2015-03

Using OpenStack for an Open Cloud eXchange (OCX)


Boston University
Using OpenStack for an Open Cloud eXchange(OCX)

P. Desnoyers*, J. Hennessey†, B. Holden‡, O. Krieger‡, L. Rudolph§, A. Young‡,
*Northeastern University. pjd@ccs.neu.edu
†Boston University. {henn,okrieg}@bu.edu
‡Red Hat. {bholden,adam}@redhat.com
§Two Sigma. larry.rudolph@twosigma.com

Abstract—We are developing a new public cloud, the Massachusetts Open Cloud (MOC) based on the model of an Open Cloud eXchange (OCX). We discuss in this paper the vision of an OCX and how we intend to realize it using the OpenStack open-source cloud platform in the MOC. A limited form of an OCX can be achieved today by layering new services on top of OpenStack. We have performed an analysis of OpenStack to determine the changes needed in order to fully realize the OCX model. We describe these proposed changes, which although significant and requiring broad community involvement will provide functionality of value to both existing single-provider clouds as well as future multi-provider ones.

II. BACKGROUND

While the implications of the OCX model are profound, in many ways it is a natural progression and can be developed by modifying and extending existing commercial and academic single-provider cloud systems.

A. Cloud Software and OpenStack

A modern cloud is a complex distributed system with many diverse hardware and software components, a huge number of parameters and complex configuration settings that must all be monitored and maintained in the presence of failures. To make this practical, scalable clouds are developed using a Service Oriented Architecture (SOA) [19], where each service provides well-defined functionality, exposes its API, and can be maintained, scaled, and evolved independently.

A good example of this architecture is the OpenStack [17] cloud middleware. Services supported by OpenStack include: Neutron for networking services, Glance for image services, Cinder for disk volumes, Nova for compute, and Keystone for authentication and authorization. Each service is itself a distributed system, but has a single well defined API that is used by other services and by customers of the cloud.

An SOA is a necessary prerequisite of the OCX model, and our implementation of an OCX is based on modifying and extending OpenStack. Moreover, OpenStack is a good starting point due to its rich open source community, and the plugin architecture where uniform APIs support a wide variety of underlying equipment and mechanisms. As an example, the networking component has 17 different plugins for different networking infrastructure technologies.

With OpenStack, for each service, a single API endpoint (identified by a URI) acts as a broker and scheduler to distribute requests to the multiple underlying components of the service. In the case of a diversity of implementation, constraints provided by the user may help determine the implementation to use.

B. Cloud Federation

Research [5, 7, 15] and industry [16, 18] cloud federation projects allow a client to take advantage of multiple cloud platforms, either to augment private clouds with as-needed capacity from a public provider, or to allow clients to concurrently exploit multiple public clouds.
The OCX model can be viewed as a natural outgrowth of the SOA of today’s cloud software, where different providers may stand up different services, and it is the client that selects which services are used. It can also be viewed as an outgrowth of cloud federation, where the federation is finer grained and at the basis of individual services rather than entire clouds.

Just as in today’s single-provider clouds, we expect most clients to interact with the cloud through higher level intermediaries (e.g., Heroku, Cloud Foundry, Rightscale, Engine Yard, OpenShift, Hadoop, Spark, StarCluster). There are however two key differences. First, in an OCX an intermediary can be a first class participant in the cloud, advertised through the exchange in the same fashion as core Infrastructure as a Service (IaaS) services and fully integrated into the user interfaces, APIs, and billing model. This is in contrast to existing single provider clouds, where there is a fundamental difference between the services of the provider and the marketplace of higher-level third party services/intermediaries.

Second, the task of selecting the right services from among many potential offerings in an OCX may be difficult for consumers, making it important to have intermediaries that provide clients with a simple model that meets their needs. For example, an intermediary could implement an IaaS service as a broker on top of multiple underlying IaaS providers. As another example, a Spark-based [21] intermediary for big-data could exploit domain-specific information to better manage the use of IaaS services.

The exchange model also enables a richer diversity of lower level services. A service with a different interface can be exposed through the exchange, and be made available to any client that knows how to use it. For example, platforms such as GENI [11] or large HPC clusters can be exposed through the exchange and the client can mix and match between resources in these services. In fact, the Hardware as a Service (HaaS) described in Section V-H is exposed through the OCX exchange in the same way as OpenStack services, allowing different OpenStack services and clients to compete to use the hardware directly.

IV. THE MOC REALIZATION

Our motivation for the Open Cloud Exchange model is as a basis for the Massachusetts Open Cloud (MOC), a new, non-profit Infrastructure as a Service (IaaS) public cloud being created as a collaboration between the Commonwealth of Massachusetts, research universities and industry. Major university partners include Boston, Harvard, Northeastern Universities, MIT and the University of Massachusetts. Industry partners include Cisco, EMC, SGI, Red Hat, Juniper, Canonical, Dell, Intel, Mellanox, Brocade, DataDirect Networks, Mathworks, Plexxi, Cambridge Computer Services, Enterprise DB and Riverbed; they are contributing engineering and operational talent, equipment, financial support and business guidance.

