Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH

Open Access

Development of template management technology for easy deployment of virtual resources on OpenStack

Yoji Yamato[*], Masahito Muroi, Kentaro Tanaka and Mitsutomo Uchimura

Abstract

In this paper, we describe the development of template management technology to build virtual resources environments on OpenStack. In recent days, Cloud computing has been progressed and also open source Cloud software has become widespread. Authors are developing cloud services using OpenStack. There are technologies which deploy a set of virtual resources based on system environmental templates to enable easy building, expansion or migration of cloud resources. OpenStack Heat and Amazon CloudFormation are template deployment technologies and build stacks which are sets of virtual resources based on templates. However, these existing technologies have 4 problems. Heat and CloudFormation transaction managements of stack create or update are insufficient. Heat and CloudFormation do not have sharing mechanism of templates. Heat cannot extract templates from existing virtual environments. Heat does not reflect actual environment changes to stack information. Therefore, we propose a new template management technology with 4 improvements. It has a mechanism of transaction management like roll back or roll forward in case of abnormal failure during stack operations. It shares templates among end users and System Integrators. It extracts templates from existing environments. It reflects actual environment changes to stack information. We implemented the proposed template management server and showed that end users can easily replicate or build virtual resources environments. Furthermore, we measured the performance of template extraction, stack create and update and showed our method could process templates in a sufficient short time.

Keywords: OpenStack; Cloud computing; IaaS; Template management server; Heat; CloudFormation

Introduction

In recent days, Cloud computing technologies such as virtualization and scale-out have been progressed and many providers have started Cloud services. Cloud services are divided into SaaS, PaaS and IaaS (Infrastructure as a Service). IaaS service provides hardware resources of CPU or Disk via a network. For examples, Amazon EC2 (Elastic Computing Cloud) [1] and Rackspace Cloud Servers [2] are production IaaS services. As IaaS infrastructure, RackSpace uses open source software OpenStack [3]. OpenStack and CloudStack [4] are major open source IaaS software and adoptions of open source IaaS software are increasing. Because OpenStack community is very active and open, we are also developing IaaS services on OpenStack.

With a spread of cloud services, technologies which deploy a set of virtual resources based on system environmental templates to enable easy building, expansion, migration of cloud virtual resources have emerged. For example, OpenStack Heat [5] and Amazon CloudFormation [6] are template deployment technologies and build stacks which are sets of virtual resources based on templates. However, these existing technologies have 4 problems. Heat and CloudFormation transaction managements of stack create or update are insufficient. Heat and CloudFormation do not have sharing mechanism of templates. Heat cannot extract templates from existing virtual environments. Heat does not reflect actual environment changes such as virtual machine deletion by OpenStack Nova API to stack information.

Therefore, we propose a new template management technology which has a mechanism of transaction management

* Correspondence: yamato.yoji@lab.ntt.co.jp
NTT Software Innovation Center, NTT Corporation, 3-9-11 Midori-cho, Musashino-shi 180-8585, Japan

like roll back or roll forward in case of abnormal failure during stack operations, template sharing among end users and System Integrators, template extraction from existing environments and reflection of actual environment change to stack information. We implemented the proposed template management (TM) server and showed that end users can easily replicate or build virtual resources environments. Furthermore, we measured the performance of template extraction, stack create and update and showed our method could process templates in a sufficient short time.

The rest of this paper is organized as follows. In Problems of existing template technologies, we review OpenStack architecture and clarify problems of existing template technologies for business use. In Proposal of template management technology, we propose a new template management technology which mediates users and OpenStack, and show how to resolve existing problems. In Template management server evaluation, we implement the TM server, confirm our proposed methods feasibility and evaluate the performance. We compare our work to related works in Related works. We summarize the paper in Conclusion.

Problems of existing template technologies

Outline of OpenStack

OpenStack, CloudStack and Eculayptus [7] are major open source IaaS software, and among them OpenStack community is active because many providers contribute developments and adopted services are rapidly increasing. Figure 1 shows architecture of OpenStack.

OpenStack is composed of the function blocks which manage logical/virtual resources deployed on physical resources, the function block which provides Single Sign On authentication among other function blocks and the function block which orchestrates a set of virtual resources. Neutron controls virtual networks. OVS (Open Virtual Switch) [8] and other software switches can be used as a virtual switch. Nova controls virtual machines (VMs). KVM (Kernel based Virtual Machine) [9], Xen [10] and others can be used as hypervisors of VMs. Cinder manages block storages and can attach a logical volume to a VM like a local disk. Swift manages object storages. Glance manages Image files. Keystone is a base which performs Single Sign On authentications of these function blocks. Heat is an orchestration deployment function to create or update virtual resource instances using Nova, Cinder or other blocks based on a text template. Ceilometer is a metering function of virtual resource usage. The functions of OpenStack are used through REST (Representational State Transfer) APIs. There is also Web GUI called Horizon to use the functions of OpenStack.

OpenStack major version is released once a half-year. Henceforth Havana which is the latest version in Feb of 2014, the new functions to catch-up Amazon EC2 will be added.

Clarification of existing template technologies problems

OpenStack Heat and Amazon CloudFormation are technologies which deploy virtual resource instances based on templates which contain information of virtual resource environment and are described by JSON, YAML or other text format. The scope of Heat and CloudFormation are same and Heat supports CloudFormation-compatible API and template format. Both call a set of virtual instances which are deployed based on a template “stack” and provide APIs to operate stacks. However, these APIs provide primitive CRUD (Create, Read, Update and Delete) operations of stack and there are some insufficient points for business use. Here, we clarify 4 major problems of existing template technologies.

