Ontology for OpenStack Service Architectures

Ales Komarek
Faculty of Informatics and Management
University of Hradec Kralove
Czech Republic
Email: ales.komarek@uhk.cz

Jakub Pavlik
Faculty of Informatics and Management
University of Hradec Kralove
Czech Republic
Email: jakub.pavlik.7@uhk.cz

Vladimir Sobeslav
Faculty of Informatics and Management
University of Hradec Kralove
Czech Republic
Email: vladimir.sobeslav@uhk.cz

***Abstract*—**

This paper explains how ontology can be used to model various OpenStack 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 any scale.

In our work we propose a formalization of OpenStack architectural model that can be automatically validated and serve suitable meta-data to configuration management tools. The OWL-DL based ontology defines service components and their relations and provides foundation for further reasoning. Provided models can support simple all-in-one architecture as as well as large architectures with service components in High Availability setup.

I. Introduction

OpenStack is the largest open-source cloud computing platform today. Many companies participate to its code, extend core functions and write new service backends to fit their business goals. The actual system consists of many 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. Detecting component inconsistencies by hand is painful and time consuming process.

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 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 manner.

A. Use Cases

OpenStack is system that has growing number of components with growing number of components and drivers. As we will show in following text there's no universal installation of OpenStack.

B. Infrastructure Modeling

Tools usually collect required metadata through web forms or answer files. This is not a conceptual way to describe model.

C. Process Automation

II. Service Architecture Models

Infrastructure as a Service platforms at it's core controls various virtualization interfaces and services and allows to launch a new virtual server from given disk image at chosen host server, connect it to provided physical or virtual network and add block storage device to it. The OpenStack platform provides services that address needed to provide the very basic compute, network and storage services. The identity provider and image store provide further services needed to provide the basic sevices.

Nebula was the predecessor of OpenStack and was superseded because it did not scale well. The services withing OpenStack communicate through 3 various communication channels

TCP/SQL - Services store it's state in SQL datastore

AMQP - Service internally communicate over asychronous communication bus which .... Compute service calls network service

HTTP - All services expose REST APIs that allow higher level integration, control ...

A. Architectural Level

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

Further versions of OpenStack introduce more sophisticated services that use basic services to Data processsing

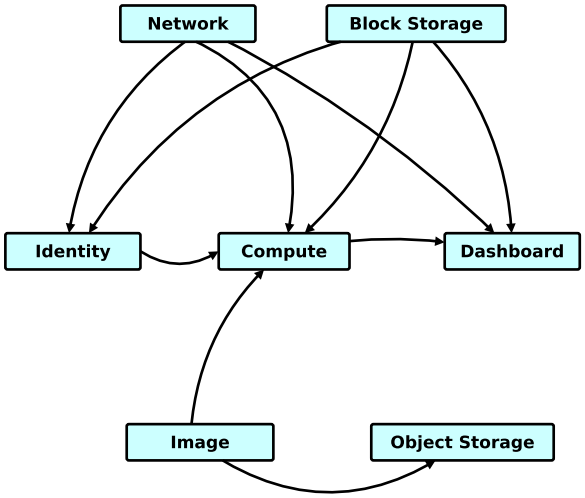

Fig. 1. Logical Model of Havana OpenStack

These core services are followed by growing number of services covering for example telemetry, orchestration or data processing. All services within OpenStack architecture have pluggable backends. This allows vendors to develop plugin for their resources, that can be accessed and managed by the OpenStack API.

Following Figure shows the basic configuration of OpenStack in Icehouse version.

