Proxmox Java API yPBL cookbook

by Aurélien Tamas-Leloup

last update: 22/12/15

Table of contents

Abstract

The goal of this cookbook is to help you to get started with the Proxmox API in Java. This is going to be a powerful tool to build cloud based applications. The Proxmox API is RESTful which means you will need a HTTP client to access it, that is exactly what I provide to you. This project was initially built by Andras Elso but he abandoned the work back in 2013. Two years later, as part of my cloud school project I forked the repository and I am working on the API ever since.

Before you start, be sure you have a working Proxmox server (either online or on VirtualBox), this tutorial will not explain how to install it. For the needs of the tutorial and for obvious security reasons, I am going to do the demonstration on my personal machine with Proxmox running on Virtualbox. Just keep in mind that it can totally be done with Proxmox online. I will start with a few general definition that you will need to follow this tutorial. Then I will give you some recipes which will guide you through the API.

Keywords

Cloud computing, hypervisor, scalability

Contributors

Authors	Reviewers
Aurélien Tamas-Leloup	

Releases

Releases	Date	Author(s)	Description
0.1	18/12/2015	A.Tamas-Leloup	Defined recipes and added abstract
0.2	20/12/2015	A.Tamas-Leloup	Recipe 1 & 2
0.3	21/12/2015	A.Tamas-Leloup	Recipe 3 & 4
1	22/12/2015	A.Tamas-Leloup	Finalizing

Introduction

Before we begin the actual tutorial, you need to understand what Promox exactly is and what is its scope into the domain of cloud computing.

Definitions and terminology

Hypervisor

An hypervisor is a system that can create and control virtual machines. It can be software or hardware. There are two kinds of hypervisors :

- **native :** run directly on the hardware. (Proxmox)
- **hosted :** runs inside another operating system

Recipes

Recipe1: Authentication

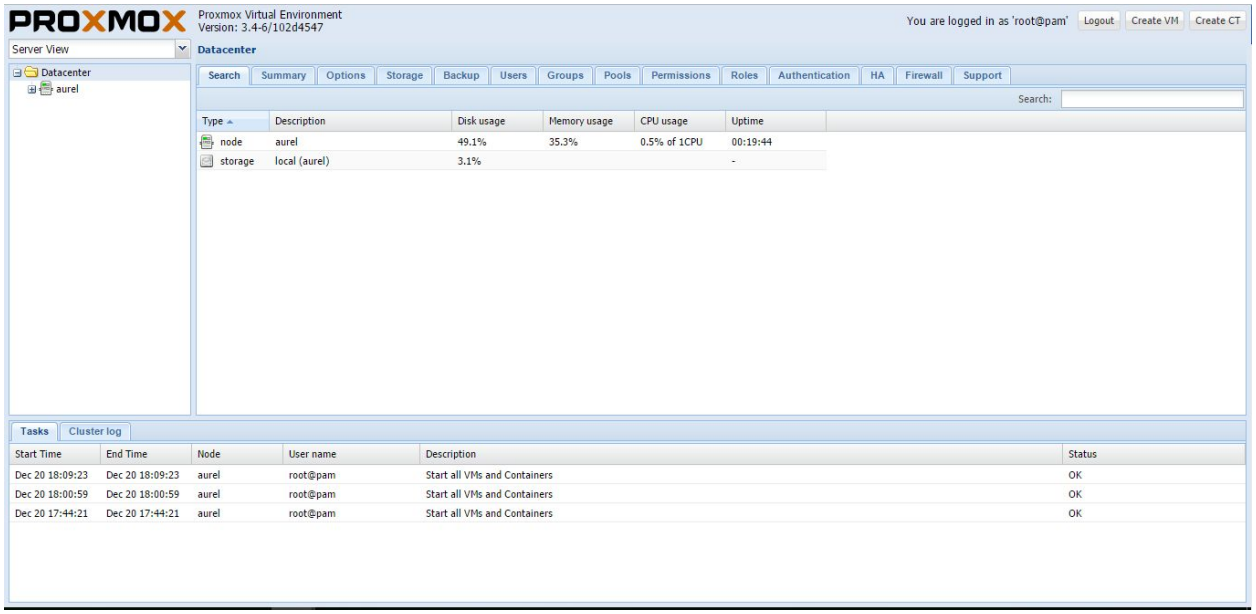
This recipe is going to teach you how to log into proxmox. Proxmox uses a token-ring based authentication. You uses your credentials to get a token and you have to reuse this token for any action you want to do.

Ingredients

- Proxmox running (either online or locally)
- A Proxmox web console access.
- The proxmox java API from [my Github]

Step1: Create an user

First log into the web proxmox console. This is the screen you should see :



Go to the Users tab and click the Add button.

You can then start typing down the user information. The only thing you need to be careful about is to chose pve over pam. Pam is reserved for the root user.