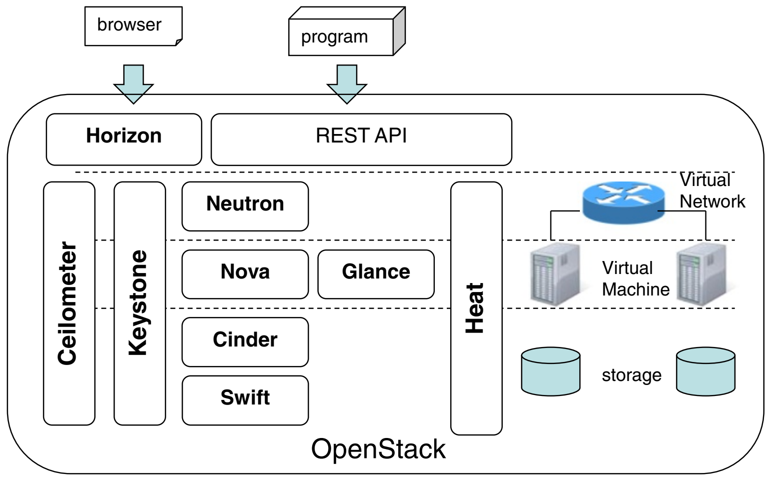


Figure 1 OpenStack architecture.

Insufficient transaction management of stack create, update or delete

Heat and CloudFormation transaction managements of stack create, update or delete are insufficient. When we create a stack, all resources need to be deployed successfully for business use. But Heat and CloudFormation sometimes fail some resources deployment and there is a possibility of end up with a half-finished stack creation. Moreover, when we update a stack, Heat and CloudFormation also fail some resources update and there is a possibility of end up with a half-finished stack update.

There is no template sharing mechanism

Templates sharing or re-use by other users are out of scope of Heat and CloudFormation. Users of Heat and CloudFormation need to describe and manage templates by themselves. However, users who do not have IT knowledge such as small business owners cannot create a template easily. It is preferable that System Integrators or distributers describe and verify templates, and end users only select templates and build virtual resource environments for their business demands.

Heat cannot extract templates from existing environments

Heat has a "template-show" API which gets a template from a stack but cannot extract a template from a non-stack environment. Thus, when users would like to replicate existing environments, they need to describe templates from the first. And because each resource belongs to only one stack in Heat specification, a template extraction is difficult for shared resources such as logical routers which connect VPN or the Internet not to belong multiple stacks.

Actual environment change is not reflected to stack information

Heat does not reflect non-Heat API operations to stack information. For example, when users shut down a VM via console or delete a VM via Nova API, there is a difference between an actual environment and stack information. When users call a stack update API, Heat checks a difference between a new template and a previous template. But in this example case, an actual environment is different from a previous template, so that there is a possibility of unexpected behavior.

Proposal of template management technology

To provide orchestration functions of OpenStack IaaS services to users, we propose a TM server which have a mechanism of transaction management like roll back or roll forward in case of abnormal failure during stack operations, template sharing among end users and System Integrators, template extraction from existing environments and reflection of actual environment change to stack information. The TM server interprets Heat JSON or HOT (Heat Orchestration Template) format templates and calls OpenStack APIs such as Nova or Cinder. The TM server has a stack operation function, template sharing function, template extraction function and environmental change reflection function. Here, we explain how to resolve 1–4 problems by the TM server.

Transaction management of stack create, update and delete

Heat and CloudFormation transaction managements of stack create, update and delete are insufficient and may end up with a half-finished stack processing. It is not acceptable for some business users because some resources failures may lead critical problems. For example, if a VM creation is successful but a logical router security setting is failed, the VM may have a risk of abuse.

Therefore, when we create a stack, it is necessary to delete and roll back all resources in case of any failures of resource creation (All resources roll back). And when we delete a stack, it is necessary to retry and delete all resources in case of any failures of resource deletion (All resources roll forward).

When we update a stack, orchestration functions check a difference between previous template and new template, then create, delete or update resources to fill up the difference. Specifically, a resource which is in previous template and is not in new template is deleted, a resource which is not in previous template and is in new template is created, a resource which is changed from previous template to new template is created after deletion or updated. In OpenStack, some resources can be updated but some cannot be. (e.g. network connection change can be updated but VM RAM size change needs to delete VM once, then create new one).

In stack update case, individual resource creation, deletion and update may be operated, so that there may be a case we cannot roll back all operations. For example, a volume is deleted successfully then a VM creation is failed, we cannot roll back because the volume is already deleted. Therefore, in stack update case, the TM server tries to roll back or roll forward for each OpenStack API transaction (not all API transactions) when each OpenStack API processing is failed.

Figure 2 shows the transaction managements of stack create, delete and update. In stack create case, when there is a problem in logical router creation, the TM rolls-back all created resources. In stack delete case, when there is a problem in logical router deletion, the TM retries and deletes all resources. In stack update case, when there is a problem in logical load balancer deletion, the TM rolls-back the logical load balancer operation because it is a creation operation.

Heat template does not describe an order of each resource processing. Our TM server interprets Heat templates and process following orders for stack create,

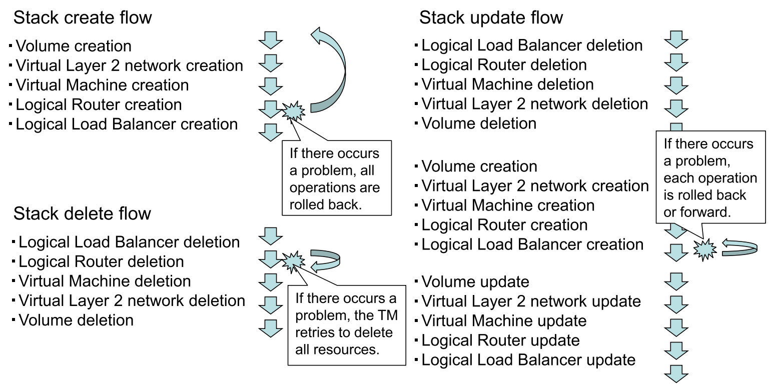

Stack create flow
・Volume creation
・Virtual Layer 2 network creation
・Virtual Machine creation
・Logical Router creation
・Logical Load Balancer creation
If there occurs a problem, all operations are rolled back.
Stack delete flow
・Logical Load Balancer deletion
・Logical Router deletion
・Virtual Machine deletion
・Virtual Layer 2 network deletion
・Volume deletion
If there occurs a problem, the TM retries to delete all resources.
Stack update flow
・Logical Load Balancer deletion
・Logical Router deletion
・Virtual Machine deletion
・Virtual Layer 2 network deletion
・Volume deletion
・Volume creation
・Virtual Layer 2 network creation
・Virtual Machine creation
・Logical Router creation
・Logical Load Balancer creation
・Volume update
・Virtual Layer 2 network update
・Virtual Machine update
・Logical Router update
・Logical Load Balancer update
If there occurs a problem, each operation is rolled back or forward.

