Cloud
Practical Work - Report
5ISS-A1
INSA Toulouse

Cécile DUTHOIT, Linn MJELSTAD

January 2017

1 Introduction

The practical works this document is related to aimed to allow us to understand and manipulate a Cloud infrastructure and to implement IaaS (Infrastructure as a Service).

The infrastructure used is based on OpenStack and is part of the INTEL-INSA-LAAS lab.

2 Understand what Cloud Computing is

Cloud Computing is exploiting the processing power or storage power of distant servers via a network (usually the Internet). The National Institute of Standards and Technology (NIST) defines Cloud Computing as the access through a telecommunications network, when the user wants, to shared and configurable resources, which consists in a relocation of the computer infrastructure. There are different levels of Cloud services:

- **SaaS:** Software as a Service
  The software is used on a distant machine and the user only control their data.
- **PaaS:** Platform as a Service
  The users gets an entire platform with an OS, containing development environments, databases, web servers, etc.
- **IaaS:** Infrastructure as a Service
  The user gets a whole computing infrastructure: (a) distant (virtual) machine(s) and services such as data storage, firewall, networking services, etc. The user is responsible of more elements than the two previous level: OS, applications, data, middleware, updates, etc.

Cloud computing is expanding for about a decade because it allows users to execute, manage, and save data from their machine on another machine that they do not need to buy (better processing or storage power means more expensive), maintain and manage. It eliminates many problematics for

the user while it offers to them the service they want. However, two aspects should be considered: the link between them and the distant server (because that link will go through the Internet to -maybe- another continent, the delay depends on that link and will always be longer than accessing data on a local server) and the security (the user does not know where the server they use is, how this server is protected, who the virtual machine next to theirs is affected to, etc.).

3 Use a Cloud infrastructure

OpenStack is a free and open-source software platform released under the terms of the Apache License. It is made of many open-source softwares that allow the user to deploy IaaS infrastructures. Different modules control different aspects of virtual machines, such as processing power, storage, or networking. It enables emulation of different OS on a unique physical machine by acting between the hardware and the different OS.

OpenStack enables the creation of instances by typing command lines, by using REST API, or by using the graphical interface. Therefore, it seems to be a platform adapted to all users, whatever their methods of work. Several instances may be generated from only one image, which enables the duplication of instances -to share an environment for example. The user can manage the accesses to these instances and images.

The request we sent to create an Instance using REST API is presented below:

- URL: http://localhost:8774/v2/bc21b551b2774ab0b64132378d744427/servers
- Headers:
  - Content-Type: application/json
  - X-Auth-Token: X-Auth-Token (our token was 4f615a89d83f4964be4f9cbeb117bcd4)
- Body:


```
{
"server": {
"name": "new-server-api-test",
"imageRef": "http://localhost:8774/v2/bc21b551b2774ab0b64132378d744427/images/b825c534-be1a-4f26-adca-a37404e7f73f",
"flavorRef": "http://localhost:8774/v2/bc21b551b2774ab0b64132378d744427}/flavor/0a06edd1-5c74-4704-9230-e13281e55876",
 "networks": [
            {
                "uuid": "c0bf1fc7-4f64-4351-bc35-100a96d51ef0"
            }
 ],
"security_groups": [
            {
                "name": "default"
            }
  ],
 "metadata": {
            "My Server Name": "Create Instance via API"
        }
 }
}
```


Figure 1: Body of our instance creation REST request.

4 Deploy and adapt in an autonomic way an IoT platform in the Cloud

4.1 Deploy a PaaS architecture based on OM2M

The OM2M project divides its architecture as well:

- **M2M Device:** a machine that runs an M2M device program.
  An M2M device can connect directly to the M2M Network or indirectly via a gateway that acts as a network proxy.
- **M2M Area Network:** part of the global network that provides connectivity between M2M devices and between those devices and the M2M gateways.
  It is based on existing technologies such as Zigbee, Phidgets, M-BUS, KNX, etc.
- **M2M Gateway:** a machine that runs an M2M Gateway program.
- **Wide Area Network:** core network that access other networks to enable M2M devices and gateways to communicate with the M2M network.
  It is based on existing technologies such as xDSL, GERAN, UTRAN, eUTRAN, W-LAN, WiMAX, etc.
- **M2M Network:** part of the global network that can be accessed by M2M Network applications (NAs) via an open interface.
- **M2M service capabilities Layer (SCL):** horizontal service layer for M2M applications that enhances M2M interoperability by abstracting the complexity of heterogeneous devices.
- **M2M application a domain-specific software application:** application that interacts with the M2M machines by querying the service layer through a set of open interfaces.

The OM2M platform could be offered to anyone who wants to implement a machine to machine communication on an elastic platform, with no need of maintaining it, and that could be available at anytime. Labs, companies, factories, or any entity that wants to install connected objects for any purpose could be interested in this offer.

Then, deploying the IN-CSE in the Cloud enables the elasticity and the availability on demand without maintenance. Indeed, thanks to the principle of elasticity of the Cloud, if too many requests are sent to the IN-CSE, a copy of its instance can be deployed to answer all the requests without delay.

However, it is inappropriate to deploy the MN-CSE in the Cloud because it is a kind of interface with the connected objects, and thus, it should be deployed on the client's side. The number of connected objects is not supposed to vary suddenly as could vary the number of requests to the IN-CSE ; therefore, there is no need of elasticity.

