[17] Nam Pham Nguyen Xuan and Souhwan Jung, "Design and Implementation of Management APIs between OpenStack Horizon and Android Containers," *KSII The 12th Asia Pacific International Conference on Information Science and Technology (APIC-IST)*, 2017.

[18] Younas, Muhammad, et al, "A Framework for Agile Development in Cloud Computing Environment," *Journal of Internet Computing and Services*, 17(5), 67-74, 2016.

[19] "Resource Isolation", Accessed 2016 [Online]. Available from: https://lwn.net/Articles/531114/

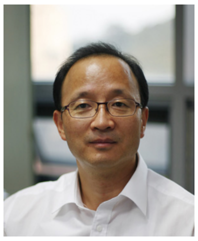
[20] "Resource Management", Accessed 2016 [Online].

Available from:
http://man7.org/linux/man-pages/man7/cgroups.7.html
[21] "Permission Mechanism", Accessed 2016 [Online]. Available from:
http://man7.org/linux/man-pages/man7/capabilities.7.html
[22] "Permission Filter Mechanism", Accessed 2016 [Online]. Available from:
http://man7.org/linux/man-pages/man2/seccomp.2.html
[23] Morris, James, Stephen Smalley, and Greg Kroah-Hartman, "Linux security modules: General security support for the linux kernel," *USENIX Security Symposium,* 2002.
https://doi.org/10.1109/fits.2003.1264934
[24] "An OpenStack Dashboard Project," Accessed 2016 [Online]. Available from:
https://docs.openstack.org/developer/horizon/
[25] Kim, Huioon, et al, "Experience in Practical Implementation of Abstraction Interface for Integrated Cloud Resource Management on Multi-Clouds," *KSII Transactions on Internet and Information Systems (TIIS),* 11(1), 18-38, 2017.
https://doi.org/10.3837/tiis.2017.01.002
[26] "Container Management Service for OpenStack", Accessed 2016 [Online]. Available from:
https://wiki.openstack.org/wiki/Zun
[27] "Management Solution for Containers Orchestration Engines", Accessed 2016 [Online]. Available from:
https://wiki.openstack.org/wiki/Magnum.

◐ 저 자 소 개 ◑

Nam Pham Nguyen Xuan
2012년 Ho Chi Minh City University of Technology and Education 공학사
2017년 숭실대학교 대학원 융합소프트웨어학과 석사
관심분야 : 클라우드 보안, 네트워크 보안
E-mail : namxuannp@gmail.com

Hojin Chun
2017년 숭실대학교 정보통신전자공학과 공학사
2017년~현재 숭실대학교 대학원 정보통신공학과 석사과정
관심분야 : 모바일 보안, 클라우드 보안
E-mail : hjjjang4@naver.com



Souhwan Jung
1985년 서울대학교 전자공학과 공학사
1987년 서울대학교 대학원 전자공학과 공학석사
1996년 University of Washington(Seattle) Electrical Engineering 공학박사
1997년~현재 숭실대학교 전자정보공학부 교수
관심분야 : 모바일 보안, 클라우드 보안, 네트워크 보안, IoT보안
E-mail : souhwanj@ssu.ac.kr