Figure 2 Transaction managements of stack operations.

delete and update. Regarding to stack create, the TM server creates resources in the following order; volume, virtual Layer 2 network, VM, logical router and logical load balancer. Regarding to stack delete, the order is logical load balancer, logical router, VM, virtual Layer 2 network and volume. Regarding to stack update, the TM server firstly deletes resources in the order same as stack delete case, secondly creates resources in the order same as stack create case and lastly updates resources in the order same as stack create case. Figure 2 also shows the orders of stack create, delete and update.

Following these orders, we can guarantee the precondition of resource creation in stack create case (e.g. a VM needs at least one volume). We can also prevent name duplication error or other precondition error in stack update case.

To follow these policies, the stack operation function of TM server manages orders of OpenStack API calls and those transactions. Because most of OpenStack APIs are asynchronous, the TM server retries a API or calls a purge API, or reverse API to decide state of OpenStack resource when a API transaction is failed. Note that a reverse API means the reverse process of each API (e.g. volume deletion API is a reverse API for volume creation). In this way, we can prevent a half-way state of stack during stack operations.

The function of stack transaction management can be generalized for deployment management with multiple resources. Resource deployment needs Create, Update and Delete transaction managements and also needs a valid order of each resource operation. Our proposal of stack transactions in case of failure and orders of each resource operation can be used also CloudStack, Eucalyptus and other Cloud platforms multiple resources provisioning because a virtual resource dependence (e.g. VM needs at least one volume) is almost same in IaaS platforms.

Template sharing

CloudFormation and Heat do not have a mechanism of template sharing. Our TM server provides a function to share templates and facilitate templates re-use. For example, when a small business owner would like to build a shopping site, a System Integrator provides a verified Web 3-tier structure template, then the small business owner selects the template and build the environment with one or two clicks. If we share templates unconditionally, there is a risk of malicious template spreading. Thus, it is necessary to limit a range of template sharing within contractual relationships. Here, we explain logics of template sharing.

– There are two methods to register a template: template extraction and template upload. The function described in Template extraction from existing tenant extracts a valid template in an extraction case. In the other side, the template sharing function validates a template in a template upload case because a template described by a user may have format or logical errors.
– Each template creator can set a scope of disclosure for each template. There are 3 options for disclosure; only the creator-self, all users who have contract with the creator and users selected by the creator. Service providers or System Integrators can share templates to subordinate users by setting a scope of disclosure. If System Integrators have multiple tier contractual relationships, there are 2nd tier subordinate users. In this case, 1st tier subordinate System Integrator downloads a template of upper tier System Integrator and registers it as its template. This is to restrict scope of disclosure within direct contractual relationships. Because each System Integrator prefers to sell its own brand, the template sharing function conceals templates upper than two tiers. Figure 3 shows an image of template sharing relations. A Cloud provider A creates and shares the template a to all users which have contracts. A System Integrator B and an End User C can use the template a. B also

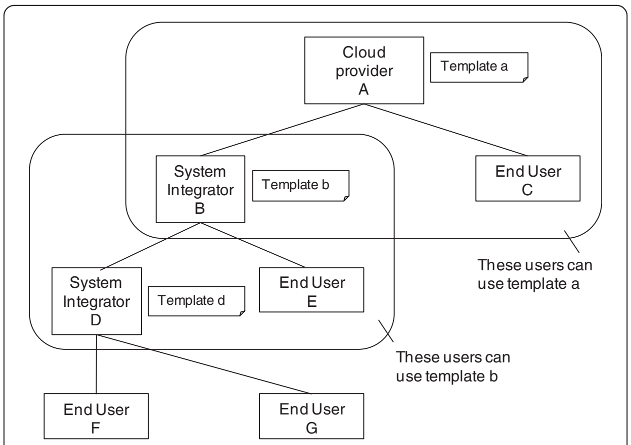


Figure 3 Template sharing among service providers, System Integrators and end users.

creates and shares the template b to users, so that D and E can use the template b.

- Users within scope of disclosure can download a template or create a stack using a template. If a template describes a reference of image ID to create a volume from the image, users also need to be able to use the image. Thus, template sharing function also manages a scope of disclosure for images. Specifically, when a user requests to create a stack, template sharing function represents to get Keystone token as upper System Integrator and create a user volume from an image of System Integrator.

In this way, users who do not have enough IT knowledge can select a shared template and build virtual resources environments easily.

The function of template sharing function shares a template among users. In Cloud services or hosting services, many System Integrators or resellers build and manage services for end users. The function of template sharing is generalized for multi-tier sharing model of cloud or hosting services. This can be used not only a text template file sharing but also restricted sharing of software with license such as OS images or restricted codes sharing for multi-vendor developments.

Template extraction from existing tenant

Heat main targets are operations of stacks and Heat cannot extract a template from non-stack environment. And there is a restriction that each virtual resource belongs to only one stack. Based on them, we propose logics to extract a template from an existing tenant.

- We extract whole virtual resources on an existing tenant to a template. If there is unnecessary resource in an extracted template, a user edits the template after downloading. This is because there is no stack that we cannot restrict corresponding resources for extraction.
- Target resources to extract are volumes, virtual Layer 2 networks, VMs, logical routers and logical load balancers. Floating IPs are IP address resources that relate logical routers which connect the Internet. The Internet connected resources are shared by multiple stacks in general and VMs or logical load balancers which are assigned floating IPs may be shared by multiple stacks. Because a virtual resource only belongs to one stack, we do not include a floating IP to a template in extraction phase. Users can assign floating IPs after stack creation based on the extracted template. In the same way, shared virtual Layer 2 networks or logical routers (e.g. VPN connected routers) are out of scope for extraction because those are used by multiple stacks.
- When users extract a template, they also can select whether to acquire images from volumes of tenant or not. When users create a stack, these images are used to replicate volumes.
- During template extraction time, we block virtual resources operations in the tenant to prevent a change of target resources for extraction.
- Extracted templates are held in the template sharing function described in Template sharing. Extracted templates can be used for stack create, update or download to edit.

