OCCIware is a project funded by the French FSN (Fonds national pour la Société Numérique), and supported by five clusters: Systematic, Minalogic, PICOM, Images & Réseaux et Solutions Communicantes Sécurisées.

Deliverable 4.2.1
Deployment Service Modeled with OCCI
Executive Summary

This deliverable presents the deployment services for an OCCI application. The OCCI runtime executes an OCCI model that triggers a set of actions for resource provisioning and application deployment and configuration. The deployment services are responsible of executing the resource provisioning and deployment of the OCCI application. They are modeled using the OCCI recommendations.

Keywords

OCCI, OCCIware, runtime, erOCCI, COAPS, ProActive, Roboconf, REST API, auto-configuration, communication, Protocols, deployment, distributed computing, cloud computing, Model Driven Engineering, MDE, component based development, resource provisioning.
**Version control**

<table>
<thead>
<tr>
<th>Version</th>
<th>Changements</th>
<th>Auteur(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Table of Content</td>
<td>Iyad Alshabani</td>
</tr>
<tr>
<td>0.2</td>
<td>Roboconf modeling with OCCI</td>
<td>Pierre-Yves Gibello</td>
</tr>
<tr>
<td>0.3</td>
<td>OCCI modeling and ProActive</td>
<td>Iyad Alshabani</td>
</tr>
<tr>
<td>0.4</td>
<td>COAPS API</td>
<td>Samir Tata</td>
</tr>
<tr>
<td>0.5</td>
<td>IaaS Service List</td>
<td>Iyad Alshabani</td>
</tr>
<tr>
<td>0.6</td>
<td>Paas Service List</td>
<td>Pierre-Yves Gibello</td>
</tr>
<tr>
<td>0.7</td>
<td>Final adjustments</td>
<td>Iyad Alshabani</td>
</tr>
<tr>
<td>1.0</td>
<td>Ready for review version</td>
<td>Iyad Alshabani</td>
</tr>
<tr>
<td>2.0</td>
<td>Ready for publication</td>
<td>Iyad Alshabani</td>
</tr>
</tbody>
</table>

**Document review**

<table>
<thead>
<tr>
<th>Revue</th>
<th>Date</th>
<th>Ver.</th>
<th>Réviseurs</th>
<th>Commentaires</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>04/15/2015</td>
<td>0.1</td>
<td>Iyad Alshabani</td>
<td></td>
</tr>
<tr>
<td>Draft</td>
<td>06/12/2015</td>
<td>1.0</td>
<td>Iyad Alshabani</td>
<td></td>
</tr>
<tr>
<td>Review</td>
<td>07/10/2015</td>
<td>1.0</td>
<td>Pierre-Yves Gibello</td>
<td>des typos, section 3.1.1 to be rewritten</td>
</tr>
<tr>
<td>Final</td>
<td>07/24/2015</td>
<td>2.0</td>
<td>Philippe Merle</td>
<td></td>
</tr>
</tbody>
</table>
### Glossary, acronyms and abbreviations

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>OCCI</td>
<td>Open Cloud Computing Interface (OGF standard)</td>
</tr>
<tr>
<td>OGF</td>
<td>Open Grid Forum</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>RM</td>
<td>Resource Manager</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>WP</td>
<td>Work package</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
</tbody>
</table>
# Table of content

1. Introduction...............................................................................................................................6

2. Service Modeling for Deployment...............................................................................................6
   2.1. PaaS/SaaS deployment modeling with Roboconf .................................................................6
   2.2. IaaS deployment modeling with ProActive Cloud Automation ............................................9

3. Examples of OCCI Deployment systems ..................................................................................10
   3.1. ProActive Cloud Automation ...............................................................................................10
   Big picture of Cloud automation ..................................................................................................10
   Examples of the use of OcciClient for Cloud Automation ..........................................................11
   3.2. COAPS .................................................................................................................................14
   COAPS overview ..........................................................................................................................14
   Resources in COAPS ....................................................................................................................14

4. OCCIWare Deployment Services .................................................................................................16
   4.1. Deployment Services for Infrastructure ..................................................................................16
   4.2. Deployment Services for Application ....................................................................................23

5. Conclusion ....................................................................................................................................25
1. **Introduction**

The deployment services in the OCCIware platform can be considered as the back-end of the OCCI runtime. Indeed, the OCCI runtime presented in the deliverable D4.1.1 (OCCI Core architecture) is composed of a front-end, which interprets the OCCI model, and a back-end, which is responsible of executing the necessary operations for the deployment of the OCCI application.

