JOINT INSTITUTE FOR NUCLEAR RESEARCH

Laboratory of Information Technologies

FINAL REPORT ON THE SUMMER STUDENT PROGRAM

Improvements in cloud bursting implementation for clouds integration

Supervisors:
Nikolay Kutovskiy
Nikita Balashov

Student:
Vadim Petrunin, Russia
Saint Petersburg State University

Participation period:
July 01 - August 26

Dubna, 2017

Abstract

Currently, one of the most important directions of cloud technologies in the scientific field is to develop new methods for integrating cloud systems and to improve existing methods and tools. Integration of the clouds - one of the solutions to the problem of lack of own resources of organizations, including scientific organizations. This document describes the main concepts one of the methods of integration of the clouds - Cloud Bursting and also describes the details on the modification of integration driver in cloud-based systems, developed by a team of JINR cloud service.

Introduction

Many research organizations have their own private clouds, however, due to the rapid growth of computing needs, there are situations when the private cloud services have exhausted their resources and are not able to handle peak computing loads. In case when such loads are only temporary, purchasing new hardware to increase cloud resources may not be the best solution, because most of the time new resources will be idle, and the cost of scaling is very high. Many organizations solve this problem by integrating their private cloud with other clouds, including the commercial ones, it might be a lot more profitable than the self-expansion of the cloud system. There are several methods of integration of cloud services, the most common of which are Federation [1] and Cloud Bursting, the last of which is described in this document in more detail. This paper provides a description of the cloud service used in Joint Institute for Nuclear Research (JINR) and based on OpenNebula [2] cloud platform, which implements Infrastructure as a Service (IaaS) service model as well as a work done during Summer Student Practice 2017 on a driver improvements for clouds integration following cloud bursting approach.

OpenNebula

OpenNebula is a cloud computing platform for managing heterogeneous distributed data center infrastructures. The OpenNebula platform manages a data center's virtual infrastructure to build private, public and hybrid implementations of infrastructure as a service. OpenNebula is free and open-source software, subject to the requirements of the Apache License version 2 [3].

Figure 1. Logo cloud service OpenNebula [2]

OpenNebula orchestrates storage, network, virtualization, monitoring, and security technologies to deploy multi-tier services (e.g. compute clusters) as virtual machines on distributed infrastructures, combining both data center resources and remote cloud resources, according to allocation policies.

The toolkit includes features for integration, management, scalability, security and accounting. It also claims standardization, interoperability and portability, providing cloud users and administrators with a choice of several cloud interfaces (Amazon EC2 Query [4], OGF Open Cloud Computing Interface [5] and vCloud [6]) and hypervisors (Xen [7], KVM [8] and VMware [9]), and can accommodate multiple hardware and software combinations in a data center [10].

Methods of integration of cloud systems

There are two most common methods of integrating IaaS cloud services - Federation and Cloud Bursting. Federation is centralized, it has a single point of entry, a single database user, a single administrator. The domain of this technology becomes clear from its description - technology is suitable for geographically remote branches of a single organization, and cannot be applied in the case of not related organizations due to centralization. Cloud Bursting doesn’t have this disadvantage and can be used for various organizations.

Cloud bursting is a model in which the local resources of a Private Cloud are combined with resources from remote Cloud providers. The remote provider can be a commercial Cloud service, such as Amazon EC2 or Microsoft Azure. Such support for cloud bursting enables highly scalable hosting environments.

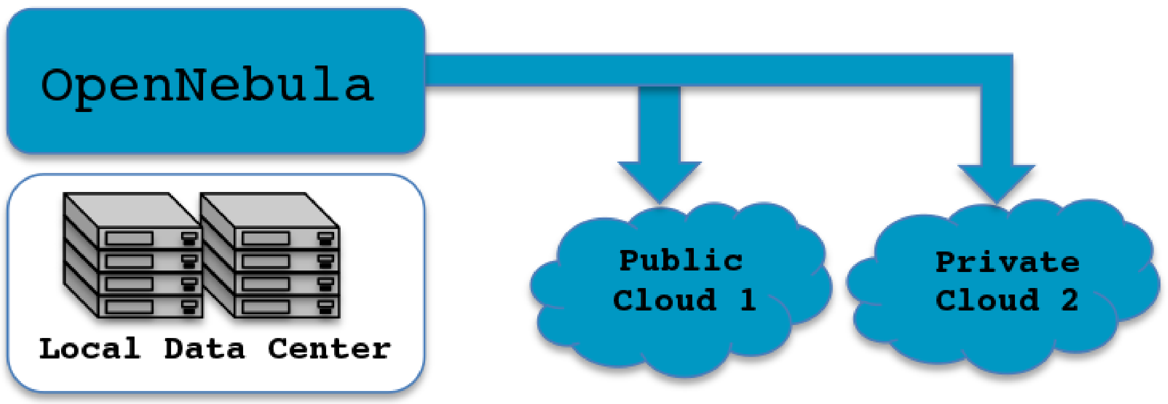

Figure 2. OpenNebula integration scheme [2]