The goals of the MOC are to:

- Provide a computational service for both university and industry.
- Support the acquisition of specialized resources that no one institution could justify acquiring.

1Arguably, an advantage of today’s inflexible and closed public cloud is that it offers a few easily-understood offerings.
Many users access the IaaS layer indirectly by using intermediaries. These might include HPC portals, Big Data platforms (e.g., Hadoop, and Spark), or various web environments (e.g., PaaS environments like Cloud Foundry, or OpenShift), and may be deployed and managed by industry partners or by the universities. These services will typically be implemented on top of resources provided by IaaS service providers, choosing these resources based on factors such as performance, price, and usage policies. For example, a life sciences Galaxy[12] service for genetic sequencing may make use of C3DDB services for most operations, but request services from commercial providers when load is high or when handling requests from users not authorized to make use of C3DDB resources.

A grant-funded limited-access resource managed by some of the MOC institutions and restricted to life sciences users.

V. SUPPORTING THE REQUIREMENTS

Features needed to support an OCX include mechanisms for (i) service discovery and (ii) requesting and configuring services; (iii) a cross-provider authentication and authorization mechanism which allows services to interact in a secure manner, (iv) consistent naming to allow interaction between services from different providers, and (v) an inter-provider communication mechanism with sufficient bandwidth. This section describes these features and points out how they differ or extend the architectural features provided by OpenStack.

A. A Global Service Directory

In OpenStack the types of services are limited, and there is a single endpoint to request services of a particular type. In the OCX model there is no bound on the number of service endpoints or service types; therefore a cloud-wide service directory is used for service location, in which providers advertise their services and features, and users in turn locate these services.

The directory provides attributes and location of services in a machine-readable form, which may then be used by client scripts or intermediaries which act on a client’s behalf to select which services to use. Service attributes in this directory such as price, SLA, and security level must have unambiguous cross-provider definitions, while allowing for specification of parameters which may be variable (e.g., price, performance) or intangible (e.g., brand). The format is extensible, allowing the description of new services and attributes which may be introduced after the directory is deployed. For example, it must allow the specification of Hardware as a Service (HaaS, Section V-H), higher-level services such as Hadoop, or more modest variations such as changes in API versions.

This new Global Services Directory makes use of OpenStack’s simple service directory (in Keystone) that maintains the list of endpoints for each of the OpenStack services in a single region. Each provider is assumed to employ OpenStack so information can be extracted from each of these per-provider directory, and used to populate the cloud-wide service directory together with extra meta data describing the characteristics of the service in the new service description language we are developing for the OCX service directory.

B. Client Controlled Selection of Services

In an OCX the client should be able to control which of many providers and versions of a service will be used to handle her requests. To provide this control, a customizable scheduling and resource allocation library is used. This library can be incorporated into intermediaries and end-applications to support modified or extended scheduling policies. This library does not replace the existing OpenStack mechanism but is instead layered above it, selecting which provider to use, after which the existing per-provider scheduler determines which specific service will be used to handle the request.

This approach is a fundamental departure from the OpenStack model of one API endpoint for each type of service, with all selection performed by a scheduler behind this endpoint considering only resource availability and a small number of user-provided constraints. Extending the OpenStack scheduling
model to an OCX would be problematic, as not only does the scheduling problem become more complex as the set of features to be selected from grows, but this shared scheduler must be trusted to fairly and accurately distribute requests across providers.

C. Multi-Provider User Interface

The user interface for a client using the OCX has several key requirements. The first is the ability for a client to use one interface to access offerings from multiple providers. For example, a client can select virtual machine compute from Northeastern and storage from EMC (Figure 2). In addition, the interface must be extensible so that new untrusted interfaces can be exposed to users in a seamless fashion. For example, a customer interacting with a Spark big data service, stood up by one provider, uses a single management panel to access that and other services that she chooses to use. Finally, the UI backend should not be a shared service that can cause one tenant to impact the security or performance of other tenants.

The first requirement means that the GUI exposes a marketplace of services in similar fashion to any app store, where the customer can shop for different services. The GUI collects the information from the OCX services directory and exposes them directly to customers.

For the second requirement, extensibility is enabled by allowing services to provide their own GUI displayed in an inline frame; common styling is being developed to better integrate these “plug-in” UIs. The OCX GUI directly interacts with only a small set of OpenStack services (Nova, Neutron, Cinder) using the OCX library to provision core IaaS services.

For the third requirement, a new instance of the GUI is provisioned in client-controlled capacity, avoiding dependency on services from any one provider. This differs from OpenStack, where the Horizon GUI is a shared web-based service from the single provider.

D. Inter-provider Interactions

A primary goal of the Open Cloud Exchange model to allow tenants to “mix and match” service components from different providers. At the infrastructure level, this means being able to combine compute services from one provider with storage services from one or more separate providers. Doing so requires a significant change from today’s model, as efficient and secure provider-to-provider interactions are required. Rather than requiring extensive trust between providers, we instead require authorization of the request from the end user.