In this deliverable we explain how the deployment services are modeled in order to execute deployment actions of an OCCI application. First we explain the modeling of the services using OCCI. Then, we explain the deployment systems that already exist in the OCCIware project. Finally we give a list of deployment operations divided into two categories, regarding IaaS and PaaS services.

2. **Service Modeling for Deployment**

2.1. **PaaS/SaaS deployment modeling with Roboconf**

Roboconf models an application as a graph of components, linked together in terms of container/content (e.g., a database contained in a virtual machine, or a web application contained in an application server) and of runtime dependencies (e.g., an application server that needs information concerning the database it should connect to).

The application can be instantiated (deployment of instances of each component), which triggers the resolution of runtime dependencies as soon as it is possible, in an asynchronous way (e.g., the instantiation of a database will trigger the sending of connection information to any instance that should use the database).

The Roboconf model, expressed using a dedicated DSL, can be seen as a general modeling for any software application deployed on a hardware infrastructure (e.g., a 3-tier application deployed on a IaaS).

That model can be represented as follows (pseudo-UML, before translation into OCCI):
Figure 1: The Roboconf model

This schema leads to the following OCCI model (the XML generated by the OCCI modeling tools has been reworked for more legibility, but is very similar to the raw one):

```
<!-- An application is a graph of components that can be instantiated -->
<kinds term="application" scheme="http://schemas.occiware.org/roboconf#" title="Application">
  <parent href="Core.xmi#///@kinds[term='resource']"/>
  <attributes name="name" required="true" type="String"/>
  <attributes name="description" type="String"/>
  <actions term="addInstance" scheme="http://schemas.occiware.org/roboconf#" title="Create new instance"/>
  <actions term="deployAndStartAll" scheme="http://schemas.occiware.org/roboconf#" title="Deploy and Start All"/>
  <actions term="stopAll" scheme="http://schemas.occiware.org/roboconf#" title="Stop All"/>
  <actions term="undeployAll" scheme="http://schemas.occiware.org/roboconf#" title="Undeploy All"/>
</kinds>

<!-- A facet is a kind of interface a component can implement -->
<kinds term="facet" scheme="http://schemas.occiware.org/roboconf#" title="Facet">
  <parent href="Core.xmi#///@kinds[term='resource']"/>
  <attributes name="name" required="true" type="String"/>
</kinds>
```
<attributes name="exportedVariables">
</attributes>
</kinds>

<-- A component is, for example, a software component that can be instantiated
(eg. a “database” component), a VM (eg. an “Openstack VM”), or anything that can be deployed -->

<kinds term="component" scheme="http://schemas.occware.org/roboconf#"
title="Component">
  <parent href="Core.xmi#//@kinds[term='facet']"/>
  <attributes name="installName" required="true">
    <type href="String"/>
  </attributes>
  <attributes name="importedVariables">
  </attributes>
</kinds>

<!-- An instance is a living copy of a specified component (eg. “my database”,
instance of a “database” component) -->

<kinds term="instance" scheme="http://schemas.occware.org/roboconf#" title="Instance">
  <parent href="Core.xmi#//@kinds[term='resource']"/>
  <attributes name="name" required="true">
    <type href="String"/>
  </attributes>
  <attributes name="status" required="true">
    <type href="InstanceStatus"/>
  </attributes>
  <attributes name="overriddenExports">
  </attributes>
  <actions term="changeInstanceState" scheme="http://schemas.occware.org/roboconf#" title="Change Instance State"/>
  <actions term="delete" scheme="http://schemas.occware.org/roboconf#" title="Delete"/>
</kinds>

