REST Case Studies

On the Layers in an IaaS

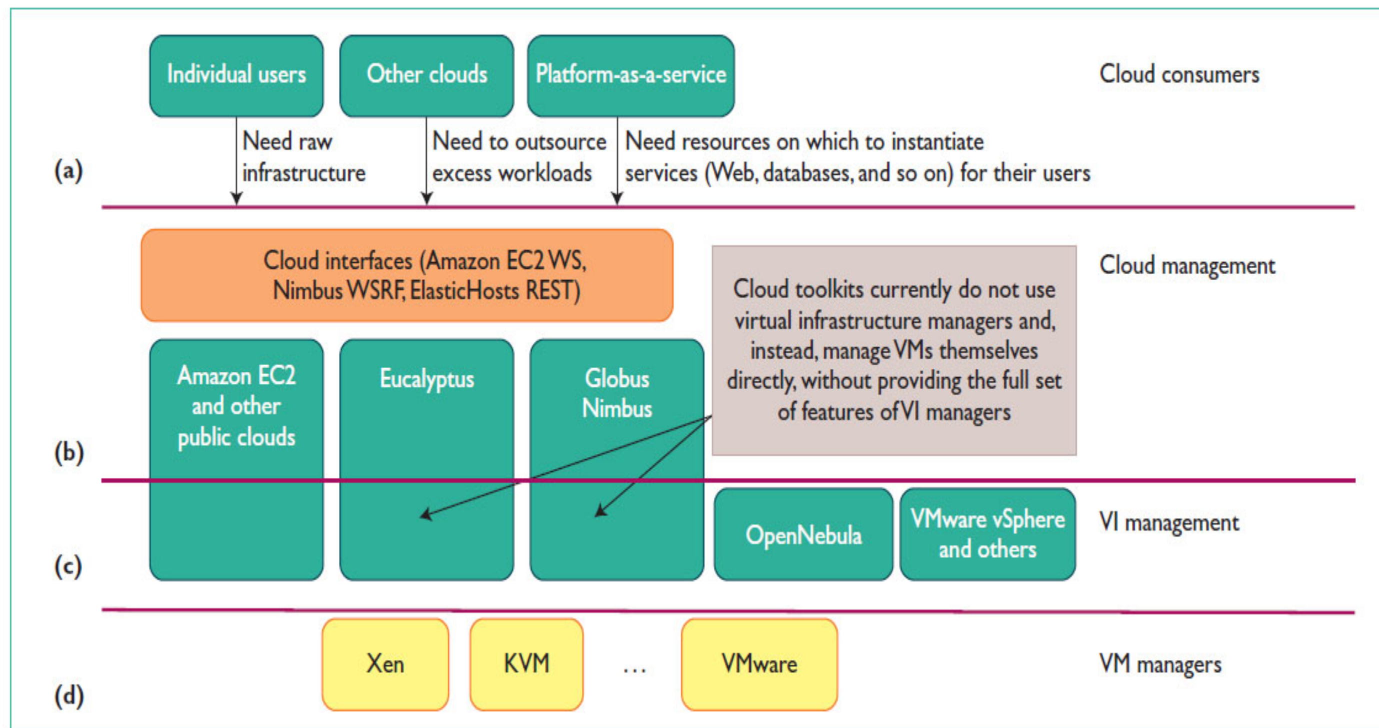


Figure 1. The cloud ecosystem for building private clouds. (a) Cloud consumers need flexible infrastructure on demand. (b) Cloud management provides remote and secure interfaces for creating, controlling, and monitoring virtualized resources on an infrastructure-as-a-service cloud. (c) Virtual infrastructure (VI) management provides primitives to schedule and manage VMs across multiple physical hosts. (d) VM managers provide simple primitives (start, stop, suspend) to manage VMs on a single host.

B. Sotomayor et al., Virtual Infrastructure Management in Private and Hybrid Clouds, IEEE Internet Computing, September/October 2009

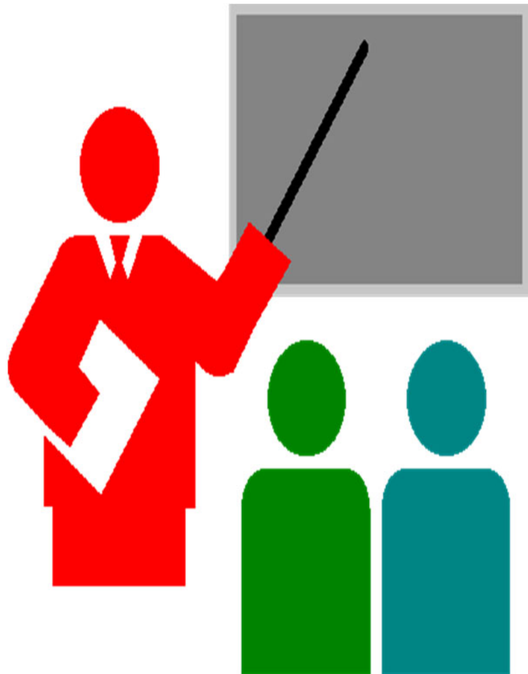
Examples of layers at which REST could be used in IaaS

- **Lowest layer**
  - Access to individual hypervisors / containers

- **Highest layer**
  - Interface between cloud users (e.g. End-user program, PaaS and cloud infrastructure), e.g.
    - OpenStack
    - AWS

Université Concordia University

REST Case studies



- REST for hypervisors (VMWARE)
- REST for containers (Docker)
- REST for cloud IaaS (Openstack)

A Tutorial on using Hypervisors and Containers through REST API

ENCS 691 K

Instructor: Dr. Roch Glitho

Presenters:

Behshid Shayesteh (b_shayes@live.concordia.ca)

Mahsa Raeiszadeh (m_raeisz@encs.concordia.ca)

Outline

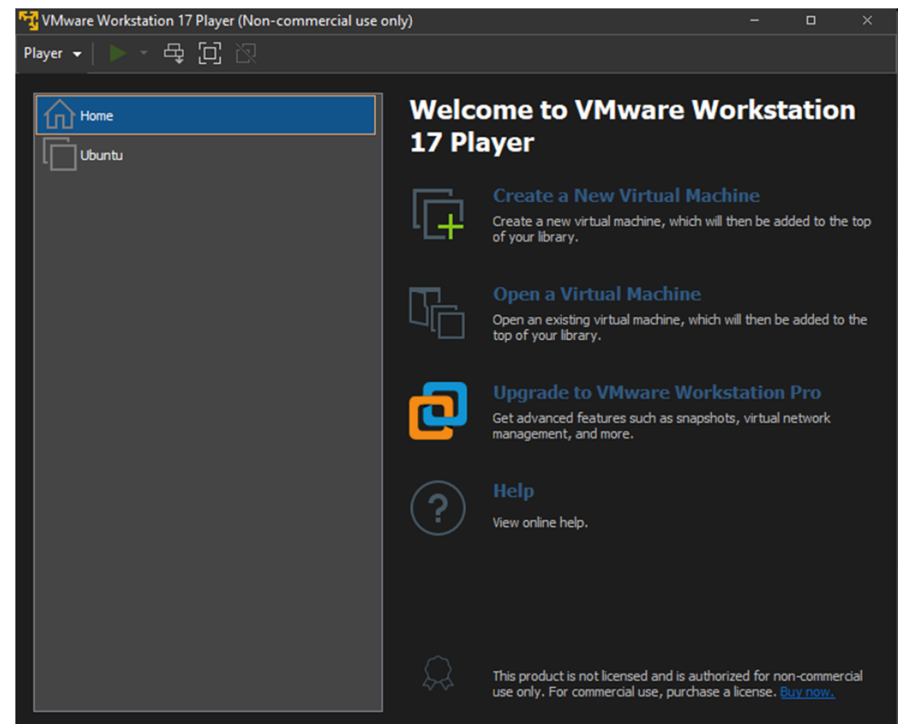
- Part One on Hypervisors
  - Introduction to VM Workstation Player
  - Introduction to REST
  - Setup VMware Workstation Player REST API HTTP server
  - Vmware Player REST API explorer
  - Try calling common VM management APIs using Python