OpenNebula's approach to cloud bursting is based on the transparency to both end users and cloud administrators to use and maintain the cloud bursting functionality. The transparency to cloud administrators comes from the fact that a AWS EC2 region or an Azure location is modeled as any other host (albeit of potentially a much bigger capacity), so the scheduler can place VMs in the external cloud as it will do in any other local host.

On the other hand, the transparency to end users is offered through the hybrid template functionality: the same VM template in OpenNebula can describe the VM if it is deployed locally and also if it gets deployed in EC2 or Azure. Therefore users just have to instantiate the template, and OpenNebula will transparently choose if that is executed locally or remotely [2].

OpenNebula natively supports cloud bursting, however, it is only implemented for a range of commercial cloud services (Amazon EC2, IBM SoftLayer [11] and Microsoft Azure [12]). Since OpenNebula is open source it's possible to expand its cloud bursting technology. The team working on cloud services at JINR developed a driver (a set of scripts for managing remote resources) using this technology. It joins the resources of their private cloud with the resources from some organizations of JINR member States to solve common research problems [13].

Cloud Bursting driver

Developed by the JINR cloud team, the driver for integration with third-party cloud services built on the platform OpenNebula, uses a combination of interfaces OCCI (or rather, its implementation in the ruby language — rOCCI [14]) and OpenNebula XML-RPC [15]. The use of two different interfaces in this implementation is due to several reasons: the interface OCCI is supported not only by the OpenNebula platform but also by other platforms (e.g., OpenStack [16]), while XML-RPC is OpenNebula specific and cannot be used for integration with other platforms. On the other hand, the OCCI interface doesn't provide functionality for monitoring the actual use of resources. Thus, the interface OCCI is used for resource management, and XML-RPC is used as an interim solution for monitoring implementation.
A typical diagram of the driver shown in Figure 3 and can be described as below. rOCCI-server converts XML-RPC requests to the OCCI requests and passes them through a proxy web server (Apache or Nginx with Passenger). To use a proxy web server it’s highly desirable to use a secure connection (https) as the XML-RPC Protocol is based on XML and it lacks an encryption. OpenNebula and rOCCI-server can be installed on a single server or on different servers depending on the load and security requirements. It needs also to create a user account in the cloud service provider on behalf of which virtual machines (VMs) will be run in an external cloud. A cloud service provider has to adjust a quota on the resources available for cloud customer. Added external cloud looks like a host with the number of cores and amount of RAM equal to the quotas set by remote provider. The administrator of the local cloud can add such host to the one of the existing clusters or create a new one [13].

