Cloud Standard API and Contextualization

XIANGHU ZHAO

IHEP COMPUTING CENTER

2015 BESIIICGEM CLOUD COMPUTING SUMMER SCHOOL

Content

- Cloud Standard API
  - OCCI
  - EC2
  - Other Approaches
- Contextualization
  - Cloud-init

Cloud Standard API

Cloud API

- Cloud API (Application Programming Interfaces)
  - API through network
  - Almost all based on the HTTP protocol
    - Independent on programming language
    - RESTful / XML-RPC / SOAP / …
  - Web portal and command lines running upon cloud API
- Cloud SDK (Software Development Kit)
  - Provide easy access to cloud API
  - Implementations for multiple programming languages

Cloud Provider Specific API

- Specific for certain cloud manager
- Each cloud manager usually has its own API
  - OpenNebula
    - OCA (XML-RPC)
  - OpenStack
    - Nova API
- May provided several SDKs for different programming languages

Cloud Standard API

- All clouds are providing similar functionalities
  - It is possible to use a unique way to manage different cloud types
- Cross-platform
  - Control different cloud types with the same interface
- Easier to deal with cloud bursting and federation

Weakness of Cloud Standard API

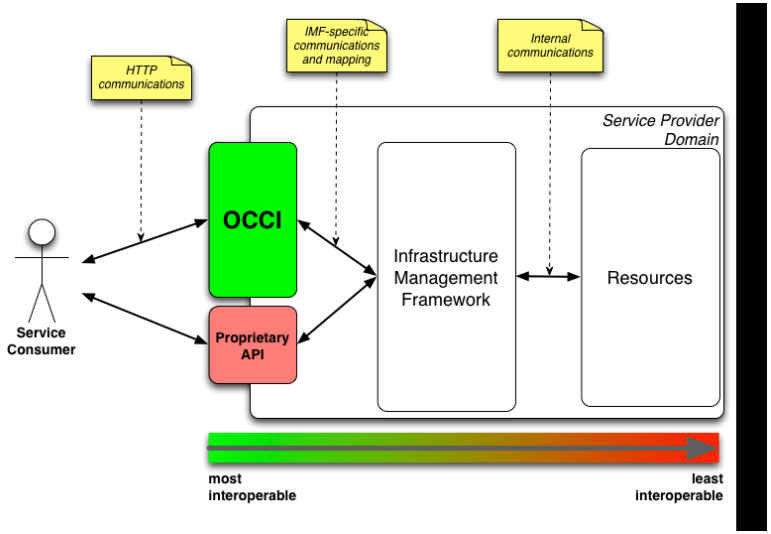
- Not full-featured API for a specified cloud
  - General design
- Need extra configuration or installation for the cloud
  - Configuration
  - Installation of some intermediate services

OCCI

Introduction to OCCI

- Open Cloud Computing Interface
- Protocol and API for management of cloud service resources
- OCCI is an open specification
  - Need implementations to put it into practice
- Used as the second API for clouds

Occi
Open Cloud Computing Interface



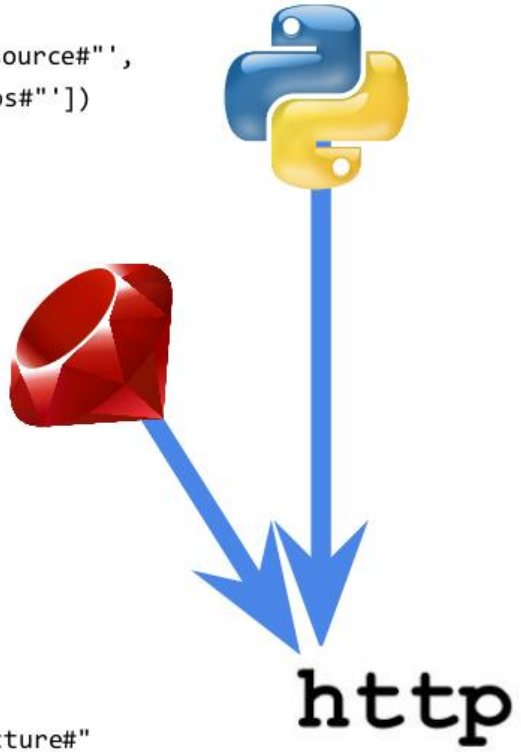
OCCI in Action

Python

```
comp = client.create_vm(token, [
           'tiny; scheme="http://schemas.openstack.org/template/resource#"',
           'cirros; scheme="http://schemas.openstack.org/template/os#"'])
```