- Part Two on Containers
  - Introduction to Docker API
  - Setup Docker HTTP server
  - Docker REST API explorer
  - Try calling common Docker APIs using Python

Part One: Hypervisors

Introduction to VMware Workstation Player

- VirtualBox does not offer REST-enabled APIs
- Vmware Workstation Player
  - Hypervisor type 2
  - Free product of VMware



Introduction to REST

- REST (Representational State Transfer) is a network architectural style for distributed hypermedia systems
  - A way to reunite the programmable web with the human web
  - Relies on HTTP and inherits its advantages
    - Adressability, statelessness, uniform interface
    - HTTP Interface
      - GET, POST, PUT, DELETE

Setup VMware Workstation Player REST API HTTP server

1. Install Vmware Workstation Player
2. Setup credentials (only first time)
    - In a terminal window, change directories to the Workstation Player installation folder and run the vmrest.exe -C command.
    - Enter a user name and password as prompted.
3. Configure REST API service for HTTP
    - In a terminal window, run the vmrest command. The command returns the IP address and port number from which you can access the HTTP service. The default IP address is 127.0.0.1:8697.
    - Open a web browser and go to http://address-returned-by-vmrest-command.
    - Click Authorize in the top-right corner of the Workstation Player API Explorer page.
    - Enter the user name and password you configured in Step 2.

Vmware Player REST API explorer

VMware Player REST API

vmrest 1.3.0 build-20800274

VM Management — Show/Hide | List Operations | Expand Operations

Method	Path	Description
GET	/vms	Returns a list of VM IDs and paths for all VMs
GET	/vms/{id}	Returns the VM setting information of a VM
GET	/vms/{id}/params/{name}	Get the VM config params
GET	/vms/{id}/restrictions	Returns the restrictions information of the VM
PUT	/vms/{id}	Updates the VM settings
PUT	/vms/{id}/params	update the vm config params
POST	/vms/registration	Register VM to VM Library
DELETE	/vms/{id}	Deletes a VM

VM Power Management — Show/Hide | List Operations | Expand Operations

Method	Path	Description
GET	/vms/{id}/power	Returns the power state of the VM
PUT	/vms/{id}/power	Changes the VM power state

VM Shared Folders Management — Show/Hide | List Operations | Expand Operations

Method	Path	Description
GET	/vms/{id}/sharedfolders	Returns all shared folders mounted in the VM
PUT	/vms/{id}/sharedfolders/{folder id}	Updates a shared folder mounted in the VM
POST	/vms/{id}/sharedfolders	Mounts a new shared folder in the VM
DELETE	/vms/{id}/sharedfolders/{folder id}	Deletes a shared folder

VM Network Adapters Management — Show/Hide | List Operations | Expand Operations

Method	Path	Description
GET	/vms/{id}/ip	Returns the IP address of a VM
GET	/vms/{id}/nic	Returns all network adapters in the VM
GET	/vms/{id}/nicips	Returns the IP stack configuration of all NICs of a VM
PUT	/vms/{id}/nic/{index}	Updates a network adapter in the VM
POST	/vms/{id}/nic	Creates a network adapter in the VM
DELETE	/vms/{id}/nic/{index}	Deletes a VM network adapter

Vmware Player REST API

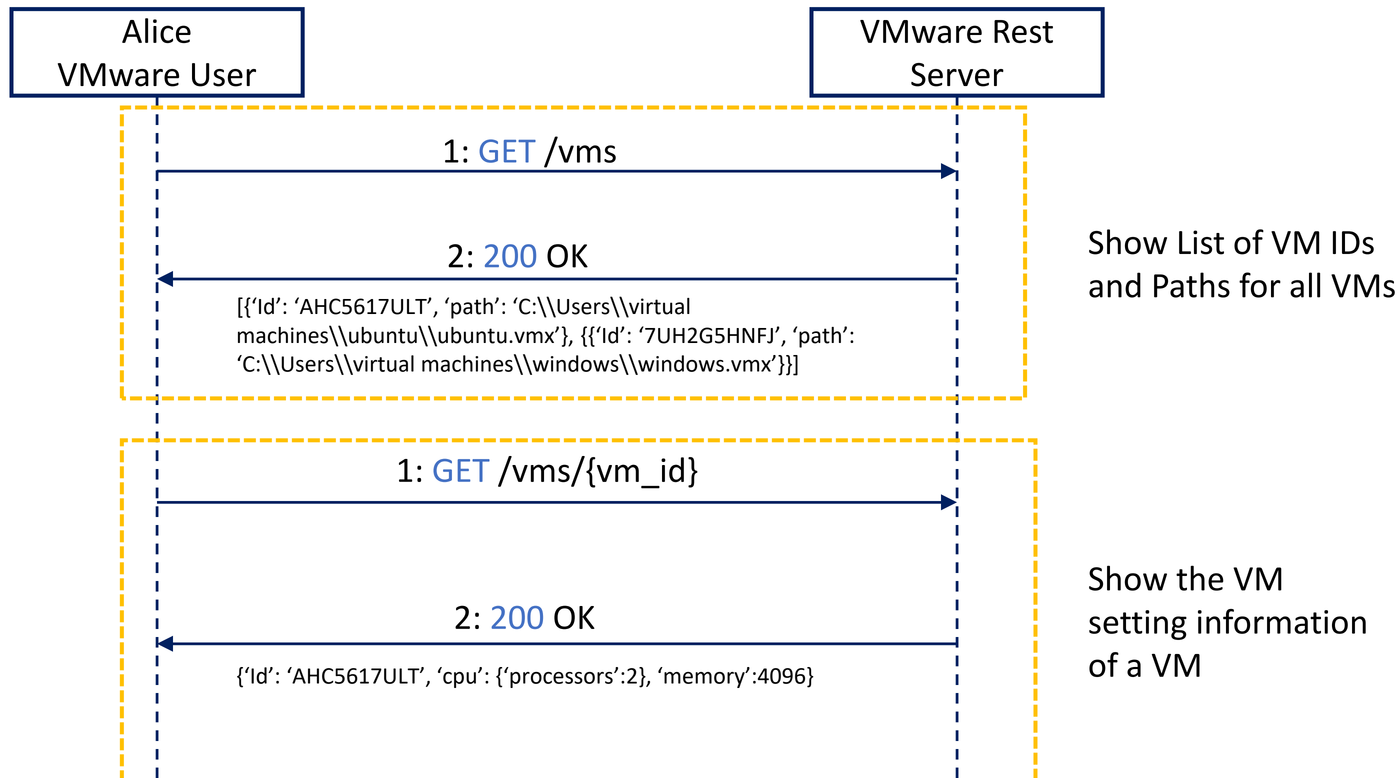
- Datasets
  - VMs
- Resources
  - Each VM is a resource
  - One special resource that lists the VMs

