High Level Models for IaaS Cloud Architectures

Ales Komarek, Jakub Pavlik, and Vladimir Sobeslav

Faculty of Informatics and Management, University of Hradec Kralove,
Rokitanskeho 62, 50003 Hradec Kralove, Czech Republic
{ales.komarek,jakub.pavlik.7,vladimir.sobeslav}@uhk.cz
http://cepsos.cz

Abstract. This paper explains how ontology can be used to model various IaaS architectures. OpenStack is the largest open source cloud computing IaaS platform. It has been gaining wide spread popularity among users as well as software and hardware vendors over past few years. It's a very flexible system that can support a wide range of virtualization scenarios at scale.
In our work we propose a formalization of OpenStack architectural model that can be automatically validated and provide suitable meta-data to configuration management tools. The OWL-DL based ontology defines service components and their relations and provides foundation for further reasoning. Model defined architectures can support simple all-in-one architecture as as well as large architectures with clustered service components to achieve High Availability.

Keywords: IaaS, OpenStack, Meta-data, Ontology, Service-Oriented Architecture, Configuration Management

1 Introduction

Nowadays IT infrastructure is the key component for almost every organization across different domains, but it must be maximally effective with the lowest investment and operating costs. For this reason, cloud Infrastructure as a Service (IaaS) is gradually being accepted as the right solution regardless hosting as private, public or hybrid form. Lots of key vendors had tried to develop own solutions for IaaS clouds during several years ago, but infrastructure is being too complex and heterogeneous. Different vendors means different technology, which caused vendor lock-in and limitations in migrations for future growth. In addition, every organization has different requirements for hardware, software and its use.

Based on the idea of openness, scalability and standardization of IaaS cloud platform NASA together with RackSpace founded in 2010 project called OpenStack, which is a free and open source cloud operating system that controls large pools of compute, storage, and networking resources across the datacenter. It is the largest open-source cloud computing platform today [2]. Community is driven by industry vendors as IBM, Hewlett-Packard, Intel, Cisco, Juniper, Red Hat, VMWare, EMC, Mirantis, Cannonical, etc. In terms of numbers the

OpenStack community contains about 2,292 companies, 8,066 individual members and 89,156 code contributors from 130 different countries [3]. These figures confirm that OpenStack belongs to the largest solution for IaaS cloud.

OpenStack is a modular, scalable system, which can run on a single personal computer or on the hundreds of thousands servers as e.g. CERN [4] or PayPal [5].

Lots of vendors and wide community mean lots of ways how OpenStack can be deployed. Each vendor tries to extend core functions and write new service backends to fit their business goals. The actual system consists of many modules and components designed with plugin architecture that allows custom implementations for various service backends. These components can be combined and configured to match available software and hardware resources and real use-case needs.

Each implementation has its own component combination and use some form of configuration management tool to enforce the service states on designated servers and possibly other network components. These tools require data that covers configuration of all components. However, there is not best practise or recommendations how to build suitable OpenStack cloud for different use cases. Detecting component inconsistencies manually is painful and time consuming process. Companies need standardization and validation process for their specific infrastructure requirements, which can help them automate whole implementation, operating and future upgrades of controlling software or physical hardware.

Our project goals are to find a solution to the following issues:

1. Propose high level architecture model definition (Logical model)
2. Implement service that transforms architecture model to solution level model for configuration management tools (physical realization)
3. Provide way how to define and validate architecture based on available hardware resources and target use case.
4. Automate the whole process from high level modelling to actual enforcement on targetted resources.

This paper is focused on designing and creation of high level architecture model, where we propose a formalization of OpenStack service architecture model, based on the approaches developed in classic knowledge representation domain, especially Service-Oriented Architecture by OpenGroup. Component definition is encoded in an ontology using the standard OWL-DL language, which enables sharing of knowledge about configurations across various systems. Reasoning can be used on the specification to automate validation of configuration changes.