<types xsi:type="ecore:EEnum" name="InstanceStatus">
  <eLiterals name="Not Deployed" literal="NOT_DEPLOYED" value="0"/>
  <eLiterals name="Deployed and Stopped" literal="DEPLOYED_AND_STOPPED" value="1"/>
  <eLiterals name="Deployed and Started" literal="DEPLOYED_AND_STARTED" value="2"/>
  <eLiterals name="Deploying" literal="DEPLOYING" value="3"/>
  <eLiterals name="Starting" literal="STARTING" value="4"/>
  <eLiterals name="Stopping" literal="STOPPING" value="5"/>
  <eLiterals name="Undeploying" literal="UNDEPLOYING" value="6"/>
  <eLiterals name="Problem" literal="PROBLEM" value="7"/>
  <eLiterals name="Unresolved" literal="UNRESOLVED" value="8"/>
</types>

<kinds term="application-template" scheme="http://schemas.occware.org/roboconf#" title="Application Template">
  <parent href="Core.xmi#//@kinds[term='resource']"/>
  <attributes name="name" required="true">
    <type href="String"/>
  </attributes>
  <attributes name="description">
    <type href="String"/>
  </attributes>
</kinds>

<kinds term="applicationFromApplicationTemplate" scheme="http://schemas.occware.org/roboconf#" title="Application created from Application Template">
  <parent href="Core.xmi#//@kinds[term='link']"/>
</kinds>
It is clear that this modeling, by its general character aiming at describing an application and its deployment, goes beyond the first goals of OCCI (IaaS resources - compute, network, storage) toward an upper level in the stack (PaaS and/or SaaS).

### 2.2. IaaS deployment modeling with ProActive Cloud Automation

ProActive Cloud Automation models infrastructure resources using a subset of the OCCI reference model. However, not all of OCCI elements are modeled in ProActive Cloud Automation. The modeled elements are:
- **Action**: is the operation/process that will be carried out on a resource
- **Resource**: is the definition of a physical or virtual resource which can be categorized as:
  - **Simple Category**
    - **IaaS**: resources at the IaaS level
      - Compute
      - Storage
      - Link
    - **PaaS**: resources at the platform and the application level
      - Platform
    - **Action Trigger**: the trigger that will acquire an action
  - **Template**: is a set of resources that will be acquire together
- **DataBase**: resources related to database
  - OrientDB
  - InMemoryDB
  - H2DB

The following chart is a pseudo class diagram of the OCCI implementation in ProActive Cloud Automation.
The implementation represents a subset of the OCCI and leaks of implementation of the Mixin. As ProActive Cloud Automation’s role is to be an OCCI deployment service, the focus is done on the “Action” element.

3. Examples of OCCI Deployment systems
3.1. ProActive Cloud Automation

Big picture of Cloud automation

In the figure below an OCCI request is sent to Cloud Automation, this request is then interpreted by using specific rules. The rules are the mean by which a specific workflow template is instantiated to match the deployment request. The generated workflow is consisting of a set of actions to provision, exploit and deprovision resources. The workflow is then submitted to the ProActive Scheduler which will execute the “Start Service” of the workflow. The “Start Service” itself loads the specific IaaS connectors, which is given by the OCCI request. Using the IaaS API, the workflow in this example will create a VM on the target IaaS. This VM contains a ProActive Resource Manager (RM) Agent. When the VM is started, the agent inside starts and is connected to the ProActive RM to be managed by it. At this stage, the node acquired in the IaaS is managed by the ProActive Resource Manager, such that any further action in the node is done using ProActive Agent. For instance, the second action in the Start Service is to install Tomcat on the target node and this action is done by the ProActive Scheduler using the Agent to be aware about the node. In the same manner, to undeploy the application and the resource, the scheduler will execute the “Stop Service” of the submitted workflow.
Examples of the use of OcciClient for Cloud Automation

