Journal of Information Systems Engineering & Management, 1:1 (2016), 65-81

ISSN: 2468-2071

Architecture and method for optimization of cloud resources used in software testing

Joana Coelho Vigário[1*], Cláudio Teixeira[1] & Joaquim Sousa Pinto[1]
[1]*Universidade de Aveiro, PORTUGAL*

ABSTRACT

Nowadays systems can evolve quickly, and to this growth is associated, for example, the production of new features, or even the change of system perspective, required by the stakeholders. These conditions require the development of software testing in order to validate the systems. Run a large battery of tests sequentially can take hours. However, tests can run faster in a distributed environment with rapid availability of pre-configured systems, such as cloud computing. There is increasing demand for automation of the entire process, including integration, build, running tests and management of cloud resources.This paper aims to demonstrate the applicability of the practice continuous integration (CI) in Information Systems, for automating the build and software testing performed in a distributed environment of cloud computing, in order to achieve optimization and elasticity of the resources provided by the cloud.

Keywords

continuous integration,
architecture,
cloud computing,
build,
testing,
software

Received: 17 Jan 2016
Revised: 5 Feb 2016
Accepted: 28 Feb 2016

DOI: 10.20897/lectito.201610

INTRODUCTION

Scope

The design of an information system is not limited to the development of code. There is a need to understand the customer's vision regarding the requested system, requirements analysis, and also to understand the industry where the same will be highlighted. The design of a functional architecture and planning tasks and product iterations is also necessary, for supporting work teams. The assessment of the product by the customer and potential users is quite important for the feedback of development teams. Finally, it is essential to test the system and all its components, in order to ensure that this is ready for publication on the market and that behaves like the client intends. The practice of software testing is essential and is an asset to the design of an information system.

Nowadays, the systems can evolve rapidly, and this growth is associated with, for example, the addition of features, the change of the perspective of the system by stakeholder requiremen(Vigário et al., 2015) or the constantly changing markets, forcing the systems to meet customers and users' needs quickly.

The need of immediate response to changes requires that software testing is carried out to ensure the correct operation of an application, both from the perspective of a whole, and a part. Software testing allow evaluating that small blocks of code integrated into the stable code (such as methods or functions) work as expected and correctly, and at the same time validate that the new integration did not damage all previous content.

The system base of this study is the Information System of Justice of Cape Verde (SIJ). The SIJ is under development since 2009, is in production since mid-2014, and includes a battery of more than 3000 tests. In a universe of more than 3000 tests it is crucial to have an architecture that helps in the preparation and execution of the tests in the least possible time. Due to the complexity and wide range of SIJ tests, the

*Correspondence to: joana.coelho@ua.pt

previous implementation took between 10 to 14 hours(Brazeta, 2014) The justification for these high times is related to the conditions of the test execution environment: sequential execution, which was unaffordable. After a work of analysis and change of the configuration of tests structure, it was possible to reduce the runtime for 2.5 to 3 hours(Brazeta, 2014) using a distributed solution.

This document is the result of an article written for the CITIS ' 2015-10th Iberian Conference of Information Systems and Technologies, entitled "Continuous Integration using Cloud Computing"(Vigário et al., 2015), and also of the development of a dissertation of the Master's program in Information Systems, titled " Optimized use of software testing resources in the cloud".

Motivation

People involved in the development of an information system must meet the requirements in an effective manner, regardless of whether the same play specific roles (composing teams) or generic roles. The teams play different tasks as analyzing requirements, draw use cases, develop code, perform tests, and evaluate the product or manage the project.

The team responsible for testing the software may be part of the project as a whole, but must be a foreign to the development team. This fact is related to the advantage of having an abstract vision about the solution, managing to be more critical and exploratory probability on the wide range of possibilities for tests to be carried out. This team usually works under pressure to ensure that all components are tested with all possible combinations in the least possible time, within the limits imposed by the work methodology followed. Despite the pressure that can be felt, the team must prepare the tests thoroughly, getting at most replicate interactions performed in the syste(Sandler et al., 2012)

The SIJ testing team works on an architecture that allows running the tests in a distributed form of cloud computing, in virtualized resources, such as virtual machines (VMs), supporting a deploy setting of automated tests in the same VMs. The configuration is manual, static and managed only by the tester. The execution of tests and results are managed by the tester. The results are only made available to the entities involved, when the tester decides to carry out the execution.

Problem

With the development of a system, several features are added, or can happen a change in the system perspective, due to changes of requirements by the stakeholders. It is necessary that the system has the operation expected with the integration of the various components, making their test validation. It is expected that when a new feature is added, do not damage the code previously correct and functional, and that the whole process is dynamic.

