Design and development of a Python testing framework installer

Pieter-Jan Robrecht

May 7, 2017

Abstract

Televic Rail has developed a Python test framework for testing various products. The software, that needs to run on different platforms, uses multiple drivers and libraries. In order to test products correctly, the framework is often updated because of the release of a new driver library or to support new products. The installation process is time consuming, error prone and should therefore be automated. By automating this process, it becomes possible to collect information about the installation and update processes. The purpose of this article is to find an efficient solution and to develop a prototype. The prototype is divided in three components: a packager, a deployment server and a deployment environment. In the first part the packager is designed. This is responsible for the assembly of the software components. Phase two consists of the development of the deployment server. The server distributes all the installers and gathers information on the deployment environment. Lastly, the deployment environment is treated. In this isolated environment, installations and updates can be done safely. After a thorough evaluation, the first prototype design may be adjusted. The prototype will be expanded in the final stage to accommodate a reporting mechanism about installed versions, deployment status, etc. .

1 Introduction

Televic Rail, a Belgian company situated in Izegem, has more than 30 years of experience in the design and maintenance of on-board communication systems. This international company combines their knowledge and experience with a constant drive for innovation to deliver cutting-edge solutions for reliable communication in trains. At the same time, Televic rail designs several mechatronic sensors and safety control systems. Each device or system is designed to comply with the railway industry standards.

To meet the stringent safety standards, Televic has designed a Python testing framework to submit their products to several test scenarios. The framework was designed to be used on a powerful testing tower but has since been adapted to be usable on smaller computers as well.

Several hardware drivers and Python libraries are used to ensure a correct functionality of the Python testing framework. The amount of drivers and libraries is growing as the number of products Televic fabricates increases. This leads to a complex installation and update process which is time-consuming and error prone. In addition to the increasing number of drivers and libraries, there is also a growth in number of users. Because of these problems, there is an increasing demand for an application which facilitates the installation and update processes and provides a administration interface for client monitoring. The purpose of this work is to find a lasting solution for these problems and develop an application.

2 Architecture

[5, 8] describe the different stages in the deployment life cycle. Typical stages such as an installation, an update and an uninstallation are all present in modern day deployments. Each stage is equally important but each stage has different needs. Environment and manageability issues are the main issues that causes a difficult deployment process [6]. Architectures and technologies such as software dock architecture, ORYA, Ansible, Electric Cloud,... were researched in order to select the most versatile. Out of these architectures and technologies, the software dock architecture was selected because it offers the most scalability and flexibility. In the following section, the various components of the architecture are discussed.

2.1 Software Dock

As previously discussed, it is necessary to support the different needs of all stages of the software life cycle together with the growing amount of users. To support these needs, the software dock architecture was selected. [7, 8] discus the software dock architecture: an agent based architecture with publish/subscribe communication between the different entities. Figure 1 shows a representation of the architecture. The architecture consists of three major components: the release dock, the field dock and the event service. [7] describes how a publish/subscribe architecture is used in the event service to enable scalable communication between the different docks.

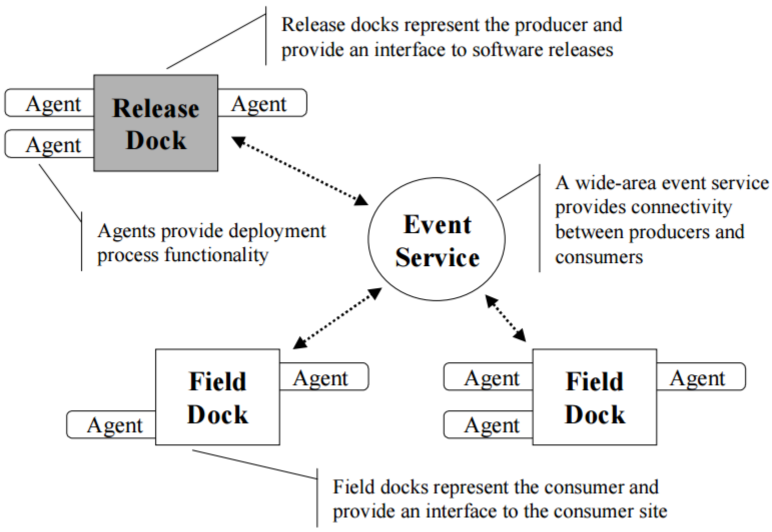

Figure 1: Software Dock Architecture [8]

The release dock is a software component running at the software producer. It contains a software repository that users can use to select the software to deploy. A Deployable Software Description (DSD) file is used to create a semantic description of the software. The DSD file together with several agents will be linked to a release [8].

