IM Documentation

Release 1.0.0

Miguel Caballer Fernandez

September 15, 2014

CONTENTS

Contents:

CHAPTER

ONE

ABOUT IM

Cloud infrastructures are becoming an appropriate solution to address the computational needs of scientific applications. However, the use of public or on-premises Infrastructure as a Service (IaaS) clouds require users to have non-trivial system administration skills.

For that, IM is a **tool that ease the access and the usability of IaaS clouds** by automating the VMI selection, deployment, configuration, software installation, monitoring and update of Virtual Appliances. **It supports APIs from a large number of virtual platforms**, making user applications cloud-agnostic. In addition **it integrates a contextualization system** to enable the installation and configuration of all the user required applications providing the user with a fully functional infrastructure.

It is a service that features a **web-based GUI, a XML-RPC API, a REST API and a command-line application**.

CHAPTER

TWO

IM VIDEOS

There are an Infrastructure Manager youtube channel with a set of videos with demos of the functionality of the platform.

Currently there are two videos available, but soon more videos will be uploaded:

The first one shows how to use the IM web interface to launch a Hadoop Cluster with a single click in a OpenNebula on-premise cloud platform and in Amazon EC2.

The second video shows a demo of how to create a cluster with a single click using the IM web interface with the EC3 tool. It also shows how CLUES works to dinamically manage the size of the cluster automatically.

YouTube IM channel

CHAPTER

THREE

IM SERVICE INSTALLATION

3.1 Prerequisites

IM needs at least Python 2.4 to run, as well as the next libraries:

- PLY, Python Lex & Yacc library for python.
- paramiko, ssh2 protocol library for python.
- PyYAML, a YAML parser.
- SOAPpy, a full-featured SOAP library (we know it is not actively supported by upstream anymore).

Also, IM uses Ansible (1.4.2 or later) to configure the infrastructure nodes.

These components are usually available from the distribution repositories. To install them in Debian and Ubuntu based distributions, do:

```
$ apt-get install python-ply python-paramiko python-yaml python-soappy ansible
```

In Red Hat based distributions (RHEL, CentOS, Amazon Linux, Oracle Linux, Fedora, etc.), do:

```
$ yum install python-ply python-paramiko PyYAML SOAPpy ansible
```

Finally, check the next values in the Ansible configuration file `ansible.cfg`, (usually found in `/etc/ansible`):

```
host_key_checking = False
transport = paramiko
record_host_keys = False
```

3.2 Optional Packages

- apache-libcloud 0.15 or later is used in the LibCloud connector.
- boto 2.19.0 or later is used as interface to Amazon EC2. It is available as package named `python-boto` in Debian based distributions. It can also be downloaded from boto GitHub repository. Download the file and copy the boto subdirectory into the IM install path.
- Spring Python framework is needed if the access to XML-RPC API is secured with SSL certificates (see `XMLRCP_SSL`). The Debian package is named `python-springpython`.
- CherryPy is needed if needed to secure the REST API with SSL certificates (see `REST_SSL`). The Debian package is named `python-cherrypy3`.

3.3 Installation

3.3.1 Form Pip

You only have to call the install command of the pip tool with the IM package:

```
$ pip install IM
```

WARNING: In some linux distributions (REL 6 or equivalents) you must unistall the packages python-paramiko and python-crypto before installing the IM with pip.

Pip will install all the pre-requisites needed. So Ansible 1.4.2 or later will be installed in the system. In some cases it will need to have installed the GCC compiler and the python developer libraries ('python-dev' or 'python-devel' packages in main distributions).

You must also remember to modify the ansible.cfg file setting as specified in the REQUISITES section.