In this way, we can extract a template from an existing tenant and replicate an environment easily.

The function of template extraction extracts a template from an existing environment. Our implementation extracts JSON or HOT template and the extracted template can be deployed both by Amazon CloudFormation and OpenStack Heat because a template is abstract text information and does not depend on IaaS platform. To generalize and adapt extracted template format to other IaaS platforms, we can use this function for Cloud migration to another platform or Cloud federation on plural platforms.

Reflection of environment change to stack information

In case of stack update, orchestration functions check a difference between previous template and new template, then create, delete or update virtual resources to fill up the difference. However, Heat can only recognize an environment change by Heat API and does not know actual environment status.

Therefore, we reflect an environment change to stack information to guarantee stack update or delete behaviors as users expect.

There are four methods to change environments.

a) Stack create, update, delete by Heat Stack API
b) Individual resource create, update, delete by other OpenStack API (Nova, Cinder and so on)
c) Resource deletion by user's manual operation. (e.g. VM shutdown via console)
d) Resource deletion by unintentional physical or virtual server down.

Regarding to a), we do not have to care it because templates of Stack API are matched to actual environments after API process.

Regarding to b), when users call OpenStack API (not Heat API), the TM server hooks the requests as OpenStack API proxy and reflects the environment change to stack information. If proxy model is difficult, the TM server may poll OpenStack DB to confirm environment changes. But each OpenStack API does not have a parameter of stack ID so that individual resource creation is not reflected to stack information. If a user would like to add a resource to a stack, a user needs to call Heat stack update API including the resource.

Regarding to c), main case is a VM shutdown by user's manual operation. VM is operated by Libvirt [11] on KVM. Therefore, the TM server can reflect the VM down status to stack information by receiving notifications of Libvirt or other monitoring agents.

Regarding to d), the TM server reflects resources down to stack information by receiving a notification of each resource monitoring agent like c) case.

Based on reflected stack information on a)-d), the TM server can update or delete stacks as users expect. Table 1 shows a comparison of reflection of actual environment change to stack information in Havana Heat case and our TM server case. Havana Heat updates environment changes to stack information only in Stack API use. Our TM updates environment changes to it except for resource create by individual OpenStack API call.

The function of environment change reflection function is generalized to a difference resolve function of actual environment and management layer. The difference has a problem not only in a stack update case but also in an individual resource provisioning case. For example, OpenStack Neutron manages a resource state in OpenStack DB and does not care an actual completion of resource provisioning after it has written the requests to DB. Thus, there is a possibility that a VM is active but access control setting of a logical router to the VM is not available. Because a Cloud provider business is charging fees for provisioned resources, it is fatal to charge a resource not created yet. The function can be used for resolving these differences to collect actual environment information by monitoring modules such as Libvirt and Pacemaker or by hooking requests as a proxy.

Template management server evaluation

We implement the TM server with proposed 1–4 functions and confirm that it can be used for carrier IaaS services. We also evaluate the performance of implemented TM server.

Template management server implementation

Figure 4 shows function blocks of the TM server, OpenStack and related systems. The TM server has three outer interfaces, Web GUI, API and OpenStack communication process. Users can extract and share templates via Web GUI. API provides Heat stack operation APIs which receive requests of stack operations and other OpenStack APIs. OpenStack communication process calls individual OpenStack API such as Nova or Cinder, confirms the request status and asks roll-back or roll-forward in case of abnormal failure. The TM server has a stack information DB which manages stack information and a template sharing DB which manages shared templates. Proposed 3.1-3.4 functions are implemented on the function blocks 1)-4) described in Figure 4.

The stack operation function receives requests of stack create, delete and update via API, makes an order of OpenStack API calls and calls individual OpenStack API via OpenStack communication process. It manages transactions of stack operations, judges to roll-back or roll-forward and calls purge or reverse APIs via OpenStack communication process in case of a abnormal failure. The

Table 1 Reflection of actual environment change to stack information

Operation type		Havana heat	Our TM server
Heat stack API	stack create	✓	✓
	stack update	✓	✓
	stack delete	✓	✓
Other OpenStack API (Nova, Cinder and so on)	resource create		
	resource update		✓
	resource delete		✓
User manual operation (e.g. VM shutdown)	resource delete		✓
Physical or virtual server down	resource delete		✓

A tick mark means that a corresponding operation result is reflected to stack information.

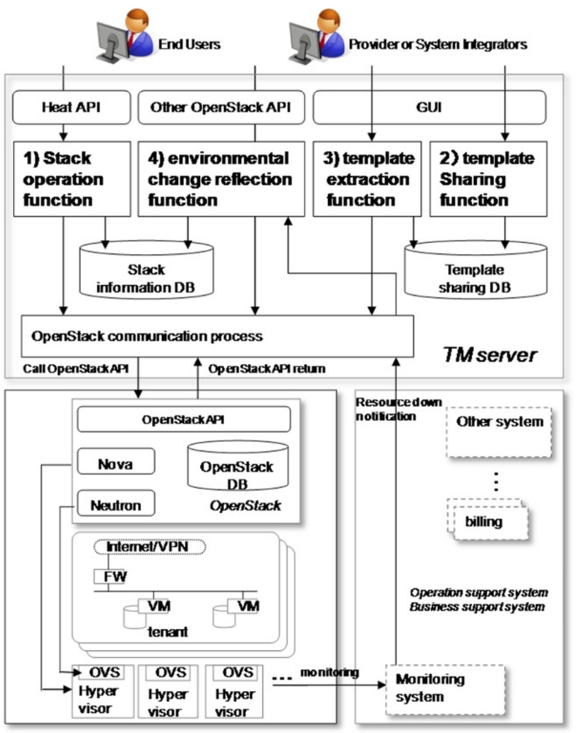


Figure 4 Function blocks of template management servers and related systems.

template sharing function retains templates which are extracted from existing environments or uploaded by users in template sharing DB. It also controls scope of disclosures of templates for users to search available templates. The template extraction function receives a request of template extraction via GUI, gathers information of current virtual resources from OpenStack DB and registers information of extracted template to template sharing DB. The environmental change reflection function behaves as a proxy for non-Heat OpenStack API such as Nova or Neutron API and reflects each resource status change to stack information DB. It also reflects virtual resources down to stack information DB when it receives a resource down notification from a monitoring system.

