An Approach for Migrating Applications to Interoperability Cloud

Binh-Minh Nguyen Ph.D.

Agenda

- Cloud Vendor Lock-In
- Portability and Interoperability
- Interoperability Approaches
- Cloud Abstraction Layer (CAL)
- Workflow-as-a-Services

Stories

- User stories:
  - We (SMB) used AWS and we had private OpenStack cloud, how can we centralize-manage both of them?
  - We used AWS, RackSpace cloud … how can we migrate data between 2 services on-demand?

From Cloudonomics

- There are hundreds of cloud vendors …
- Due to
  - Business competition
  - Lack of official standards

  → **Each vendor imposes its own stack of technologies**
- Differences among the stacks: hypervisor, networking infrastructure, data storage facilities, management means, …
- Vendor lock-in issue:
  - Lock cloud users into services provided by only one vendor!
  - Can you transfer data and applications to and from the clouds at the same time?

Some critics, such as Richard Stallman*, have called it "a trap aimed at forcing more people to buy into locked, proprietary systems that will cost them more and more over time"

*Richard Stallman is founder of GNU Project and Free Software Foundation

Impacts of Lock-in on Cloud Actors

Cloud Users

High Cost for Poor Services (no choice)

Incompatible Technologies

Cloud Providers

Strategy to avoid customer loss

Promoting Particular Technologies

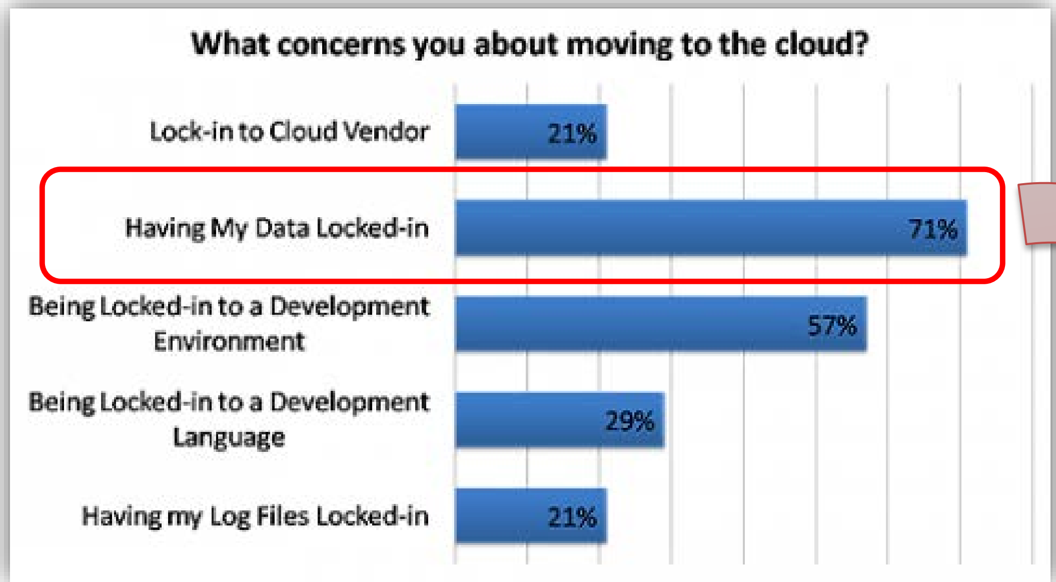
Pricing Power (monopoly)

Cloud Market

Entry Barriers for New Entrants

Detrimental to Cloud Computing Adoption

Vendor Lock-in Figures



Source: RighScale Report [1]

Need of innovative solutions => appear keywords in context of cloud computing

Interoperability

Portability

Federation

Avoid Vendor Lock-in = > More Service Choices => Lower Cost

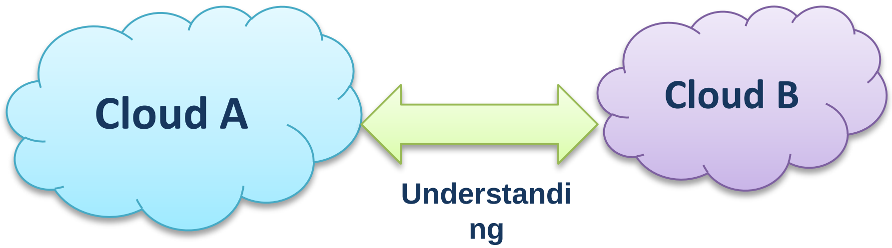
[1] http://www.rightscale.com/blog/cloud-management-best-practices/skinny-cloud-lock

Why Interoperability?

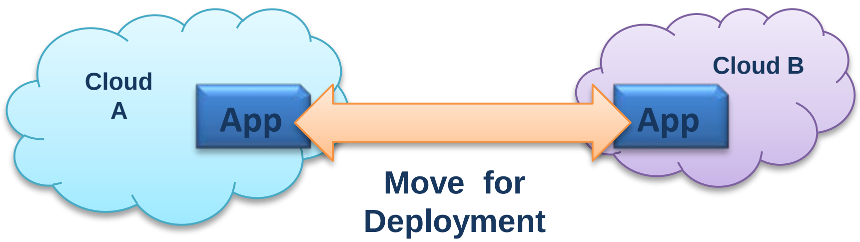
- Avoid vendor lock-in
- Take full advantages of the different clouds
- Develop applications/services once, deploy anywhere
- Open research directions:
  - Enable hybrid clouds
  - Brokering cloud services
  - Cloud service marketplace

