OpenStack Federation in Experimentation Multi-cloud Testbeds

Juan Angel Lorenzo del Castillo, Kate Mallichan, Yahya Al-Hazmi

HP Laboratories
HPL-2013-58

Keyword(s):

Cloud Federation; cloud computing; openstack

Abstract:

There is an increasing number of cloud platforms emerging in both academia and industry. They often federate resources from multiple infrastructures in order to benefit from the unique features that each presents. After introducing the main capabilities and features of OpenStack, this article addresses the integration of OpenStack-based platforms into larger, heterogeneous multi-cloud infrastructures, taking the EU FP7 BonFIRE project as an integration use case. Ultimately, we aim to contribute to the state of the art and provide guidelines to integrators looking to federate Open Stack testbeds into more complex architectures.

External Posting Date: August 21, 2013 [Fulltext] Approved for External Publication
Internal Posting Date: August 21, 2013 [Fulltext]
To be published in UNICO 2013: Workshop, UsiNg and building CLOud Testbeds

OpenStack Federation in Experimentation Multi-cloud Testbeds

Juan Ángel Lorenzo del Castillo
Hewlett-Packard Laboratories, Bristol
Email: juan.lorenzo-del-castillo@hp.com

Kate Mallichan
Hewlett-Packard Laboratories, Bristol
Email: kate.mallichan@hp.com

Yahya Al-Hazmi
Technical University Berlin, Berlin
Email: yahya.al-hazmi@tu-berlin.de

***Abstract*—There is an increasing number of cloud platforms emerging in both academia and industry. They often federate resources from multiple infrastructures in order to benefit from the unique features that each presents. After introducing the main capabilities and features of OpenStack, this article addresses the integration of OpenStack-based platforms into larger, heterogeneous multi-cloud infrastructures, taking the EU FP7 BonFIRE project as an integration use case. Ultimately, we aim to contribute to the state of the art and provide guidelines to integrators looking to federate Open Stack testbeds into more complex architectures.**

I. Introduction

Resource federation is recognised as a promising mechanism aimed at interconnection of heterogeneous resources across several independent infrastructures. In providing a larger-scale and higher performance infrastructure, federation enables on-demand provisioning of complex services. Due to the heterogeneity of such infrastructures in terms of provided features, APIs, tools, etc., there is a need for federation mechanisms so that resources can interoperate in a standardised manner. The federated architecture has to be designed to include these mechanisms and fulfil all collaboration requirements. Several architectural approaches are possible, each has its pros and cons. Among those introduced in [1] are:

- **A central management system:** Separate infrastructures replace their management systems with a common one. This allows for easy deployment, management, and maintenance. However, it implies adoption of new software and potential compatibility issues.
- **Homogeneous federation running the same management system on all infrastructures:** Each infrastructure manages its resources independent from a central point, but each management system must be replaced and users must adopt the new tools.
- **Central front-end point:** Infrastructures and their tools are listed on a portal. Each infrastructure keeps its management system. This is the easiest and cheapest approach, although there is no real federation and users still need to use multiple accounts, tools, etc.
- **Common APIs on top of distinct independent management systems:** Site resources are managed at federation level through standardised interfaces. This requires sub-interfaces for the individual management mechanisms such as provisioning, control, etc. The interface definition is not a trivial task, but allows for an easy integration of further infrastructures.

The last approach might be the most suitable approach for large and dynamic federations with heterogeneous infrastructures. However, for small and stable federations, as those studied in this article, the centralised approach works well. This article focuses on two key aspects of the federation: observability and controllability.

Observability is an important topic for monitoring (watching for system changes) and metering (watching system usage, typically for billing purposes). Most commercial clouds provide that information only at application level. Apart from further tools that users install or implement on their VMs, little or no information is available about the VMs or the underlying infrastructure. However, in the absence of those restrictions, much other information can be exposed. Measurements can be taken from the physical infrastructure, such as the degree of machine occupancy or the memory and CPU available on a given host. Information at VM-level might relate to status and performance such as network performance or the disk read/write rate. At application level, custom metrics can be gathered by users according to the applications in use such as the number of open database connections or the load in a webserver. Monitoring at cloud management level is usually provided by a notification service that collects events relating to instance, network, or storage resources, and additional information such as the current VM allocation policy. Other sources of monitoring information include log files, status pages or accounting services.