When dealing with hundreds of components with thousands of properties and relations, keeping track of changes throughout its life cycle is very challenging. Current approaches are ad hoc, even OpenStack Fuel (Mirantis OpenStack deployment tool [6]) has severe limitations, there exists no standard for specifying common OpenStack architectural model. The question how to convert the

proposed OWL-DL schema to metadata format that configuration management tools can process is discussed. We are working on external node classification service that uses graph database to serialize the OWL ontology with REST API that configuration management tools can use as metadata provider. This can streamline the process of adopting new services and service backends in predictable way.

In Section 2 we describe service architecture models, which contain OpenStack moduls, deployment, etc. After we explain ontology models for IaaS cloud in Section 3. Finally we show how to implement high level model into ontology in Section 4.

2 Service Architecture Models

This section describes modularity and complexity of OpenStack IaaS platform including core and supporting services. However, there is not so much place for detail description of all components, since the main idea is contained in Sections 3 and 4. The goal of this section is to show that OpenStack modules are independent services, which can be implemented in many different of ways.

2.1 IaaS Architecture Core Modules

OpenStack is complete Infrastructure as a Service platform. It allows to create virtual servers on virtual networks using virtual block devices.

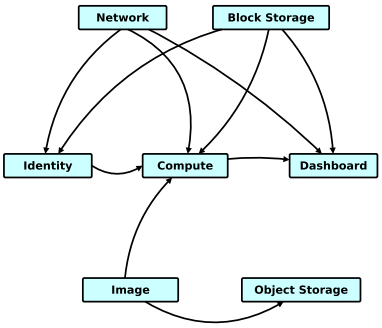

Fig. 1. Logical Model of Icehouse OpenStack service achitecture

Further versions of OpenStack introduce more complex services that use basic services to provide for example Data processsing, Database, Message Queue or Orchestration. All services or modules within OpenStack architecture are independent and have pluggable backends or drivers. This allows vendors to develop plugin for their resources, that can be accessed and managed by the OpenStack API.

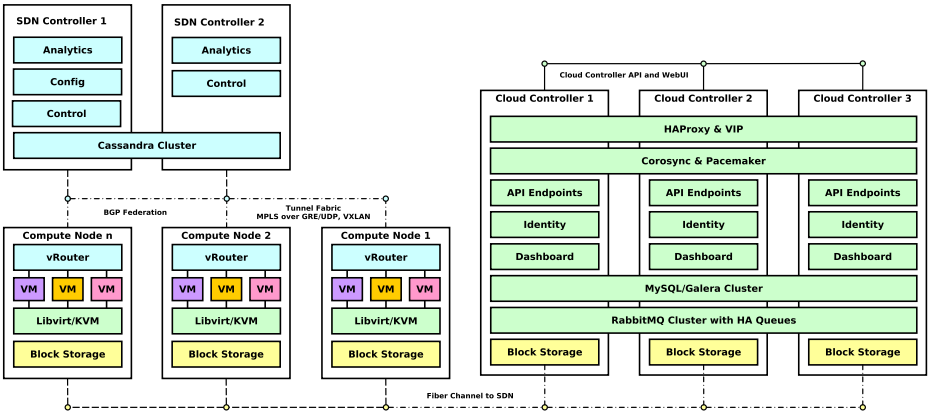

Fig. 2. Locality 2 Architecture

Figure 1 shows the core modules of OpenStack included in Icehouse release. Each module is briefly described including a serveral backends or plugins.

Identity - Keystone is an OpenStack project that provides Identity, Token, Catalog and Policy services for use specifically by projects in the OpenStack family. *Backends/plugins*: sql, ldap

Image - Glance service provides services for virtual disk images. Compute service uses image service to get the starting image of the virtual server. *Backends/plugins*: dir, Swift, Amazon S3

Compute - Nova service is designed to provision and manage large networks of virtual machines, creating a redundant and scalable cloud-computing platform. *Backends/plugins*: KVM, Hyper-V, VMware vSphere, Docker

Network - Neutron is an OpenStack networking project focused on delivering networking as a service. It makes hard to deploy advanced networking services because of wide range of plugins. *Backends/plugins*: Nova flat networking, OpenVSwitch gre/vxlan, OpenContrail, VMware NSX, etc.