3.3.2 Form Source

Once the dependences are installed, just download the tarball of *IM Service* from Download, extract the content and move the extracted directory to the installation path (for instance `/usr/local` or `/opt`):

```
$ tar xvzf IM-0.1.tar.gz
$ sudo chown -R root:root IM-0.1.tar.gz
$ sudo mv IM-0.1 /usr/local
```

Finally you must copy (or link) $IM_PATH/im file to /etc/init.d directory:

```
$ sudo ln -s /usr/local/IM-0.1/im /etc/init.d
```

3.4 Configuration

If you want the IM Service to be started at boot time, do

1. Update the value of the variable `IMDAEMON` in `/etc/init.d/im` file to the path where the IM im_service.py file is installed (e.g. /usr/local/im/im_service.py), or set the name of the script file (im_service.py) if the file is in the PATH (pip puts the im_service.py file in the PATH as default):

   ```
   $ sudo sed -i 's/`IMDAEMON=.*/`IMDAEMON=/usr/local/IM-0.1/im_service.py'/etc/init.d/im
   ```

2. Register the service.

To do the last step on a Debian based distributions, execute:

```
$ sudo update-rc.d im start 99 2 3 4 5 . stop 05 0 1 6 .
```

or the next command on Red Hat based:

```
$ sudo chkconfig im on
```

Alternatively, it can be done manually:

```
$ ln -s /etc/init.d/im /etc/rc2.d/S99im
$ ln -s /etc/init.d/im /etc/rc3.d/S99im
$ ln -s /etc/init.d/im /etc/rc5.d/S99im
```

```
$ ln -s /etc/init.d/im /etc/rc1.d/K05im
$ ln -s /etc/init.d/im /etc/rc6.d/K05im
```

IM reads the configuration from `$IM_PATH/etc/im.cfg`, and if it is not available, does from `/etc/im/im.cfg`. There is a template of `im.cfg` at the directory `etc` on the tarball. The options are explained next.

3.4.1 Basic Options

DATA_FILE
Full path to the data file. The default value is `/etc/im/inf.dat`.

MAX_VM_FAILS
Number of attempts to launch a virtual machine before considering it an error. The default value is 3.

WAIT_RUNNING_VM_TIMEOUT
Timeout in seconds to get a virtual machine in running state. The default value is 1800.

LOG_FILE
Full path to the log file. The default value is `/var/log/im/inf.log`.

LOG_FILE_MAX_SIZE
Maximum size in KiB of the log file before being rotated. The default value is 10485760.

3.4.2 Default Virtual Machine Options

DEFAULT_VM_MEMORY
Default principal memory assigned to a virtual machine. The default value is 512.

DEFAULT_VM_MEMORY_UNIT
Unit used in `DEFAULT_VM_MEMORY`. Allowed values: `K` (KiB), `M` (MiB) and `G` (GiB). The default value is `M`.

DEFAULT_VM_CPUS
Default number of CPUs assigned to a virtual machine. The default value is 1.

DEFAULT_VM_CPU_ARCH
Default CPU architecture assigned to a virtual machine. Allowed values: `i386` and `x86_64`. The default value is `x86_64`.

DEFAULT_MASTERVM_NAME
Default name of virtual machine with the *master* role. The default value is `vmmaster`.

DEFAULT_DOMAIN
Default domain assigned to a virtual machine. The default value is `localdomain`.

3.4.3 Contextualization

CONTEXTUALIZATION_DIR
Full path to the IM contextualization files. The default value is `/usr/share/im/contextualization`.

RECIPES_DIR
Full path to the Ansible recipes directory. The default value is `CONTEXTUALIZATION_DIR/AnsibleRecipes`.

RECIPES_DB_FILE
Full path to the Ansible recipes database file. The default value is `CONTEXTUALIZATION_DIR/recipes_ansible.db`.

MAX_CONTEXTUALIZATION_TIME
Maximum time in seconds spent on contextualize a virtual machine before throwing an error. The default value is 7200.

3.4.4 XML-RPC API

XMLRCP_PORT
Port number where IM XML-RPC API is available. The default value is 8899.

XMLRCP_SSL
If `True` the XML-RPC API is secured with SSL certificates. The default value is `False`.

XMLRCP_SSL_KEYFILE
Full path to the private key associated to the SSL certificate to access the XML-RPC API. The default value is `/etc/im/pki/server-key.pem`.

XMLRCP_SSL_CERTFILE
Full path to the public key associated to the SSL certificate to access the XML-RPC API. The default value is `/etc/im/pki/server-cert.pem`.