In the case of centrally-managed, horizontally-federated infrastructures, most monitoring information is gathered from each member and made available through a unique access point. In order to federate a cloud infrastructure into a larger one, the former needs to provide mechanisms to be observed and controlled. Observed, meaning that the main infrastructure receives timely *meaningful* state notifications. And controlled, so that it can receive and interpret commands like the rest of the federated sites. Note that this implies an agreement in the degree of homogeneity about the information sent and received from any site.

It should be noted that some of the monitoring capabilities are agnostic of the Cloud platform used. Nagios [2] or Zabbix [3] can be used to monitor hosts and support any cloud platform thus will not pose a further concern regarding federation.

We have chosen OpenStack [4] to study its federation capabilities in multi-cloud, heterogeneous environments. OpenStack is the Open Source Cloud Computing platform most widely adopted in Industry. Meant to be simple to implement,

this Infrastructure as a Service (IaaS) is open and massively scalable. Examples of companies that use OpenStack are Hewlett-Packard, Rackspace and Red Hat, to cite some.

The rest of this article is structured as follows: Section II introduces the observability and controllability features in OpenStack. Section III presents the multi-cloud, federated infrastructure BonFIRE. Section IV provides an example of OpenStack integration with BonFIRE and discusses the current limitations of OpenStack in terms of federation capabilities, suggesting solutions for them. Finally, the main conclusions and future work are presented in Section V.

II. OpenStack

OpenStack comprises several modules, the core ones being Nova (computing), Glance (VM repository), Neutron (networking) and Swift and Cinder (storage). As a virtual infrastructure platform, OpenStack manages a group of interdependent *virtual resources* such as virtual machines, virtual networks and volumes, and defines the several levels of virtual resource granularity. *Resources* exist in most other cloud platforms, and are comprised of computes, networks and storage. The term *Composite* is not defined by OpenStack itself but, in this article, when referring to OpenStack, we will use this term to describe an interdependent group and configuration of resources.

OpenStack offers several mechanisms to interact with it, from a REST and Python APIs to a complete notification service. In addition, a separate project provides an implementation of the Open Cloud Computing Interface (OCCI) [5]. These features make it easy for external systems to control OpenStack and obtain system status. The next sections elaborate on this.

A. Observing events in OpenStack

OpenStack modules use notifications to intercommunicate and provide information about their current status. These are in JSON format and managed by an AMQP (Advanced Message Queuing Protocol) broker. From a monitoring point of view, events on a queue are more convenient than a set of log files on a host. Events are pre-structured and queues can be accessed programmatically. The queue assures delivery and consistency of events. This is extremely suited to services that require high reliability such as billing. Examples of events are *resource created/deleted*, *IP assigned/detached* or *security group/s changed*. A complete list of events for system usage is available in [6] and [7]. A list of exchanges and queues created by OpenStack is detailed in [8]. Details of the events' payload can be found in [9].

Currently there exist several projects and tools to gather, access and utilise the events generated by OpenStack. At the moment of writing this article, the most relevant are:

- **Ceilometer [7]:** Collects metering data and offers a single point of contact for billing systems implemented on top of it. Data are offered through a REST API to facilitate access from external resources.
- **Kombu [10]:** A messaging framework for Python that encapsulates the AMQP API to facilitate its access and provide a high-level to interact with the AMQP.
- **Yagi [11]:** Highly reliable, it is an external publisher that gives support to AtomHopper, a system that turns notifications into Atom feeds that may be consumed by other systems like billing.
- **Marconi [12]:** A messaging and notifications queue system. Still in development stage, it provides an open alternative to SQS (producer-consumer) and SNS (pub-sub) in OpenStack. It will provide a REST API.
- **StackTach [13]:** A debugging tool to collect and report notifications sent by Nova. A daemon consumes them from the AMQP queues and makes them available through a SQL database. Notifications can be displayed on a web UI or accessed by CLI tools.