Ruby

```
cmpt = client.get_resource "compute"
cmpt.mixins << client.find_mixin 'cirros', 'os_tpl'
            << client.find_mixin 'tiny', 'resource_tpl'
client.create cmpt
```



On-the-wire

```
> POST /compute/ HTTP/1.1#
> User-Agent: occi-client/1.1 (linux) libcurl/7.19.4 OCCI/1.1
> Host: localhost:8888
> Accept: text/plain
> Content-type: text/plain
>
> Category: compute; scheme="http://schemas.ogf.org/occi/infrastructure#"
> Category: cirros; scheme="http://example.com/templates/os#"
> Category: tiny; scheme="http://example.com/templates/compute#"
```

rOCCI

- rOCCI is a modular Framework for OCCI written in ruby
- Current supported backends
  - OpenNebula
  - EC2
- Contain 4 parts
  - rOCCI-core
  - rOCCI-api
  - rOCCI-cli
  - rOCCI-server

Adding OCCI support for OpenNebula

- Install rOCCI-server for OpenNebula
- rOCCI-server is actually a web service
  - Based on Ruby on Rails framework
- rOCCI-server Could be located on any place
  - Virtual machine is also OK
- Configure the web server
  - apache / nginx / …
- Installation details
  - https://wiki.egi.eu/wiki/rOCCI:ROCCI-server_Admin_Guide

rOCCI Command Line

- rOCCI-cli is a ready-to-use shell client for OCCI enabled services
- Provide full features to access rOCCI server
- Installation
  - Need Ruby >= 1.9.3
  - gem install occi-cli

rOCCI-cli Examples

- List all images
  - occi --endpoint https://<ENDPOINT>:<PORT>/ --action list --resource os_tpl --auth x509
- List all resource types (CPU, Memory, …)
  - occi --endpoint https://<ENDPOINT>:<PORT>/ --action list --resource resource_tpl --auth x509

rOCCI-cli Examples

- List all VM instances
  - occi --endpoint https://<ENDPOINT>:<PORT>/ --action list --resource compute --auth x509
- Create a new VM instances
  - occi --endpoint https://<ENDPOINT>:<PORT>/ --action describe --resource compute --auth x509
- Detail information about the VM instance
  - occi --endpoint https://<ENDPOINT>:<PORT>/ --action describe --resource /compute/<OCCI_ID> --auth x509

EC2

Introduction to EC2 API

- EC2 API is original used for management of Amazon EC2
- Became a kind of standard by the powerful influence of AWS in cloud computing
- It is supported by many cloud managers

Tools for EC2 API

- Amazon EC2 command line tools
  - EC2 official tools written in java
- euca2ools
  - Compatible with Amazon EC2 and IAM APIs
- econe tools provided by OpenNebula
  - Suitable for testing the OpenNebula econe service

Configure EC2 in OpenNebula

- Modify the configuration file /etc/one/econe.conf
- Start econe service
  - Econe-server start
- EC2_ACCESS_KEY is the user name
- EC2_SECRET_KEY is the SHA1 hashed password
  - oneuser show user-name
- Detailed configuration
  - http://docs.opennebula.org/4.8/advanced_administration/public_cloud/ec2qcg.html

econe Examples

- Common environments for econe commands
  - EC2_URL
  - EC2_ACCESS_KEY
  - EC2_SECRET_KEY
- Image

```
$ econe-upload /images/gentoo.img
Success: ImageId ami-00000001
$ econe-register ami-00000001
Success: ImageId ami-00000001
```

```
$ econe-describe-images -H
Owner        ImageId       Status         Visibility   Location
------------------------------------------------------------------------------
helen        ami-00000001  available      private      19ead5de585f43282acab4060bfb7a07
```

econe Examples

- Instance

```
$ econe-run-instances -H ami-00000001
Owner       ImageId                 InstanceId InstanceType
-------------------------------------------------------------------------------
helen       ami-00000001            i-15       m1.small
```

```
$ econe-describe-instances  -H
Owner       Id    ImageId      State         IP              Type
-------------------------------------------------------------------------------
-----
helen       i-15  ami-00000001 running      147.96.80.33     m1.small
```

```
$ econe-terminate-instances i-15
Success: Terminating i-15 in running state
```