Concepts

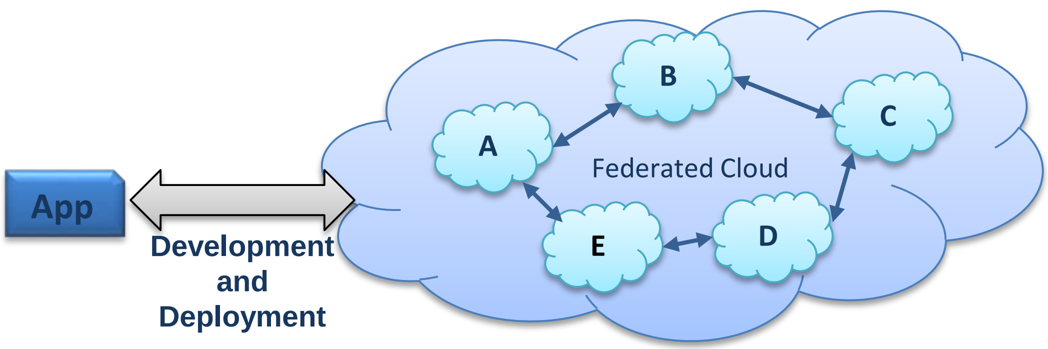
- **Interoperability**: Ability for different cloud to talk to each other



- **Portability**: Ability to move application, data, tools from one cloud to another

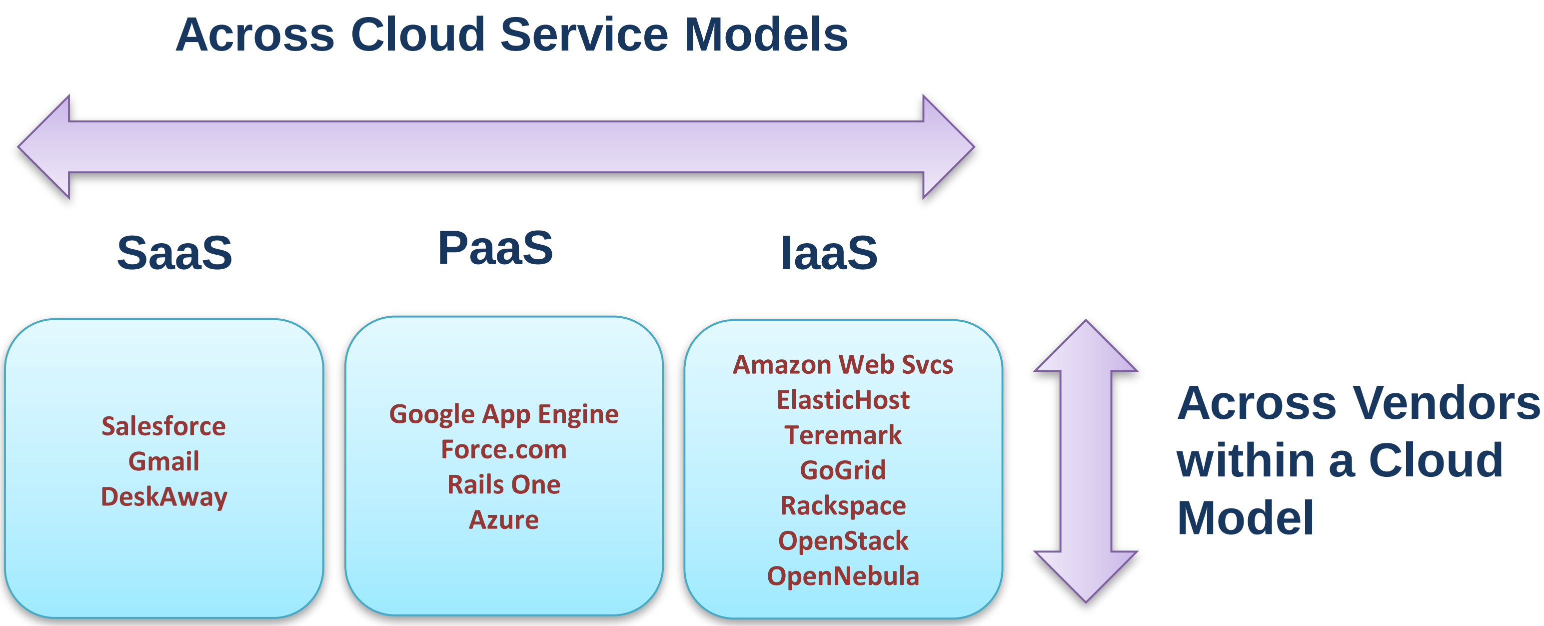


- **Federation**: Ability to bring together services from various cloud vendors to provide a solution



Interoperability between Clouds?

