CERN

SFT GROUP

SUMMER STUDENT REPORT

Enabling the μCernVM for the interactive use case

Author:
Vasilis NICOLAOU

Institute:
University of Manchester

Supervisor:
Dr. Jakob BLOMER

September 2013

Abstract

The μCernVM will be the successor of the CernVM as a new appliance to help with accessing LHC for data analysis and development. CernVM has a web appliance agent that facilitates user interaction with the virtual machine and reduces the need for executing shell commands or installing graphical applications for displaying basic information such as memory usage or performing simple tasks such as updating the operating system. The updates are done differently in the μCernVM than mainstream Linux distributions. Its filesystem is a composition of a read-only layer that exists in the network and a read/write layer that is initilised on first boot and keeps the user changes afterwards. Thus, means are provided to avoid loss of user data and system instabilities when the operating system is updated by fetching a new read-only layer.

Title of project: Enabling the μCernVM for the interactive use case

Author: Vasilis Nicolaou

Supervisor: Dr. Jakob Blomer

September 2013

Contents

Chapter 1

Introduction

The project aimed to develop a web appliance agent for the μCernVM[1] the successor of CernVM[2]. The μCernVM is a software appliance that facilitates development and accessing data for analysis from the LHC both on a local machine and on the Cloud. The project also considers problems related to updating the Linux distribution of the virtual machine. Those problems arise from the fact that μCernVM filesystem consists of two layers, a read-only (R-O) layer that is mounted from network and a read/write (R/W) layer that stays on the local machine and contains user changes.

The web application is an agent for managing and getting basic information from the virtual machine. The μCernVM has some specific tasks different from standard Linux distributions due to the nature of its file system. For instance, the update mechanism is different since it is achieved in two stages. The first stage is the update of the system packages in the network. The second is the update done on a local machine that runs an instance of the μCernVM. The updates of the two layers must be merged by avoiding package conflicts that may affect negatively the state of the system. The appliance agent has to be able to handle such special cases and communicate effectively with the virtual machine.

The project also considered the creation of the update scripts that merge the user changes with the system updates. There are two rpm databases, two passwd databases, two group databases etc. On the user side those databases are read/write, so the user changes have to be persistent up to certain a point.

Chapter 2

Software Development

This chapter concerns the development process of the various project objectives. They consist of the web appliance agent and the scripts responsible to merge the updates from the read-only layer and the read/write one.

2.1 Web appliance agent

The web appliance agent is a web application that runs under Apache server. Its core is written in Python using the Common Gateway Interface (CGI). It has been implemented using technologies such as HTML5, CSS3, XML, Javascript and AJAX with jQuery[3].

2.1.1 Configuring and using Apache

Apache server[4] has the daemon service `/etc/init.d/httpd` . The service is responsible for handling the requests Apache serves or responses to. Its behaviour depends on mainly two configuration files. The one is the main configuration which handles http requests and the other is for configuring SSL functionality to serve https requests. Many instances of httpd can be executed with different configuration files. This is very handy in terms of allowing the user to keep using the Apache service at the default TCP ports 80 (http) and 443 (https) and also use a different paths to run various web applications.

2.1.2 Development with Python and CGI

Web applications usually consist of a set of small stand alone applications independent from each other which make development and testing easy. Python is great for developing web applications since its syntax and toolset help in creating small applications fast. Finally, it supports CGI functionality for both development and debugging.

CGI is an easy way of developing web applications and it is supported by most web servers. In principle, any executable file can be run by apache as a CGI program.

Use of the CSS frameworks Blueprint[5] and Bootstrap[6] along with the python modules was done in order to provide the appropriate feel and 'umami' to the the user interface. Javascript was also imported to provide live feed information using Ajax via the jQuery library.

2.2 The update mechanism

The update mechanism consists of a script that has the capabilities of checking for new updates, ensuring the sanity of the updates against the user changes, applying the updates and patching the update on reboot with the user changes. Patching is done by adding to the new rpm database the user packages that were detected on pre-reboot stage and by executing the post installation scripts of any new packages that the update installed. During the rebase which takes place every time a new update is installed, the user database is also synchronised with the corresponding database that is loaded from upstream.