In the architecture of the SIJ, the configuration of the test execution is static, implying that the resources of the cloud are always consumed, even when they are not necessary for the execution of the tests. For the tester would be useful that the test execution were an automatic process, at all stages, since getting the latest version of CSV, compilation without errors, the copy of services/interface/database files, the initialization of the tests and the collection of results. All these steps will facilitate the work of the team. The savings of OpenNebula resources could also be achieved by the automation of the control of the VMs, i.e., the resources would be consumed only at the time of the test execution. In addition to the resources savings, the automation of the process could still allow that the VMs were created with the configuration necessary for the performance of tests, depending on the desire of the tester, enabling cloud elasticity. I.e., the tester would have freedom of choice of the number of VMs that would perform the tests, making the management of creation of VMs and controlling the speed of execution of the same. With this structure, the profitability of the OpenNebula would be far greater, especially at night, this being the most favorable time for the execution of the tests, by a smaller number of users.

STATE OF THE ART

Contextualization

According to the content presented in "Continuous Integration using Cloud Computing(Vigário et al., 2015) continuous integration (Martin Fowler, 2000) is a software development practice where members of a team integrate their work frequently during the day, at least once. Each of these integrations is checked by an automatic build and can be validated by integration tests. The CI process is presented in Figure 1.

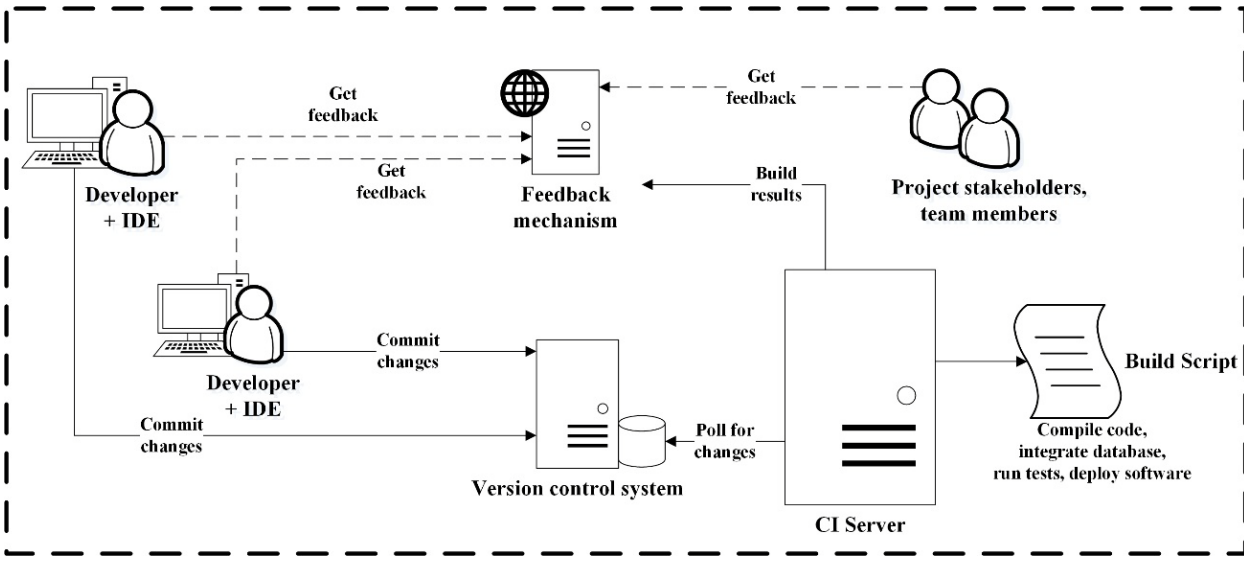

Figure 1. CI process

In Figure 1, it is possible to see that there are key actors that interact with each other, that are programmers, version control system, CI server, build script and feedback system, and these actors allow a valid execution of the CI process. The programmer, after completing a task, compiles the solution to verify that no errors occurred, and then integrates the changes into the repository. The CSV allows the tester to manage the code and other aspects related to software development, and the implementation of CI is on the main version of the repository. The CI server interacts with the CSV, searching for changes considered to be significant, and if they exist, performs a build and test the code. The build script is used to compile, test, inspect or publish the software. Finally, the feedback mechanism makes it possible to view the result.

Software integration may not be a problem in a project with small dimensions and with few external dependencies, but as the complexity of a project grows, it is necessary to integrate and ensure that the software components work together cohesively. The option to wait for the end of development can lead to poor quality of the software and the resolution of potential problems will have a higher cost, and therefore may delay the delivery of the product to customers. The adoption of this practice allows the development team to feel greater confidence in the product and more security in each code integration. It is intended that the concept of build to be seen in a wider perspective, i.e., not for just the process of compiling code, but rather a process of integration, build, test, and deployment of software(Duvall et al., 2007).