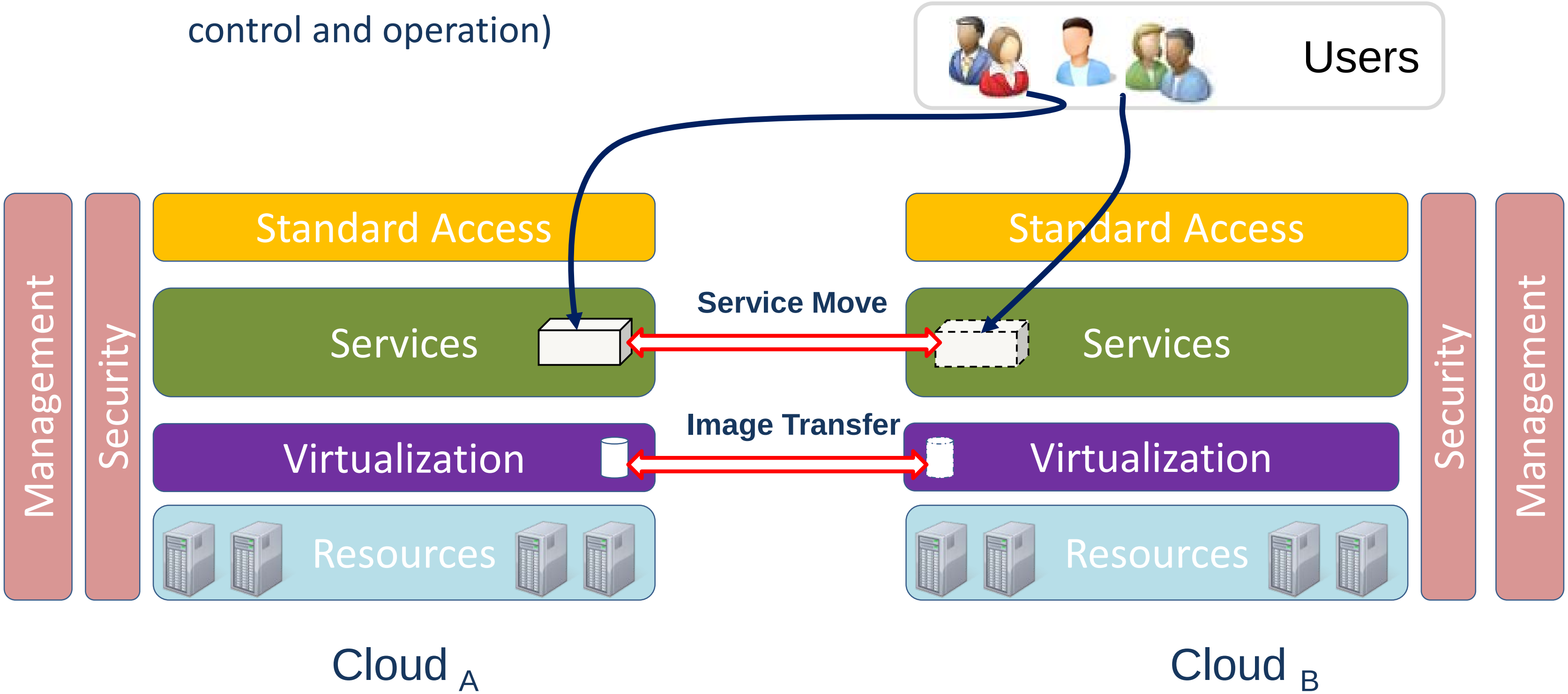
- Ability to use the cloud services provided by multiple vendors
  - Across vendors within a cloud model
  - Across cloud service models
- Ability to move data and code from one cloud to another or back to the enterprise (portability)



A Cloud Standardization? A Solution does not depend on Cloud providers? Or both?

