5e année ISS 2018/19

Cloud Computing: Adaptability and Autonomic Management

Lab 2 : Paravirtualization over Cloud Hypervisors

S. Yangui (INSA/LAAS)

Access link : goo.gl/H7YEVD

Reminder: proxmox servers assignement : goo.gl/DDbdDt

Objectives:

During the first lab, you acquired the skills necessary to the creation and configuration of Proxmox containers (including setting up their accessibility to/from the Internet), as well as, the creation of snapshots, restoring and migrating containers manually via the Proxmox Web interface.
This second lab extends the previous one. It is meant for you to master technical skills related to the following objectives:

- Objective 1 : Being able to use the Proxmox API instead of its Web interface to automate the following objectives 2 to 5.
- Objective 2 : Automatically creating and configuring a CT.
- Objective 3 : Automatically monitor the resources of the CSN Proxmox server platform, as well as, the CT created on this platform.
- Objective 4 : Automatically migrate CT between servers of the CSN platform.
- Objective 5 : Create an automatic monitoring application for the CSN platform resources.

NB : The Proxmox API is REST-based. For the ones who are not familiar with REST, the lab was adapted so that you can invoke the primitives of this API via a provided (native) Java API. Actually, the Java API maps the calls to the Proxmox REST operations.

Expected output:

- On the shared Google sheet: Logic/details of the designed algorithms with the choices that you made.
- A github repo (mentioned in the shared Google sheet) that contains the developed code.

First PART
(objectives 1 to 4)

1. Using the Proxmox API (related to objective 1)

Proxmox exposes a REST API (over HTTP) enabling the automation of tasks you performed in Lab 1 using the Web API. Actually, the Web interface is one of the prospective clients of the Proxmox API.

An HTTP REST client is required to be able to interact with Proxmox API. To that end, multiple clients were developed by the Proxmox community. These clients are available in different languages: php, perl, python, java, c#, ruby. You may use one of these, or write your own client in the language you prefer (including shell with the cURL tool), as long as it stills inline with the Proxmox specifications.

2. Expected work

Based on a Java library that implements a subset of the Proxmox REST API:

- Collect informations about a Proxmox server:
    - CPU usage (%), disk usage (%), memory usage (%)
- Create and start containers.
- Collect information about a container
    - status, CPU usage (%), disk usage (%), memory usage (%), host server name.

In addition to the previously provided resources, the documentation consists of the following material:

- Proxmox hypervisor manual:
    - Wiki Proxmox : *http://pve.proxmox.com/wiki/Main_Page*
    - Documentation : *http://pve.proxmox.com/wiki/Documentation*
- API Proxmox
    - *https://pve.proxmox.com/wiki/Proxmox_VE_API*
    - *https://pve.proxmox.com/pve-docs/api-viewer/index.html*

Second PART
(related to objective 5)

1. Application description

Preamble: the CSN platform's servers are shared by multiple teams/duo (max 5 teams per server), you are only allowed 16% of the RAM of the hosting server for your containers (only 80% of each server resources will be used).

In this part, you are expected to design and implement two applications:

- ***a CT generator***: this application will be in charge of the creation of CT, randomly on the two servers (66% on server 1, 33% on server 2) attributed to your team. The creation of a new CT may be periodical, or driven by a statistic distribution (uniform, exponential, …).
    - **IMPORTANT!**: You must monitor the resources stats (in particular, its memory). If on any of the two servers the RAM usage exceeds **16%**, the generation of containers must stop.
- ***A cluster manager***: This application will periodically monitor server status, and acts on the deployed containers based on the following algorithm:
    - As soon as the load of one of your CT exceeds 8% of the host memory (i.e. half the resources accessible on the server), perform load balancing between your servers ;
    - As soon as the load of one of the servers exceeds **12%** of the total load, stop the oldest CT to support the creation of new ones.

Fig. 1 describes these specifications.