OpenStack has OpenStack DB, OpenStack modules such as Nova and OpenStack API. OpenStack receives a request via OpenStack API and controls a virtual resource using OpenStack modules like Nova or Neutron and retains virtual resource information in OpenStack DB. OSS (Operation Support System) and BSS (Business Support System) supports an operation of Cloud services. Monitoring system of OSS monitors virtual or physical resource availability and it sends a notification to the TM server when it detects a resource down.

We implemented the TM server on OpenStack Folsom. We implemented it on Ubuntu 12.04 OS and Apache Tomcat 6.0.36 by Java language (JDK1.6.0.38) and Python.

We confirmed 1–4 functions validity on a test environment of Figure 5. It is confirmed that stack is updated based on actual environment information and if there is an failure during stack operation, the TM server roll back or roll forward as expected. And it is also confirmed that System Integrators can extract a template from an existing tenant, share a template among users who have contracts to System Integrators and replicates a virtual resource environment to a user tenant using the template.

Performance evaluation of proposed method

The TM server mediates a user and OpenStack and manages a template processing. We measure the processing time of 3 type processings and confirm that an overhead of our proposed method is sufficient low. The graph values of processing time are average of 3 times measurements.

Performance measurement conditions

This subsection describes measurement conditions of what time is measured, what and how much resources are used and number of concurrent processing number in the measurements. We extract a template from a stack in the 1st measurement, create s stack using the extracted template in the 2nd measurement and update a stack by a new template in which some virtual resources are changed in the 3rd measurement.

The 1st measurement is a template extraction.

– Measured time: Template extraction processing time from a tenant.
– Extracted template: A tenant has 2 VMs, 4 volumes, 2 virtual Layer 2 networks, 1 logical router and 1 logical load balancer.
– Concurrent processing numbers: 1 and 3.

Note that processing time of Image acquisition from a volume is out of scope because it depends on network bandwidth between Cinder and Glance.

The 2nd measurement is a stack create.

– Measured time: Stack creation processing time.
– Template for stack creation: A template has 2 VMs, 4 volumes, 2 virtual Layer 2 networks, 1 logical router and 1 logical load balancer. (the same template in 1st measurement)
– Concurrent processing numbers: 1 and 3.

Note that processing time of volume creation from an image is out of scope because it depends on network bandwidth between Cinder and Glance.

The 3rd measurement is a stack update.

– Measured time: Stack update processing time.

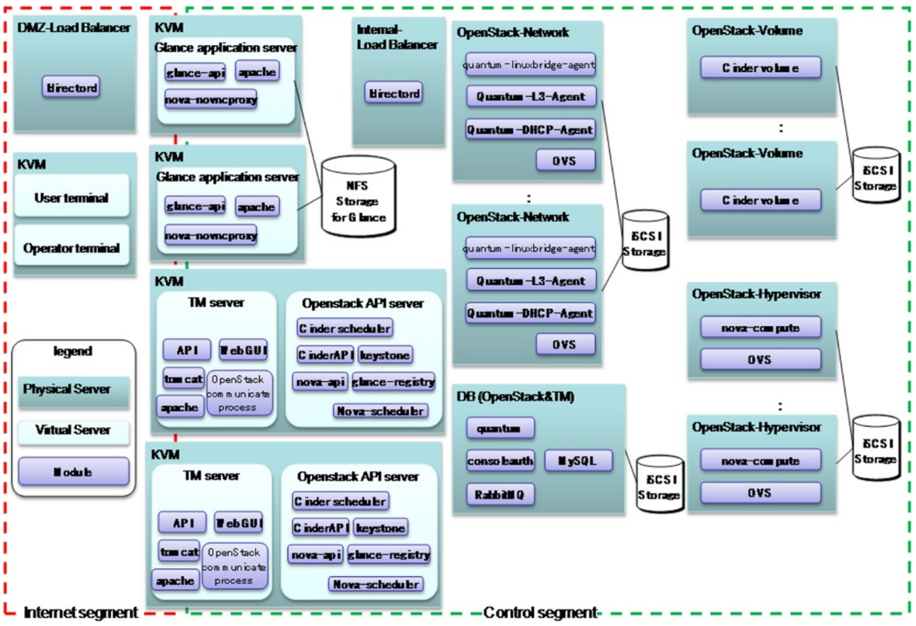
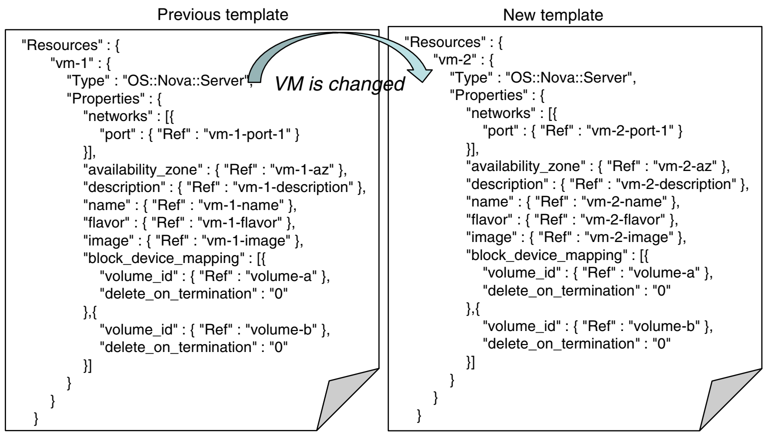


Figure 5 Performance measurement environment.