EC2 SDK Example

```
#!/usr/bin/env python

import boto
from boto.ec2.regioninfo import RegionInfo

accessKey = 'SOMEACCESSKEY'
secretKey = 'SOMESECRETKEY'
region = RegionInfo(name='cn-north-1', endpoint='ec2.cn-north-1.amazonaws.com.cn')

conn = boto.connect_ec2(aws_access_key_id = accessKey,
                        aws_secret_access_key = secretKey,
                        is_secure = False,
                        region = region,
                        path = '/',
                        port = 80,
                        debug = False)

print(len(conn.get_all_images(owners = 'self')))
print(conn.get_all_instances())
print(conn.get_all_instance_status())
print(conn.get_image('ami-ba2dbf83'))
```

Other Approaches

Unified API from Client

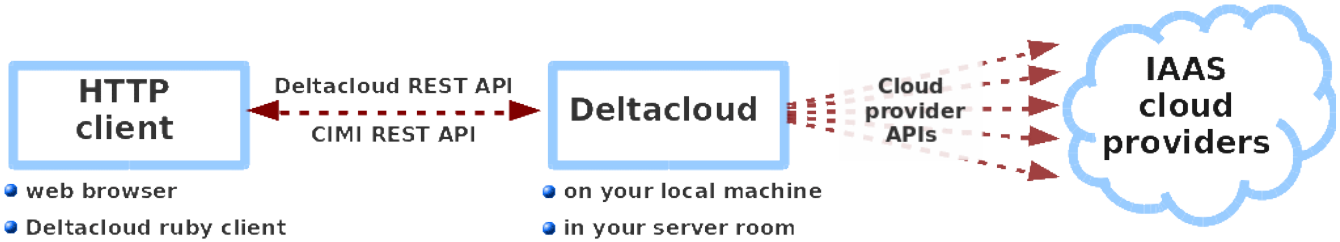
- Interacting with many cloud service providers using a unified API
- Provide different drivers for many clouds
- Do not need to change anything from the cloud side
- Related projects
  - Apache Libcloud (python)
  - Fog (ruby)
  - Apache Deltacloud
  - Libcloud REST
  - …

Libcloud

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver

cls = get_driver(Provider.RACKSPACE)
driver = cls('username', 'api key', region='iad')

sizes = driver.list_sizes()
images = driver.list_images()

size = [s for s in sizes if s.id == 'performance1-1'][0]
image = [i for i in images if 'Ubuntu 12.04' in i.name][0]