XMLRCP_SSL_CA_CERTS
Full path to the SSL Certification Authorities (CA) certificate. The default value is `/etc/im/pki/ca-chain.pem`.

3.4.5 REST API

ACTIVATE_REST
If `True` the REST API is activated. The default value is `False`.

REST_PORT
Port number where REST API is available. The default value is 8800.

REST_SSL
If `True` the REST API is secured with SSL certificates. The default value is `False`.

REST_SSL_KEYFILE
Full path to the private key associated to the SSL certificate to access the REST API. The default value is `/etc/im/pki/server-key.pem`.

REST_SSL_CERTFILE
Full path to the public key associated to the SSL certificate to access the REST API. The default value is `/etc/im/pki/server-cert.pem`.

REST_SSL_CA_CERTS
Full path to the SSL Certification Authorities (CA) certificate. The default value is `/etc/im/pki/ca-chain.pem`.

CHAPTER

FOUR

RESOURCE AND APPLICATION DESCRIPTION LANGUAGE (RADL)

The main purpose of the *Resource and Application description Language* (RADL) is to specify the requirements of the scientific applications needed to be deployed in a virtualized computational infrastructure (cloud). Using a declarative scheme RADL considers distinct features related to

- hardware, like CPU number, CPU architecture, and RAM size;
- software, like applications, libraries and data base systems;
- network, like network interface and DNS configuration; and
- contextualization, extra steps to set up an adequate environment for the application.

RADL is intended to be more abstract that other standards to specify virtual appliances, like OVF, and easily extensible with other tools, like contextualization languages such as Ansible.

4.1 Basic structure

An RADL document has the next general structure:

```
network <network_id> (<features>)

system <system_id> (<features>)

configure <configure_id> (<Ansible recipes>)

contextualize [max_time] (
  system <system_id> configure <configure_id> [step <num>]
  ...
)

deploy <system_id> <num> [<cloud_id>]
```

The keywords `network`, `system` and `configure` assign some *features* or *recipes* to an identity `<id>`. The features are a list of constrains separated by `and`, and a constrain is form by `<feature name> <operator> <value>`. For instance:

```
system tomcat_node (
   memory.size >= 1024M and
   disk.0.applications contains (name='tomcat')
)
```

this RADL defines a *system* with the feature `memory.size` greater or equal than `1024M` and with the feature `disk.0.applications` containing an element with `name` `tomcat`.

The sentences under the keyword `contextualize` indicate the recipes that will be executed during the deployment of the virtual machine.

The `deploy` keyword is a request to deploy a number of virtual machines. Some identity of a cloud provider can be specified.

4.2 Use Cases

RADL is not limited to deploy different configurations of virtual machines easily. In many applications infrastructures need management during their life cycle, like deploying virtual machines with new features, changing the features of already deployed virtual machine and undeploying some of them. Next we detail valid RADL examples for every use.

4.2.1 Create a New Infrastructure

A common RADL defines a network and at least one kind of virtual machine and deploys some virtual machines. However the minimum RADL document to create an infrastructure is an empty one.

4.2.2 Add New Definitions

After the creation of the infrastructure, new networks, systems and recipes can be defined. The new definitions can refer to already defined elements, but they must be mentioned. For instance, an infrastructure is created as:

```
network net (outbound = 'no')
system small_node (
   cpu.arch = 'x86_64' and
   cpu.count = 1 and
   memory.size >= 512M and
   net_interface.0.connection = 'net' and
   disk.0.os.name = 'linux'
)
```

A new system with more memory and CPUs, and in the same network can be defined as:

```
network net
system big_node (
   cpu.arch = 'x86_64' and
   cpu.count = 4 and
   memory.size >= 3G and
   net_interface.0.connection = 'net' and
   disk.0.os.name = 'linux'
)
```

4.2.3 Deploy New Virtual Machines

In the same way, new virtual machines from already defined systems can deployed. For instance, this example deploys one `small_node` and other `big_node`:

```
system small_node
system big_node

deploy small_node 1
deploy big_node 1
```

4.3 Network Features

Under the keyword `network` there are the features describing a Local Area Network (LAN) that some virtual machines can share in order to communicate to themselves and to other external networks. The supported features are:

outbound = yes|no Indicate whether the IP that will have the virtual machines in this network will be public (accessible from any external network) or private. If `yes`, IPs will be public, and if `no`, they will be private. The default value is `no`.