Creation of a compute resource

This example shows how the OcciClient connects to the server, specifies arguments for the creation of a new Compute resource (VM provided by OpenStack), creates the VM, waits for it to be up, prints one of the category attributes (occi.networkinterface.address), and stops it.

```java
import org.ow2.proactive.brokering.occi.client.OcciClient

OcciClient client = new OcciClient("http://try.activeeon.com/cloudautomation/");

// Create compute
def argu = ["provider":"openstack","rule":"try","occi.compute.hostname":"Vm-" + new Random().nextInt(10000)]
def compute = client.createResource("compute", argu);

// Wait for compute to be up
while (!"up".equalsIgnoreCase(compute.get("occi.compute.state"))) {
}
```
Creation of a Elasticsearch platform with elasticity

This example shows how the OcciClient can create a new Elasticsearch multi-VM platform, with elasticity.
The initial amount of VMs is 2, they will be configured to connect to each other by Cloud Automation.
Elasticity is provided thanks to a poll-based monitoring of the Elasticsearch platform API; a provided rule script evaluated regularly can trigger the platform scale-out (add or remove a VM from the pool of VMs).
The state of the platform can be obtained by performing GET on its category instance and reading its attributes. It is possible to navigate through categories’ instances children if they are category instances too. In other words, some of the attributes point to other category instances, and they can be navigated. For instance, for the platform category instance of Elasticsearch, there will be a link to the Elasticsearch master VM, specified by the attribute “occi.paas.elasticity.masterplatform”.

```java
package org.ow2.proactive.brokering.occi.client.examples
import org.ow2.proactive.brokering.occi.client.OcciClient
import org.ow2.proactive.brokering.occi.client.ResourceInstance

// Instantiate an OCCI client for Cloud Automation API
OcciClient client = new OcciClient("http://try.activeeon.com/cloudautomation/");

// Create elastic platform
def argu = [
    "provider":"openstack","rule":"try","elasticity.vm.count.maximum":2,"elasticity.vm.count.minimum":0,
    "application":"elasticsearch","flavor":"elastic"]
def elasticPlatform = client.createResource("platform", argu);

// Wait so that the bitPlatform updates its attributes
while(!"done".equalsIgnoreCase(elasticPlatform.get("action.state"))) {
    elasticPlatform.refresh(client)
    Thread.sleep(1000*2)
}

// Get information about resources created
```

Thread.sleep(1000*20)
compute.refresh(client)
}
def ip = compute.get("occi.networkinterface.address")
println ip

// Stop compute
client.updateResource(compute, [:], "stop");

def masterLocation = elasticPlatform.get("occi.paas.elasticity.masterplatform")
def masterPlatform = new ResourceInstance(masterLocation).refresh(client)

def triggerLocation = elasticPlatform.get("occi.paas.elasticity.trigger")
def trigger = new ResourceInstance(triggerLocation).refresh(client)

// Show resources created
println "Elastic platform: " + elasticPlatform.toString()
println "Master platform: " + masterPlatform.toString()
println "Trigger platform: " + trigger.toString()

// Install kibana for elasticsearch in the master node
masterPlatform.update(client, [application:"kibana"], "install")

// Shutdown the whole platform
platformbig = client.updateResource(elasticPlatform, [], "stop");

A OcciClient Code Snippet

Code for developers, from within the OcciClient implementation.

public ResourceInstance createResource(String category, Map<String, String> attributes, String action) throws ResourceCreationException, ResourceReadingException {

    String urlRaw = internalCreateResource(category, attributes, action);
    return new ResourceInstance(urlRaw).refresh(this);
}

private String internalCreateResource(String category, Map<String, String> attributes, String action) throws ResourceCreationException {

    String url = generateUrl(endpoint + "/", category, action);

    HttpPost post = new HttpPost(url);
    post.setHeader("X-OCCI-Attribute", Utilities.buildCommaSeparatedAttributesString(attributes));

    try {
        HttpResponse response = httpClient.execute(post);
        if (isSuccessful(response)) {
            String resourceUrlRaw = getEntityAsString(response);
            return resourceUrlRaw;
        } else {
            throw new InternalError(getEntityAsString(response));
        }
    } catch (Exception e) {
        throw new ResourceCreationException(e);
    }
}


COAPS overview

COAPS is a generic API that allows human and/or software agents to provision and manage applications in a target PaaS provider. COAPS provides an abstraction layer and a middleware for existing and proprietary PaaS solutions to allow application provisioning in a unified manner. It exposes a generic interface that can be implemented according to the different actions exposed by a PaaS (e.g. Cloud Foundry, OpenShift, etc.) (see Figure below). Thanks to COAPS, each company’s Cloud is now able to seamlessly interact with different and heterogeneous PaaS providers (e.g. Cloud Foundry, OpenShift, etc.), which reinforces and ensures the PaaS providers cooperation and federation goals.