Given the amount of similar initiatives, several efforts are being made to unify them into a single system. The solution that is gaining weight among the OpenStack community is Kombu. Listing 1 shows a Python code snippet that demonstrates how events may be easily collected from resources and used. After defining a connection to a given queue (the `notifications.info` queue from Nova, in this example), the code goes into an infinite loop consuming the received events and printing them on the screen. Kombu facilitates the development of tools to adapt and integrate event management from different cloud sites to a single point.

Listing 1 Capturing OpenStack events with Kombu

```
from kombu import Connection, Exchange, Queue
from pprint import pprint

nova_x = Exchange('nova', type='topic', durable=False)
info_q = Queue('notifications.info', exchange=nova_x, durable=False,
routing_key='notifications.info')

def process_msg(body, message):
    print '='*80
    pprint(body)
    message.ack()

with Connection('amqp://guest:guest@openstackserverip//') as conn:
    with conn.Consumer(info_q, callbacks=[process_msg]):
      while True:
        try:
          conn.drain_events()
        except KeyboardInterrupt:
          break
```

B. Controllability in OpenStack

OpenStack defines the term *Server* to refer to a virtual machine resource. We will use the terms "compute", "compute resource", "VM" and "server" synonymously in the rest of this article. OpenStack exposes 2 types of APIs: REST based and a Python CLI [14]. Both can be used to perform similar tasks, though the Python language bindings are particularly useful for programmatic control of OpenStack. The following paragraph describe the different levels of controllability in OpenStack.

1) ***Resource-level controllability****:* OpenStack allows create, fetch, update and delete operations at per-resource level. Update operations provide features such as setting resource metadata, changing the power state of a VM, attaching/detaching volumes and networks from a VM. For a complete list refer to the OpenStack API reference [15].

*2) **Composite-level controllability**:* The OpenStack Heat Project [16] provides a service to allow creation of composite cloud applications by submitting a descriptive template file.

*3) **Infrastructure-level controllability**:* OpenStack's APIs allow not only interaction with virtual resources, but also provide a limited amount of interaction with the underlying hypervisors and physical hosts [15].

*4) **Data persistence**:* OpenStack provides a distributed object storage platform that is accessible from applications, for persistent data storage. An additional API for block storage allows block devices to be attached to VMs.

*5) **Application-level controllability**:* Compute resources can be pre-configured with application-specific parameters, like SSH keys. This is achieved by injecting metadata supplied through the API at creation time.

*6) **Elasticity**:* OpenStack does not provide automatic scaling, so any such functionality must be implemented in application code. Vertical elasticity of compute resources is manually allowed by an API.

III. BonFIRE

BonFIRE [17] offers a federated facility that supports large-scale testing of applications, services and systems over multiple, geographically distributed, heterogeneous cloud and network testbeds. Users can control and monitor the execution of their experiments to a degree not found in traditional cloud facilities. Architecturally, BonFIRE uses a central broker that exposes a common API to interact with private and public clouds. BonFIRE defines two aspects of granularity: Resources and Experiments. *Resources* exist in most other cloud platforms and are comprised of computes, networks and storage. Experiments is a term particular to BonFIRE. This can be thought of as a container for a group of resources, which may be interdependent and are a typically a custom configuration, according to the user's requirements. The *experiment* concept is managed by the BonFIRE internal components. The testbed infrastructures themselves have no knowledge of it.

A. Observability in BonFIRE

BonFIRE defines several types of observability with multiple entry points.

*1) **Experiment Message Queue**:* The Experiment Message Queue is a BonFIRE service that consumes messages specific to each experiment. Those messages record both experiment and resource status such as created, deleted, updated, as well as state changes (active, suspend, etc.). Messages are tagged with a unique id to identify the experiment and posted to the queue by both BonFIRE's internal components and the testbeds. The messages are in JSON format, as shown in Listing 2:

As well as being accessible by BonFIRE internal services, software running on user VMs are able to connect to this queue. With access granted to all VMs, the queue can also be used as a messaging service between user VMs if desired.

BonFIRE has an internal accounting service which subscribes to experiment message queue and uses the timestamped messages to store information in a database that could support accounting and billing systems on a per experiment or per-user basis. It exposes a REST interface to present usage reports, though this is currently not available to users.