4.4 System Features

Under the keyword `system` there are the features describing a virtual machine. The supported features are:

image_type = vmdk|qcow|qcow2|raw Constrain the virtual machine image disk format.

virtual_system_type = '<hypervisor>-<version>' Constrain the hypervisor and the version used to deploy the virtual machine.

price <=|=|=> <positive float value> Constrain the price per hour that will be paid, if the virtual machine is deployed in a public cloud.

cpu.count <=|=|=> <positive integer value> Constrain the number of virtual CPUs in the virtual machine.

cpu.arch = i686|x86_64 Constrain the CPU architecture.

cpu.performance <=|=|=> <positive float value>ECU|GCEU Constrain the total computational performance of the virtual machine.

memory.size <=|=|=> <positive integer value>B|K|M|G Constrain the amount of *RAM* memory (principal memory) in the virtual machine.

net_interface.<netId> Features under this prefix refer to virtual network interface attached to the virtual machine.

net_interface.<netId>.connection = <network id> Set the virtual network interface is connected to the LAN with ID `<network id>`.

net_interface.<netId>.ip = <IP> Set a static IP to the interface, if it is supported by the cloud provider.

net_interface.<netId>.dns_name = <string> Set the string as the DNS name for the IP assigned to this interface. If the string contains `#N#` they are replaced by a number that is distinct for every virtual machine deployed with this `system` description.

disk.<diskId>.<feature> Features under this prefix refer to virtual storage devices attached to the virtual machine. `disk.0` refers to system boot device.

disk.<diskId>.image.url = <url> Set the source of the disk image. The URI designates the cloud provider:

- `one://<server>:<port>/<image-id>`, for OpenNebula;
- `ost://<server>:<port>/<ami-id>`, for OpenStack; and
- `aws://<region>/<ami-id>`, for Amazon Web Service.

Either `disk.0.image.url` or `disk.0.image.name` must be set.

disk.<diskId>.image.name = <string> Set the source of the disk image by its name in the VMRC server. Either `disk.0.image.url` or `disk.0.image.name` must be set.

disk.<diskId>.type = swap|iso|filesystem Set the type of the image.

disk.<diskId>.device = <string> Set the device name, if it is disk with no source set.

disk.<diskId>.size = <positive integer value>B|K|M|G Set the size of the disk, if it is a disk with no source set.

disk.0.free_size = <positive integer value>B|K|M|G Set the free space available in boot disk.

disk.<diskId>.os.name = linux|windows|mac os x Set the operating system associated to the content of the disk.

disk.<diskId>.os.flavour = <string> Set the operating system distribution, like `ubuntu`, `centos`, `windows xp` and `windows 7`.

disk.<diskId>.os.version = <string> Set the version of the operating system distribution, like `12.04` or `7.1.2`.

disk.0.os.credentials.username = <string> and disk.0.os.credentials.password = <string> Set a valid username and password to access the operating system.

disk.0.os.credentials.public_key = <string> and disk.0.os.credentials.private_key = <string> Set a valid public-private keypair to access the operating system.

disk.<diskId>.applications contains (name=<string>, version=<string>, preinstalled=yes|no) Set that the disk must have installed the application with name `name`. Optionally a version can be specified. Also if `preinstalled` is `yes` the application must have already installed; and if `no`, the application can be installed during the contextualization of the virtual machine if it is not installed.

4.5 Configure Recipes

Contextualization recipes are specified under the keyword `configure`. Only Ansible recipes are supported currently. They are enclosed between the tags `@begin` and `@end`, like that:

```
configure add_user1 (
@begin
---
  - tasks:
    - user: name=user1   password=1234
@end
)
```

To easy some contextualization tasks, IM publishes a set of variables that can be accessed by the recipes and have information about the virtual machine.

IM_NODE_HOSTNAME Hostname of the virtual machine (without the domain).

IM_NODE_DOMAIN Domain name of the virtual machine.