Resources in COAPS

There are two types of resources used in COAPS API: platform-level resources and application-level resources.

The Platform resource types that we defined are derived from Resource entity at OCCI core level, whereas platform resource mixins are derived from the Mixin entity and the interfaces, which link resources between them, are derived from Link entity.

The main defined OCCI platform resources are:
- Database, which are data store resources for platform applications processing persistent Data.
- Container, which are engines to host and run applications
- Router, which are resources that provide protocols, message format transformations and routing

We also define a set of links to connect and interact with these resources:
- ContainerLink to connect to Container resources
- RouterLink to connect to Router resources,
- DatabaseLink to connect a Container resource to a Database resource

The Application resource types that we defined are derived from Resource entity at OCCI core level and the interfaces, which link resources between them, are derived from Link entity.

The main defined OCCI Application resources are:

- Environment, which represents a set of "settings" needed to host and run an Application - i.e. the needed runtime (java 7, java 6, ruby, etc.), the needed frameworks/containers (spring, tomcat, ruby, etc.) and possibly needed services (databases, messaging, etc.).
- Application, which is the software or program that can be deployed on top of a PaaS
- Deployable, which represents the Application deployables

We also define the ContainerLink resource to connect an Environment to an Application.

More information about COAPS can be found at http://www-inf.it-sudparis.eu/SIMBAD/tools/COAPS/
4. OCCIWare Deployment Services

The deployment services are organized into the following categories:

- Deployment services for Infrastructure, which are the services responsible of resource provisioning at the infrastructure level regarding virtual machines, operating systems, network, storage.
- Deployment Services for Applications.

4.1. Deployment Services for Infrastructure

The deployment services of the OCCIWare platform are part of the ProActive Cloud automation using OCCI. We provide in the table below a list of the services regarding the Resource Provisioning at the IaaS level with examples.

<table>
<thead>
<tr>
<th>Service VM</th>
<th>REST API Specifications</th>
</tr>
</thead>
</table>
| Virtual Resource Provisioning | Machine Characteristics | CPU (Architectures, Cores) | Provisioning resources at the Virtual machine level and giving its characteristics such as the CPU features, number of cores, architecture, memory speed and the local storage. This is done using the compute category element inside an OCCI request. OCCI-Attributes (prefix occi.compute):
  - category=Compute
  - cores= N
  - hostname = ""
  - architecture = ""x86"
  - memory = ""1.2"
  - speed= ""2.4"

Example:
```
curl -X POST --header 'Content-type: text/occi' --header 'Category: compute;
```

<table>
<thead>
<tr>
<th>Service VM</th>
<th>REST API Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td></td>
</tr>
<tr>
<td>Local Storage</td>
<td></td>
</tr>
</tbody>
</table>
| NIC cards | Provisioning resource at the virtual machine level for a network interface. In the request, the element Link is used to give the network interface characteristics.
  additional X-OCCI-Attributes:
  - networkinterface.interface=""2"" |
### System Image
At the system level the resource is provisioned using the category "os_image" for the image of the system.

**additional class:**
- Mixin

**additional X-OCCI-Attribute:**
- term=""

**Example:**
curl -X POST --header 'Content-type:text/occi' --header 'Category:compute;

http://localhost:8182/procci-SNAPSHOT/compute/

### Parameters (Metadata)
The parameters of the system image are given using a list of <key, value> in the request with the category "os_para".

**additional class:**
- Mixin

**additional X-OCCI-Attribute:**
- term=""

**Example:**
curl -X POST --header 'Content-type:text/occi' --header 'Category:compute;

### SLA Gold
SLA properties are given in the request by adding the category "SLA_Gold" as a mixin element, and the property sla=Gold for a given Virtual machine provisioning.