– Differences between a previous template and a new template: virtual resources numbers are same but 1 VM, 1 virtual Layer 2 network and 1 logical router in a previous template are removed and 1 VM, 1 virtual Layer 2 network and 1 logical router are newly added. Figure 6 shows parts of templates which are used for this measurement. Both templates have a VM but VM properties are different, so that a VM is deleted firstly and another VM is created secondly in stack update.
– Concurrent processing numbers: 1 and 3. Because concurrent update of one stack is blocked by the TM, we measure parallel 3 different updates in this experiment.

Performance measurement environment

Figure 5 is a performance measurement environment. It shows physical and virtual servers and modules in each server. For example in TM server case, a TM server is a virtual server, is in both Internet segment and Control segment and has modules of API, Web GUI, OpenStack communicate process, tomcat and apache. Two servers are for redundancy. Other servers are a user terminal and an operator terminal, Glance application servers for image upload, a NFS storage for image, OpenStack API servers, a DB for OpenStack and TM, OpenStack servers for virtual resources such as network, volume or VM and iSCSI storages and load balancers for load balancing. Figure 5 omits maintenance servers such as syslog or


Previous template
New template
VM is changed

```
"Resources" : {
    "vm-1" : {
        "Type" : "OS::Nova::Server",
        "Properties" : {
            "networks" : [{
                "port" : { "Ref" : "vm-1-port-1" }
            }],
            "availability_zone" : { "Ref" : "vm-1-az" },
            "description" : { "Ref" : "vm-1-description" },
            "name" : { "Ref" : "vm-1-name" },
            "flavor" : { "Ref" : "vm-1-flavor" },
            "image" : { "Ref" : "vm-1-image" },
            "block_device_mapping" : [{
                "volume_id" : { "Ref" : "volume-a" },
                "delete_on_termination" : "0"
            },{
                "volume_id" : { "Ref" : "volume-b" },
                "delete_on_termination" : "0"
            }]
        }
    }
}
```

```
"Resources" : {
    "vm-2" : {
        "Type" : "OS::Nova::Server",
        "Properties" : {
            "networks" : [{
                "port" : { "Ref" : "vm-2-port-1" }
            }],
            "availability_zone" : { "Ref" : "vm-2-az" },
            "description" : { "Ref" : "vm-2-description" },
            "name" : { "Ref" : "vm-2-name" },
            "flavor" : { "Ref" : "vm-2-flavor" },
            "image" : { "Ref" : "vm-2-image" },
            "block_device_mapping" : [{
                "volume_id" : { "Ref" : "volume-a" },
                "delete_on_termination" : "0"
            },{
                "volume_id" : { "Ref" : "volume-b" },
                "delete_on_termination" : "0"
            }]
        }
    }
}
```

Figure 6 Parts of sample templates of performance measurement.

backup servers and redundant modules such as heartbeat. These servers are connected with Gigabit Ethernet.

Table 2 shows each server specification and usage. For example in DB case (6th row), the hardware is HP ProLiant BL460c G1, the server is a physical server, the name is DB, the main usage is OpenStack and TM DB, CPU is Quad-Core Intel Xeon 1600 MHz*2 and Core number is 8, RAM is 24 GB, assigned HDD is 72 GB and NIC (Network Interface Card) number is 4.

Performance measurement results

Template extractions need only DB checks and we can extract templates about 1–3 sec when concurrent processing numbers are 1 and 3. Because the design of system environment takes much time in general, we think quick template extraction is effective to replicate virtual resources environment.

Figure 7(a) shows the processing time of stack create. When concurrent processing number is 1, we can create a stack about 300 sec and when it is 3, it takes about 350 sec. Figure 7(b) shows divisions of resource creation times when concurrent processing number is 1. TM server processing takes 90 sec in orchestration and other times are OpenStack resources creation times (load balancer and VM creation take almost all time of OpenStack processing). Proposed TM server manages transactions of all virtual resources creation. We think an overhead of TM server transaction management is sufficient low. Note that actual stack create may need volume creation from an image and data transfer from Glance to Cinder may take 10–30 minutes.

Figure 8(a) shows the processing time of stack update. When concurrent processing number is 1, we can update a stack about 210 sec and when it is 3, it takes about 270 sec. Figure 8(b) shows divisions of resource update times when concurrent processing number is 1. TM server processing takes 90 sec in orchestration and other times are OpenStack resources update times (VM creation take much time of OpenStack processing). Proposed TM server manages transactions of updating virtual resources and judges roll back or roll forward for each OpenStack API processing. We think an overhead of TM server differential checks and transaction management is sufficient low. Note that actual stack update may need volume deletion and take 10–30 minutes because Cinder overwrites all data with zero for volume deletion.

Related works

Like OpenStack, OpenNebula [12], Ecalyptus [7] and CloudStack [4] are open source IaaS software. OpenNebula is a virtual infrastructure manager of IaaS building. OpenNebula manages VM, storage, network of company and virtualizes system resources to provide Cloud services. Eucalyptus characteristic is an interoperability of Amazon EC2, and Xen, KVM or many hypervisors can be used on Eucalyptus. Our group also contributes to developments of OpenStack itself. Functions of load balancer and some bug fixes of OpenStack are our group contributions.

Amazon CloudFormation [6] and OpenStack Heat [5] are two major template deployment technologies on IaaS platform. Our work has resolved 4 problems of these technologies. The paper [13] is a work of OpenStack federation using Heat and users need to describe a Heat template first. Because our work can extract templates from existing environments and reflects actual environment changes to stack information, users can replicate or federate virtual resource environments more easily. The paper [14] is a work to construct multitier cloud-based services. Our work provides a mechanism of template sharing in multi-tier contracts, so that end users can build virtual resource environments using templates of upper tier System Integrators. The paper [15] is a work to realize a transitional implementation of meta cloud (cloud abstraction layer) for solving cloud vendor lock-in problems. Our work also targets Cloud migration or transition and template extraction or template sharing function support it. Our work also faces transaction managements of resource deployment.