2.2.1 Merging the user database

The user database consists of three files; /etc/passwd, /etc/group and /etc/shadow. The /etc/-passwd file contains information such as the username and user id of each user as shown in Table 2.1. The users' passwords are stored under /etc/shadow. Its structure is illustrated on Table 2.3. The /etc/group contains the system groups with their information such as the group id and the group members as shown on Table 2.2.

On boot time, when CVM-FS is mounted as a read only layer there are two versions of each file. One version in the path /root.ro/cvm3/etc/ and one in /root.rw/persistent/etc . The solution to the problem requires the creation of a third version based on the previous two. The rules of the algorithm that merges the databases are:

1. If an entity with the same username/groupname exists on both versions keep the user version. If id's are different, update the map file by putting a line `<rw id> <ro id>`.

2. If an entity on the RO version has the same id with an entity in the user version, assign a new id and update the map file as on step 1.

3. If shadow files have different passwords for a username, keep the user defined version.

4. If group entities contain different users as members, merge the lists to one avoiding double insertions.

5. All id changes are recorded in a map file. There is a map file for group ids and one for user ids.

The merging has to be done before any service is executed. On boot time, only some core modules are loaded as a ram file, which initialise the rest of the system. The merging of the user databases has to be part of this ram file.

2.2.2 Merging the RPM databases

The Red Hat Package Manager (RPM) is the default package manager of a Red Hat based Linux distribution such as Fedora or Enterprise Linux. An rpm package normally contains a spec file with the information of the application to be installed and the scripts to be executed. Files needed from the application can be inside an rpm or downloaded by the installation scripts on running rpm install.

A divergence from the upstream RPM database might occure since users are allowed to install their own packages. Thus, the rpm database on the RO layer and the one on the R/W

Table 2.1: Structure of `/etc/passwd` entries which are separated with ':'

Field	Property
1	username used for login, unique for each user
2	password, usually an x which indicates that password is stored encrypted in /etc/shadow
3	User id, unique for each user, 0 is root
4	Group ID of the primary group that the user belongs to
5	Contains extra information about the user
6	The absolute path of the home directory of the user
7	The absolute path of a command shell that is given to a user on login

Table 2.2: Structure of `/etc/group` entries which are separated with ':'

Field	Property
1	The name of the group
2	password, generally not in use thus it is left empty
3	Group id, unique for each group
4	Group list, contains the usernames of the users that are members of this group

Table 2.3: Structure of `/etc/shadow` entries which are separated with :

Field	Property
1	username used for login
2	the password, encrypted
3	date indicating the last time the password was changed
4	Minimum number of days that the user is allowed to change their password
5	Maximum number of days for which the password is valid
6	Number of days before password expiration that a warning will be sent to the user
7	Number of days after password expiration that the account of the user will be disabled
8	Date indicating when the password expires

layer are not anymore considered consistent. Every time there is a system update, the two databases have to be merged and the produced version must replace the database on the R/W layer.

The RPM database contains information of the installed rpm packages. It is not a relational database but a Berkeley database[7] which provides data in key/value format. The database is also maintained and used by the RPM community. That makes the approach of how to merge the two databases a major problem which requires a well thought development decision. The reason is that the requirements are only to put the header information of a package in the rpm database without touching the rest of the system.

The process for merging the two RPM databases is fairly straightforward. Before the update, the user packages are detected. Then the replicas of these packages[1] are constructed and kept under /etc/cernvm-update/packages/rpms. On restart, a new system database is available. It is copied in a temporary directory and the rpm is executed to try and install the packages but it is called with the –dbpath argument to point to the database under the temporary directory. If the process is successful, the RPM database in the RW layer is replaced with the temporary one.

Before rebooting and installing the updates however, a lot of checks have to be done to ensure that no conflicts exist between the new system update and the user packages. Furthermore, there are some post installation scripts that might be defined in an RPM package.

2.2.3 Applying and patching the updates

If a conflict is found the user is prompted to either cancel the update or agree in removing the packages that cause the conflict. When conflicts are resolved, then the system is ready for applying the updates.

The applying of the updates happens by removing the CVM-FS_snapshot file that contains the tag which represents the current system version. Then the target update must be registered in a context file. As soon as everything is ready, the post installation scripts are prepared.