Listing 2 An example of BonFIRE network create message.

```
uk-hplabs.network.create
{
    "timestamp":"1370946265",
    "groupId":"usergroup",
    "objectData":{
        "experimentId":"3013",
        "address":"10.25.2.48",
        "name":"netwkHP",
        "size":"6"
    },
    "source":"uk-hplabs",
    "objectId":"subnet-3-19",
    "userId":"user",
    "eventType":"create",
    "objectType":"network"
}
```

*2) **Infrastructure metrics**:* Provides information about the underlying hypervisors system capabilities and network performance. Typical examples are the load on the hypervisor's CPU, the number of VMs on the same CPU, or network QoS metrics such as bandwidth utilisation, pack loss rate or delay. These allow experiments to measure, understand and, ultimately, account for uncontrollable outside influences that affect the execution of an experiment.

*3) **Application-level monitoring**:* The application metrics will be defined by the experimenter according to the specific software applications being used by the experiment.

*4) **VM-level monitoring**:* VM metrics provide system information about the status of the VM regarding CPU, memory, disk space, etc.

*5) **API to access monitoring information**:* Graphical tool (based on Zabbix GUI) to display metric values and metric value graphs in real time while an experiment is executing. The Network resources allow any VM or site to connect to and are used to manage and monitor the VMs.

B. Controllability in BonFIRE

BonFIRE exposes 3 APIs to the user: A REST-based one at both resource and experiment level, and a set of command line tools which can be used to manage bonfire experiment(s) from a user application. The REST interface is an implementation of the Open Cloud Computing Interface (OCCI) [5].

*1) **Resource-level controllability**:* A user can perform basic create, delete and update operations at a per-resource level. Update applies to computes only. It allows controllability of power state and saving a customised compute volume to a new golden image so that further computes may be booted from it.

*2) **Experiment-level controllability**:* From an empty experiment, a user may construct their desired configuration at either resource level granularity, or more powerfully by submitting a document-based description of it, referred to as an *experiment descriptor*. BonFIRE parses it, works out dependencies, schedules appropriately and then instantiates the collection of resources. This is similar to the functionality provided by OpenStack's Heat project.

*3) **Data persistence**:* BonFIRE allows for block storage creation both in and out of the context of an experiment. Such storage can be made persistent, so that it can be attached and detached from VMs, whilst preserving all data that is written to it. A customised VM image can also be saved as a new golden image to be reused to boot further VMs.

*4) **Application-level controllability**:* Compute resources can be configured with application specific contextualisation, which is configuration information provided to the VM and made available to applications when the machine is started.

*5) **Elasticity**:* Elasticity can be controlled by either user application logic, or by BonFIRE itself at sites that allow automatic vertical and/or horizontal elasticity. Both use the BonFIRE monitoring API to detect thresholds and take action.

*6) **Connectivity to external facilities**:* Thanks to an Amazon EC2 interconnect facility [18].

After evaluating the current capabilities of OpenStack, subsequent sections will summarise their potential to satisfy BonFIRE requirements and propose solutions for identified shortcomings.

IV. Use Case: Federating OpenStack in BonFIRE

This section studies, as a use case, the set of features that a new OpenStack testbed must fulfil in order to be federated in BonFIRE. To do so, issues such as event observability, controllability and API compatibility are considered.

There exist two types of cloud federation: Vertical and Horizontal. Vertical federation allows integration of new infrastructures with the purpose of providing new capabilities. Horizontal federation refers to expanding the capacity of the existing cloud by integrating a new site. When working on a federated heterogeneous infrastructure, the most common scenario is one in which an existing infrastructure needs to increase its offer of resources to their users. In the case of BonFIRE, a central service provides a single access point (REST API, CLI, Web Portal). Still taking into account the heterogeneity and different features of its members, the current objective in BonFIRE is to provide enough capacity to host the experiments carried out on it. Because of these reasons, this section focuses on horizontal federation.

Figure 1 shows the lowest layers in the BonFIRE architecture. A Web Portal and a REST API provided by an Experiment Manager (not shown) will capture the user requisites to create and manage experiments. Requests are then translated into OCCI requests that are sent to the Resource Manager or Broker. The Broker maintains the set of resources in use by each experiment and the lifecycle state of an experiment. It also writes experiment and resource state changes to the BonFIRE Message Queue.