VM management APIs

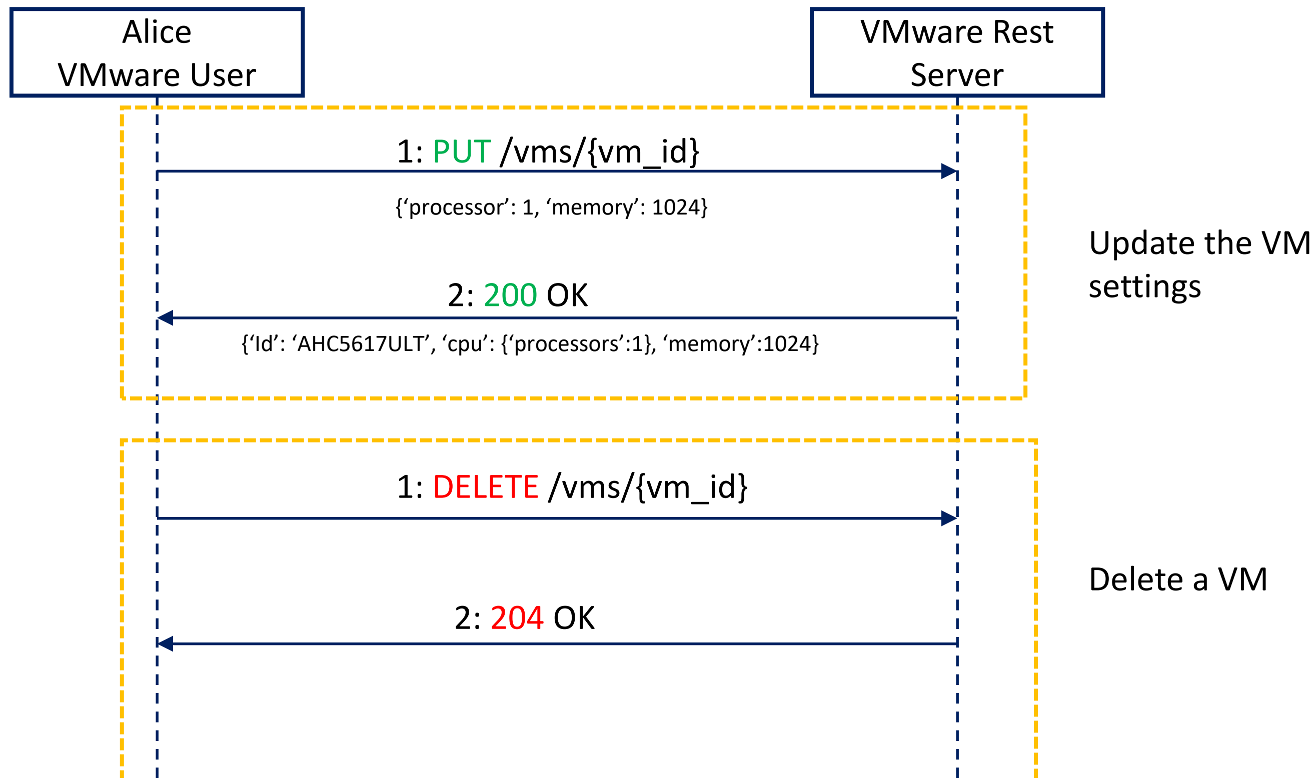
- Common VM management APIs that will be called during this tutorial
  - Get list of exisiting VMs
  - Get the configuration of a specific VM
  - Update the resource configuration of a specific VM
  - Delete a specific VM

- Link to other APIs:
  - https://developer.vmware.com/apis/1042/#api

Interacting with Vmware Player API - Example



Interacting with Vmware Player API - Example



Python Code to interact with Wmware Player API - Example

```python
api_url = "http://localhost:8697/api/vms"
headers = {'Accept': 'application/vnd.vmware.vmw.rest-v1+json',
           'Authorization': 'Basic TOKEN'}
response = requests.get(api_url, headers=headers)
if response.status_code == 200:
    print("Response:")
    print(response.json())
else:
    print(f"Failed to retrieve data: {response.status_code}")
```

Part Two: Containers

Introduction to REST API

- **REST API**
  - REST (Representational State Transfer) is a set of architectural principles for designing networked applications.
  - RESTful APIs allow you to access and manipulate resources over the internet via HTTP methods.
  - Docker Engine, a containerization platform, exposes a RESTful API for container management.

- **Python and Docker:**
  - Python can be used to interact with Docker Engine's REST API to automate container operations.

Docker REST API

- Docker Engine provides a RESTful API that exposes endpoints for container management.
- Key endpoints include /containers, /images, /networks, and more.
- API calls are made using HTTP methods such as GET, POST, PUT, and DELETE.

- To interact with Docker's REST API in Python, you need:
  - Docker Engine installed and running.
  - Python installed on your system.
  - The “request” library for making HTTP requests.

- **Python and request library**
  - The “request” library simplifies making HTTP requests.
  - You can use it to GET, POST, PUT, and DELETE requests to Docker's API endpoints.

Common API Operations

- With Docker's REST API and Python, you can:
  - Create and start containers.
  - Stop and remove containers.
  - Build and manage custom images.
  - Access container logs and statistics.
  - Configure network settings, and more.

- Visit the website below for Docker’s Engine API:
  https://docs.docker.com/engine/api/v1.43/

Docker Engine REST API

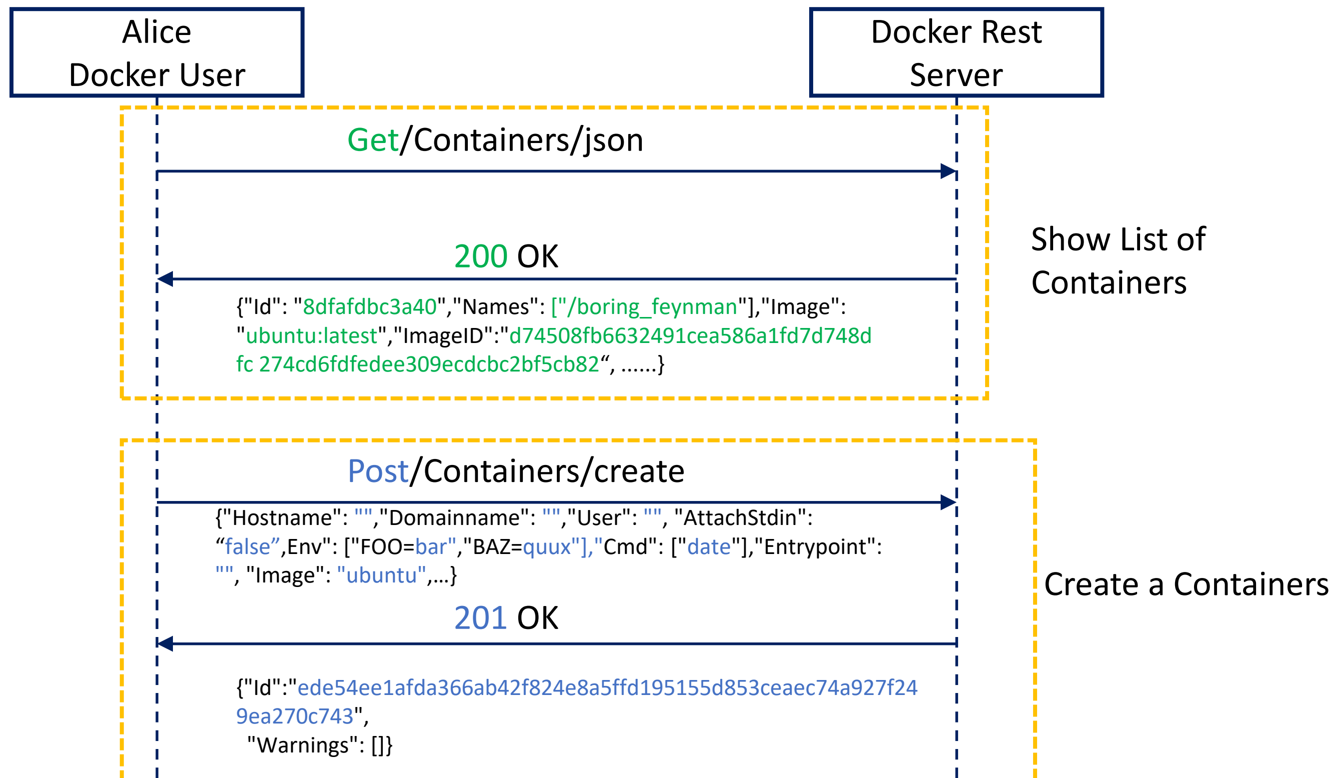
- Datasets
  - Containers
  - Images
  - Networks