**additional class:**
- Mixin

**additional X-OCCI-Attribute:**
- sla=Gold
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affinities with other Vms</td>
<td>Affinity rules and anti-affinity rules tell the virtual machine hypervisor platform to keep virtual entities together or separate</td>
<td></td>
</tr>
<tr>
<td>Service Network</td>
<td>VLAN</td>
<td>The network service is used to provision virtual network. For VLAN, the request will use the kind Category: network Class: kind X-OCCI-Attribute (prefix occi.network) : - vlan=&quot;&quot; - state=&quot;active/non active&quot; - title=&quot;&quot;</td>
</tr>
<tr>
<td>Service</td>
<td>Storage</td>
<td>SAN/NFS</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Category</td>
<td>storage</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>kind</td>
<td></td>
</tr>
<tr>
<td>X-OCCI-Attributes:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- occi.storage.size=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- occi.storagelink.deviceid=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
<td>API Request</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
Migration

Make a migration operation on the virtual resource.
method = migrate

Example:
curl -X POST --header 'Content-type:text/occi' --header 'X-OCCI-Attribute: method=migrate'

Machine reconfiguration

Add memory

Update a configuration of the VM by adding more memory.
Category:
X-OCCI-Attribute:
- hostname="***"
- memory="the amount of memory to configure"

Example:
curl -X PUT --header 'Content-type:text/occi' --header 'Category: compute; scheme="http://schemas.ogf.org/occi/infrastructure#"; class="kind"'
--header 'X-OCCI-Attribute: occi.compute.hostname=VIRTUAL occi.compute.memory=2.4' http://localhost:8182/procci

Service Network

VLAN

Up
An action to start a VLAN network.
method = up

Example:
curl -X POST --header 'Content-type:text/occi' --header 'X-OCCI-Attribute: method=up'

Down
An action to stop a VLAN network.
method = down

Example:
curl -X POST --header 'Content-type:text/occi' --header 'X-OCCI-Attribute: method=down'

Service Storage

online
An action to make a storage available online.
method = online

Example:
curl -X POST --header 'Content-type:text/occi' --header 'X-OCCI-Attribute: method=online'
http://localhost:8182/procci-SNAPSHOT/storage/69f42572-f6e4-411d-a223-44c641106e4f?action=online --header 'Category: online; scheme="http://schemas.ogf.org/occi/infrastructure/storage/action#"; class="action"'

offline
An action to make a storage unavailable online.
method = offline

Example:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>


**Virtual Resource De provisioni ng**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Network</td>
<td>DELETE</td>
<td>Delete a network resource.</td>
<td>Example: curl -X DELETE <a href="http://localhost:8182/procci-SNAPSHOT/network/971ae010-cd54-4c74-a2ef-3c5cea8c7907">http://localhost:8182/procci-SNAPSHOT/network/971ae010-cd54-4c74-a2ef-3c5cea8c7907</a></td>
</tr>
<tr>
<td>Service Storage</td>
<td>DELETE</td>
<td>Delete a storage resource.</td>
<td>Example: curl -X DELETE <a href="http://localhost:8182/procci-SNAPSHOT/storage/c480466b-b1c3-4046-8f97-f4dd4b9b5d">http://localhost:8182/procci-SNAPSHOT/storage/c480466b-b1c3-4046-8f97-f4dd4b9b5d</a></td>
</tr>
</tbody>
</table>
4.2. Deployment Services for Application

Roboconf defines an application as a graph of components that can be instantiated. To make the parallel with object-oriented model, components are kinds of “classes”, and the application itself, when deployed, is a set of living “instances” of those classes.

Basically, we can distinguish 2 kinds of instances:
- “Root instances” that are generally virtual machines or equivalents (containers), and can be considered IaaS computation units (in most cases, a “root instance” is a IaaS VM with networking and storage).
- Plain instances that are in general “software components” (e.g., a database or an application server).

There may be several levels of “plain instances” (deployed one inside another, like an instance of “web application” deployed inside an instance of “web server”), all of them deployed on a “root instance” (the machine, generally).