The Enactor is a client to each of the testbed OCCI servers. It transforms the XML payloads of these messages to and from a common OCCI XML schema that is understood by the Broker. The Enactor uses a set of adaptors (labelled *endpoints* in Figure 1) for each type of testbed. Horizontal federation only requires that the new site supports the requests made by the Enactor. In this case, a new adaptor will have to be developed at the Enactor. We have identified the following adaptations

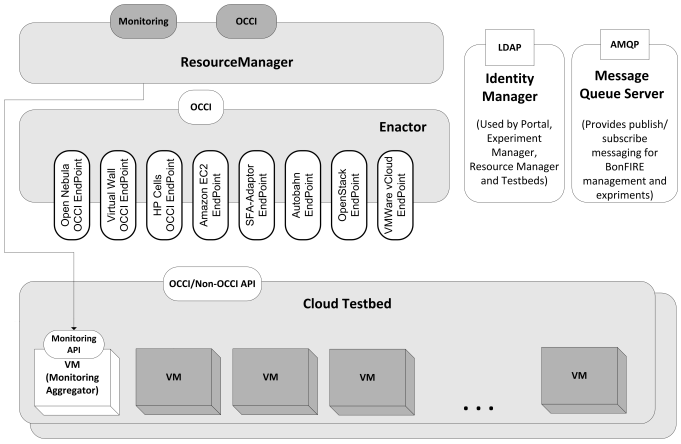

Fig. 1. Bottom layers of the BonFIRE architecture.

that will be required at the OpenStack site in order to meet BonFIRE's requirements:

A. Mapping of BonFIRE tasks to the OpenStack REST APIs

Table 1 shows a mapping of the actions that can be performed through BonFIRE, against the equivalent operations exposed by the OpenStack REST API. Only the HTTP request type and relative URLs are provided. The REST API endpoint is the URL root.

B. Controlling instance placement

OpenStack can be configured to allow host-level controllability of VM placement. This can be done by defining an availability zone in the configuration of each host. Each host should be assigned a unique zone id. This id can then be passed to OpenStack as a parameter in a compute create request, through the REST API; the OpenStack internal scheduler will then place the resource appropriately. For the user to choose and specify a host for a new compute resource, they need to be able to view information about the hosts that are available. Section II-B shows how the OpenStack APIs can be used to list the available hosts, their capacities and current resource usage. To create a compute resource on a specific host, specify the availability_zone parameter in the JSON body of the create request as shown in Listing 3.

Listing 3 Setting the availability zone of a new OpenStack server

```
{
    "server": {
        "flavorRef": "http://openstack.example.com/openstack/flavors/1",
        "imageRef": "http://openstack.example.com/openstack/images/70a599e0-31e7-
49b7-b260-868f441e862b",
        "metadata": {
        "My Server Name": "Apache1"
        },
        "name": "new-server-test",
        "security_groups": [{"name": "sg1"}]
        "availability_zone": "host1"
    }
}
```

C. Retrieving Event information from OpenStack

A set of BonFIRE-defined events are generated by each of the federated sites and sent to the Central Services for monitoring purposes, billing, etc. Table II shows the BonFIRE events

TABLE I. MAPPING OF BONFIRE OPERATIONS TO THE OPENSTACK REST APIS

Category	Operation	OpenStack REST API Call	
		HTTP Type	**URL**
Compute	List all VMs	GET	/servers/detail?image=⟨imageRef⟩&flavor=⟨flavorRef⟩&name=⟨serverName⟩&status=⟨serverStatus⟩&marker=⟨markerID⟩&limit=⟨int⟩&changes-since=⟨dateTime⟩
	Create a new VM	POST	/servers
	Create a new VM with host placement	POST	/servers (specify availablity_zone parameter in the request body[?]http://api.openstack.org/api-ref.html#compute_servers)
	Get description of a VM	GET	/servers/⟨id⟩
	Delete a VM	DELETE	/servers/⟨id⟩
	Update a VM	PUT	/servers/⟨id⟩
	Save a modified VM volume as a new golden image	POST	server/⟨id⟩/⟨action⟩ (action = createImage)
Network	List all Networks	GET	/networks
	Create a new Network	POST	/networks
	Get description of a Network	GET	/networks/⟨netid⟩
	Delete a Network	DELETE	/networks/⟨netid⟩
Storage	List all Storage	GET	/volumes/details (datablocks) GET /v2/images (golden images)
	Create a new Storage	POST	/volumes (datablock)*
	Get description of a Storage	GET	/volumes/⟨volid⟩ (datablock) GET /v2/images/⟨imageid⟩ (golden image)
	Delete a Storage	DELETE	/volumes/⟨volid⟩ (datablock) DELETE /v2/images/⟨imageid⟩