IM_NODE_FQDN Complete FQDN of the virtual machine.

IM_NODE_NUM The value of the substitution `#N#` in the virtual machine.

IM_MASTER_HOSTNAME Hostname (without the domain) of the virtual machine doing the *master* role.

IM_MASTER_DOMAIN Domain name of the virtual machine doing the *master* role.

IM_MASTER_FQDN Complete FQDN of the virtual machine doing the *master* role.

IM_<application name>_VERSION The version installed of an application required by the virtual machine.

IM_<application name>_PATH The path to an installed application required by the virtual machine.

4.6 Including roles of Ansible Galaxy

To include a role available in Ansible Galaxy a special application requirement must be added: it must start with: "ansible.modules" as shown in the following example. In this case the Ansible Galaxy role called "micafer.hadoop" will be installed:

```
network net (outbound = "yes")

system node_ubuntu (
   cpu.arch = 'i686' and
   memory.size >= 512M and
   net_interface.0.connection = "net" and
   disk.0.os.name = "linux" and
   disk.0.os.flavour = "ubuntu" and
   disk.0.applications contains (name="ansible.modules.micafer.hadoop")
)
```

Then the configuration section of the RADL can use the role as described in the role's documentation. In the particular case of the "micafer.hadoop" role is the following:

```
configure wn (
@begin
---
 - roles:
    - { role: 'micafer.hadoop', hadoop_master: 'hadoopmaster' }

@end
)
```

4.7 Examples

4.7.1 Hello Cloud!

The next RADL is a simple example that launches two virtual machines in the default cloud provider with at least 512M of RAM:

```
system node (
   memory.size >= 512M
)
deploy node 2
```

4.7.2 Deploy ten Ubuntu

The next RADL deploys ten Ubuntu of 32 bits with version 12.04 at least, that can be accessed from extern networks and with DNS names `node-0`, `node-1`, ..., `node-9`:

```
network net (outbound = "yes")

system node_ubuntu (
   cpu.arch = 'i686' and
   memory.size >= 512M and
   net_interface.0.connection = "net" and
   net_interface.0.dns_name = "node-#N#" and
   disk.0.os.name = "linux" and
```

```
   disk.0.os.flavour = "ubuntu" and
   disk.0.os.version >= "12.04" and
   disk.0.applications contains (name="toncat")
)

deploy node_ubuntu 10
```

4.7.3 Including a recipe from another

The next RADL defines two recipes and one of them (`add_user1`) is called by the other (`add_torque`):

```
configure add_user1 (
@begin
---
  - tasks:
    - user: name=user1   password=1234
@end
)

configure add_torque (
@begin
---
  - tasks:
    - include: add_user1.yml
    - yum: pkg=${item} state=installed
      with_item:
      - torque-client
      - torque-server
@end
)
```

CHAPTER

FIVE

IM XML-RPC API

IM Service can be accessed through the API that follows the XML-RPC specification. The port number and the security settings are controlled by the options listed in *XML-RPC API*.

The last parameter in every call refers to the credentials for the IM Service, the VMRC and cloud providers. Every credential is represented as a struct datatype, whose keys and values are described in *auth-file*. Then the parameter is an array of these structs.

This is the list of method names:

`GetInfrastructureList`

parameter 0 `auth`: array of structs

ok response [true, `infIds`: array of integers]

fail response [false, `error`: string]

Return the ID associated to the infrastructure created by the user.

`CreateInfrastructure`

parameter 0 `radl`: string

parameter 1 `auth`: array of structs

ok response [true, `infId`: integer]

fail response [false, `error`: string]

Create and configure an infrastructure with the requirements specified in the RADL document passed as string. Return the ID associated to the created infrastructure.

`GetInfrastructureInfo`

parameter 0 `infId`: integer

parameter 1 `auth`: array of structs

ok response [true, struct(`cont_out`: string, `vm_list`: array of integers)]

fail response [false, `error`: string]

Return in `vm_list` a list of IDs associated to the virtual machine of the infrastructure with ID `infId`. If the contextualization process has finished, `cont_out` may have a message indicating why the process failed.