When software is marked for release, the software will be transferred from the release dock to the field dock together with the DSD file and several agents. These agents use the DSD file to perform a part in the deployment process. Agents are capable of subscribing to different events. Every change to the release causes the triggering of an event which in term will trigger the appropriate agents [8].

The field dock provides information about the resources and configuration of the user system. With this information, it is possible to build a context in which the release are deployed. When software is being released, the proper agents will dock themselves in the field dock and use the interface provided to install the release. This agent based approach and the use of an interface to describe the users ensures that each release can be personalized [8].

An important part of the software dock architecture is the event service. This deals with communication between the different docks and is a pivotal component in the architecture. By using a publish/subscribe architecture, it is possible for brokers to guide incoming messages to the right recipient. [10, 4] describe how the event-service might be implemented.

2.2 Software packaging

The Python testing framework consists of several software libraries and hardware drivers. Each library and driver could be seen as a separate package. Libraries and drivers will typically be written in different programming languages and will be handled differently. [3] states that the use of extra software enables the programmer to bridge the gap between different technologies. Various technologies such as Qt Installer framework, Chocolatey, NSIS, . . . can be used to combine packages into one installer which can then be used to install every package. To implement the packer functionality, a system based on the Qt installer is designed. Other software packaging solutions (such as NSIS and WiX Toolset) also provide a solution for combining software packages. The Qt installer framework is chosen because of the better separation of software packages. This gives a better overview of what is present in an installer and which installation scripts from the meta folder affect the data.

2.3 Docker

As mentioned previously, environment and manageability issues are the main causes for an error prone deployment process. Different strategies can be used to counter these issues. Rollback strategies might be implemented to negate the negative effects of a bad installation. [11] discusses several strategies. Some strategies are more laborious to realize than others. [5] suggests virtualization as an alternative.

With Docker, an alternative virtualization technology, it is possible to create small containers in which the installation take place. A comparison between the normal virtualization architecture and the Docker architecture is visible in Figure 2. [9] states that the resource use of Docker is lower in comparison with virtual machines.

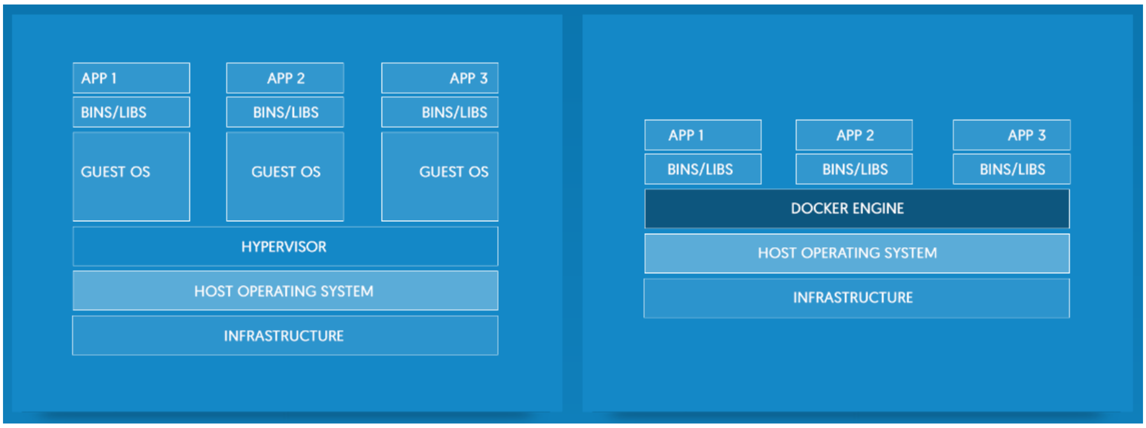

Figure 2: Comparison between VM's and Docker [1]

3 Implementation

Current architectures and technologies do not meet expectations and provide no conclusive solution to all Televic issues. Most technologies offer a partial solution. In order to solve the problems, an architecture is designed that consists of a combination of the discussed architectures and technologies.

The final design of the application consists out of the four parts: a deployment server, a packager, a database and a deployment client. The software dock architecture is used as architecture. By using this architecture, it is possible to combine the deployment server with the packager and the database into a release dock and deploy the deployment client as a field dock. The use of the software dock architecture provides a scalable architecture which allows an increase of users and software producers. Both the release dock and the field dock should not track which docks are present in the network as that is the broker's task.