On next boot the updates will be installed. Then the updates that come with the system have to be patched with the user changes. The update script merges the RPM databases. Then the post installation scripts are executed via installing an empty rpm replica which contains those scripts and uses a temporary -initially empty- database. The `nodeps` argument is specified to avoid checking for dependencies in an empty database.

[1] rpm packages containing only header information, no executable scripts

Chapter 3

Results and deliverables

3.1 Web appliance agent

The web appliance agent provides three types of functionality and interaction. A live feed menu provides real time information about memory usage, disk usage and notifications for updates. There is also static information for the kernel release and the hostname ip address. The other two menus are used for navigating.

3.1.1 Core functionality

The actions that were developed as part of the system are:

1. Power Management: The μCernVM can be shut-down or rebooted from this section (see Figure 3.1). Unlike the desktop interface this does not require password authentication to take action.

2. Change Password: Action that uses `htpasswd` utility of Apache to change the password of the web application admin account (see Figure 3.2). Authenticates with the current password, checks if the two new password entries match and then changes the admin password.

3. Services: An embedded web application that displayes the levels of each service and allows the user to enable or disable a service via its corresponding on/off widget (see Figure 3.3).

4. API Management: Enables the API functionality and returns the username and the key to be used for validating API post requests as illustrated in Figure 3.4.

5. Updates: Provides information about new updates and an easy way for applying any new updates (see Figure 3.5).

3.1.2 Shell based actions

The shell based actions are bash commands that their result is displayed on the screen. For each command a title is specified. Then, the title of the command is shown with the result below it. Figure 3.6 shows the results of three commands with 3 different titles and their results. The

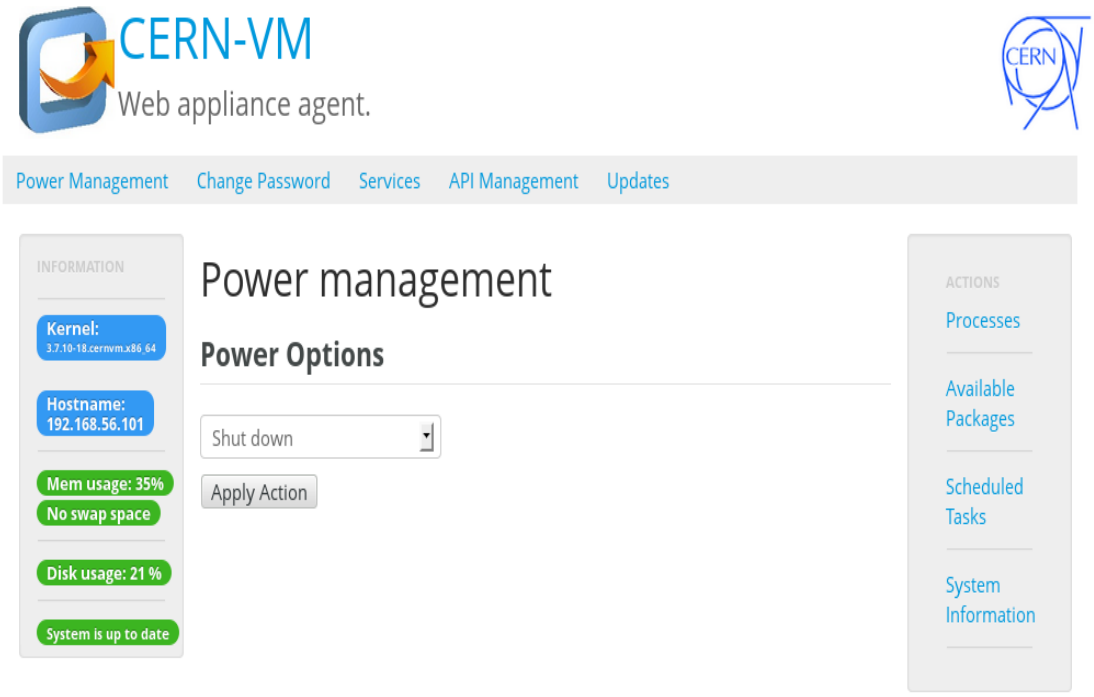

Figure 3.1: Power management section of the appliance agent.