TABLE II. BONFIRE EVENTS AND CORRESPONDANCE TO OPENSTACK

Category	Event		Description
	BonFIRE	OpenStack	
Compute	vm.create	compute.instance.create.start/.end	VM has been created
	vm.delete	compute.instance.delete.start/.end	VM has been deleted
	vm.state.reserved	compute.instance.power_on.start	Resources allocated, but VM not booted yet
	vm.state.active.up	compute.instance.exists	VM in booted and running state
	vm.state.stopping.off	compute.instance.shutdown.start/.end	VM has entered a shutdown state
	vm.state.active.standby	-	VM has entered a standby/suspended state
	vm.state.failed.error	compute.instance.create.error	VM has entered an error state
Network	network.create	network.create	Network created
	network.delete	network.delete	Network deleted
Storage	storage.create	volume.create.start/.end	storage has been created
	storage.delete	volume.delete.start/.end	storage has been deleted
	storage.state.locked	compute.instance.snapshot.start/.end	save-as transfer is in progress to this target volume
	storage.state.ready	compute.instance.snapshot.end	save-as transfer completed. Image ready to use
	storage.state.error	-	save-as transfer or volume creation has failed

and a subset of corresponding OpenStack events. OpenStack generates two events for every activity; one when the activity starts and another when the activity ends. BonFIRE events are only triggered on completion. As seen in the table, most of them match easily to similar events in OpenStack. Since BonFIRE does not require information about time taken for activity completion, it could safely discard those *.start* events and match only the *.end* OpenStack events to the BonFIRE notification model.

For some BonFIRE events, there are no OpenStack equivalents or only partial matches. We suggest some alternatives:

- **vm.state.reserved:** There is no event in OpenStack to denote resource allocation while waiting for a VM to boot. Instead, the event *compute.instance.create.start* can be used to denote the moment at which resources are allocated to start booting a VM in OpenStack.
- **vm.state.active.standby:** There is no event to denote when a VM is suspended. However, one of the possible server states in OpenStack is SUSPENDED, so it should be possible to obtain this state from the API.
- **compute.instance.create.error:** Sent when a VM enters an error state. OpenStack only sends this event if an error occurs when booting. For a running VM, check the event *compute.instance.exists*, which is periodically generated by a cron daemon.
- **storage.state.error:** An alternative for this event would be also to periodically check that the *volume.exists* event is generated.

The information about network events in Neutron (formerly Quantum) is very scarce. According to the Ceilometer code available in GitHub [19], it is possible to capture *network*, *subnet*, *port* and *router* events. In addition, there is an in-progress initiative to make network events available to Neutron administrators [20].

D. Sending events to BonFIRE from the OpenStack site

In order to make OpenStack events usable by BonFIRE, it would be necessary to develop a software component to run at the site alongside the OpenStack deployment. This would consist of 2 parts: one to parse and reformat event data into the BonFIRE format, and a second one to act as a RabbitMQ producer client to post these messages to the queue. Since the OpenStack event retrieval code would likely be implemented in Python, it would make sense to also develop this component in Python, using one of the many client libraries available [21].

E. Security model compatibility

The recommended method for user authentication in OpenStack is by auth-token. BonFIRE does not use this system, so BonFIRE would have to generate this on behalf of the BonFIRE LDAP user and additionally deal with token expiry. If it is assumed that the BonFIRE experiment concept can be equated with the OpenStack tenant concept, we will encounter incompatibilities with this parallel such as the BonFIRE experiment Walltime, which does not apply to the OpenStack tenant concept. An additional task would be how to map BonFIRE group permissions to the OpenStack security model.