Volume - Cinder provides an infrastructure for managing volumes in OpenStack. It uses storage drivers for volumes direct mapping into virtual instances through FibreChannel or iSCSI. *Backends/plugins*: LVM driver, SAN driver, EMC VNX, IBM Storwize, CEPH, Gluster, etc.

OpenStack is not only about its core services, but there are many services at infrastructural level that are essential as well.

High Availability Cluster software is responsible for clustering OpenStack services and creation High Availability in active/active or active/passive mode. *Backends/plugins*: corosync/pacemaker, keepalived

Communication Service is messaging between components of same OpenStack module. *Backends/plugins*: RabbitMQ, QPid, ZeroMQ

Database Services is responsible for storing persistent data of all modules. *Backends/plugins*: MySQL/galera, PostgreSQL

2.2 Use Cases

We participate in operations of several real OpenStack deployments in Central Eastern Europe. Each of them is different, which means that uses different backends in modules. Because of lack of space we decided to show only one use case. Figure 2 shows the logical architecture of TCP Virtual Private Cloud.

3 IaaS Service Ontology

What is an ontology? An ontology is a specification of a conceptualization. According to Gruber [8] an ontology defines a set of representational primitives with which to model a domain of knowledge. These primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application.

In general, any enterprise application can benefit from use of ontologies. They are used in field of semantics-based health information systems, interoperable reference ontologies in biology and biomedicine as summarised in Open Biological and Biomedical Ontologies[13].

The formal definition of cloud computing ontology was introduced by Youseff [15]. It maps the complete domain of Cloud computing from software to hardware resources. It is divided into 5 layers of services.

1. Servers (physical and virtual)
2. Core Infrastructure Services (DNS, NTP, config management)
3. Storage (NAS and SAN)
4. Network (Routers, Switches, Firewalls, Load Balancers)
5. Facilities (Power, Cooling, Space)

The scope of IaaS Service Ontology covers the level 1 and 2 with core infrastructure services and servers running OpenStack services. The levels 3 and 4 with network and storage devices will be adopted in further versions of the ontology. The level 5 will be implemented as last as no services are directly configured.

3.1 Ontological Standards

The ontologies define the relations between terms, but does not prescribe exactly how they should be applied. Following ontologies serve as the starting point for creating new ontologies including the IaaS Service Ontology.

Service-Oriented Architecture The SOA ontology specification was developed in order to aid understanding, and potentially be a basis for model-driven implementation of software systems. It is being developed by Open Group and was updated to version 2 in april 2014. The ontology is represented in the Web Ontology Language (OWL) defined by the World-Wide Web Consortium (W3C). The ontology contains classes and properties corresponding to the core concepts of SOA [10].

OSLC Configuration Management OSLC Configuration Management Resource Definitions [11] is a common vocabulary for versions and configurations of linked data resources. It provides suitable classes and properties from configuration management domain.

Dublin Core Metadata Initiative The Dublin Core Metadata Intiative terms provide vocabularies for common resource definition. It is foundational meta-data vocabulary for many other schemas and covers the basic properties.

3.2 Ontology Serialization Formats

There are several ways how to serialize ontology. In it's core ontology representation is a linking structure that forms a directed, labeled graph, where the edges represent the named relation between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for ontologies and is often used in easy-to-understand visual explanations. They differ by reading and writing speed.

RDF/OWL-DL Documents Ontologies are stored in Web Ontology Language (OWL) defined by the World-Wide Web Consortium (W3C). OWL has three increasingly expressive sub-languages: OWL-Lite, OWL-DL, and OWL-Full [9]. The sub-language OWL-DL provides the greatest expressiveness possible while retaining computational completeness and decidability. RDF is a standard model for data interchange, it as features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed. The format is used by ontology editors

Graph Databases Graph databases can store ontologies very well as they have graph format very similar to RDF format which is standard format of any XML based graph database, just very different implementation. Graph database uses graph structures with nodes, edges, and properties to represent and store data. A graph database is any storage system that provides index-free adjacency. This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary.