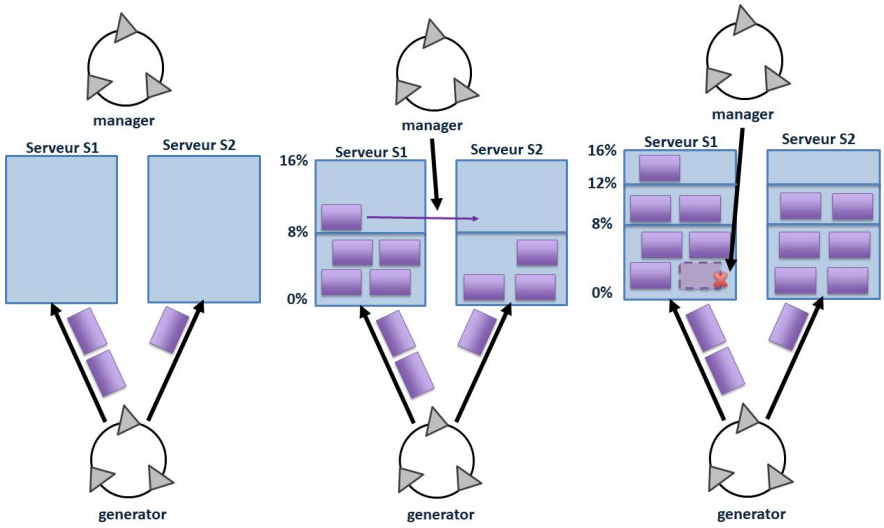

Figure 1

2. Expected work

Develop these two applications in Java based on the REST client mentioned earlier.

NB :

- An Eclipse project provides a base is available **here**.
- Fig. 2 depicts a functional view of the architecture of the code and its interfacing with the REST API exposed by Proxmox.

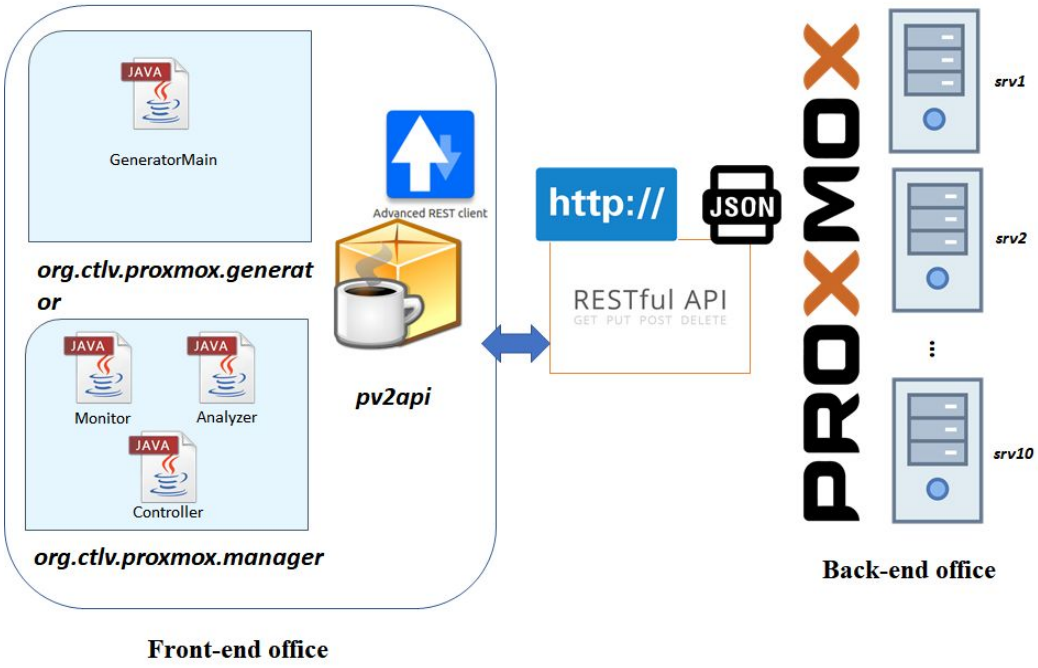

Figure 2

3. Notable information

CT naming rule:

Basic rule (first lab reminder) :

- container 33 created by team 2 of group A : ct-tpgei-virt-bA2-ct33

In the first lab, the identifier of the CT was predefined in the UI, you had no need to define or modify it. In this lab, you must define this identifier based on this rule:

- for all the CT you create via the API, use as a seed the range from xy00 à xy99, where :
  - x is 1 for group A, 2 for group B
  - y for the team number
- For instance, team A2 : identifiers range from 1200 to 1299
- NB: If your program runs for a certain while, you must implement a strategy to reuse IDs

of your own range.

Coin flip with heterogeneous probabilities:

Exemple: Random flip of a coin, 75% head and 25% tails.

```
u = random(0.0, 1.0)
if (u ≤ 0.75) then
      result = HEAD
else
      result = TAILS
```

Generating events under a uniform law:

Example: the duration between two events follows a uniform law $U \mapsto \mathcal{U}_{[0, P]}$ with P the desired average duration (30sec for instance).

```
u = random(0.0, 1.0)
nextEvt = now() + u*P
```

Generating events based on an exponential law:

Example : the duration between two events follows an exponential law $X \mapsto \mathcal{E}(\lambda)$.

```
u = random(0.0, 1.0)
x = - log(u)/λ
nextEvt = now() + x
```