- Resources
  - Each container is a resource
  - Each image is a resource
  - Each network is a resource
  - One special resource that lists containers
  - One special image that lists containers
  - One special network that lists containers

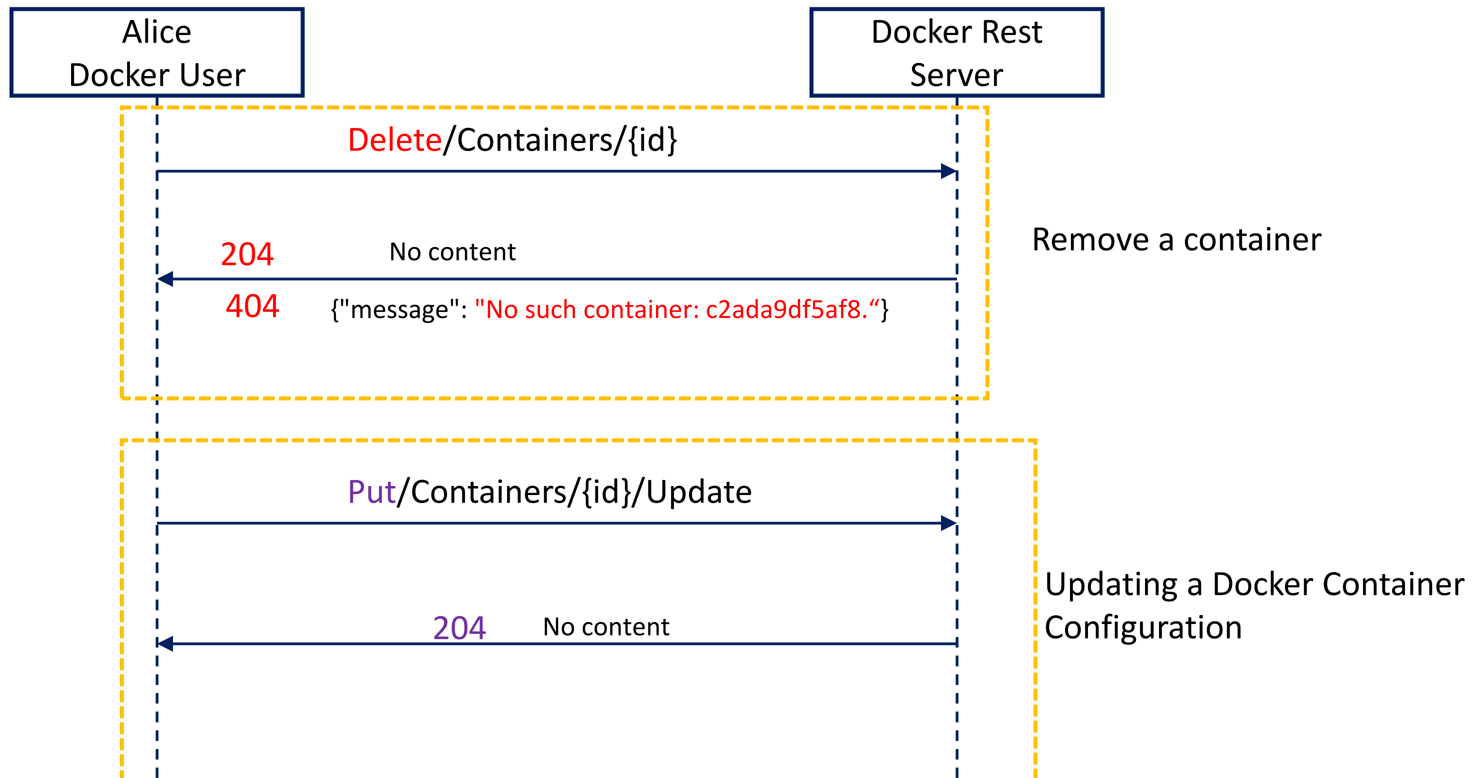
Docker Engine API - Name Resources with URIs

Container URI	Image URI	Network URI
GET `/containers/json` List containers	GET `/images/search` Search an image	GET `/networks` List networks
POST `/containers/create` create a container	POST `/build` build an image	POST `/networks/create` Create a network
PUT `/containers/{id}/archive` Extract an archive of files or folders to a directory in a container	DELETE `/images/{name}` remove an image	DELETE `/networks/{id}` Remove a network
DELETE `/containers/{id}` remove a container		