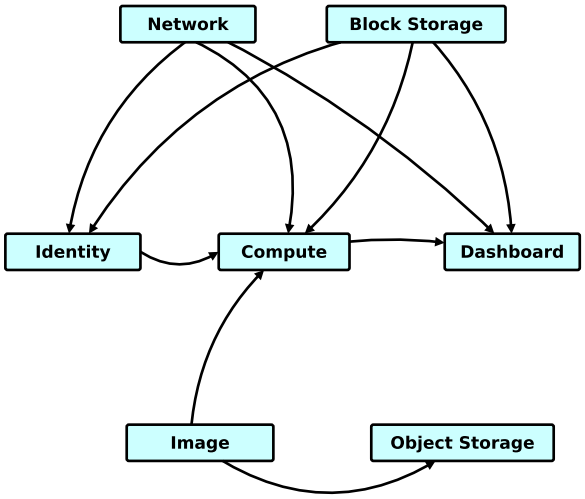

Fig. 2. Logical Model of Icehouse OpenStack service achitecture

Klasicky logicky model openstack architektury

2) OpenStack architecture moduls

Database

Message queue

Time service

Identity - Keystone

Image - Glance

Compute - Nova

Network - Neutron

Volume - Cinder

B. IaaS Controller Support Services

This section covers the

Real implementations of Architectural models

1) High Availability Services: Cluster software - corosync/-pacemaker - keepalived

2) Communication Services: RPC

- rabbitmq

- qpid

- 0mq

3) Database Services: Database

- mysql/galera

- postgressql/xtradb

4) Time Synchronization Services: time

- ntp

C. Iaas Controller Core Services

APIs

Pluggable backends

1) Identity Service: Keystone is an OpenStack project that provides Identity, Token, Catalog and Policy services for use specifically by projects in the OpenStack family.

- sql

- ldap

2) Image Service: OpenStack Image service (code-named Glance) provides services for virtual disk images. Compute service uses image service to get the starting image of the virtual server.

OpenStack Image service handles variety of disk image formats, including Raw, Machine (kernel/ramdisk outside of image, also known as AMI), VHD (Hyper-V), VDI (VirtualBox), and qcow2 (Qemu/KVM).

- dir

- swift

- s3

3) Compute Service: OpenStack Compute service (codename Nova) is designed to provision and manage large networks of virtual machines, creating a redundant and scalable cloud-computing platform. It provides the software required to orchestrate compute instances through using libvirt or or other virtualization client libraries. OpenStack Compute service is both hardware and hypervisor agnostic, currently supporting a variety of standard hardware configurations and major hypervisors.

- kvm

- qemu

- docker

- hyper-v

- vsphere

4) Network Service: Neutron is an OpenStack networking project focused on delivering networking as a service. Neutron has replaced the original networking application program interface (API) in OpenStack. Neutron is designed to address deficiencies in "baked-in" networking technology found in cloud environments, as well as the lack of tenant control (in multi-tenant environments) over the network topology and addressing, which makes it hard to deploy advanced networking services.