Using a similar structure as the Qt installer framework, a packager is designed that can handle different types of software packages. The Qt installer framework itself is not enough as it is not possible to create an installer that works on both Linux and Windows on one operating system. Since the packager itself is designed, it is possible to choose any programming language. Therefore, a language that is fully operating system-independent is chosen. By combining the packager with the software dock architecture, it is possible to personalize the installation of each software package separately together with the personalization of each step in the deployment process. The packager is included in the release dock code and is used to produce installers.

A final element of the architecture is the deployment environment that will consist of the field dock. The purpose of the deployment environment is to create a secure environment in which the installer of the release dock can be installed and updated. As already indicated, the installation process and update process are error prone and a solution must be found for this. This will be solved by using Docker containers. Information about the success or failure will be made available to the release dock by using a database. This database will store all important information about the client system and installation processes.

4 Evaluation

4.1 Strength

The use of the software dock architecture provides a scalable and flexible application that supports the continuous increase of user and software packages. The designed application provides a good basis for Televic to expand.

In addition, Docker provides a safe installation environment. This causes errors during the installation process to have no effect on the system and a working version of the software will always be available.

4.2 Weaknesses

A weakness of the architecture lies with the broker. As the broker has to handle all messages, it forms a bottleneck in the architecture. However, there are solutions where the broker returns a handle to the docks [2]. Another strategy is to use multiple brokers.

Security is also creates a weakness. The sending and receiving of messages is currently not safe because all messages are sent unencrypted. This can be easily solved by adding cryptographic capabilities to the application.

The main weakness consist of further completing the application. In addition to the installation process, it must also be possible to perform an update process. There are already some methods available that can be used.

4.3 Opportunities

With the designed architecture, it is possible to deploy different software to a group of software users. This opens up the opportunity for Televic to collaborate with different software vendors to distribute software between software users.

4.4 Threats

A threat to the application lies with the use of Docker. In case of problems with the Docker Python module or Docker for Windows, Televic is dependent on a separate group of developers who should resolve these issues as quickly as possible.

4.5 Tests

Several tests were conducted to test the capabilities of the prototype. Different scenarios were conceived to check whether the designed methods for software distribution and installation were viable. Test were run to check whether the prototype could be used on multiple operating systems. These test have shown that the designed methods are viable but also that several functionalities are not yet supported.

5 Conclusion

The purpose of this thesis was to design a solution for the complex installation and update processes. In addition, it must support an increasing number of user and software packages. It was important to provide a solution that is both scalable and flexible.

To solve this problem, the software dock architecture is used. By combining this architecture with a software packager whose architecture is based on the Qt installer framework and docker, it was possible to provide a flexible and scalable solution. The implementation of the design provides a good starting application that Televic can easily expand and adapt.

Several tests were run to test the capabilities of the prototype. These tests show that the designed methods for software distribution and installation are a viable solution for the problem.

References

[1] Docker Main Page. https://www.docker.com/, 2016. [Online; consulted 16-08-2016].

[2] Richard M Adler. Distributed coordination models for client/server computing. *Computer*, 28(4):14–22, 1995.

[3] John R Callahan. Software packaging. Technical report, 1998.

[4] Antonio Carzaniga, David S Rosenblum, and Alexander L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[5] A. Dearie. Software deployment, past, present and future. In *Future of Software Engineering, 2007. FOSE '07*, pages 269–284, May 2007.

[6] Eelco Dolstra. *The purely functional software deployment model.* Utrecht University, 2006.

[7] Richard S Hall, Dennis Heimbigner, Andre Van Der Hoek, and Alexander L Wolf. An architecture for post-development configuration management in a wide-area network. In *Distributed Computing Systems, 1997., Proceedings of the 17th International Conference on*, pages 269–278. IEEE, 1997.

[8] Richard S Hall, Dennis Heimbigner, and Alexander L Wolf. A cooperative approach to support software deployment using the software dock. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 174–183. IEEE, 1999.

[9] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[10] Peter R Pietzuch and Jean M Bacon. Hermes: A distributed event-based middleware architecture. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 611–618. IEEE, 2002.

[11] Sudarshan M Srinivasan, Srikanth Kandula, Christopher R Andrews, Yuanyuan Zhou, et al. Flashback: A lightweight extension for rollback and deterministic replay for software debugging. In *USENIX Annual Technical Conference, General Track*, pages 29–44. Boston, MA, USA, 2004.