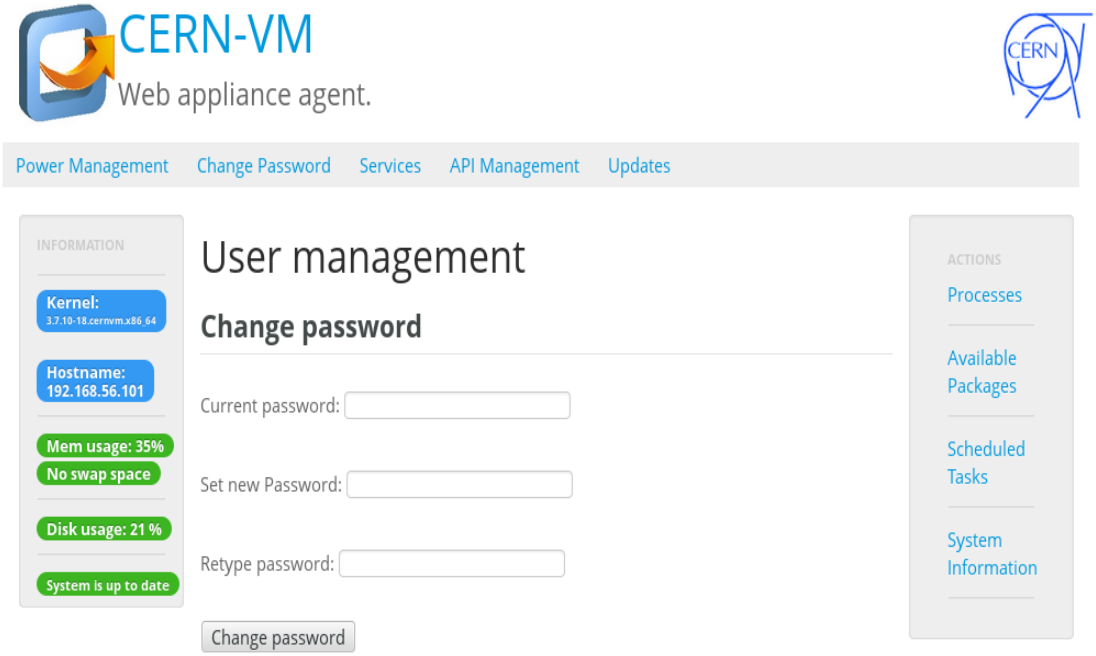

Figure 3.2: Section of the appliance agent that provides the capability for changing password.