`GetVMInfo`

parameter 0 `infId`: integer

parameter 1 `vmId`: string

parameter 2 `auth`: array of structs

ok response [true, struct(`info`: string, `cloud`: string, `state`: string)]

fail response [false, `error`: string]

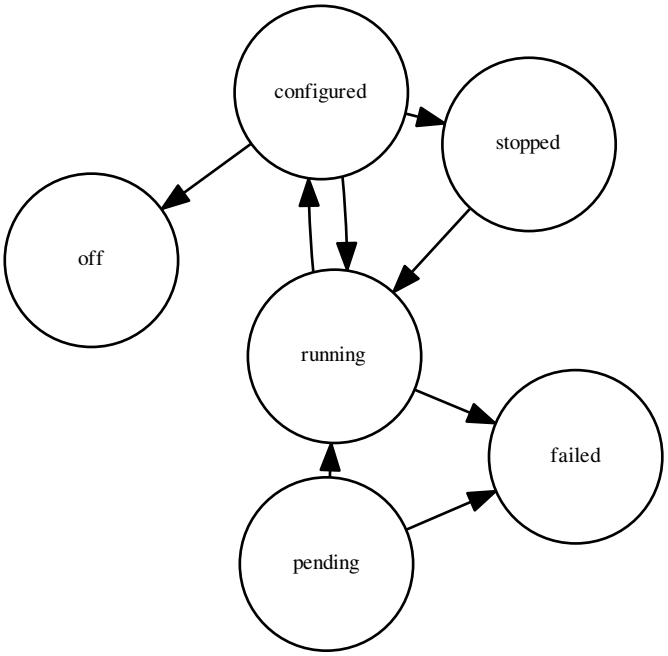
Return a struct with information about the virtual machine with ID `vmId` in the infrastructure with ID `infId`. The returned struct is composed by

- `info`, information about the virtual machine in RADL format;
- `cloud`, information about the cloud; and
- `state`, state of the virtual machine.

The `state` can be

- `pending`, launched, but still in initialization stage;
- `running`, created successfully and running, but still in the configuration stage;
- `configured`, running and contextualized;
- `stopped`, stopped or suspended;
- `off`, shutdown or removed from the infrastructure;
- `failed`, an error happened during the launching or the contextualization; or
- `unknown`, unable to obtain the status.

The next figure shows a state diagram of virtual machine status.



`AlterVM`

parameter 0 `infId`: integer

parameter 1 `vmId`: string

parameter 2 `radl`: string

parameter 3 `auth`: array of structs

ok response [true, struct(`info`: string, `cloud`: string, `state`: string)]

fail response [false, `error`: string]

Change the features of the virtual machine with ID `vmId` in the infrastructure with with ID `infId`, specified by the RADL `radl`. Return a struct with information about the virtual machine, like *GetVMInfo*.

`DestroyInfrastructure`

parameter 0 `infId`: integer

parameter 1 `auth`: array of structs

ok response [true, string of length zero]

fail response [false, `error`: string]

Undeploy all the virtual machines associated to the infrastructure with ID `infId`.

`AddResource`

parameter 0 `infId`: integer

parameter 1 `radl`: string

parameter 2 `auth`: array of structs

ok response [true, `infId`: integer]

fail response [false, `error`: string]

Add the resources specified in `radl` to the infrastructure with ID `infId`. The `deploy` instructions in the `radl` must refer to *systems* already defined. If all the *systems* defined in `radl` are new, they will be added. Otherwise the new *systems* defined will be ignored.

`RemoveResource`

parameter 0 `infId`: integer

parameter 1 `vmIds`: string

parameter 2 `auth`: array of structs

ok response [true, `infId`: integer]

fail response [false, `error`: string]

Updeploy the virtual machines with IDs in `vmIds` associated to the infrastructure with ID `infId`. The different virtual machine IDs in `vmIds` are separated by commas.

`StopInfrastructure`

parameter 0 `infId`: integer

parameter 1 `auth`: array of structs

ok response [true, string of length zero]

fail response [false, `error`: string]

Stop (but do not undeploy) all the virtual machines associated to the infrastructure with ID `infId`. They can resume by *StartInfrastructure*.