These issues are related to the modification of the BonFIRE central services, not to OpenStack itself and are thus beyond the scope of this article. However, we make the suggestion that a single tenant and OpenStack user be used for authentication with the BonFIRE central services, and that BonFIRE user/group information be stored as metadata with each virtual resource. This model is the one used for the integration of HP Cells and avoids the complications of attempting to merge two conflicting/incompatible security models.

F. Suitability and Status of the OpenStack OCCI API

The OpenStack OCCI project is currently under development and lacks basic create/delete functionality for networks. The BonFIRE Enactor adaptor would instead need to interact with the testbed site through the native OpenStack REST API.

V. Conclusion

OpenStack is one of the most promising cloud computing platforms to date, with a user base in academia and industry. This article has presented and discussed OpenStack's capabilities in observability and controllability. We have shown how these features make OpenStack suitable for horizontal federation into larger, heterogeneous infrastructures. Finally, we have studied, as a use case, the work required to integrate OpenStack in BonFIRE, a cloud testbed for research and experimentation which presents more strict requirements than a conventional cloud infrastructure.

For efficient integration, a new testbed must provide the means to be observed and controlled. OpenStack can be observed at VM, application and infrastructure levels. At cloud operating system level, its notification service provides a useful source of information. In fact, we showed how by just providing a subset of the available events to BonFIRE, it was sufficient for a proper federation.

Control of OpenStack turned out to be more challenging. We demonstrated that BonFIRE requests can be mapped to OpenStack with some modifications to some modifications to the OpenStack API or intermediate layers. An OCCI API project for OpenStack is under development, but it still lacks the necessary features to be used as the only point of exchange between OpenStack and the federating infrastructure.

Based on this study, further work will involve the development of a component to translate and transmit the event data from OpenStack to BonFIRE and another one for the BonFIRE central services to interact with the OpenStack REST API.

Acknowledgment

This work was undertaken in the context of the BonFIRE project which is funded by the European Union 7th Framework Programme under grant agreement number 257386. The authors would like to thank all of the BonFIRE development team for their contribution and support.

References

[1] W. Vandenberghe *et al.*, "Architecture for the heterogeneous federation of future internet experimentation facilities," *Future Network and Mobile Summit*, 2013.

[2] *Nagios monitoring tool*, http://www.nagios.org.

[3] *Zabbix: Open source monitoring solution*, http://www.zabbix.com.

[4] *OpenStack*, http://www.openstack.org.

[5] *OCCI - OpenStack*, https://wiki.openstack.org/wiki/Occi.

[6] *Event types and payload data in OpenStack*, https://wiki.openstack.org/wiki/SystemUsageData.

[7] *The Ceilometer Project*, http://docs.openstack.org/developer/ceilometer/measurements.html.

[8] *Queues and Exchanges in OpenStack*, http://ilearnstack.com/2013/04/24/messaging-in-openstack-using-rabbitmq.

[9] *Notification payloads in OpenStack*, https://wiki.openstack.org/wiki/NotificationEventExamples.

[10] *Kombu*, http://kombu.readthedocs.org/en/latest/.

[11] *Yagi*, https://github.com/Cerberus98/yagi.

[12] *Marconi*, https://wiki.openstack.org/wiki/Marconi.

[13] *StackTach*, http://www.sandywalsh.com/2012/10/debugging-openstack-with-stacktach-and.html.

[14] *OpenStack API language bindings*, http://docs.openstack.org/developer/language-bindings.htm.

[15] *OpenStack Complete API Reference*, http://api.openstack.org/api-ref.html.

[16] *OpenStack Orchestration: Heat*, https://wiki.openstack.org/wiki/Heat.

[17] A. C. Hume *et al.*, "Bonfire: a multi-cloud test facility for internet of services experimentation," in *8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, Proceedings*. Ghent University, Department of Information technology, 2012, pp. 1–16.

[18] *BonFIRE's connector to Amazon EC2*, http://doc.bonfire-project.eu/R3.1/reference/amazon-connector.html.

[19] *Notification management with Python in Ceilometer*, https://wiki.openstack.org/wiki/Neutron/APIv2-specification.

[20] *Quantum Notifications*, https://wiki.openstack.org/wiki/QuantumNotifications.

[21] *RabbitMQ Python Client Libraries*, http://www.rabbitmq.com/devtools.html#python-dev.