3.3 Plain Meta-data Serialization

The domain of cloud computing services can be mapped not just by ontologies but in a less formal data structures. The most common examples are YAML or JSON files with plain or nested data structures. The schema is enforced by documentation and no semantic validation can be used.

Hierarchical Databases The more complex meta-data can be stored in hierarchical databases. These systems allow to define service parameters through class inheritance, which can be overridden. Hierarchical classes can be featured as sets, commonalities, or as roles. You can assemble your infrastructure definition from smaller bits, eliminating duplication and exposing all important parameters to a single location. Within hierarchical databases parameters can reference other parameters in the very hierarchy that are actually assembling.

4 Ontology Usage

The journey to mapping high level models in ontologies was long. It was a process of describing everchanging OpenStack cloud architectures over past 2 years. The OpenStack foundation has released 4 major versions with 12 minor versions over that period. The number of managed software components grew from 5 to 17. At the beginnings we started mapping meta-data models in separate files containing complete meta-data set of all services for each server or device. The meta-data was encoded in YAML format. This approach was fine at earlier versions of OpenStack we tested as just the basic set of OpenStack services existed at the time.

Simple meta-data in YAML format

```
service_name:
  service_role:
    data_parameter1: data_value1
    data_parameter2: data_value2
    data_parameter3:
    - data_value3a
    - data_value3b
    object_parameter:
     object1:
       data_parameter1a:
```

The next step was storing the service meta-data in hierarchical databases. The meta-data was split into separate files. The final meta-data for given resources is assembled using the two simple methods. The deep merging of several service definition fragments and parameter interpolation where parameters can be referenced. The reclass [**?**] was used to implement the desired data manipulation behaviour.

Meta-data in YAML format, parameter interpolation

```
service_name:
  service_role:
    data_parameter1: data_value1
    data_parameter2: {service_name:service_role:data_paramer1}
    object_parameter:
     {otherobject:object1}
```

This was elegant solution that could easily model growing number of services, but still lacked mechanisms to validate given data or encapsulate semantics. It helped to determine the domain and scope of new ontology. The ontology should use existing standards and provide vocabulary to define OpenStack services, core services and later network and storage resources. The ontology should provide mechanisms to validate schema for integrity issues, as missing parameters, disjoint services or values out of proper value domain.

All OpenStack services can be very well described by ontology as they communicate over common message bus, serialize their state and expose services through interfaces in a same way. The resource within ontology have data properties that are derived mostly from Dublin Core metadata terms. Other extensively used standard is OCLS Configuration Management resource definitions and Service-Oriented Architecture Ontology. Resources can have object property types that describe more complex relations. Following example shows excerpt from Glance image service definition on the the controller node:

Meta-data in OWL-DL format

```
<owl:Class rdf:about="#CinderVolumeService">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#VolumeService"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#NovaVolumeService"/>
  </owl:disjointWith>
</owl:Class>
```

4.1 Implementation details

Initial work on creating our Ontology was done in Protege, open-source ontology editor and framework for building intelligent systems. It took some time to evolve ontology classes with propertice and create necessary dictionaries [16] to cover first IaaS architectures.

The ontology is transformed into graph database using our python-bases service named django-ENC that can read and write ontology from OWL-DL XML files created by Protege and communicates with neo4j graph database through REST API. The graph databases are part of family of NoSQL databases and offer much better performance at any volume of data.

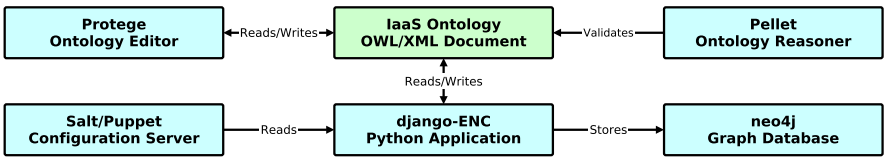

Fig. 3. Ontology Service Architecture

The django-ENC service use web framework Django to provide web services and asynchronous task queue Celery to perform time consuming tasks like on-

tology assertions and synchronizations between XML and graph database. The HTTP REST API that can be consumed by configuration management tools like Salt or Puppet through their External Node Classification interface. The meta-data passed to configuration management tools is valid for level 1, 2 and 3 of unified cloud computing ontology [15].