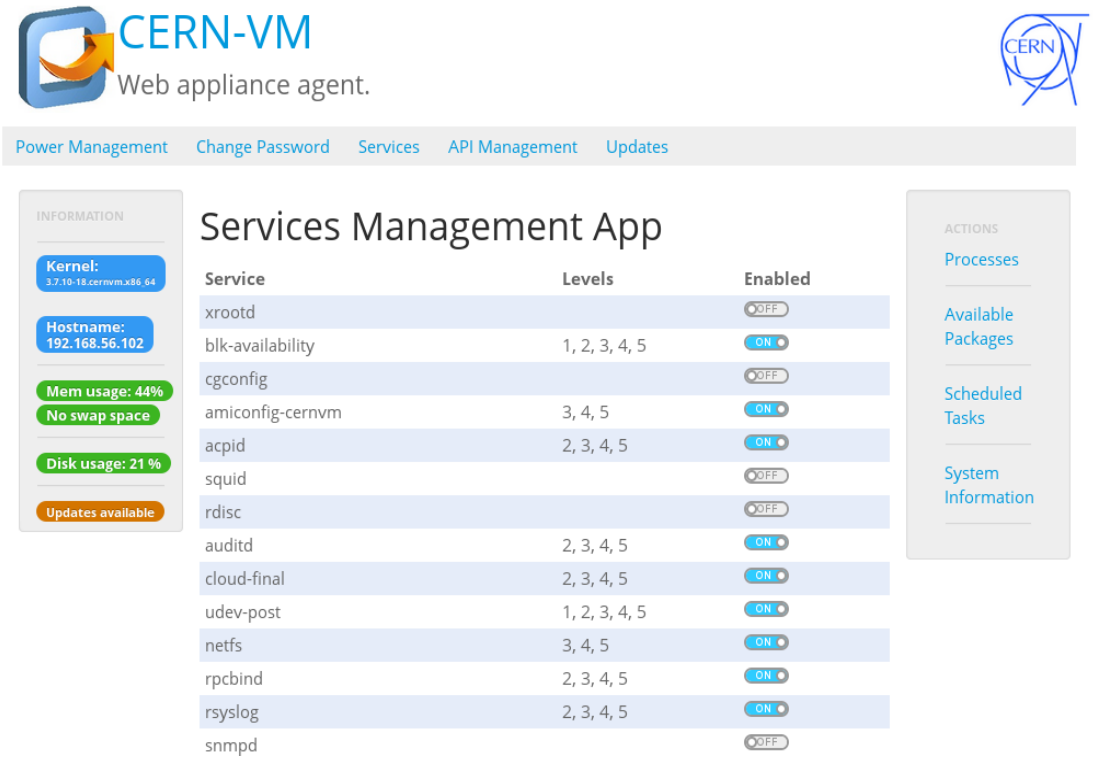

Figure 3.3: Embedded app for managing services.

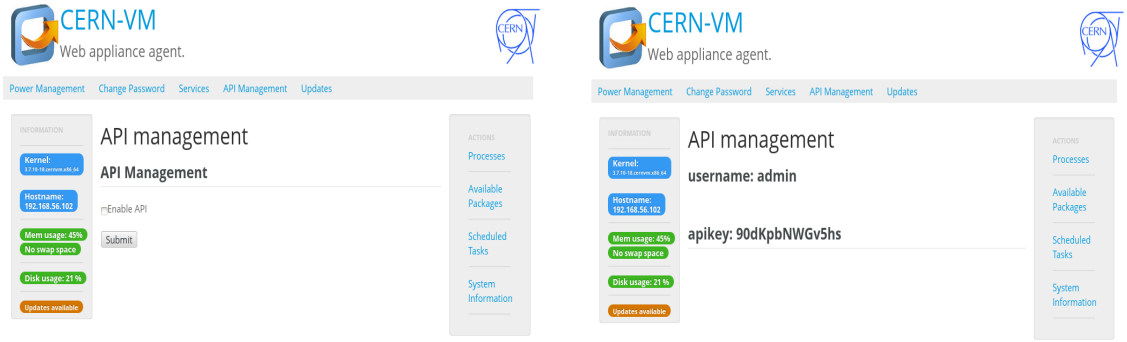

Figure 3.4: Section for enabling/disabling the API capability and getting information to be used for authentication inside the post request.