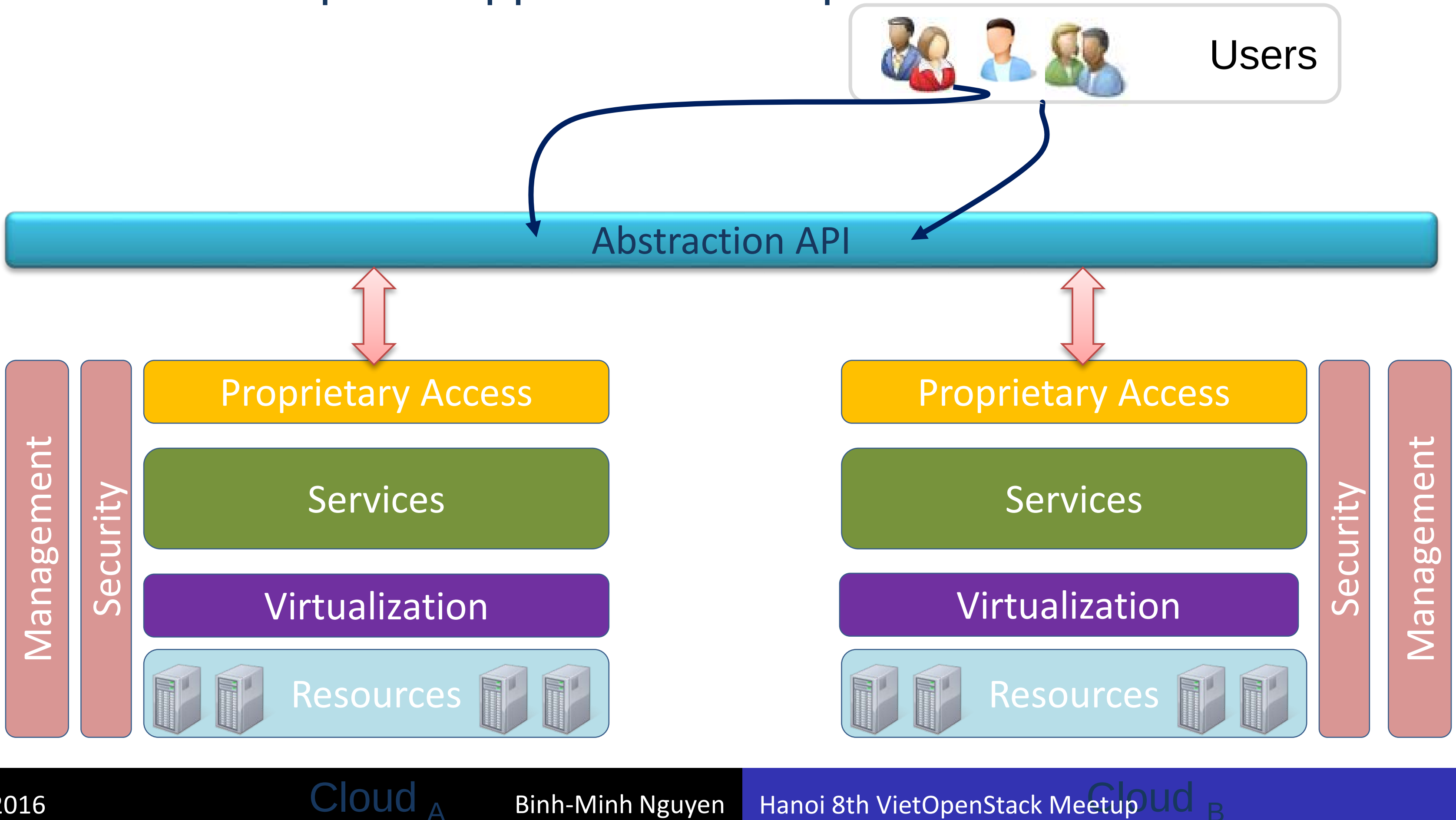
Current Standardization Approaches

- Standard deployment packaging format
  - IaaS level: standard **images** for vendor hypervisors
  - PaaS level: **application packaging** standards for programming languages
- Standard common cloud API
  - For both IaaS and PaaS level: standard **interface** for service managements (access, control and operation)

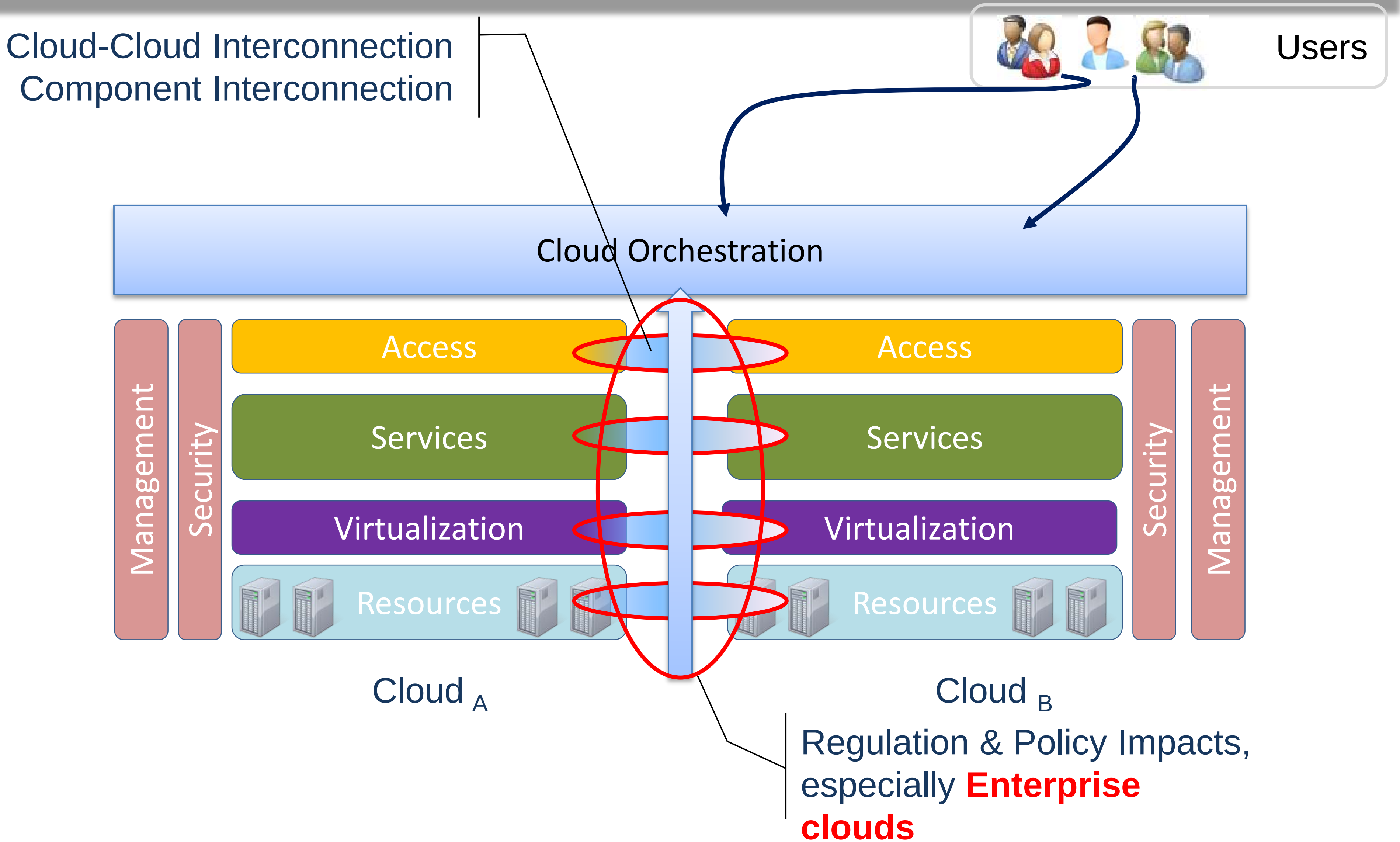


Vendor Independence Approach

- Program library in various language (e.g. PHP, Ruby, Java, Python)
- Abstracting different APIs to provide single unified interface
- Do not require support and acceptance from cloud vendors