node = driver.create_node(name='libcloud', size=size, image=image)
print(node)
```

Driver could be change to accommodate different clouds

Deltacloud



- web browser
- Deltacloud ruby client
- curl
- your custom client
- any other client that can make HTTP requests

- on your local machine
- in your server room
- on your network
- a public Deltacloud instance

There is No Silver Bullet for Cloud API

- There are still many standards
- Choose the one fits your needs best
  - Supported cloud types
  - The way to manage the cloud
  - Need for cloud specific features

Contextualization

What is Contextualizaton

- Contextualization provides boot time customization for cloud and virtualization instances.
- Service runs early during boot, retrieves user data from an external provider and performs actions
- Could build various VM instances with the same image

Contextualization Methods

- Amiconfig
  - User-data
- HEPIX
- Cloud-init
- Vmcontext in OpenNebula
- …

Cloud-init

- Supported user data formats:
  - Shell scripts (starts with #!)
  - Cloud config files (starts with #cloud-config)
    - Standard YAML syntax available for many common configuration operations.
  - MIME multipart archive.
    - Custom part handling also available.
- Modular and highly configurable.

Cloud-init Modules

- cloud-init has modules for handling:
  - Disk configuration
  - Command execution
  - Creating users and groups
  - Package management
  - Writing content files
  - Bootstrapping Chef/Puppet
- Additional modules can be written in Python if desired.

Data Categories

- meta-data is provided by the cloud platform.
- user-data is a chunk of arbitrary data the user provides.
- Retrieved from data source and saved to /var/lib/cloud/

What can cloud-init Do

- You may already be using it!
  - Injects SSH keys.
  - Grows root filesystems.
- Other module support tasks such as
  - Setting the hostname.
  - Setting the root password.
  - Setting locale and time zone.
  - Running custom scripts.

Examples

- User-data Examples
- Upgrading and installing packages:
  - #cloudconfig
  - package_upgrade: true
  - packages:
  - - git
  - - screen
  - - vimenhanced

Examples

- Run an arbitrary command:
- #cloudconfig
- runcmd:
- - rhnreg_ks activationkey=3753...
- Or:
- #!/bin/bash
- rhnreg_ks activationkey=3753...

Enable Cloud-init in Image

- Use cloud-init enabled image
- Install cloud-init package via yum in the guest OS
  - Enable EPEL repository
  - yum install cloud-init
- Make new image from the above instance

How does it work - OpenStack / EC2

- Accesses metadata service at
  - http://169.254.169.254/latest/meta-data
  - http://169.254.169.254/latest/user-data
- NAT rules on your network controller make this work.
- Service provided by nova-api (accessed via per-router neutron-metadata-proxy when using Neutron).

How does it work - OpenNebula

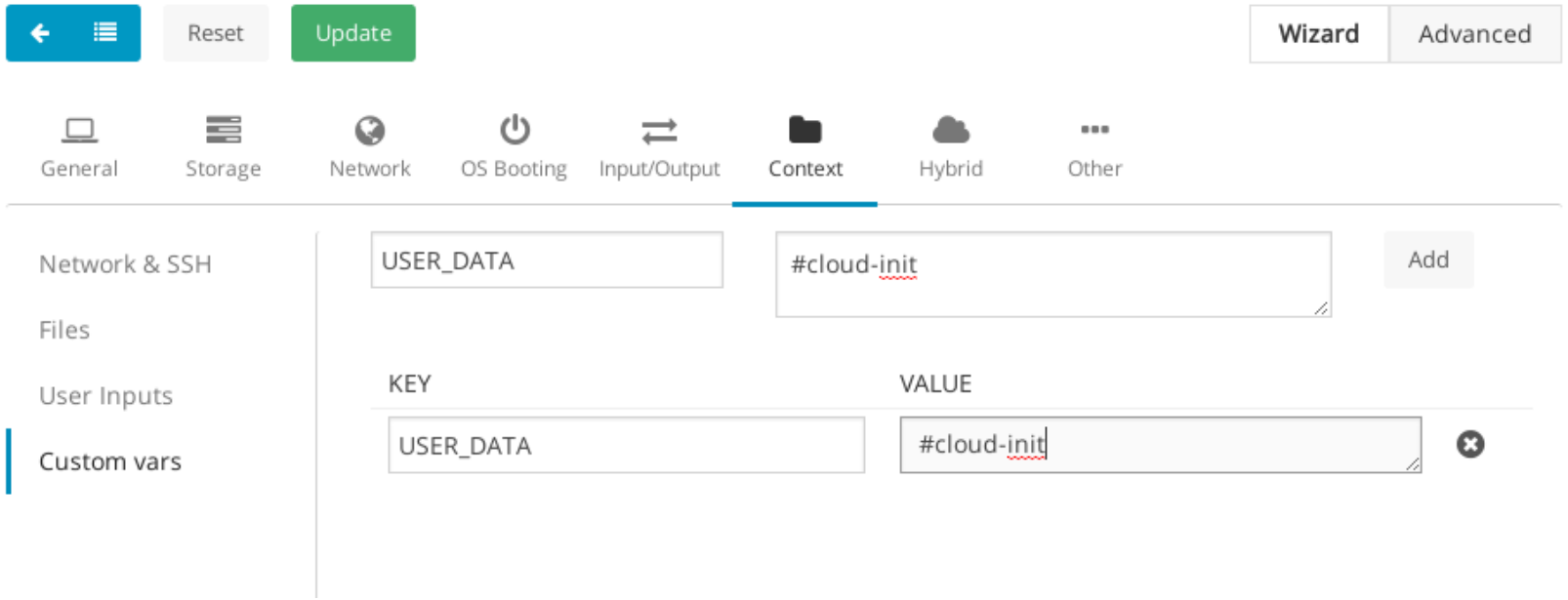
- Try to find the CONTEXT ISO disk created by OpenNebula
- Find the context.sh file in the ISO image
  - Configure the network with the variables
  - Get the USER_DATA and run it as cloud-init script

Cloud-init for OpenNebula

- Need a little modification of configuration
- Use the OpenNebula data source
- Edit /etc/cloud/cloud.cfg in VM and add:

```
disable_ec2_metadata: True
datasource_list: ['OpenNebula']
```

Set USER_DATA



Thanks