Step2: Auth as root / as a regular user

Now let's open my project in your favorite Java IDE. Create a new class and the first thing you want to do is to instantiate this api. Once done, you can call `login()`, it should look like this :

```java
package cookbook;

import net.elbandi.pve2api.Pve2Api;

public class Recipe1 {
    public static void main(String[] args) {

        Pve2Api api = new Pve2Api("localhost", "root", "pam", "admin");

        try{
            api.login();
            System.out.println("You're in !");

        } catch(Exception e) {
            System.out.println("Something went wrong");
        }

    }

}
```

The Api needs 4 String parameters which are in order :

- The hostname of the machine where your proxmox is running, if it is online use the IP adress. If it is on a virtual machine you can use a port redirection, just make sure that both host and guest port are 8006
- The username
- The realm : pam for root, pve for anybody else
- The password

Run the program with your root account and the user you created during the previous step. If the success message is displayed you are ready to continue.

Recipe2: Create and Destroy a container

Ingredient

For this recipe, we are going to use the root account.

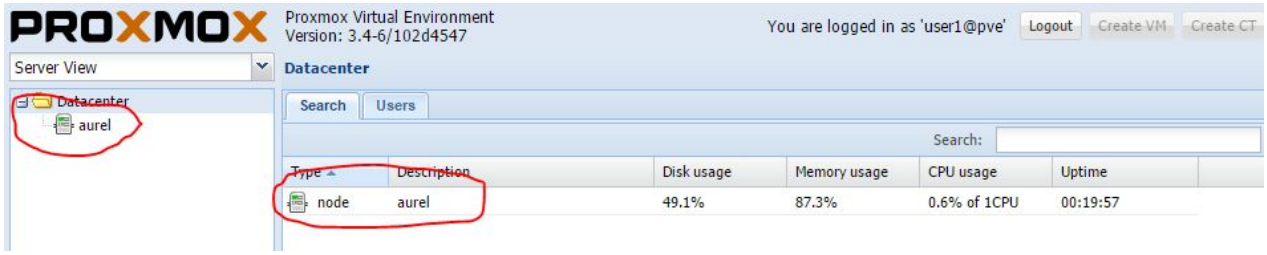
Step1: Configuring our container

We need to programmatically configure the container to create, but there are a few step that we need to achieve before. First, we need to be logged as root, you can take your code from recipe one. Second, we need to get our main node. This code will do it :

```
Node node = api.getNode("aurel");
```

Instead of "aurel", you need to put your main node name, you will find it on the web

console:



Now, we are going to create a VmOpenvz container.
First we call the default constructor to set the default settings.
`VmOpenvz vm = new VmOpenvz();`
Then you just need to manually set the parameters you want for your container.
This is the list of the ones you can set using the java code.

- ostemplate : the exact name of the template with its absolute path(see the following example)
- memory : the amount of memory (Mb)
- swap : the amount of swap space (Mb)
- IP adress : the IP adress of your container, this is optionnal
- name : hostname of your container
- node : the node where you want your container to be
- id : the unique identifier of the container

Here is an example of configuration :

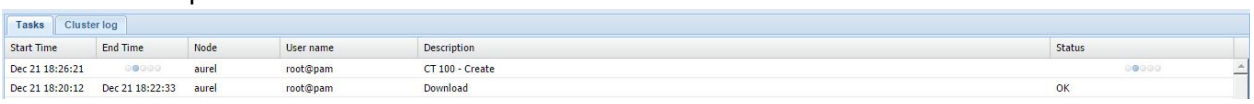
```
vm.setMemory(512);
vm.setSwap(512);
vm.setVmid(100);
vm.setNode(node.getName());
vm.setOstemplate("local:vztmpl/ubuntu-10.04-standard_10.04-4_i386.tar.gz");
vm.setName("aurel.ct1");
```

For the other ones, you are going to need to call : `setConfig(JSONObject data)`
You will need to create a JSON file and parse it in Java. This can be really useful to set up Paas or Iaas.

Both styles are equally good to me, keep in mind that everything is open source so you can basically adapt my code to your needs.

Step2: Create the container

Once you're finished with the configuration, creating the container is very easy. You just need to call : `api.createOpenvz(vm);`
This is the output :



Step3: Destroy the container

You can destroy it by calling:

```
api.deleteOpenvz(node.getName(), 100);
```

You need to replace 100 with the id you choose.

Dec 21 18:43:39		aurel	root@pam	CT 100 - Destroy	
Dec 21 18:26:21	Dec 21 18:27:29	aurel	root@pam	CT 100 - Create	OK
Dec 21 18:20:12	Dec 21 18:22:33	aurel	root@pam	Download	OK

Recipe3: Start and stop a container

Ingredients

Keep the same project. Create a few containers either with the web console of from your code.