Cloud Federation



Standardization + Independent Solutions => More Easily in Building Cloud Federation

Our Motivation

- Actual IaaS clouds are too low-level
  - Cloud users are forced to be admins of their virtual machines and have to install and configure everything by themselves
- PaaS are special purposed and limited to concrete platforms
  - e.g. Google App Engine can be used only for short requests, need to re-implement legacy apps while deploying into clouds

Example

- User: I need to create a cluster with shared home directory and MPI
- Provider: You are admin of your VMs, you can install/configure whatever you want (and do it yourself)
- Options for users:
  - Learn how to install and configure clusters
  - Hire experts (IT support staffs) to do it
  - Use services from third-party companies

Objectives of CAL

- General-purposed easy-to-use interface for cloud users (IaaS)
- Abstraction of cloud resources
- Design complex system and deploy it by single command
- Platform independent, interoperability
- Automatic optimization in background

Design and implementation

- Object-oriented approach: resources are represented by objects
- Inheritance and compound objects for creating complex systems
- Enable default parameters: users have to specify only their special requirements
- Implemented in Python

Abstraction of a Virtual Machine

- Represented by Instance object

```
t = Instance()   // create a default instance
t.start()                // start the instance
t.upload(“myapp.exe, input.dat”, “”)
t.execute(“app.exe input.dat output.dat”)
t.download(“output.dat”)
t.shutdown()
t.delete()
```

Using default parameters

- Users should specify only parameters they need to change

```
t = Instance()   // create a default instance
t = Instance(type=large) // create a strong VM
t = Instance(type=large, os=linux, version=“ubuntu-
12.04”)

// and this is a very concrete machine
t = Instance(image=myimage, keypair=mykeypair,
cloud=openstack)
```

Inheritance and customization

- Via inheritance, developer can create new abstract class for concrete type of virtual machine.
- E.g MySQLServer is an instance with image containing MySQL server, and new method upload_database()

```
MySQLServer: Instance
    __init__
          Instance(image=“mysql-server”)
    config()
          ...............
    upload_database(data)
          Instance.upload(data, “”)
          Instance.exec(“mysql ….”)
```

MySQLServer: Consideration

- Generic images
  - Developers can choose to create new image with MySQL server or use generic images and install mysql-server package:

```
__init__
        t =Instance()   //generic machine
        t.install("mysql-server") //install the package
```

  - Advantage of generic images: maintained by provider/developers, always up-to-date, portability
  - Disadvantages: additional overhead at start

MySQLServer: Consideration

- Code reuse:
  - No need to low-level coding (IP address, manual login to server)
  - Easy to maintain and extend
- Use of the abstract object is very simple

```
m = MySQLServer()
m.start()
m.config()
m.upload_database()
```

Optimization capabilities in background

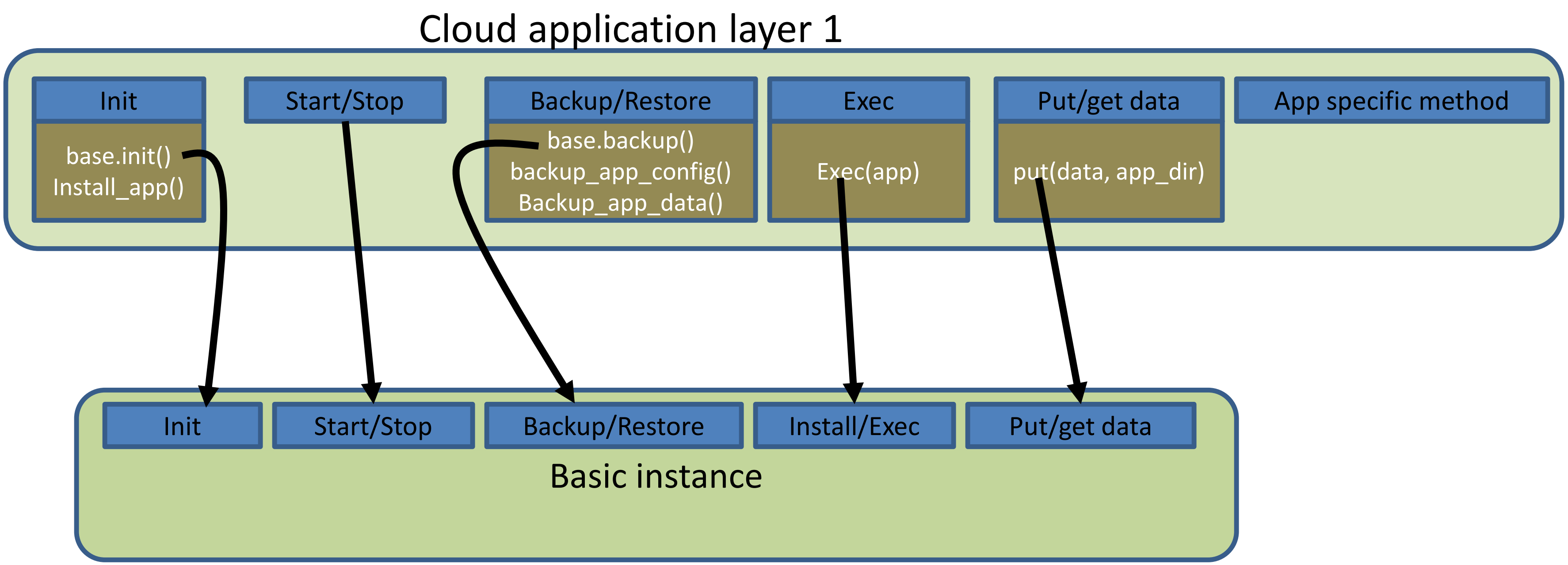
- There are many places we can do optimization in background
  - compress data before transfer to save bandwidth
  - choose best provider (availability, price, …)
  - search and choose suitable images, cloud
- All optimization can be done automatically without user interference