We have successfully tested service status enforcement of several complete OpenStack installations by SaltStack configuration management tool with meta-data acquired from Ontology Service API. The deployment process is not yet fully automated as there is need of setting up network and storage resources manually (only servers are proviced), but the progress in both configuration management tools and network and storage will allow better automation of these components by in-place agents or access protocols like SSH in the future.

5 Conclusion

We have managed to do the first steps in formalization of IaaS Architecture high level models. The representation of models, the ontologies, can be used to create and validate meta-data for individual OpenStack cloud installations. The ontology provides schema for the meta-data for each installation so the overall service integrity is ensured.

We created a python-based web service django-enc that use data from the ontology to generate the suitable meta-data for configuration management tools. The service provide simple interface for manipulating the ontology as well as interfaces for ontology editors. The ontology defines the basic services of OpenStack Havana and Icehouse versions. New components and service backends can be easily defined and included.

We plan to expand ontology from virtual and physical servers to network and storage resources by better adoption of configuration management tools. Ontology model is suitable for software agent processing and their rational decisions. It is possible to define agents that will maintain the state of services according to the high-level model. The more parts of the process are modelled and their deployment automated the more manageable the whole system becomes.

References

1. NIST, (2011). The NIST Definition of Cloud Computing [online] Available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf [Accessed 10 Oct. 2014].
2. OpenStack.org, (2014). OpenStack Open Source Cloud Computing Software. [online] Available at: http://openstack.org [Accessed 12 Oct. 2014].
3. Stackalytics.com, (2014). Stackalytics — OpenStack community contribution in Kilo release. [online] Available at: http://stackalytics.com [Accessed 12 Oct. 2014].
4. Information-technology.web.cern.ch, (2014). OpenStack Information. [online] Available at: http://information-technology.web.cern.ch/book/cern-private-cloud-user-guide/openstack-information [Accessed 12 Oct. 2014].

5. Openstack.org, (2014). PayPal : OpenStack Open Source Cloud Computing Software. [online] Available at: http://www.openstack.org/user-stories/paypal/ [Accessed 12 Oct. 2014].
6. Wiki.openstack.org, (2014). Fuel - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/Fuel [Accessed 13 Oct. 2014].
7. Ivan Ivanov, Marten van Sinderen and Boris Shishkov, editors, (2012). Cloud Computing and Services Science. Springer Science, New York, USA.
8. Ling Liu and M. Tamer zsu, editors, (2009).Encyclopedia of Database Systems, Springer Science, New York, USA.
9. W3C.org, (2004). Web Ontology Language (OWL). [online] Available at: http://www.w3.org/2004/OWL [Accessed 12 Oct. 2014].
10. The Open Group, (2014). Service-Oriented Architecture Ontology, Version 2.0, Open Group, New York, USA.
11. OASIS, (2013). Configuration Management Resource Definitions. [online] Available at: https://tools.oasis-open.org/version-control/browse/wsvn/oslc-ccm/trunk/specs/config-mgt.html [Accessed 8 Oct. 2014].
12. Dublin Core Metadata Initiative, (2012). DCMI Metadata Terms. [online] Available at: http://dublincore.org/documents/dcmi-terms/ [Accessed 11 Oct. 2014].
13. Berkeley Bioinformatics Open Source Project, (2014). The Open Biological and Biomedical Ontologies. [online] Available at: http://www.obofoundry.org/ [Accessed 10 Oct. 2014].
14. W3C.org, (2001). Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. [online] Available at: http://www.w3.org/2001/sw/BestPractices/SE/ODA/ [Accessed 5 Oct. 2014].
15. Youseff L., Butrico, M., Da Silva, D., (2008). Toward a Unified Ontology of Cloud Computing. Grid Computing Environments Workshop GCE '08, IEEE, USA.
16. Lucas de Oliveira Arantes, Ricardo de Almeida Falbo, Giancarlo Guizzardi, (2009). Evolving a Software Configuration Management Ontology, IEEE