RightScale is a product for cloud service management which enables automatic operations of system monitoring, alert, auto scaling. ServerTemplates of RightScale [16] is an abstract template approach. When a user deploys a template to a Cloud such as Amazon Web Services or RackSpace Cloud Servers, RightScale sets a configuration adapting to each component of deployed Cloud. This concept is similar to our template extraction to enable easy Cloud migration or replication. However, RightScale ServerTemplates transaction managements or reflections of environment change depend on each Cloud and remain unresolved. Our work resolves them by the TM server. Amazon OpsWorks [17] is also a technology for cloud service management such as automatic scaling, scheduling, monitoring and deployment. Amazon recommends using OpsWorks in high layer management and using CloudFormation for low layer deployment. Support range of OpsWorks is limited in application oriented Amazon Web Services resources and support range of CloudFormation is much larger. Our work scope is low layer deployment and we will use current management technologies such as RightScale for high layer management.

The paper [18] is a research of dynamic resource allocation on OpenStack. As same as [18], our work is also a resource deployment technology on OpenStack but our work targets to resolve problems of plural resources deployment like stack operation transactions. There are some works of resource arrangement on hosting services to use physical server resources effectively [19]. We have

Table 2 Each server specification and usage

Hardware	Physical or VM	Name	Main usage	CPU		RAM (GB)	HDD	NIC
				Model name	Core		Logical (GB)	
HP ProLiant BL460c G6	Physical	KVM host		Quad-Core Intel Xeon 2533 MHz x 2	8	48	300	4
	VM	OpenStack API server	OpenStack stateless process such as API		assign: 4	assign: 8	assign: 60	
	VM	Template management server	Proposed template management server		assign: 4	assign: 8	assign: 60	
HP ProLiant BL460c G6	Physical	KVM host		Quad-Core Intel Xeon 2533 MHz x 2	8	48	300	4
	VM	Glance application server	Received requests related to glance		assign: 8	assign: 32	assign: 150	
HP ProLiant BL460c G1	Physical	DB (OpenStack & TM)	OpenStack and TM DB	Quad-Core Intel Xeon 1600 MHz x 2	8	24	72	4
HP ProLiant BL460c G1	Physical	OpenStack-Network	Used for OpenStack logical network resources	Quad-Core Intel Xeon 1600 MHz x 2	8	18	72	6
HP ProLiant BL460c G1	Physical	OpenStack-Volume	Used for OpenStack logical volume resources	Quad-Core Intel Xeon 1600 MHz x 2	8	18	72	6
HP ProLiant BL460c G1	Physical	OpenStack-Hypervisor	Used for OpenStack VM resources	Quad-Core Intel Xeon 1600 MHz x 2	8	24	72	4
IBM HS21	Physical	DMZ-Load Balancer	Load Balancer for internal access	Xeon E5160 3.0GHz x 1	2	2	72	1
IBM HS21	Physical	Internal Load Balancer	Load balancer for internal access	Xeon E5160 3.0GHz x 1	2	2	72	1
IBM HS21	Physical	KVM host		Xeon E5160 3.0GHz x 1	2	2	72	1
	VM	User VM	VM for user terminal		assign: 1	assign: 1	assign: 20	
	VM	Operator VM	VM for operator terminal		assign: 1	assign: 1	assign: 20	
EMC VNX 5300	Physical	iSCSI storage	iSCSI storage for user volume				500	
EMC VNX 5300	Physical	NFS storage	NFS storage for Image				500	

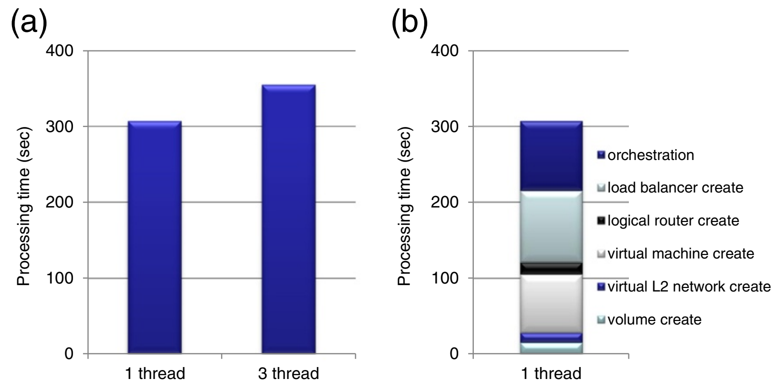


Figure 7 Stack create processing time. (a) Total processing time. **(b)** Divisions of resource creation times.

already developed a scheduler which determines an appropriate physical server for virtual resource deployment [20]. Because a stack has many virtual resources and need effective allocations to enhance total system performances, we will combine [20] and this work in the future.

Conclusion

In this paper, we proposed the template management technology for end users or System Integrators to build virtual resources environments on OpenStack. To resolve existing technologies problems, we designed the TM server which had a mechanism of roll back/roll forward in case of abnormal failure of stack operations, shared templates among end users and System Integrators, extracted templates from existing environments and reflected actual environment change to stack information. We implemented the proposed TM server on OpenStack Folsom, confirmed functions feasibilities and measured performances.

It was confirmed that the TM server prevented a half-finished stack because it rolled back all operations in case of abnormal failure of stack create and rolled back or rolled forward each operation in case of abnormal failure of stack update. Template sharing to end users who have contracts to System Integrators or providers can replicate virtual resource environments on new tenants easily. Our server extracted a template from non-stack environment except for shared resources. Users could update stack as expected because our server reflected actual environment change such as VM shut down or other OpenStack API operations to stack information. Moreover, we showed the effective performance of template management. Template extraction took 1–3 sec, stack create took 300 sec and stack update took 210 sec through experiments of sample templates with 10 virtual resources.