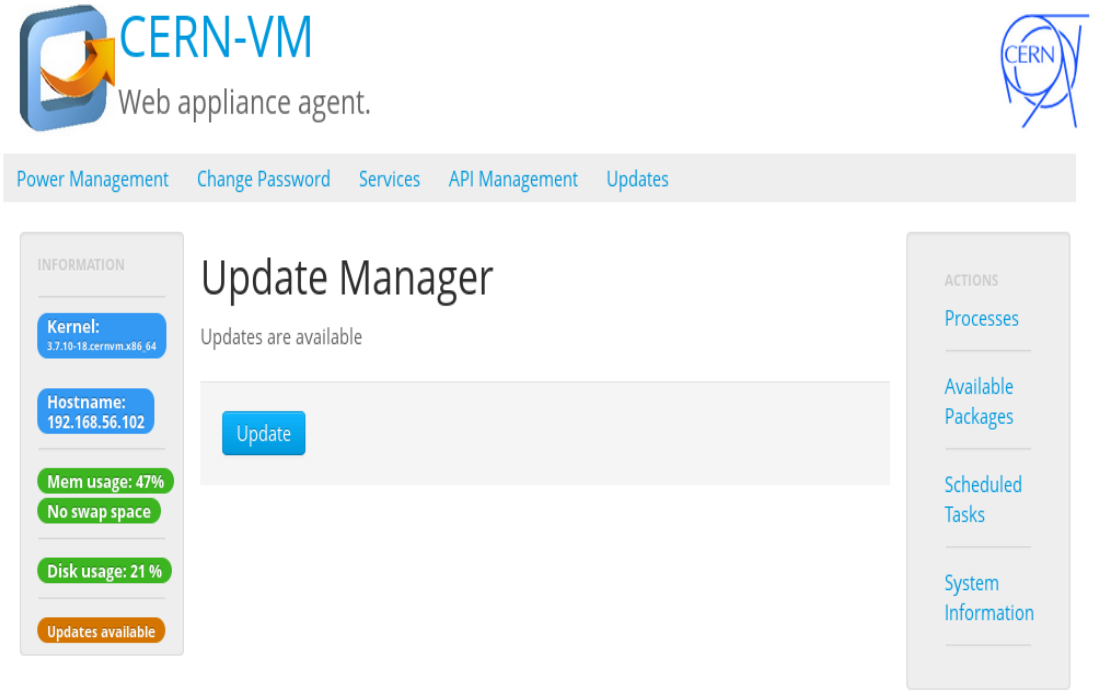

Figure 3.5: Section of the appliance agent to make the update process easy.

result of a set of commands is displayed inside a blue frame. Such a set is called a command-group in this context.

There are two ways to expand the action list of the shell based actions and add more. The one way is to directly add extra actions in the actions file which is in the configuration directory of the web application[1]. Since modifying configuration files is always risky an API is supplied.

3.2 The update mechanism

The update mechanism consists of two parts. The first is the `cernvm-update` script which uses the functions developed for merging the rpm databases. The script that merges the user databases is deployed in the init scripts. The second is a script that is deployed as a cron job and checks periodically for updates. It notifies at most once per day except if there is a newer update than the last notification specified. The notifications are send to the desktop environment via a `notify-send` call. The result is shown in Figure 3.7.

[1] `/usr/libexec/cernvm-appliance-agent/etc/config/.actions`

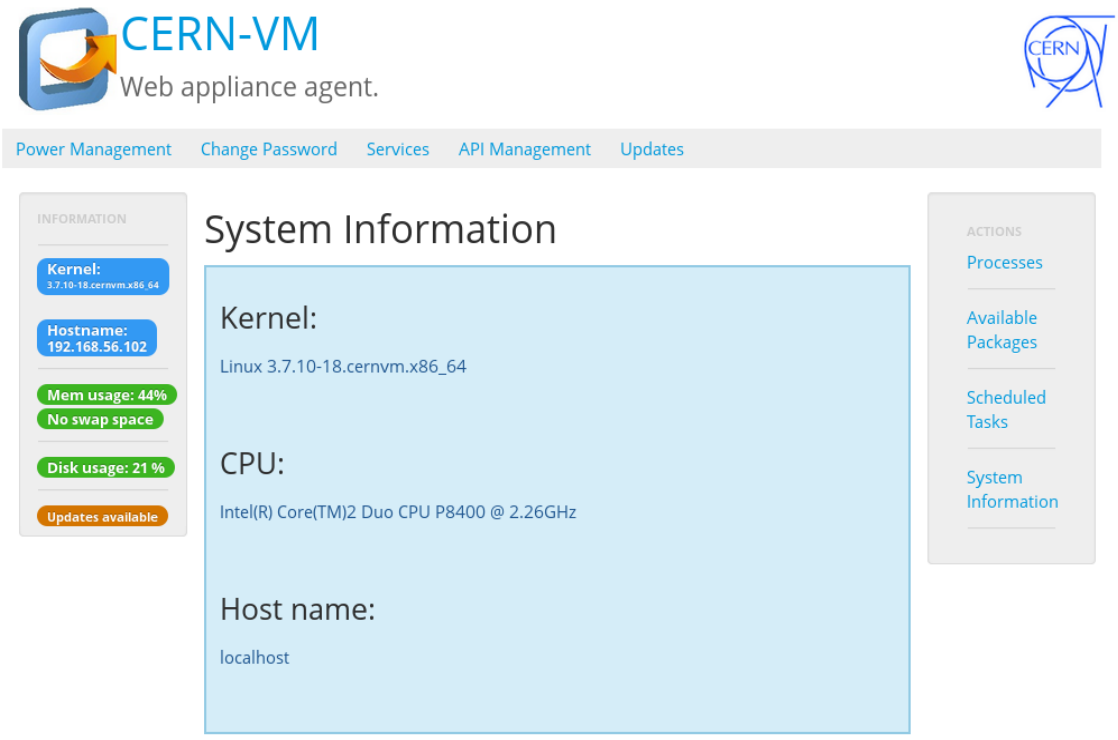

Figure 3.6: When bash script commands are executed the web applications displays the results using a blue frame.