Interacting with Docker API - Example

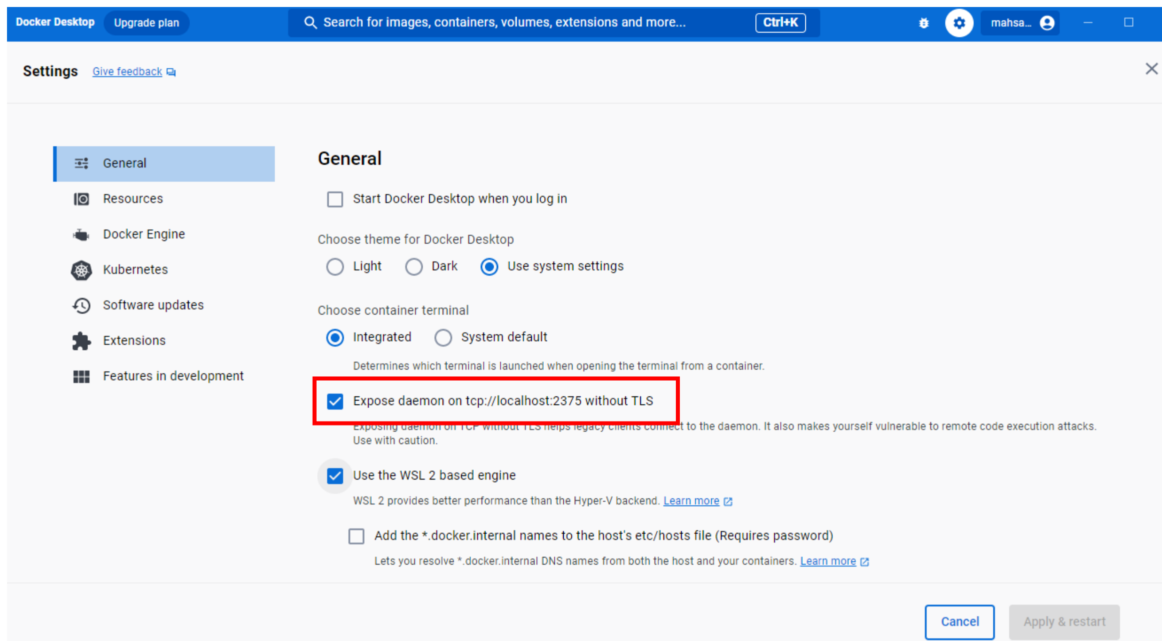


Interacting with Docker API - Example



Enable Docker API Port

- Docker Desktop -> Setting -> General



Example Python code for listing containers

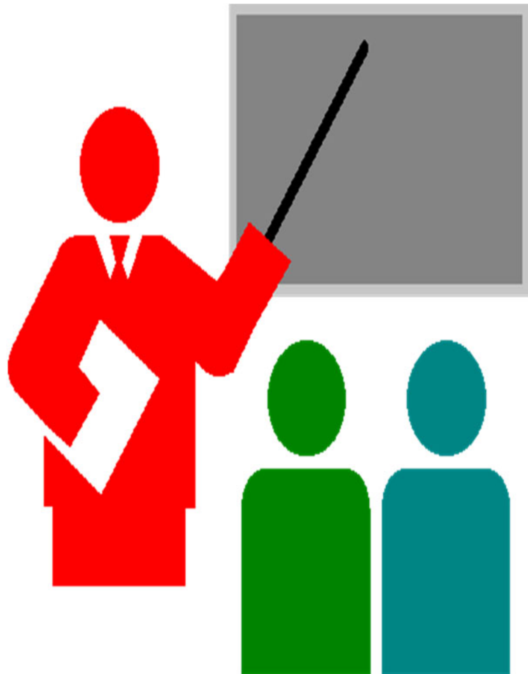
```python
import requests

docker_base_url = "http://localhost:2375"
containers_endpoint = "/v1.43/containers/json"

response = requests.get(f"{docker_base_url}{containers_endpoint}")

if response.status_code == 200:
    containers = response.json()
    for container in containers:
        print(f"Container ID: {container['Id']}")
        print(f"Container Name: {container['Names'][0]}")
        # ... and more container details
```

OpenStack Compute API



- REST Modelling procedure
- OpenStack Compute key concepts
- Applying the procedure

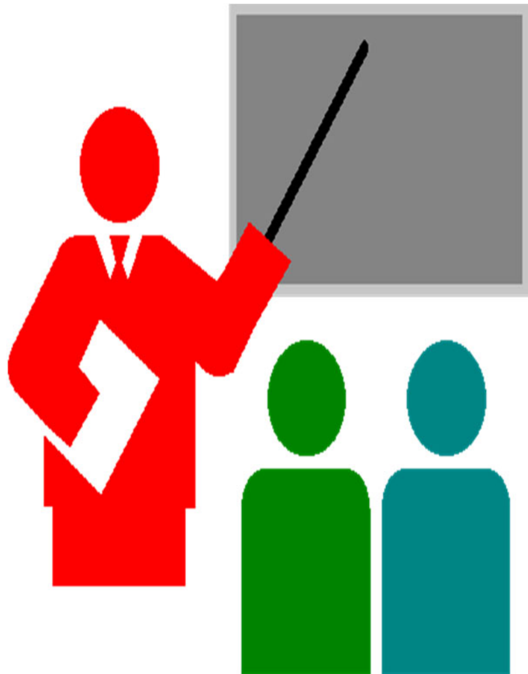
Université Concordia University

Examples of REST Modelling

(OpenStack - Compute)

Note: Slides prepared by Yassine Jebbar, Teaching Assistant

OpenStack Compute API



- REST Modelling procedure
- OpenStack Compute key concepts
- Applying the procedure

Université Concordia University

The procedure – First Part

- Figure out the data set
- Split the data set into resources

The procedure – Second Part

For each resource:

- Name the resources with URIs
- Identify the subset of the uniform interface that is exposed by the resource
- Design the representation(s) as received (in a request) from and sent (in a reply) to the client
- Consider the typical course of events by exploring and defining how the new service behaves and what happens during a successful execution

OpenStack Compute (REST-based) Key Concepts

- OpenStack Compute is a compute service that provides server capacity in the cloud.
- Compute Servers come in different flavors (virtual hardware configuration) of memory, cores, disk space, and CPU, and can be provisioned in minutes.
- Interactions with Compute Servers can happen programmatically with the OpenStack Compute API.

OpenStack Compute Key Concepts

- **Server:** A virtual machine (VM) instance, physical machine or a container in the compute system.

- **Flavor:** Virtual hardware configuration for the requested server. Each flavor has a unique combination of disk space, memory capacity and priority for CPU time.

- **Image:** A collection of files used to create or rebuild a server. Operators provide a number of pre-built OS images by default.

OpenStack Key Concepts

- **Server Management:** Enable all users to perform an action on a server.

Example:
- ➢ Create/Delete/Resize/Reboot Server
- ➢ Show Server(s) Details

- **Flavor Management:** Show and manage server flavors.

Example:
- ➢ Create/Delete/Update Flavor
- ➢ Show Flavor(s) Details

- **Image Management:** Show details and manage images.

Example:
- ➢ List Images
- ➢ Show Image Details
- ➢ Delete Image

Applying the procedure – Data Set

- Servers
- Flavors
- Images

Applying the procedure – Split Data Set into Resources

- Each server is a resource
- Each flavor is a resource
- Each image is a resource
- One special resource that lists servers
- One special resource that lists flavors
- One special resource that lists images

Applying the procedure – Name Resources with URIs

Server URI

Method	URI	Operation
GET	/servers	**List Servers**
POST	/servers	**Create Server**
GET	/servers/detail	**List Servers Detailed**
GET	/servers/ {server_id}	**Show Server Details**
PUT	/servers/ {server_id}	**Update Server**
DELETE	/servers/ {server_id}	**Delete Server**
POST	/servers/ {server_id} /action	**Reboot Server (reboot Action)**
POST	/servers/ {server_id} /action	**Resize Server (resize Action)**