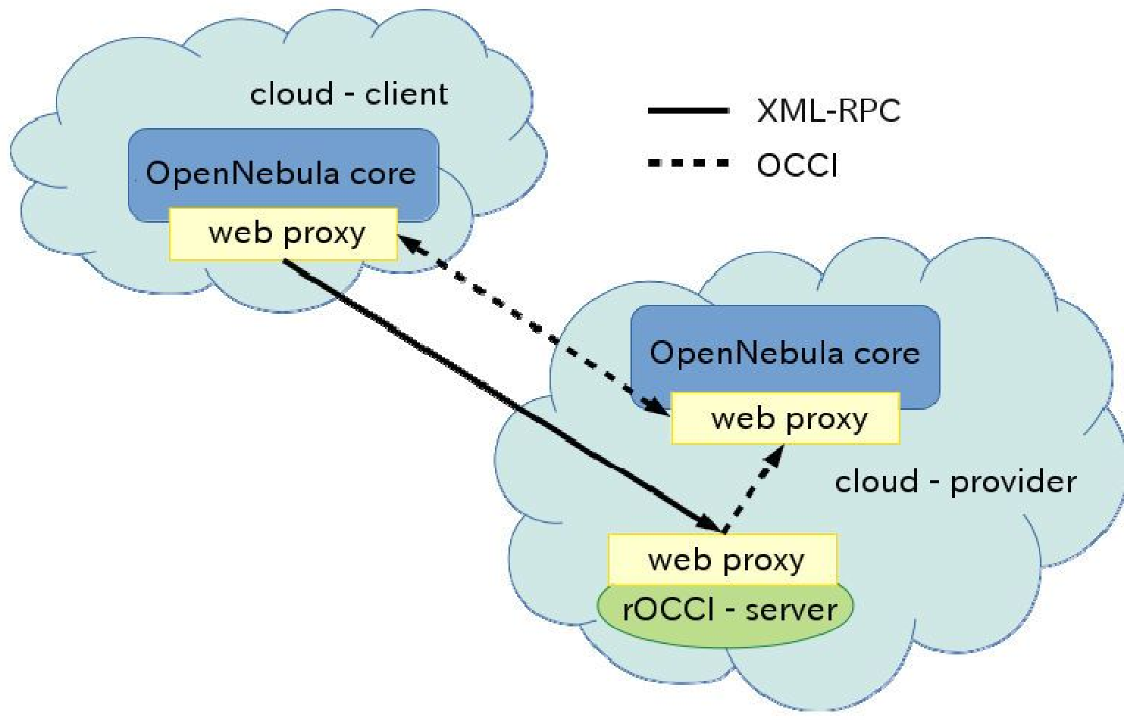

Figure 3. Scheme of the cloud bursting driver [13]

Developed driver integrating cloud systems expands the scope of OpenNebula cloud bursting, adding the ability to integrate with other clouds based on OpenNebula as well as OpenStack software. Development and maintenance of the driver are still ongoing.

Current version of the driver implements the minimum required functionality but it's certain operations is not a part of the driver's specification (in particular, setting the correct status of the VM).

The purpose of the project

The aim of this project is to improve driver for cloud bursting integration, developed by a team of JINR cloud service. The expected improvements include handling of exceptional situations during configuration as well as the monitoring and interaction of clouds involved in integration.

The most important task of this work is the completion of the synchronization of VM statuses in accordance with the current specification of the OpenNebula drivers.

Description of the process of creating VM in cloud bursting

For a further understanding of this document one needs to describe a process of VM creation in the cloud integrated with another one following cloud bursting model. It should be noted that VM is always controlled by the user of the local cloud (further such cloud will be referred as main one). If necessary, create a VM, making sure to pre-customize templates and images for each of the clouds participating in VM deployment. Initially the VM is initialized on the main cloud. When VM is deployed it gets a *local id* (the id assigned to the VM by the main cloud) and *remote id* (the id that VM gets on a remote cloud). Then VM can be monitored from the first cloud despite the fact that the second cloud doesn't exist yet.
After receiving the *remote id* on the second cloud a VM deployment starts. In case of success VM get running on the remote cloud. State of this VM can be tracked from the main cloud as well as various management operations on it can be performed within OpenNebula specification.

Used tools

To update the cloud bursting driver for the integration of private cloud platforms, two testbeds with pre-installed with SL 6.5 x64 [17] and OpenNebula 4.12 has been used, each of which consisted of three nodes:
1) Front-end node (FN);
2) Two working node (cloud/cluster node CN).
Each of CNs was a OpenVZ-enabled KVM based VM (i.e. KVM VM with OpenVZ support inside).
The driver described earlier in this document was maintained on the main cloud (let's call it a first one). The driver's code is written in the programming language - Ruby.

The main problems of the current version of the driver and their solution

As a first step a revision of the driver for clouds integration was performed and it revealed issues what can be divided into three main categories:

1. Handling exceptions associated with a driver configuration,
2. Handling exceptions related to the monitoring and interaction,
3. Code refactoring.

Handling exceptions associated with a driver configuration

1. A virtual machine template on a remote cloud

The virtual machine is deployed from a special template that contains the requirements for the VM. In the case of integration in addition to these requirements there is a special section PUBLIC_CLOUD . Example of a virtual machine template is listed below:

```
CPU="1"
DISK=[
  IMAGE="centOS",
  IMAGE_UNAME="oneadmin" ]
MEMORY="1024"
OS=[
  ARCH="x86_64",
  BOOT="hd" ]
PUBLIC_CLOUD=[
  PROVIDER_TEMPLATE_ID="0",
  TYPE="opennebula" ]
```