Compound objects: cluster

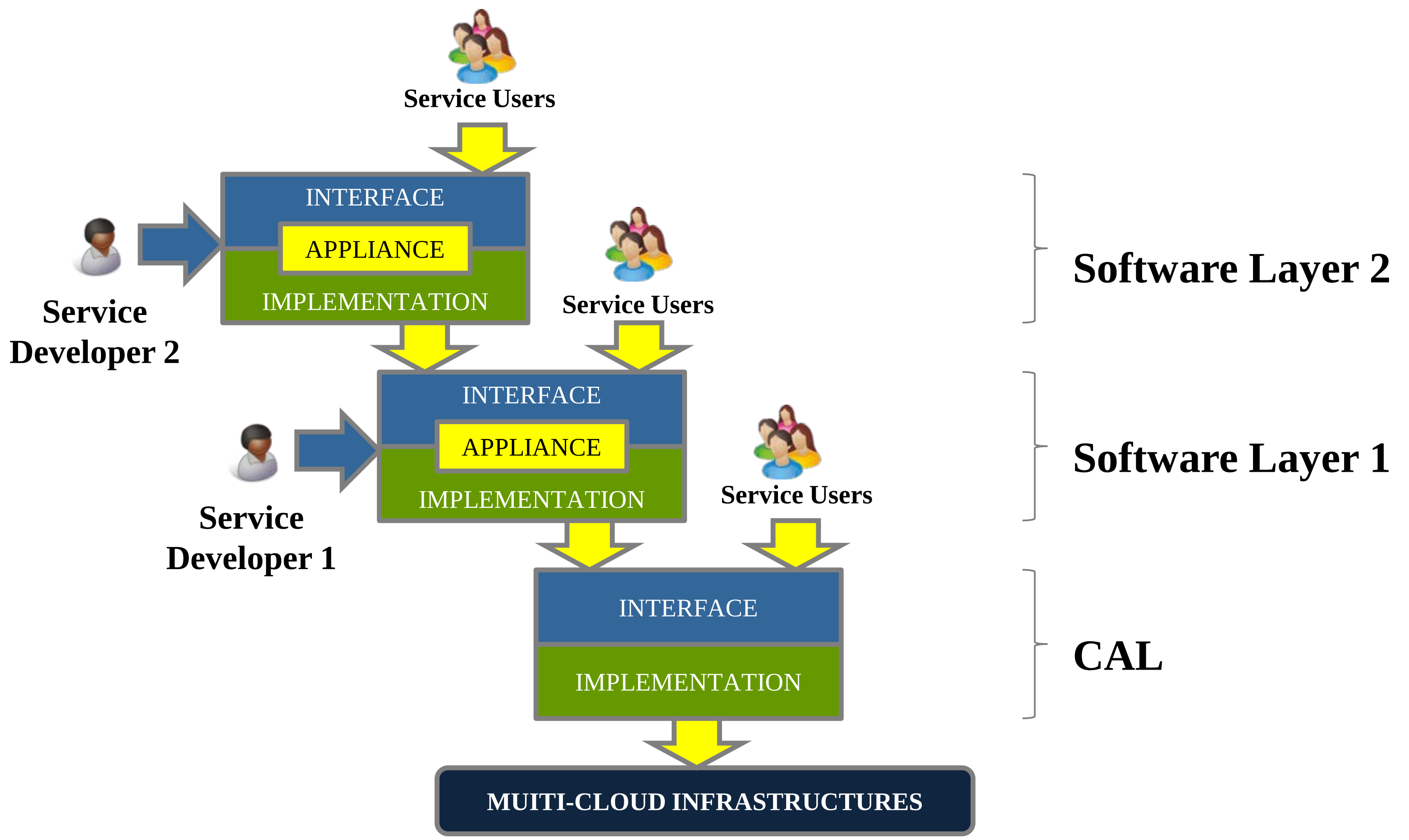
- Complex systems e.g. clusters can be implemented using compound objects:

```
c = Cluster(worker=8)   // create a cluster with
c.start()               // start the cluster
c.upload_and_distribute(“additional_software”, “”)
c.execute(“mpirun ...”)
c.shutdown()
c.delete()
```

Inheritance



Software Layering



Example 1 – Service Development

- **Develop once**

```
class myservice(CAL):                          #inherit CAL functions
     def setCloud(self,cloud):
        CAL.setCloud(self,cloud)               #choose cloud

     def start(self,OS,VM_type):               #start service
        CAL.start(self,OS,VM_type)             #start VM
        CAL.put_data(self,my_app)              #upload app or data
        CAL.execute(self,install_my_app)       #install the app
```

Example 2 – Service Deployment and Usage

- Developers/users choose base software with OS and deploy the installation packages.

```
sv = myservice()
sv.setCloud(OpenStack)          #choose OpenStack driver
sv.start('Ubuntu', 'small')     #start the service
```

- Simple service deployment and use: automatic app. installation after VM start.