Roboconf allows to provision instances, then manage their lifecycle by changing their states: after the OCCI modeling proposed in this document (part 2.1), we can derive a REST API specification to provide real-world instance provisioning, as follows:

<table>
<thead>
<tr>
<th>Application instances-level provisioning</th>
<th>REST APIs specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances CRUD operations</td>
<td>GET /app/{name}/children</td>
</tr>
<tr>
<td>List instances under a given one (parent)</td>
<td>{name} is the application name. Other parameters:</td>
</tr>
<tr>
<td></td>
<td>● instance-path: the path of the parent instance (null to get root instances).</td>
</tr>
<tr>
<td></td>
<td>● all-children: true to get all the children, false to only get the direct children.</td>
</tr>
<tr>
<td></td>
<td>Returns an array of instances, including for each of them the following information:</td>
</tr>
<tr>
<td></td>
<td>● name: the instance name.</td>
</tr>
<tr>
<td></td>
<td>● path: the instance path</td>
</tr>
<tr>
<td></td>
<td>● status: the instance status (NOT_DEPLOYED, DEPLOYING, DEPLOYED_STOPPED, STARTING, DEPLOYED_STARTED, PROBLEM).</td>
</tr>
<tr>
<td></td>
<td>● component: information about the deployed component, including its name and installer (e.g. puppet or bash).</td>
</tr>
<tr>
<td></td>
<td>● data: custom data fields maintained by the instance (e.g. the IP address).</td>
</tr>
<tr>
<td>Add a new instance</td>
<td>OCCI mapping: addInstance action in Application resource. Roboconf REST operation:</td>
</tr>
</tbody>
</table>
| **Change instance state** (deploy / undeploy or start / stop) | **OCCI mapping:** changeInstanceState action in Instance resource.  
Roboconf REST operation:  
**POST /app/{name}/change-state**  
{name} is the application name.  
Other parameters:  
- **new-state:** the new state (NOT_DEPLOYED, DEPLOYED_STOPPED, DEPLOYED_STARTED).  
- **instance-path:** the path of the instance whose state must be changed. |
| --- | --- |
| **Delete an instance** | **OCCI mapping:** delete action in Instance resource.  
Roboconf REST operation:  
**DELETE /app/{name}/instances**  
{name} is the application name.  
Other parameters:  
- **instance-path:** the path of the instance to delete. |
| **Application global provisioning** | **Global instance operations** | **OCCI mapping:** deployAndStartAll action in Application resource.  
Roboconf REST operation:  
**POST /app/{name}/deploy-all**  
{name} is the application name.  
Other Parameters:  
- **instance-path:** the path of the instance to start (including all its children). Null to match all the root instances. |
| **Deploy and start all children of a given instance (or all instances).** | **Stop all children of a given instance (or all instances).** | **OCCI mapping:** stopAll action in Application resource.  
Roboconf REST operation:  
**POST /app/{name}/stop-all**  
{name} is the application name.  
Other Parameters:  
- **instance-path:** the path of the instance to stop (including all its children). Null to match all the root instances. |
| **Undeploy all children of a given instance (or all instances).** | **OCCI mapping:** undeployAll action in Application resource.  
Roboconf REST operation:  
**POST /app/{name}/undeploy-all**  
{name} is the application name.  
Other Parameters:  
- **instance-path:** the path of the instance to undeploy (including all its children). Null to match all the root instances. |
5. Conclusion

In this deliverable, we presented the deployment services of an OCCI application. To do so, the deliverable was divided into three main sections to be able to give the specification of the deployment services.

First we presented how deployment systems are using OCCI modeling in order to be conform to the OCCI recommendation and to be able to deploy an OCCI application. We presented two main systems: Roboconf for application deployment and configuration, and ProActive Cloud Automation for the infrastructure resource provisioning and deployment of the OCCI application.

Then in the second part of the deliverable, we presented examples of deployment systems such as Cloud Automation and COAPS and gave more detailed information with examples how to use them.

Finally we gave a list of deployment services in two categories for the application and the infrastructure deployments. It is important to note that the list of services that are presented in this deliverable is non exhaustive and some new services can be added while implementing the deployment system.