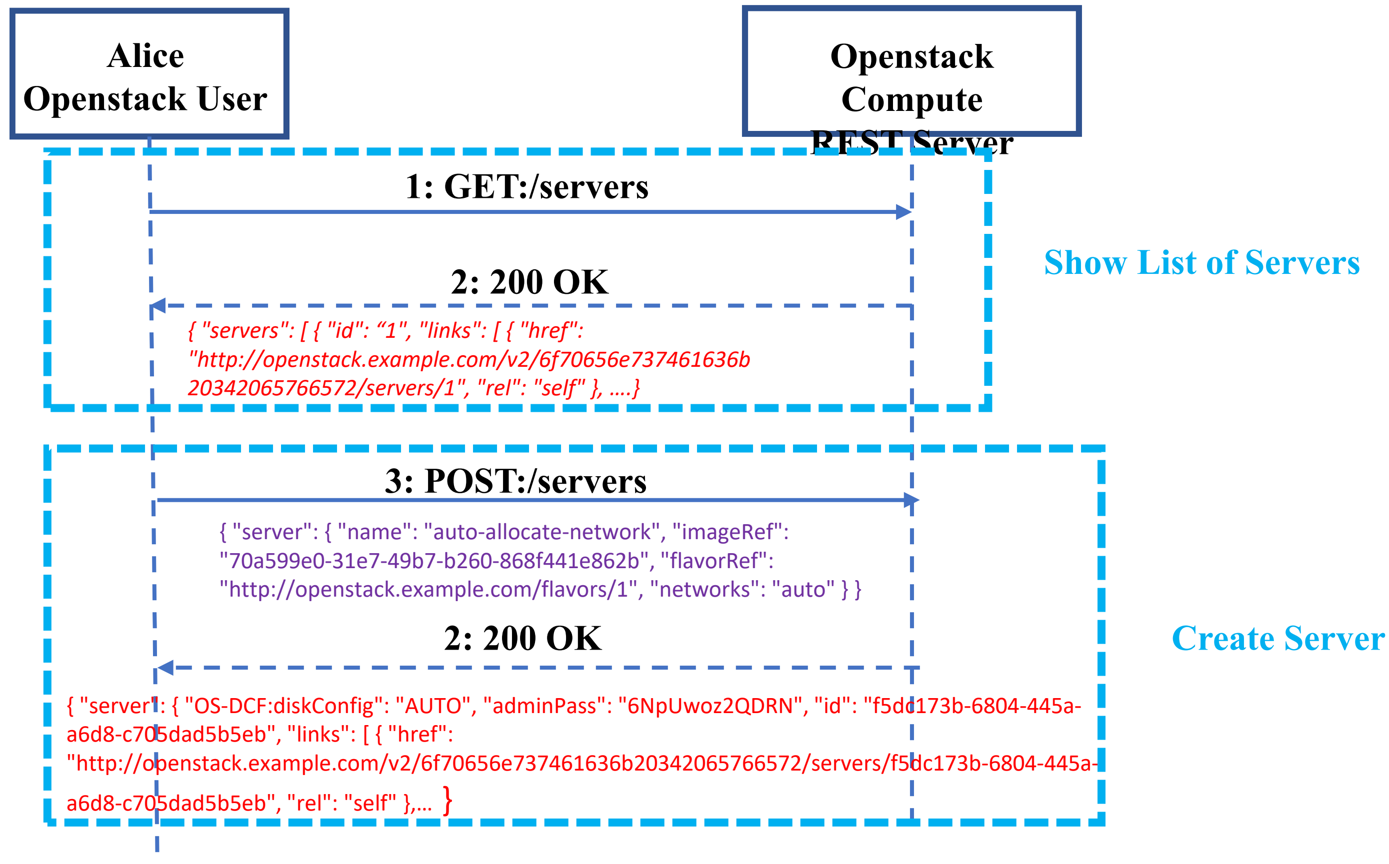
Flavor URI

Method	URI	Operation
GET	/flavors	**List Flavors**
POST	/flavors	**Create Flavor**
GET	/flavors/detail	**List Flavors With Details**
GET	/flavors/ {flavor_id}	**Show Flavor Details**
PUT	/flavors/ {flavor_id}	**Update Flavor Description**
DELETE	/flavors/ {flavor_id}	**Delete Flavor**

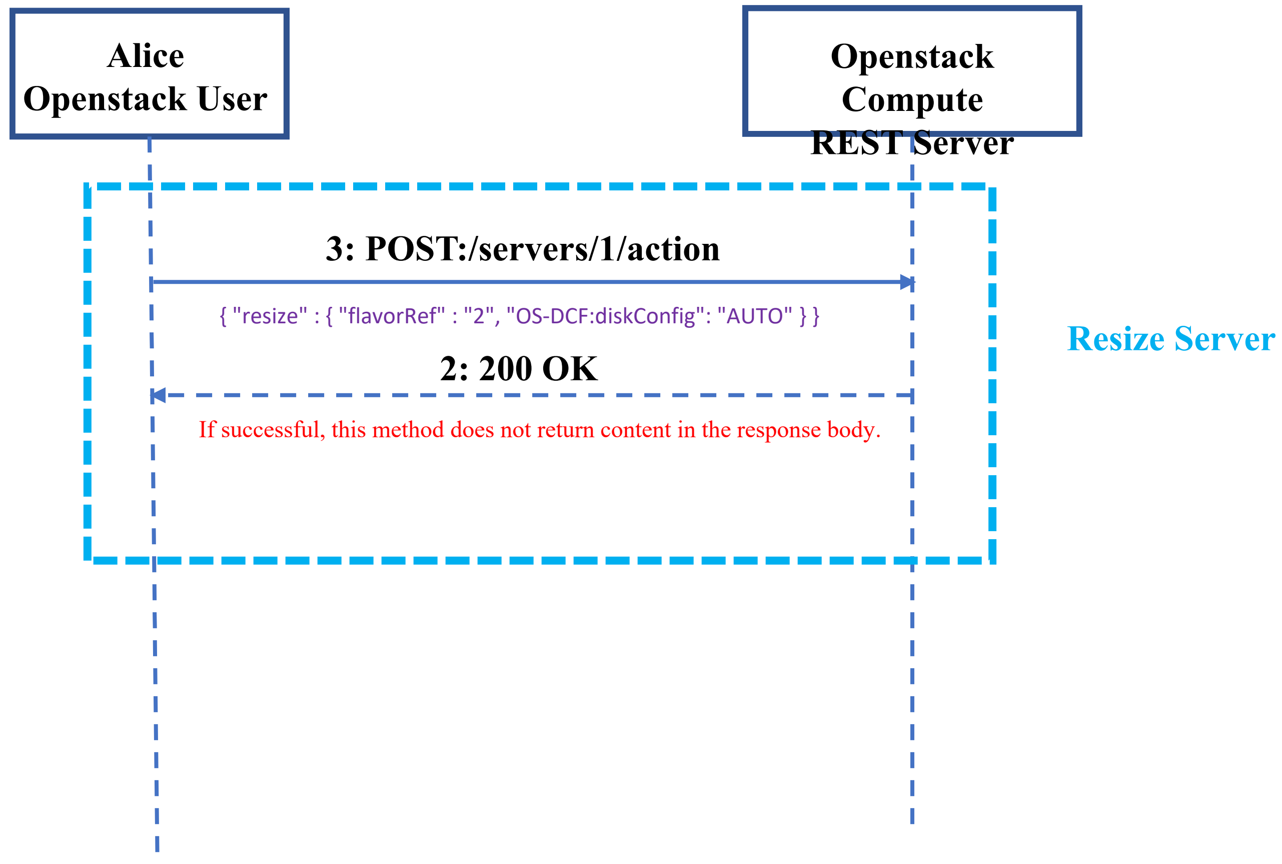
Image URI

Method	URI	Operation
GET	/images/detail	**List Images With Details**
GET	/images/ {image_id}	**Show Image Details**
DELETE	/images/ {image_id}	**Delete Image**

Example: Listing and Creating Server



Example: Resizing Server



References

https://docs.openstack.org/api-guide/compute/general_info.html

https://docs.openstack.org/api-ref/compute/?expanded=

Case Study – REST for Conferencing

http://users.encs.concordia.ca/~glitho/

References

- F. Belqasmi, C. Fu, R. Glitho, Services Provisioning in Next Generation Networks: A Survey, IEEE Communications Magazine, December 2011
- F. Belqasmi, J. Singh, S. Bani Melhem, and R. Glitho, SOAP Based Web Services vs. RESTful Web Services: A Case Study for Multimedia Conferencing Applications, *IEEE Internet Computing*, July/August 2012

Université Concordia University

Examples of REST Modelling
(Messaging)