Such service can work regardless of cloud middleware or hypervisor (deploy anywhere)
=> enabling service interoperability

Comparison Between CAL and other Solutions

- IaaS tools

Feature \ Solution	CAL	OVF	OCCI	Simple Cloud API	Apache Libcloud	Deltacloud	jclouds	boto	Apache Cloudstack
General approach	A	S	S	A	A	A	A	A	A
Resource management	x		X	X	X	X	X	X	X
Service development	X								
Service deployment	X		x	x	x	x	x		
Interoperability	X	X	x	x	x	x	x	x	x

A – **A**bstraction approach; S –**S**tandardization approach;

X – major feature; x – support feature

Comparison Between CAL and other Solutions (contd.)

- Distributed Computing Configuration Tools

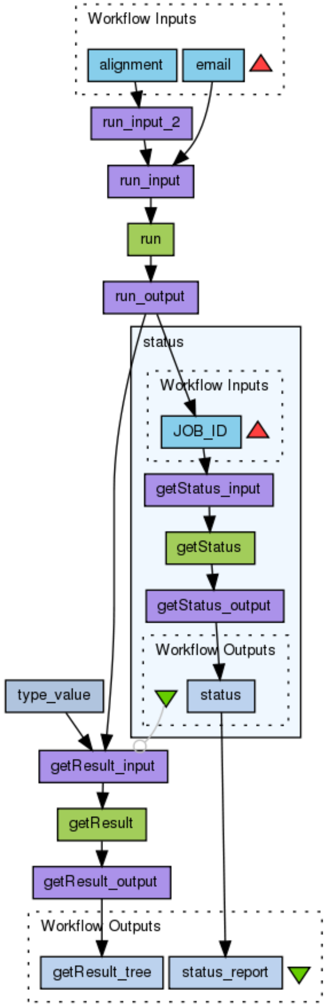
Feature \ Solution	CAL	CFEngine	Puppet	Chef	Bcfg2
Configuring easily legacy applications	X	X	X	X	X
Resource management	x				
Service development and deployment	X	x	x	x	x
Interoperability	X	x	x	x	x