In case if the main cloud does not have the required resources for VM deployment then a section PUBLIC_CLOUD that specifies the template on the remote cloud is used. Despite the fact that the format specifies the name of the template and methods for its preparation are described in the examples mentioning the driver documentation [18], there is still some probability of an error during template name specification. A use of the original version of the driver would lead to attempt to deploy VM even in case of incorrect template name. Although an error messages would appear in the log files, a root cause of such error would be hard to guess for the end user who does not know the driver implementation details of the driver. An example of such error messages in the log file are listed below.

```
Fri Jul 21 17:03:09 2017 [Z0][ReM][D]: Req:4096 UID:0 VirtualMachineDeploy invoked
, 143, 1, false, -1
Fri Jul 21 17:03:09 2017 [Z0][DiM][D]: Deploying VM 143
Fri Jul 21 17:03:09 2017 [Z0][ReM][D]: Req:4096 UID:0 VirtualMachineDeploy result
SUCCESS, 143
Fri Jul 21 17:03:09 2017 [Z0][VMM][D]: Message received: LOG I 143 Successfully
execute network driver operation: pre.
Fri Jul 21 17:03:12 2017 [Z0][ReM][D]: Req:2112 UID:0 VirtualMachineInfo invoked ,
143
Fri Jul 21 17:03:12 2017 [Z0][ReM][D]: Req:2112 UID:0 VirtualMachineInfo result
SUCCESS, "<VM><ID>143</ID><UID..."
Fri Jul 21 17:03:12 2017 [Z0][VMM][D]: Message received: LOG I 143 Command
execution fail: /var/lib/one/remotes/vmm/opennebula/deploy
'/var/lib/one/vms/143/deployment.0' 'cloud_fn2' 143 cloud_fn2
Fri Jul 21 17:03:12 2017 [Z0][VMM][D]: Message received: LOG I 143
/usr/local/lib/ruby/gems/2.2.0/gems/occi-core-4.3.5/lib/occi/core/mixins.rb:20:in `<<':
undefined method `attributes' for nil:NilClass (NoMethodError)
Fri Jul 21 17:03:12 2017 [Z0][VMM][D]: Message received: LOG I 143 from
/var/lib/one/remotes/vmm/opennebula/one_bursting_driver.rb:313:in `deploy'
Fri Jul 21 17:03:12 2017 [Z0][VMM][D]: Message received: LOG I 143 from
/var/lib/one/remotes/vmm/opennebula/deploy:16:in `<main>'
Fri Jul 21 17:03:12 2017 [Z0][VMM][D]: Message received: LOG I 143 ExitCode: 1
Fri Jul 21 17:03:12 2017 [Z0][VMM][D]: Message received: LOG I 143 Failed to execute
virtualization driver operation: deploy.
Fri Jul 21 17:03:12 2017 [Z0][VMM][D]: Message received: DEPLOY FAILURE 143 -
```

To handle such kind of errors the following solution was implemented*:* additional checks for the existence and correctness of the template on a remote cloud were added. These checks result to VM deployment in case of the correct template name and in case of errors a user will receive the message, clearly pointing to the cause of the error.

2. Incorrect owner of the remote template

As described earlier in this document, the technology of cloud bursting assumes an existence of a special dedicated user in the remote cloud on whose behalf VMs are created. If a template in the remote cloud belongs to a different user then such template will be unavailable for a user in local cloud and he won't be aware of a reason.

An output of additional error messages in the log file seems a proper *solution* to the problem described above.

3. Incorrect datastore configuration on the remote cloud

One of the datastore configuration option is the choice of Transfer Mode (TM_MAD attribute), which can take one of three values: *ssh*, *shared*, *qcow2*. In the case of cloud bursting the valid value is only *ssh*, however, if the user specifies a different value, this may lead to negative consequences. Since VM deployment takes place twice in both clouds - local and remote ones (see Description of the process of creating VM in cloud bursting) then wrong parameter for TM_MAD attribute will cause to successful VM deployment on the local cloud and to a failure on the remote one although the status of such VM will correspond to normal termination of operation what leads to discrepancy in VM statuses in local and remote clouds.

The following *solution* was implemented*:* pre-inspection of settings and in case of detection of incorrect parameters, VM deployment won't happen even in the local cloud and thus in the remote one too with printing an appropriate messages in a log file.