- flat networking

- ovs-gre/vxlan

- sdn

- opencontail

- nsx

5) Volume Service: Cinder

- lvms

- sans

Různé způsoby nasazení ukázky reálné architektury - promapovat ve 4 na ontologii

D. Use Cases

Why we choose different openstack setups

1) Locality 1: At CEPSOS laboratory have deployed

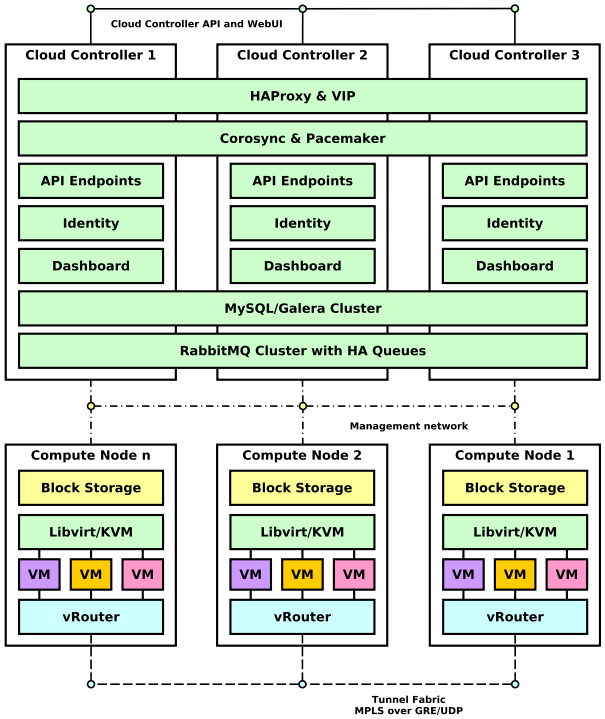

Fig. 3. Locality 1 Architecture

- 20 hypervisors, kvm, ovs-gre, local hdd

2) Locality 2: At Cloudlab in datacenter in Pisek we tested

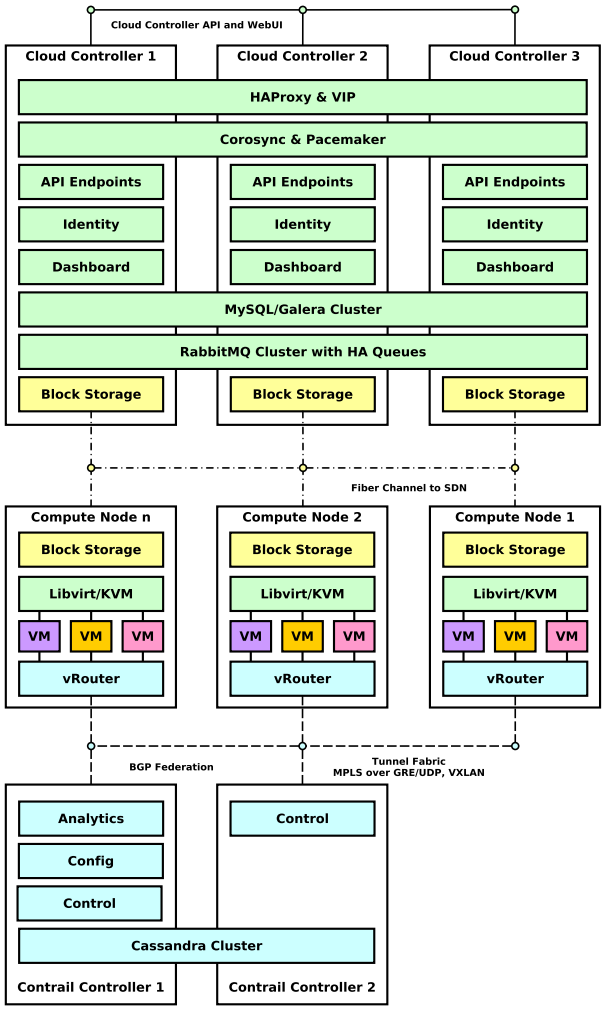

Fig. 4. Locality 2 Architecture

- 4 hypervisoers, kvm, sdn-contrail, san

3) Locality 3: At Cloudlab in datacenter in Pisek we tested

- 3 hypervisors, ... l2 networking

III. OpenStack Deployment Options

There are many ways how to deploy OpenStack infrastructure which are more or less automated. Some of them require to fill in answer files, some configuration files. Some tools have graphiceal user interface and allow to provision entire hardware infrastructure as some just configure the services on the provisioned servers.

Model je popsanej dokumentem a není to čitelný, automatizace. Není validita modelu. Chyby se debugují na úrovni reality.

A. Development Environment Installers

For testing and developing OpenStack ...

1) PackStack: Packstack is a utility that uses Puppet modules to deploy various parts of OpenStack on multiple pre-installed servers over SSH automatically. Currently only Fedora, Red Hat Enterprise Linux (RHEL) and compatible derivatives of both are supported.

2) Devstack: DevStack has evolved to support a large number of configuration options and alternative platforms and support services. That evolution has grown well beyond what was originally intended and the majority of configuration combinations are rarely, if ever, tested.

B. Production Environment Managers

For production installations of OpenStack ...

1) Fuel: Fuel is an open source deployment and management tool for OpenStack. Developed as an OpenStack community effort, it provides an intuitive, GUI-driven experience for deployment and management of OpenStack, related community projects and plug-ins.

2) Foreman: You can setup Foreman to deploy RDO. The metadata is provided in Host Groups.