X – major support; x – support feature

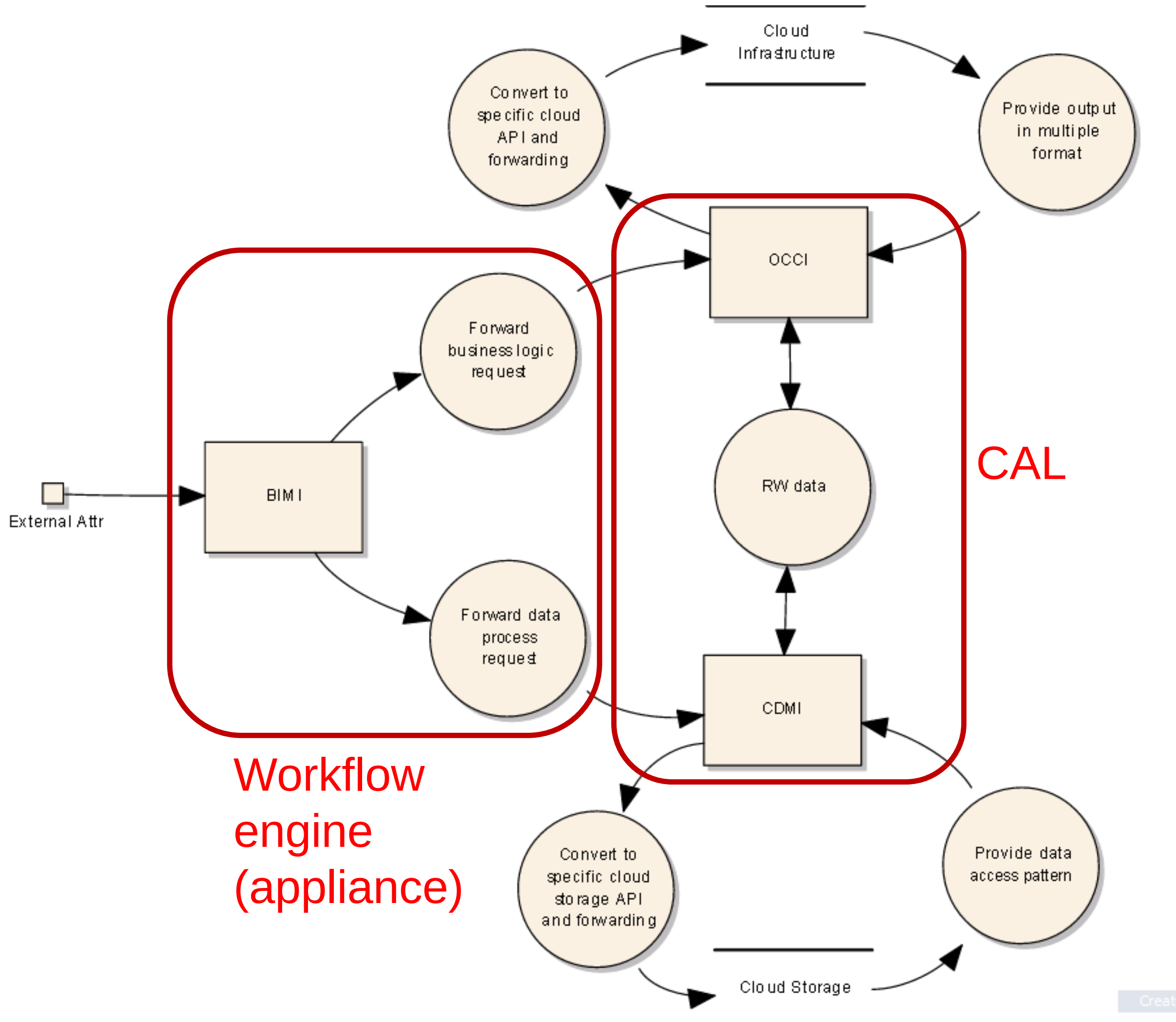
Workflow-as-a-Service

- Prototype model using CAL, aim at migrating formal Bio-Informatics workflow to cloud env.
  - Bio tools: clustalw, BLAST ..
- BIMI: Bio-Informatics Management Interface
- Another approaches for migrating apps into

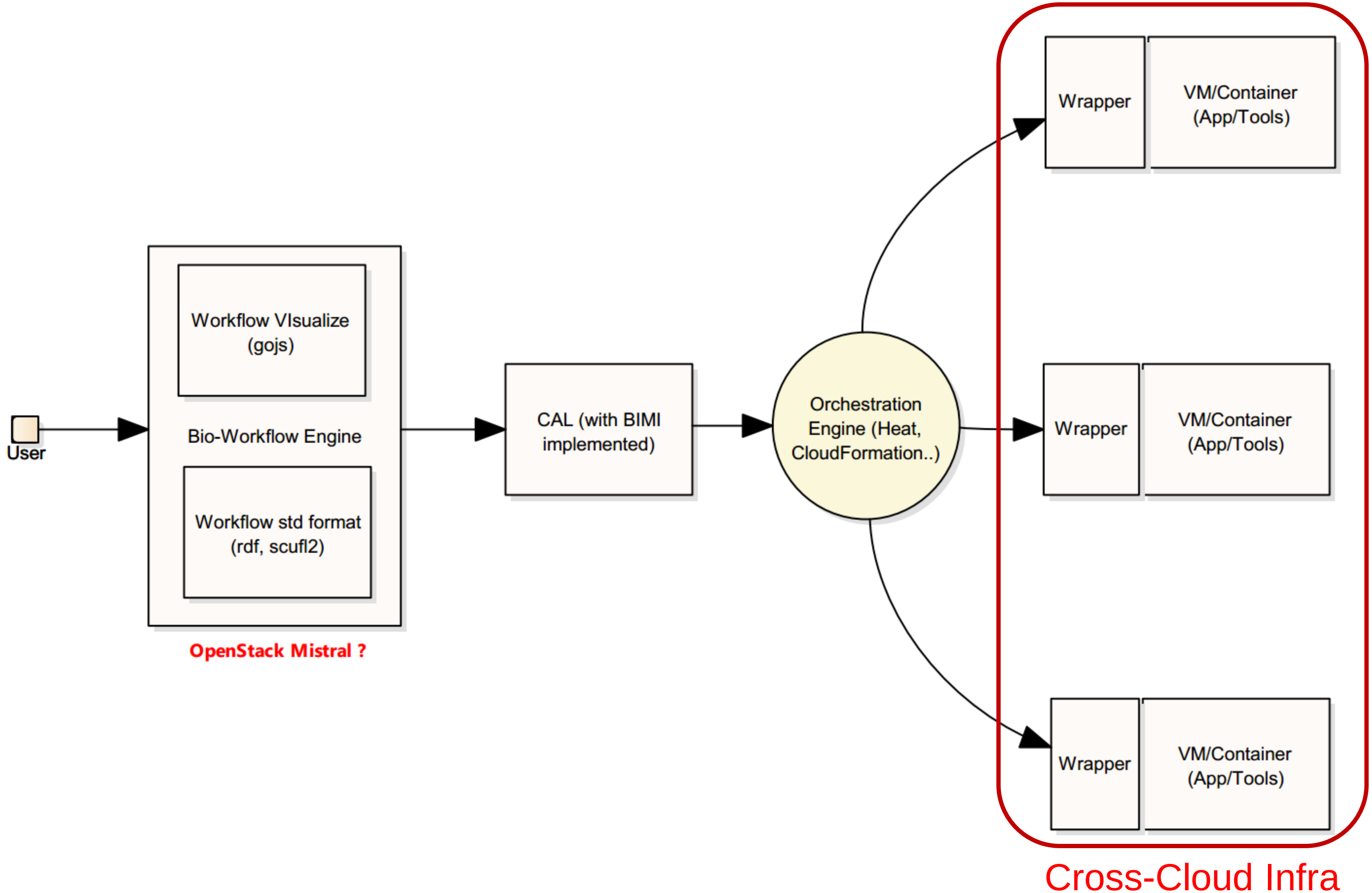
Cloud: Open Service Catalog Manager (OSCM)



BIMI High Level Design



Current Design Status (contd.)



DEMO

Bio-Informatics Workflow-as-a-Service on OpenStack

(Work-in-progress)

Discussion

Thank for your attention!

Q&A

minhnb@soict.hust.edu.vn