Handling exceptions related to monitoring and interaction

1. Monitoring of not yet undeployed VM

Information about a VM deployed on a remote cloud is delivered to the main (local) cloud (and further affect the display of the VM on the primary cloud). Such VM monitoring occurs within certain intervals of time. The results of each monitoring cycle are recorded in several special log files (central OpenNebula and VM-specific ones) for further analysis and to collect statistics. It might happen that during VM monitoring cycle a VM deployed at the local cloud hasn't yet achieved the same status on the remote one thus leading to a discrepancy in monitoring results and causing an appearance of an error message in a log file about void values since a DEPLOY_ID does not exist yet at that stage on the remote cloud. An example of such error message in the log file is listed below.

```
Fri Jul 21 18:48:34 2017 [Z0][ONE][E]: Error parsing host information: syntax error,
unexpected EQUAL, expecting COMMA or CBRACKET at line 13, columns 200:201.
Monitoring information:
HYPERVISOR=opennebula
PUBLIC_CLOUD=YES
PRIORITY=-1
CPUSPEED=1000
HOSTNAME="fn2.localdomain"
TOTALMEMORY=10485760
```

```
TOTALCPU=1000
USEDMEMORY=0
USEDCPU=0.0
VM_POLL=YES
VM=[
  ID=146,
  DEPLOY_ID=,
  POLL="USEDCPU=0.0 NETTX=0 NETRX=0 NAME= USEDMEMORY=0 STATE=a
GUEST_IP=10.0.0.151]
```

Similar error message are mentioned a few times in each of the logs depending on the duration of VM deployment on a remote cloud and the duration of intervals of monitoring cycle adding redundant records to the log file.

An implemented solution is an additional conditions check allowing to avoid getting extra records in the appropriate logs.

2. Restoring the previously stopped VM

In the VM life cycle it might happen a situation that requires stopping a VM by one of the operations (suspend, poweroff, stop, etc.) with the subsequent resume of such virtual machine. Due to changes in specifications OpenNebula, solution implemented in the original version of the driver, no longer supported and VM recovery is not possible.

As a *solution* a revision of the driver in accordance with the updated OpenNebula specification has been made allowing to fully restore the functionality of the recovery VM after stopping.

3. VM statuses synchronization on integrated clouds

The synchronization statuses of VM refers to the compliance status of the same VM that is visible from the remote cloud and the local cloud. There were only 3 different VM statuses visible from the main cloud in the original release driver. However according to the updated specifications it is possible to obtain 6 statuses now. It should be noted that the specification of the OpenNebula cloud bursting model imposes a significant limitation comparing with more than 40 possible statuses each VM can reach in each cloud locally.

A *solution* is to handle additional statuses what has been implemented in the updated version of driver. However the full sync status is not covered by the specification.

Code refactoring

An important part of software development is code refactoring. The driver for clouds integration based on cloud bursting model is no exception and as part of this work a its code refactoring has been performed both the original one and added by the author of that document. Repeating blocks of code were moved into a separate functions, constants added which allowed to reduce the number of magic numbers and strings in the program code. The code was tuned to more correspond to the style of the Ruby programming language. As a result a readability of code was increased without loss of software functionality as well as to ease a further improvements in the driver's code.

An updated code of the driver is available at [18].

Conclusion

Thus, during participation in SSP'17 the author of this project sufficiently improved previously created by the JINR cloud team driver for clouds integration based on the cloud bursting model. A handling of more exceptions as well as of more VM statuses were added into the driver. Apart from that its code was updated and refactored to match a current version of OpenNebula specification.

References

1. OpenNebula Federation (website)
   https://docs.opennebula.org/5.2/advanced_components/data_center_federation/overview.html
2. Cloud service OpenNebula (website)
   https://opennebula.org/
3. Apache License (website)
   https://www.apache.org/licenses/LICENSE-2.0
4. Amazon EC2 (website)
   https://aws.amazon.com/ec2/
5. Open Cloud Computing Interface (website)
   http://occi-wg.org/
6. vCloud (website)
   http://vcloud.vmware.com
7. XEN (website)
   https://www.xenproject.org/

8. KVM (website)
https://www.linux-kvm.org/page/Main_Page
9. VMware (website)
https://www.vmware.com/
10. Wikipedia: OpenNebula (website)
https://en.wikipedia.org/wiki/OpenNebula
11. IBM SoftLayer (website)
http://www.ibm.com/Cloud_Softlayer
12. Microsoft Azure
https://azure.microsoft.com
13. A.V. Baranov et al. Approaches to cloud infrastructures integration // Computer Research and Modeling, ISSN: 2076-7633 (Print), 2077-6853 (Online), 2016, Vol. 8, No. 3, P. 583 – 590 (in Russian)
14. rOCCI (website)
http://occi-wg.org/2012/04/02/rocci-a-ruby-occi-framework/
15. OpenNebula: XML-RPC API (website)
https://docs.opennebula.org/5.2/integration/system_interfaces/api.html
16. OpenStack (website)
https://www.openstack.org
17. Scientific Linux (website)
https://www.scientificlinux.org/
18. GitHub: JINR Cloud bursting driver (website)
https://github.com/JINR-LIT/ONE-cloudbursting-driver