However, its database could be deployed in the Cloud since it does not need to be "near" the objects, and it erases the need of a server with a certain capacity of storage since it is always possible to increase the amount of storage we use in the Cloud (in exchange for payment of course).

4.2 Make a PaaS architecture autonomic

4.2.1 Autonomic Computing

Autonomic computing is a computing model that is self-managing. It controls the functions of associated applications without input from a user. We have used the tool Frameself. There are four different modules to it's loop; the monitor, the analyzer, the planner, and the executor.

- The monitor gets values from the sensors and creates symptoms according to them.
- The analyzer analyzes the different symptoms and creates Request For Change (RFC) according to them.
- The planner plans wich actions to perform in according to the RFC's that it receives.
- The executor executes the planned actions.

For the monitor, the analyzer and the planner you have to implement rules that decides how the system should interpret the different inputs. This will decide the behavior of the system based on external factors.

There are four different automatic computing proprieties, it should be:

- Self-configuring
- Self-healing
- Self-optimizing
- Self-protecting

4.2.2 Snapshot

One can take a snapshot of an image/virtual machine at a given time. When doing this the current state of the virtual machine will also be saved, and one can therefore have an image that can be deployed where the different services are already configured and started.

4.2.3 Our PaaS service

This PaaS platform will manage the congestion of access demands to a database. One can interrogate the database through an IN-CSE. Initially there is only one master IN-CSE deployed on the cloud through which one can access the database.

We have a snapshot that contains an IN-CSE already running and correctly configured. This snapshot can be used to deploy a new IN-CSE if there is a risk of congestion.

The PaaS will according to the number of request and the number of current IN-CSE deployed decide whether or not to create an additional IN-CSE to manage all the request. If the demand gets low again, it can also delete certain IN-CSE. By implementing a set of rules in the PaaS service, it will on its own be able to decide what actions to carry out based on the external inputs.

According to the rules implemented in the Frameself:

- If the number of requests gets higher than two and the number of IN-CSE currently deployed are less than two: The service will create a new IN-CSE
- If the number of requests are less then two and the number of IN-CSE are higher than two: The service will delete one IN-CSE

The rules we established to manage each component are readable below.

```
rule "add RemoveIN-Dup"
    when
       Rfc(category == "DecreaseIN-CSENumber")
    then
       ArrayList<Attribute> attributes = new ArrayList<Attribute>();
       attributes.add(new Attribute("state","false"));
       Action action = new Action();
       action.setCategory("IN-Dup");
       action.setName("DecreaseIN-CSENumber");
       action.setAttributes(attributes);
       action.setEffector(new Effector("Dispatcher"));
       action.setTimestamp(new Date());
       insert(action);
end

rule "add AddIN-Dup"
    when
       Rfc(category == "IncreaseIN-CSENumber")
    then
       ArrayList<Attribute> attributes = new ArrayList<Attribute>();
       attributes.add(new Attribute("state","false"));
       Action action = new Action();
       action.setCategory("IN-Dup");
       action.setName("IncreaseIN-CSENumber");
       action.setAttributes(attributes);
       action.setEffector(new Effector("Dispatcher"));
       action.setTimestamp(new Date());
       insert(action);
end
```

Figure 2: Analyzer rules.

```
rule "add HighRequestsRate"
    when
       Event($id: id, category == "RequestsRate",  $value: value, Integer.parseInt(value) >= 2)
    then
        Symptom symptom = new Symptom();
        symptom.setCategory("HighRequestsRate");
        symptom.setValue($value);
        symptom.setTimestamp(new Date());
        symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
       insert(symptom);
end

rule "add LowRequestsRate"
    when
       Event($id: id, category == "RequestsRate",  $value: value, Integer.parseInt(value) < 2)
    then
        Symptom symptom = new Symptom();
        symptom.setCategory("LowRequestsRate");
        symptom.setValue($value);
        symptom.setTimestamp(new Date());
        symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
       insert(symptom);
end

rule "add HighIN-CSENumber"
    when
       Event($id: id, category == "IN-CSENumber",  $value: value, Integer.parseInt(value) >= 2)
    then
        Symptom symptom = new Symptom();
        symptom.setCategory("HighIN-CSENumber");
        symptom.setValue($value);
        symptom.setTimestamp(new Date());
        symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
       insert(symptom);
end

rule "add LowIN-CSENumber"
    when
       Event($id: id, category == "IN-CSENumber",  $value: value, Integer.parseInt(value) < 2)
    then
        Symptom symptom = new Symptom();
        symptom.setCategory("LowIN-CSENumber");
        symptom.setValue($value);
        symptom.setTimestamp(new Date());
        symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
       insert(symptom);
end
```

Figure 3: Monitor rules.

```
rule "add IncreaseIN-CSENumber rfc"
    when
       Symptom(category == "HighRequestsRate")
       Symptom(category == "LowIN-CSENumber")
    then
       Rfc rfc = new Rfc();
        rfc.setCategory("IncreaseIN-CSENumber");
        rfc.setTimestamp(new Date());
        rfc.setExpiry(new Date(System.currentTimeMillis()+4000));
       insert(rfc);
end

rule "add DecreaseIN-CSENumber rfc"
    when
       Symptom(category == "LowRequestsRate")
       Symptom(category == "HighIN-CSENumber")
    then
       Rfc rfc = new Rfc();
        rfc.setCategory("DecreaseIN-CSENumber");
        rfc.setTimestamp(new Date());
        rfc.setExpiry(new Date(System.currentTimeMillis()+4000));
       insert(rfc);
end
```

Figure 4: Planner rules.