C. Configuration Management Tools

You can install OpenStack by configuration management tool

1) Puppet: It's already used by Fuel an Foreman

2) Salt: Salt is another approach to install OpenStack.

IV. OpenStack Platform Ontology

Let's start what the ontology is, the short is answer:

An ontology is a specification of a conceptualization.

The broader answer may be:

In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational 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.

http://tomgruber.org/writing/ontology-definition-2007.htm

There formal definition of cloud comptuting architecture

1. Servers (physical and virtual)

2. Core Infrastructure Services (DNS, DHCP, NTP, image management)

3. Storage (NAS and SAN)

4. Network (Routers, Switches, Firewalls, Load Balancers)

5. Facilities (Power, Cooling, Space)

A. Standard Ontologies

The ontologies define the relations between terms, but does not prescribe exactly how they should be applied.

OWL has three increasingly expressive sub-languages: OWL-Lite, OWL-DL, and OWL-Full [OWL]. The sub-language OWL-DL provides the greatest expressiveness possible while retaining computational completeness and decidability.

1) 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

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. The formal OWL definitions are supplemented by natural language descriptions of the concepts, with graphic illustrations of the relations between them, and with examples of their use. For purposes of exposition, the ontology also include.

2) OSLC Configuration Management:

B. Ontology Serialization Formats

Způsoby serialiazce ontologie

1) XML Documents: RDF is a standard model for data interchange on the Web. RDF has 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.

RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.

2) Graph databases: Preserving RDF format, just very different implementation

In computing, a graph database is a database that 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. General graph databases that can store any graph are distinct from specialized graph databases such as triplestores and network databases.

je to servica, tzn overhead oproti xml filu, ale zas ma api atd ...

C. Plain Meta-data Serialization Formats

It's tree structure

1) Hierarchical Databases: Subject (id) or property driven

reclass allows you to define your nodes through class inheritance, while always able to override details further up the tree (i.e. in more specific nodes). Think of classes as feature sets, as commonalities between nodes, or as tags. Add to that the ability to nest classes (multiple inheritance is allowed, well-defined, and encouraged), and you can assemble your infrastructure from smaller bits, eliminating duplication and exposing all important parameters to a single location, logically organised. And if that isn't enough, reclass lets you reference other parameters in the very hierarchy you are currently assembling.

D. Ontology structure

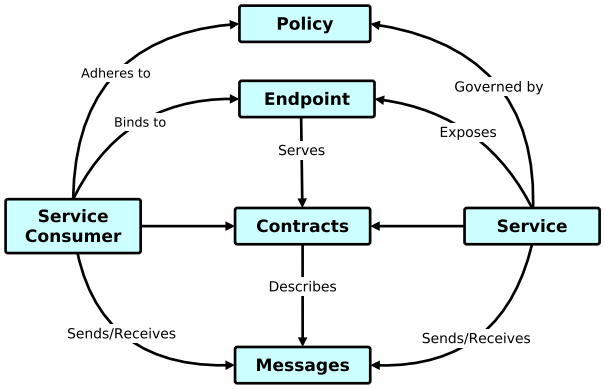

Fig. 5. SOA service to consumer

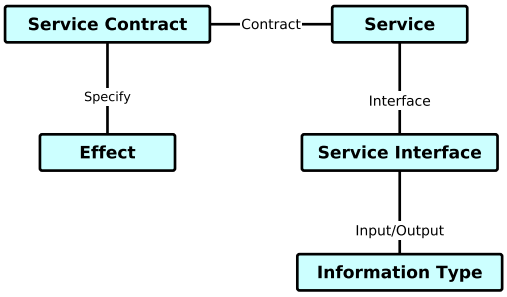

Fig. 6. SOA service properties

E. Comparison

- speed - parsing / scaling

- integration, maintenance costs

- security issues

V. Ontology Usage

We started mapping the very

The Ontology can support many individual implementations at the time

A. Implementation details

Initial work on creating our Ontology was done in Protege, open-source ontology editor and framework for building intelligent systems.