`StartInfrastructure`

parameter 0 `infId`: integer

parameter 1 `auth`: array of structs

ok response [true, string of length zero]

fail response [false, `error`: string]

Resume all the virtual machines associated to the infrastructure with ID `infId`, previously stopped by *StopInfrastructure*.

`Reconfigure`

parameter 0 `infId`: integer

parameter 1 `radl`: string

parameter 2 `auth`: array of structs

ok response [true, string of length zero]

fail response [false, `error`: string]

Update the infrastructure with ID `infId` using the *configuration sections* in the RADL `radl`. Some virtual machines associated to the infrastructure may be reconfigured.

CHAPTER

SIX

IM REST API

Optionally, IM Service can be accessed through a REST(ful) API. The port number and the security settings are controlled by the options listed in *REST API*.

Every HTTP request must be companied by the header `AUTHORIZATION` with the content of the *auth-file*, but the lines separated with "\n" instead. If the content cannot be parsed successfully, or the user and password are not valid, it is returned the HTTP error code 401.

Next table summaries the resources and the HTTP methods available.

HTTP method	/infrastructure	/inf/<infId>	/vms/<infId>/<vmId>
GET	**List** the infrastructure IDs.	**List** the virtual machines in the infrastructure `infId`	**Get** information associated to the virtual machine `vmId` in `infId`.
POST	**Create** a new infrastructure based on the RADL posted.	**Create** a new virtual machine based on the RADL posted.	
PUT		**Stop**, **start** or **reconfigure** the infrastructure.	**Modify** the virtual machine based on the RADL posted.
DELETE		**Undeploy** all the virtual machine in the infrastructure.	**Undeploy** the virtual machine.

GET `http://imserver.com/infrastructure`

Content-type text/uri-list

ok response 200 OK

fail response 409

Return a list of URIs referencing the infrastructures associated to the IM user.

POST `http://imserver.com/infrastructure`

body `RADL document`

Content-type text/uri-list

ok response 200 OK

fail response 409

Create and configure an infrastructure with the requirements specified in the RADL document of the body contents. If success, it is returned the URI of the new infrastructure.

GET `http://imserver.com/inf/<infId>`

Content-type application/json

ok response 200 OK

fail response 409

Return a JSON object with two elements:

- vm_list: list of URIs referencing the virtual machines associated to the infrastructure with ID `infId`.
- cont_out: contextualization message.

POST `http://imserver.com/inf/<infId>`

body `RADL document`

Content-type text/uri-list

ok response 200 OK

fail response 409

Add the resources specified in the body contents to the infrastructure with ID `infId`. The RADL restrictions are the same as in *RPC-XML AddResource*. If success, it is returned a list of URIs of the new virtual machines.

PUT `http://imserver.com/inf/<infId>`

input fields `op`, `radl` (compulsory if `op` is `reconfigure`)

Content-type text/uri-list

ok response 200 OK

fail response 409

Perform an action on the infrastructure with ID `infID` indicated by the value of `op`:

- `stop`: stop (but do not undeploy) all the virtual machines in the infrastructure.
- `start`: resume all the virtual machines in the infrastructure.
- `reconfigure`: update the configuration of the infrastructure as indicated in `radl`. The RADL restrictions are the same as in *RPC-XML Reconfigure*.

DELETE `http://imserver.com/inf/<infId>`

ok response 200 OK

fail response 409

Undeploy the virtual machines associated to the infrastructure with ID `infId`.

GET `http://imserver.com/vms/<infId>/<vmId>`

Content-type application/json

ok response 200 OK

fail response 409

Return information about the virtual machine with ID `vmId` associated to the infrastructure with ID `infId`. See the details of the output in *GetVMInfo*.

PUT `http://imserver.com/vms/<infId>/<vmId>`

body `RADL document`

ok response 200 OK

fail response 409

Change the features of the virtual machine with ID `vmId` in the infrastructure with with ID `infId`, specified by the RADL document specified in the body contents.

DELETE `http://imserver.com/vms/<infId>/<vmId>`

ok response 200 OK

fail response 409

Undeploy the virtual machine with ID `vmId` associated to the infrastructure with ID `infId`.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*