In the future, we will modify the TM server for OpenStack new versions. IceHouse or Juno is a new major version of OpenStack and provides new functions to catch up Amazon Web Services. We will also propose our technologies of transaction managements and reflections of actual environment change to stack information to community Heat. Furthermore, we will improve the software quality of the TM server and verify the feasibility of existing OSS / BSS interconnections to provide production carrier IaaS services based on OpenStack.

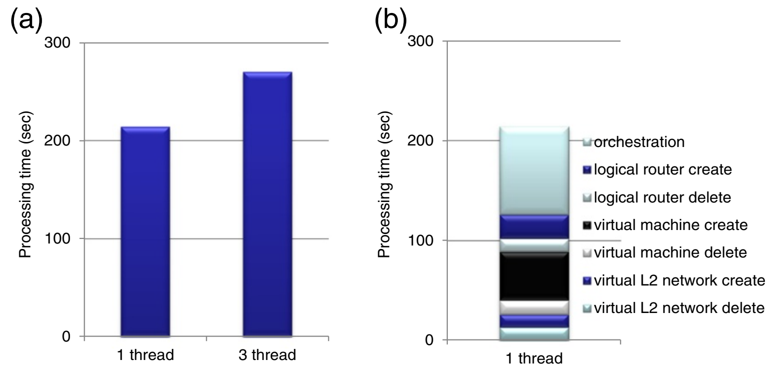


Figure 8 Stack update processing time. (a) Total processing time. **(b)** Divisions of resource update (delete and create) times.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
YY carried out the template management technology studies, participated in the design and implementation of the template management server, evaluated it, surveyed the related technologies and drafted the manuscript. MM, KT, MU participated in the design and implementation of the template management server. All authors have read and approved the final manuscript.

Authors' information
Yoji Yamato received his B. S., M. S. degrees in physics and Ph.D. degrees in general systems studies from University of Tokyo, Japan in 2000, 2002 and 2009, respectively. He joined NTT Corporation, Japan in 2002. Currently he is a RESEARCHER of NTT Software Innovation Center. There, he has been engaged in developmental research of Cloud computing platform, Peer-to-Peer computing, Service Delivery Platform. Dr. Yamato is a member of IEICE, Japan.
Masahito Muroi is a RESEARCHER of NTT Software Innovation Center, NTT Corporation.
Kentaro Tanaka is a RESEARCHER of NTT Software Innovation Center, NTT Corporation.
Mitsutomo Uchimura received his Bachelor's degree in Mechanical Engineering from University of Tokyo, Japan in 1997. He joined NTT Corporation, Japan in 1997. Currently he is a senior research engineer of NTT Software Innovation Center. There, he has been engaged in Cloud SE Project.

Acknowledgements
We thank Hiroshi Sakai and Hikaru Suzuki who are managers of this development.

Received: 24 January 2014 Accepted: 15 May 2014
[redacted]

References

1. Amazon Elastic Compute Cloud web site. http://aws.amazon.com/ec2/. Accessed 24 Jan 2014
2. Rackspace public cloud powered by OpenStack web site. http://www.rackspace.com/cloud/. Accessed 24 Jan 2014
3. OpenStack web site. http://www.openstack.org/. Accessed 24 Jan 2014
4. CloudStack web site. http://CloudStack.apache.org/. Accessed 24 Jan 2014
5. OpenStack Heat web site. https://wiki.openstack.org/wiki/Heat. Accessed 24 Jan 2014
6. Amazon CloudFormation web site. http://aws.amazon.com/cloudformation/. Accessed 24 Jan 2014
7. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D (2009) The Eucalyptus Open-source Cloud-computing System. In: Proceedings of Cluster Computing and the Grid, 2009 (CCGRID '09). 9th IEEE/ACM International Symposium on., p 124
8. Pfaff B, Pettit J, Koponen T, Amidon K, Casado M, Shenker S (2009) Extending Networking into the Virtualization Layer. In: Proceedings of 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII), Oct. 2009
9. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) kvm: the Linux virtual machine monitor. In: Proceedings of OLS '07: The 2007 Ottawa Linux Symposium. p 225
10. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. In: Proceedings of the 19th ACM symposium on Operating Systems Principles (SOSP'03). p 164
11. Libvirt web site. http://libvirt.org/. Accessed 24 Jan 2014
12. Milojicic D, Llorente IM, Montero RS (2011) OpenNebula: a cloud management tool. IEEE Internet Comput 15(2):11–14
13. Castillo L, Angel J, Mallichan K, Al-Hazmi Y (2013) OpenStack Federation in Experimentation Multi-cloud Testbeds. In: Technical reports of HP Laboratories, HPL-2013-58
14. Bahga A, Madisetti VK (2013) Rapid prototyping of multitier cloud-based services and systems. IEEE Comput 46(11):76–83
15. Satzger B, Hummer W, Inzinger C, Leitner P, Dustdar S (2013) Winds of change: from vendor lock-in to the meta cloud. IEEE Internet Comput 17(1):69–73
16. RightScale ServerTemplates web site. http://www.rightscale.com/blog/cloud-management-best-practices/rightscale-servertemplates-explained. Accessed 13 Apr 2014
17. Amazon OpsWorks web site. https://aws.amazon.com/opsworks/. Accessed 13 Apr 2014
18. Wuhib F, Stadler R, Lindgren H (2012) Dynamic resource allocation with management objectives - Implementation for an OpenStack cloud. In: Proceedings of Network and service management, 2012 8th international conference and 2012 workshop on systems virtualiztion management., p 309
19. Liu X, Zhu X, Padala P, Wang Z, Singhal S (2007) Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. In: Proceedings of the IEEE Conference on Decision and Control., p 3792
20. Yamato Y, Yokozeki D, Hirai T, Yuhara M, Muroi M, Tanaka K (2013) Japanese patent application No. 2013–244205

doi:10.1186/s13677-014-0007-3
Cite this article as: Yamato *et al.*: Development of template management technology for easy deployment of virtual resources on OpenStack. *Journal of Cloud Computing: Advances, Systems and Applications* 2014 **3**:7.