Examples of RESTful Web Services

Resources	URL Base URL: http://{serverRoot}/{apiVersion}/smsmessaging	HTTP action
Outbound SMS message requests	/outbound/{senderAddress}/requests	GET: read pending outbound message requests POST: create new outbound messages request
Outbound SMS message request and delivery status	/outbound/{senderAddress}/requests /{requestId}	GET: read a given sent message, along with its delivery status
Inbound SMS message subscriptions	/inbound/subscriptions	GET: read all active subscriptions POST: create new message subscription
Individual inbound SMS message subscription	/inbound/subscriptions/{subscriptionId}	GET: read individual subscription DELETE: remove subscription and stop corresponding notifications

Table 2. *A subset of ParlayREST SMS resources.*

Examples of RESTful Web Services

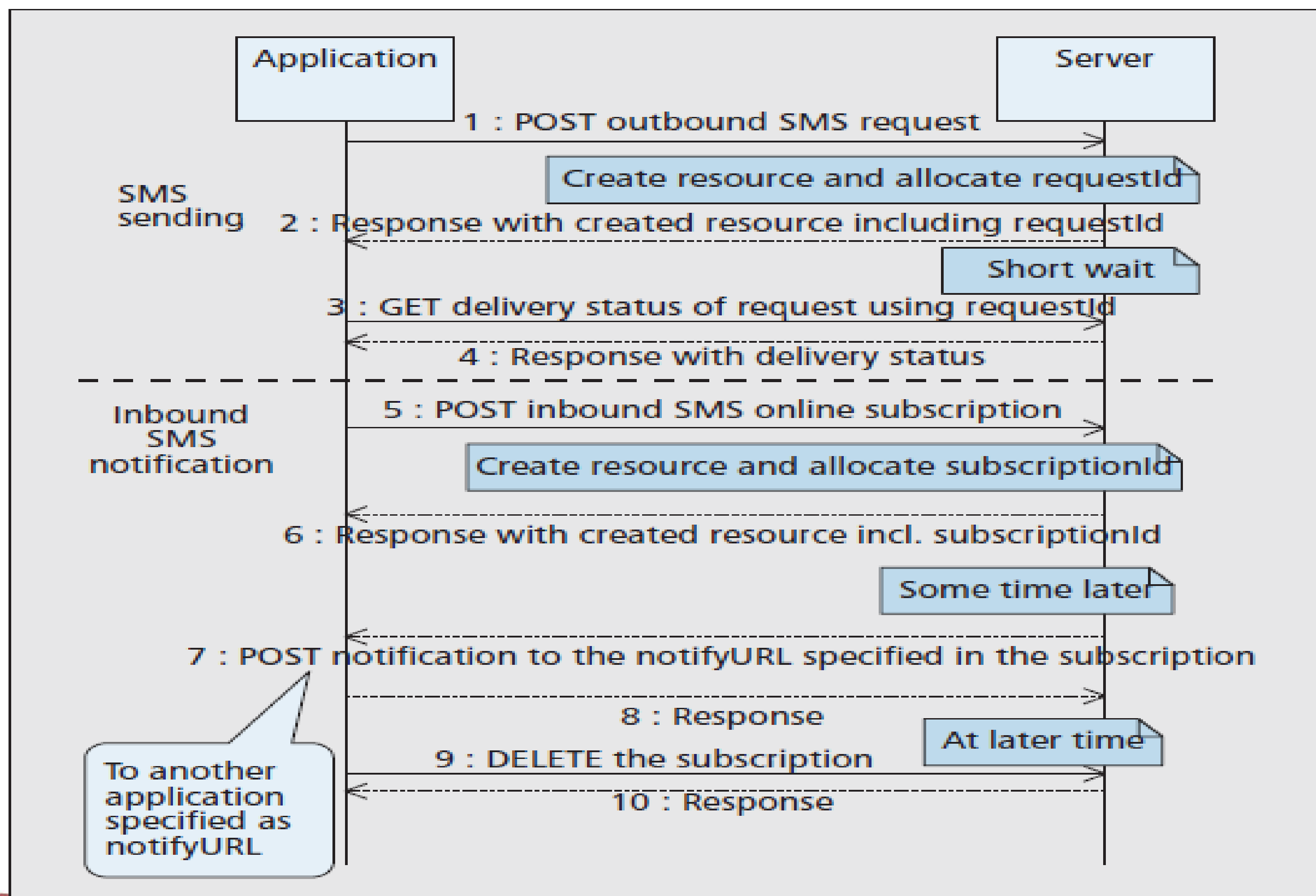
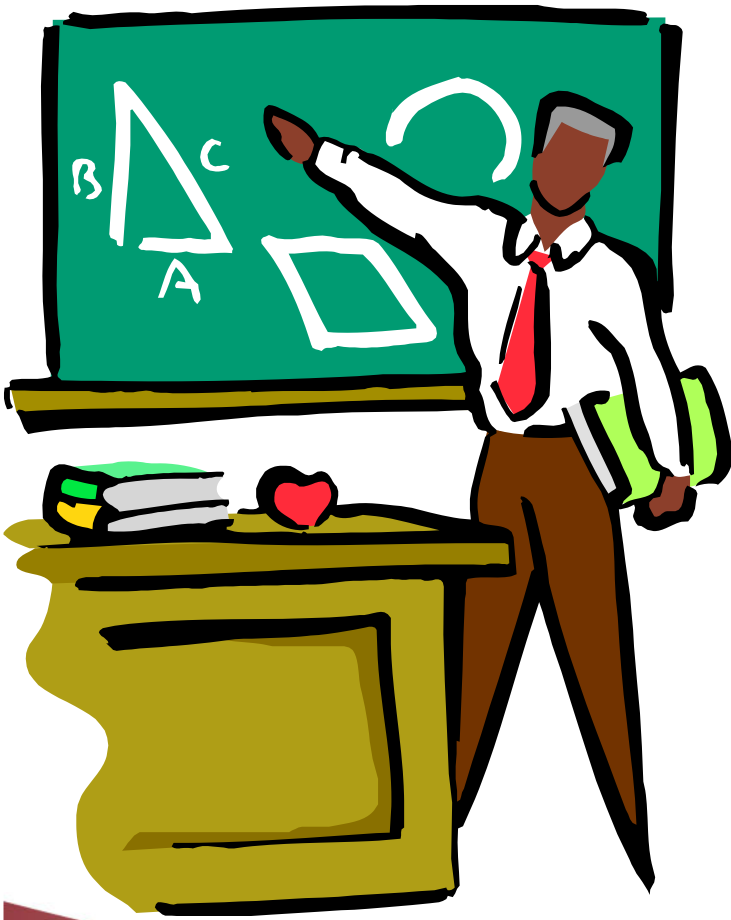


Figure 4. *Sample scenario for SMS handling.*

Examples of REST Modelling
(Conferencing)

Case Study On Conferencing



1. A stepwise procedure
2. On conferencing semantics
3. Applying the procedure to conferencing

The procedure – First Part

- Figure out the data set
- Split the data set into resources

The procedure – Second Part

For each resource:

- Name the resources with URIs
- Identify the subset of the uniform interface that is exposed by the resource
- Design the representation(s) as received (in a request) from and sent (in a reply) to the client
- Consider the typical course of events by exploring and defining how the new service behaves and what happens during a successful execution

On Conferencing semantics

- The conversational exchange of multimedia content between several parties
  - About multimedia
    - Audio, video, data, messaging
  - About participants
    - Any one who wants to participates the conference

On Conferencing semantics

Classification:

- Dial-in / dial-out
- Open/close
- Pre-arranged/ad hoc
- With/without sub-conferencing (i.e. sidebar)
- With/without floor control

Université Concordia University

On conferencing semantics

- Case considered in the use case
  - Create a service that allows a conference manager to :
    - Create a conference
    - Terminate a conference
    - Get a conference status
    - Add users to a conference
    - Remove users from a conference
    - Change media for a participant
    - Get a participant media

Université Concordia University

Applying the procedure – First part

1. Data set

- Conferences
- Participants
- Media

Applying the procedure – First part

2. Split the data set into resources
   - Each conference is a resource
   - Each participant is a resource
   - One special resource that lists the participants
   - One special resource that lists the conferences (if we consider simultaneous conferences)

Université Concordia University

Applying the procedure – Second part

3. Name the resources with URIs

- I'll root the web service at *http://www.confexample.com/*
- I will put the list of conferences at the root URI
- Each conference is defined by its ID: *http://www.confexample.com/{confId}/*
- A conference participants' resources are subordinates of the conference resource:
    - The lists of participants: *http://www.confexample.com/{confId}/participants/*
    - Each participant is identified by his/her URI:

http://www.confexample.com/{confId}/participants/{participantURI}/

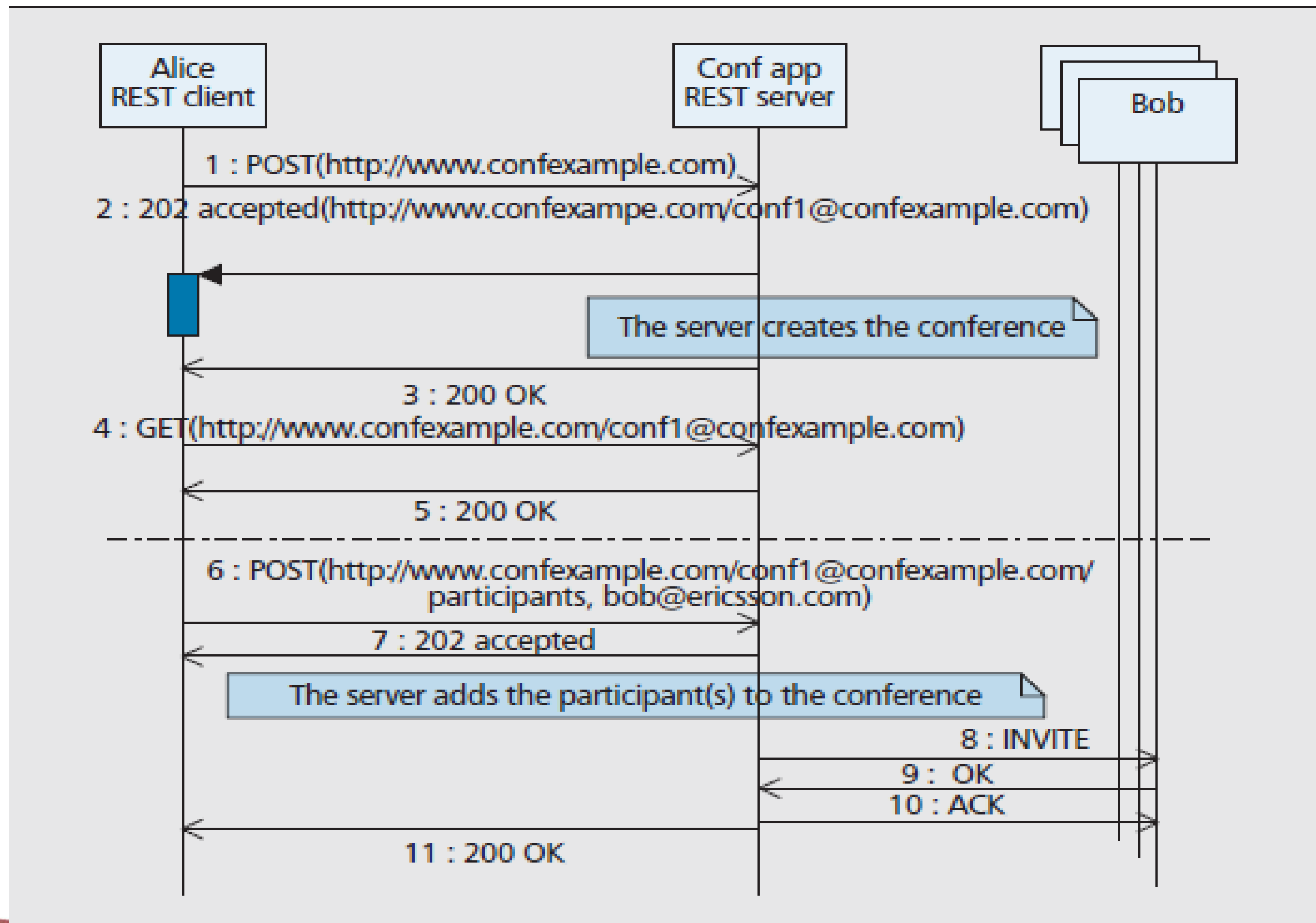
Université Concordia University

Applying the procedure – Second part

Resource	Exposed subset of the uniform interface		Data representation operation	
	Operation	**HTTP action**	**Client->server**	**Server->client**
Conference	Create: establish a conference	POST: http://confexample.com/	<conference> <description> discuss project </description> <maxParticipants>10</maxParticipants> </conference>	http://www.confexample/conf23@example.com
	Read: Get conference status	GET: http://confexample.com/{confId}	None	<status>Active</status>
	Delete: end a conference	DELETE: http://confexample.com/{confId}	None	None
List of participant(s)	Read: Get list of participants	GET: http://confexample.com/{confId}/participants	None	<participants> <participant> <uri>alice@ericsson.com</uri> <status>Connected</status> </participant> .... </participants>
	Create: Add a participant	POST: http://confexample.com/{confId}/participants	<participant> alice@ericsson.com </participant>	<participant> <uri>alice@ericsson.com</uri> <link>http://confexample.com/{confId}/participants/alice@ericsson.com</link> </participant>
	Read: Get a participant status	GET: http://confexample.com/{confId}/participants/{participantURI}	None	<status>Invited</status>
	Delete: remove a participant	DELETE: http://confexample.com/{confId}/participants/{participantURI}	None	None

conference?

Applying the procedure – Second part



Applying the procedure – Second part

9. What might go wrong?
   - Conference

Operation	Server->Client	Way it may go wrong
Create (POST)	Success: 200 OK Failure: 400 Bad Request	The received request is not correct (e.g. has a wrong body)
Read (GET)	Success: 200 OK Failure: 404 Not Found	The targeted conference does not exist
Delete (DELETE)	Success: 200 OK Failure: 404 Not Found	The targeted conference does not exist

Applying the procedure – Second part

9. What might go wrong?
   - Participant(s)

Operation	Server->Client	Way it may go wrong
Create (POST)	Success: 200 OK Failure: 400 Bad Request Failure: 404 Not Found	• The received request is not correct (e.g. has a wrong body) • The target conference does not exist
Read (GET)	Success: 200 OK Failure: 404 Not Found	• The target conference does not exist • The target participant does not exist
Update (PUT)	Success: 200 OK Failure: 400 Bad Request Failure: 404 Not Found	• The received request is not correct • The target conference does not exist • The target participant does not exist
Delete (DELETE)	Success: 200 OK Failure: 404 Not Found	• The target conference does not exist • The target participant does not exist

The End