Step1: Get all the available containers

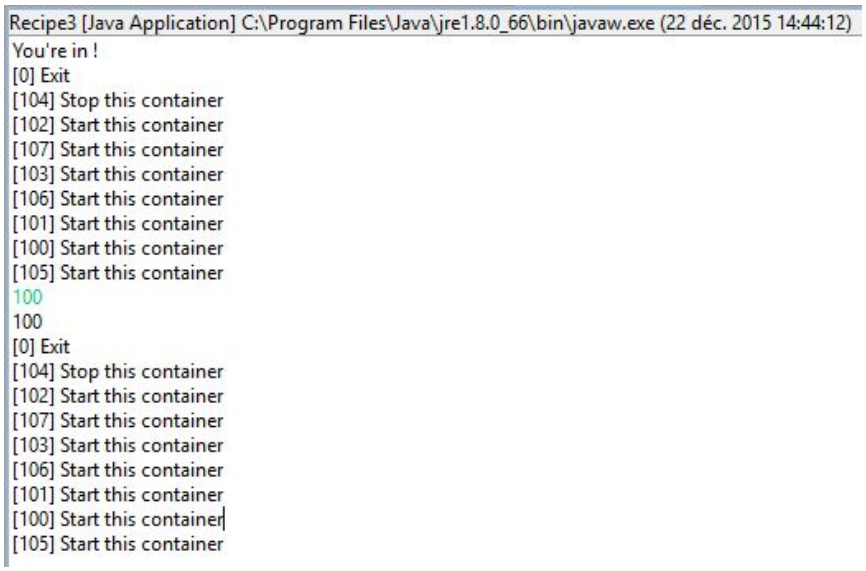
To get the list of containers, there is only one method to call :

```
api.getOpenvzCTs("aurel");
```

With "aurel" being replaced with the name of your main node. This method will return a list of `VmOpenvz`.

Step2: A simple controller

Now let's build a very simple console application. Take the code from the class Recipe3 and try to understand it. This is just a simple loop that ask the user which container to turn off/on. Note the fact that the API is RESTful which means that the calls are asynchronous. That's why you can see this kind of execution :



We want to start the container 100 but once we've done it, the list doesn't update ! Of course it worked but what is the problem with my code then ? In fact, getting the list is very quick and container creation a bit slow. Calling wait wouldn't be a good idea because we cannot really estimate the time to wait. You need to keep this trouble in mind when you will be using the API.

You need to remember that a stopped container does not consume any resource except from disk space. In most case, it is a better solution to manually create all your containers than dynamically create them.

Recipe4: Resource monitoring

Ingredients

Same project, a few container created.

Step1: Get the resources

To get the resources of a container, you just need to get the VmOpenvz object that represents it.

- From the list : `List<VmOpenvz> getOpenvzCTs(String node)`
- From the vmid : `VmOpenvz getOpenvzCT(String node, int vmid)`

Step2: A simple monitoring loop

Take my code from Recipe4 and try to understand what it does. It is a very basic monitor which is the first component of the well known loop of Autonomic computing.

```
You're in !
VmOpenvz [cpu=3.9745172E-4, cpus=1, disk=Infinity, diskread=0, diskwrite=0, maxdisk=0, maxmem=536870912, maxswap=536870912, mem=29822976, name=CT104, netin=0, netout=0, nproc=10, status=
VmOpenvz [cpu=0.0, cpus=1, disk=0.0, diskread=0, diskwrite=0, maxdisk=9223372036854775807, maxmem=536870912, maxswap=536870912, mem=0, name=CT102, netin=0, netout=0, nproc=0, status=sto
VmOpenvz [cpu=0.0, cpus=1, disk=0.0, diskread=0, diskwrite=0, maxdisk=9223372036854775807, maxmem=536870912, maxswap=536870912, mem=0, name=CT107, netin=0, netout=0, nproc=0, status=sto
VmOpenvz [cpu=0.0, cpus=1, disk=0.0, diskread=0, diskwrite=0, maxdisk=9223372036854775807, maxmem=536870912, maxswap=536870912, mem=0, name=CT103, netin=0, netout=0, nproc=0, status=sto
VmOpenvz [cpu=0.0, cpus=1, disk=0.0, diskread=0, diskwrite=0, maxdisk=9223372036854775807, maxmem=536870912, maxswap=536870912, mem=0, name=CT106, netin=0, netout=0, nproc=0, status=sto
VmOpenvz [cpu=0.0, cpus=1, disk=0.0, diskread=0, diskwrite=0, maxdisk=9223372036854775807, maxmem=536870912, maxswap=536870912, mem=0, name=CT101, netin=0, netout=0, nproc=0, status=sto
```

From the information you will get here, you can identify the problems (Analyse), decide what to do (Plan) and finally put your decisions in application (Execute).

The Proxmox Java API can provide everything you need to achieve such an application.

Recommended documentation

- Proxmox API
- Hypervisor

Feedback

If you encounter any problem trying to run my code or if you have any improvement or suggestion to make, you can contact me : aurelien.tamasle@gmail.com