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 database through REST API. The graph databases are part of family of NoSQL databases and offer much better performance at any volume of data.

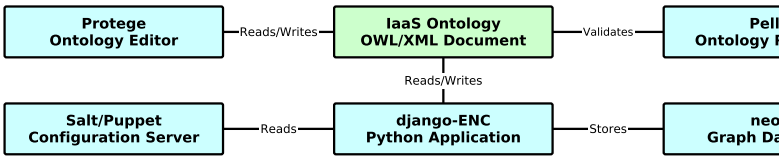

Fig. 7. Ontology Service Architecture

The django-ENC service use web framework Django to deliver web services and asynchronous task queue Celery to perform time consuming tasks like ontology assertions and synchronizations between XML and graph database. Service expose it's owns HTTP REST API that can be consumed by configuration management tools like Salt or Puppet through their External Node Classification option.

The metada passed to CM tools is valid for 1st level of Cloud computing ontology [cite]

We have tested SaltStack configuration management tool to instantiate

The process is not yet fully automated as there is need of setting up network and storage components manually, but the progress in both configuration management tools and network and storage will allow better automation of these components by on-place agents or access protocols like SSH.

B. Ontology samples

Given use case scenario Lab1 we have 3 virtual servers providing OpenStack and core services in HA mode. These servers are virtualised in common . 20 physical servers

```
<owl:Class rdf:about="#Glance">
  <owl:disjointWith>
    <owl:Class rdf:about="#"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Composition"/>
  </rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:about="#Glance">
  <owl:disjointWith>
    <owl:Class rdf:about="#"/>
  </owl:disjointWith>
```

```
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Composition"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Composition"/>
    </rdfs:subClassOf>
  </owl:Class>



  <owl:Class rdf:about="#Glance">
    <owl:disjointWith>
      <owl:Class rdf:about="#"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Composition"/>
    </rdfs:subClassOf>
  </owl:Class>



  <owl:Class rdf:about="#Glance">
    <owl:disjointWith>
      <owl:Class rdf:about="#"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
  </owl:Class>
```

VI. Conclusion

Ontologická reprezentace prostředí, která je vhodná pro agentové prostředí, aby bylo možné provádět autonomní rozhodnutí.

In our work we created ontology of OpenStack services.

Vytvoření ideálního prostředí.

Způsoby přenesení ontologie do realizace

A. Future work

Low level realizace prostřednictvím configuration management tools a jejich transformací z modelu.

Acknowledgment

This paper is published thanks to the financial support of the European Operational Programme Education for Competitiveness project INDOP CZ.1.07/2.3.00/45.0014 and UHK specific research project no. 2101.

References

[1] *NIST. The NIST Definition of Cloud Computing* http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[2] Ivan Ivanov, Marten van Sinderen and Boris Shishkov, editors. *Cloud Computing and Services Science* Springer Science, 978-1461423256, New York, USA, 2012

[3] *OpenStack. Fuel Wiki* https://wiki.openstack.org/wiki/Fuel

[4] *CloudStack. Open Source Cloud Computing: Apache CloudStack* http://cloudstack.apache.org/about.html

[5] *Fedora Project. OpenStack devstack* http://fedoraproject.org/wiki/OpenStack_devstack

[6] *reclass. Recursive external node classification* http://reclass.pantsfullofunix.net/

[7] *CFEngine. FCEngine 3.5 Dcoumentation* https://cfengine.com/docs/3.5/index.html

[8] *Puppet Labs. Creating Hierarchies* http://docs.puppetlabs.com/hiera/1/hierarchy.html

[9] *The Foreman. The Manual: Compute Resources* http://theforeman.org/manuals/1.4/index.html# 5.2ComputeResources

[10] *Configuration Management Resource Definitions* http://open-services.net/wiki/configuration-management/Configuration-Management-Resource-Definitions/

[11] *Configuration Management Resource Definitions* http://www.w3.org/2004/OWL

[12] *Web Ontology Language (OWL)* http://www.w3.org/2004/OWL