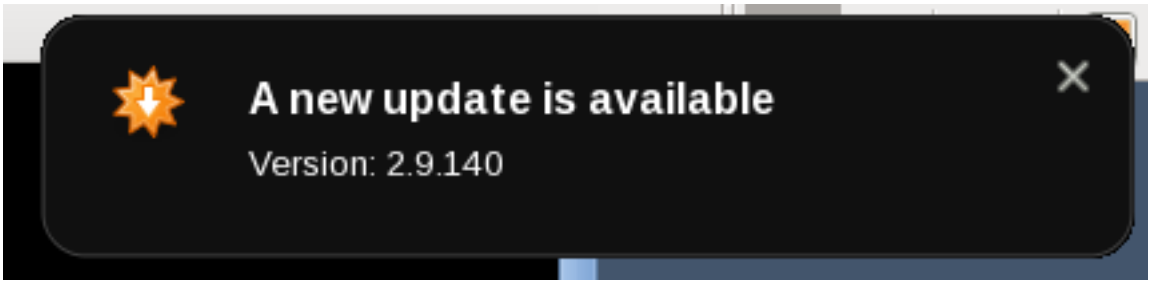

Figure 3.7: Notifications pop up once a day except if there is a newer update than the last check.

Chapter 4

Discussion

There are three distinguishable parts implemented during this project. The web appliance agent, the cernvm-update script that manages the updates on pre and post stage and the account-merge.sh script that merges the user database of the R-O and R/W layers into a new version for the R/W layer.

Although a test driven development was followed, the cernvm-update script and the account-merge.sh need to be tested by users on the real system to identify if they cause glitches or unstable behaviour despite that they were tested against the specified requirements. The question is whether the changes to the databases have a negative effect on the system.

The web appliance agent works out of the box and can be used theoretically on any Linux distribution with few exceptions that contain calls of shell commands specific to the μCernVM such as the update script.

Developers that will maintain the web application can use the framework capabilities in order to expand it with actions more complicated than simple bash commands. This can be done via calling the view module as summarised in listing 4.1.

Listing 4.1: Illustrates how the framework can be used for displaying contents on the screen

```
sys.path.append(os.environ['MY_HOME']+'/etc/config')
sys.path.append(os.environ['MY_HOME']+'/cgi-bin/chrome')
from cern_vm import Configuration
from view import View
#[..]#
config=Configuration()
view = View(config.system.actions)
view.setContent("My application", "Hello World")
```

4.1 Conclusion

In the new framework was built in a very short time frame and the final result is a very lightweight application that is easy to use and expand by both users and developers. The goal was to follow some specifications of the previous appliance agent to maintain familiarity of the new program. For instance, port definitions, initial authentication credentials and some

functional capabilities were kept. However, there are additions such as the API which lets the users define their own commands.

The update scripts work as specified by their requirements. They were tested extensively in an isolated environment. Testing was done on a real system as well but it was not enough to ensure that there will be no conflicts in the future.

To sum up, a web application was delivered with a nice interface that it can be easily expanded by both users and developers. Core functionality of the cernvm-update script appears to be working. However a beta version should be released for identifying bugs early.

Bibliography

[1] Cern, "μCernVM Preview." http://cernvm.cern.ch/portal/ucernvm, August 2013.

[2] Cern, "CernVM Software Appliance." http://cernvm.cern.ch/portal/, October 2012.

[3] jQuery, *version 1.10.2*. The jQuery Foundation, 2013.

[4] A. H. S. Project, *version Apache 2.2.15 (Unix)*. The Apache Software Foundation, 2013.

[5] O. Bjorkoy, "blueprint." http://www.blueprintcss.org/, 2007.

[6] J. T. Mark Otto, *bootstrap 2.3.2*. Twitter, 2013.

[7] B. C. Eric Foster-Johnson, Stuart Ellis, "Rpm guide." http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/, November 2011.