There are two issues that must be addressed. One is a naming issue to allow resources in one service provider to be able to name the service in another provider. Another issue is trust or lack of trust, to ensure that a service in one provider that is used on behalf of a service in a second provider has the appropriate restrictions.

Resource addressing: In OpenStack each resource object is identified by a UUID; the type of the object, and thus the endpoint which may operate on it, is implied by context. In contrast, resource objects in an OCX system will be identified by both UUID and the URI of the service endpoint, allowing a subsystem, e.g. Nova, to route requests to destinations on other providers.

Trust delegation: In a multi-provider cloud it is important to limit the scope of trust delegation when a client invokes a service that must interact with other services on behalf of that client. In the example shown in Figure 3, an untrusted compute provider (red) hosts a user’s virtual machine and accesses a storage volume on a third-party storage provider. The same user allocates other, more sensitive virtual machines (blue) on a more reliable compute provider, with separate storage volumes hosted by the same storage provider. Per-object access control is needed to ensure that the red provider can only access the red storage and not the blue storage. Work is underway with the Keystone team to add mechanisms which will allow the required level of fine-grained access delegation.

E. Federated authentication and authorization

It is necessary in an OCX for each provider to be able to have their own authentication and authorization services to support their own administrators and users. On the other hand a global authentication and authorization service is needed to allow a client to concurrently use multiple services (that may in turn invoke other services) from different providers. Consequently per-provider services must federate with the OCX service. Explicit sharing agreements dictate data elements such as domain and project definitions that need to be common across two or more keystone instances owned by different providers.

For example, in Figure 2 Harvard might be unwilling to trust a service stood up by Northeastern for authorizing its administrators to access a Harvard-managed Nova service. At the same time, we want customers to the cloud to be able to be able to log in once and access services from all the providers.

F. Network spanning and security of underlying protocol

Any OCX solution requires establishing network connectivity between services deployed on different providers. In the
near term, simple tunneling via a gateway server (normally a VM) between each provider will be used. In the long run, however, better mechanisms will be needed, as data centers often have massive bisectional bandwidth and such gateway servers will introduce severe bottlenecks. Also, the simple tunneling solution will not support rich networking functionality, such as being common with network function virtualization.

In the long run, SDN (Software Defined Network) functionality will be developed to provide the connection between the service providers. The SDN’s for each provider negotiate efficient distributed paths and provide rich network functions. This is an area of research as to-date nearly all the development has been on north and south interfaces for networking controllers, i.e., up from the compute resource to the top-of-rack switch and either further up or back down.

G. Securing underlying protocols

In addition to control interfaces, there are also underlying protocols used for the data paths, and in an OCX these have to be more secure than the protocols used in today’s (e.g., OpenStack) single provider cloud (e.g. NFS and iSCSI). In the short term, network isolation can be used to establish pairwise isolated relationships when existing non-secure protocols are used. For example, a Cinder service will have a separate isolated network for each Nova it supports, and volumes being exposed to a particular Nova service will only be exposed over a network shared with that Nova service.

H. HaaS

In a typical OpenStack cloud, one set of administrators controls all the hardware in the entire cloud whereas in an OCX, the hardware is just a service upon which other services can be layered. Sharing of physical resources in OCX is via Hardware as a Service (HaaS) [14] (Figure 4). The main difference between HaaS and related systems such as Emulab [1], Maas [6], and others [2, 8, 20] is that, rather than providing (and requiring) a single solution for both scheduling and OS provisioning, HaaS enables the use of arbitrary pre-existing schedulers and provisioning systems as well as its own.

HaaS uses its knowledge of compute nodes and network connectivity to enforce isolated configurations. To accomplish connectivity, it maintains a list of network switch ports, the NICs of compute nodes, and how the two are connected. Switches are supported via model-specific drivers. Currently VLANs are supported for connecting and isolating networks, though the model could accommodate other mechanisms such as OpenFlow.

In the example shown in Figure 2, Harvard and Red Hat both operate Nova instances, each using Dell resources, and (within the limits of the underlying resources) able to expand and contract their usage as needed. Such flexibility in redeployment of resources is key to the efficient utilization of cloud hardware; indeed, VCL [9] demonstrated major improvements in utilization and economics by being able to shift resources between a Cloud and HPC clusters.
most part, consistent with directions the community is already going.

ACKNOWLEDGMENTS

Partial support for this work was provided by the MassTech Collaborative Research Matching Grant Program, National Science Foundation awards #1347525 and CNS-1149232, and several commercial partners of the Mass Open Cloud who may be found at http://www.massopencloud.org. We are also grateful to the many contributors towards the MOC project, including Azer Bestavros, Ian Denhardt, George Silvis III, Zhaoliang Liu, Andrew Mohn, Jonathan Bernard, Valerie Young, Jonathan Bell, Rohan Garg, Ritesh Singh, Abhishek Raju Charanarthi, Ryan Abouzahra, and Vignesh Venugopal.

References Cited