There are some validation strategies in CI for the build of the project, namely daily, weekly, among others(Kawalerowicz and Berntson, 2011). In this way integration problems can be significantly reduced, problems that are often found by the merge operation(Fowler, 2006). With the daily application of operations as merge, update and check in/commit, the integrations may be smaller and less complex, and will be easier to fix the conflicts. One of the possible strategies in CI, for example, ensures that the repository is built before and after each check-in. So, if after check-in the code, the project in question, no longer compile, the programmer must take the error correction as a priority. Typically, beyond simple compilation of code, can also run unit tests for detection of possible problems.

Martin Fowler lists best practices that should be followed by the team for CI:

- Keep a single code repository: In a software project, the files must have update/revert operations and ability to view the history of the versions of the same. These versions are managed by tools that are an integral part of the development. The tools, which do not have all the same options or the same name, has as main objective the management of versions of code;
- Automate the build: Some software development tools integrate automated compilers, such as the .NET framework, which integrates into development environment the MSBuild tool. With tools like this it is possible to compile and deploy the system with a single command. The compiler has an associated script that must contain all the necessary commands for compiling and running on any machine;
- Do the build self-testing: a program can run but this does not guarantee that it is correct. To prevent mistakes the team should made the inclusion of automated tests in the build process;

- All programmers do commit to the main version every day: the normal commit cycle goes through, previously, do an update to the code, compile, and if no errors occur (or conflict), the programmer can send the changes to the repository. If this happens often during the day, the conflicts between modified files by different programmers have a tendency to decrease;
- Every commit should build the main version in the integration machine: Some factors may influence the malfunction of the compilation, like for example, programmers don't do the normal process of commit (with update and build) or the programmer's development environment is different. So, it should be confirmed that the compilation performed before the integration works as supposed;
- Resolve failed builds right away: the CI process appears to work in a more stable version of a solution, and as such, keep the solution without errors is priority, and so if there is an error in a compilation, it has to be discovered and solved as soon as possible;
- Keep the build fast: the feedback must be available quickly, i.e., if the compilation take a long time, the team must find mechanisms to speed up the process and provide feedback to programmers;
- Test the project in an environment that simulates the production environment: it is important to develop and test the project in a clone of production environment, because often these environments are different. In this way, the programmer can avoid integration errors in relation to the production environment;
- Make it easier for anyone to get the latest executable version: all the people involved in the system should be able to see the latest production version of the system, because it may be necessary to do exploratory testing or statements;
- Everyone can see what is happening: the practice of CI wants to strengthen the system integrations and communication between the people involved. Thus, it is essential that any member has knowledge of the state of the system and of the changes that are made over time;
- Automate the deployment of the project: Sometimes, for the practice of CI, it may be necessary to have multiple test environments, (as the example of virtual machines for tests) and as such the automatic deployment in them is something very useful, by the facility that provides. Thus, the system must have a script that allows an easily deploy of the application on any development environment.

Martin considers that the best benefit that the CI practice gives is the reduction of the risk. I.e., because with its use, the programmers do not invest so much time on integration, already know at what point is the system, and it is known, at the moment, if the contributions to the system cause errors, and if so, then corrects the problem before cause greater errors in the complete system. In the book Continuous Integration: Improving Software Quality and Reducing Risk(Duvall et al., 2007) it is explored in more detail the value of CI. With the same view of Martin, the book authors affirm that the value of the CI is the reduction of risk, the reduction of manual processes carried out repeatedly, the production of stable software ready at any time, a better visibility of the project and the establishment of greater confidence in the product. To make multiple daily integrations reduce risks as the belated discovery of errors, the lack of cohesive software, low-quality and visibility of the project. CI generates software ready to publish at any moment, being this one of the most relevant aspects of the adoption of this practice. Finally, in the book is mentioned that CI establishes greater confidence in the product, because at each compilation, the development team knows the impact of the changes made in the code, and get the results of tests performed by checking the behavior of the software, for obtaining a product functional and tested. Thus, CI allows monitoring of the state of the product, several times a day, reducing the time of the appearance of a defect and the time that the same is corrected by improving the quality of the entire software(Duvall et al., 2007).

In order to adopt this practice to the desired solution, it is necessary to know the tools available to be able to choose the one that best fits the problem.

Tools for CI

For the realization of a CI architecture is essential the analysis and choice of a CSV, a CI server, a manager for compiling and testing. It can also be supplemented by a feedback mechanism and a code analysis tool. Taking into consideration the concrete case study, must also be evaluated if the tools currently in use are the most indicated for the inclusion of the process of CI.

The Version Control System (CSV)

An CSV, or also called revision control system, code control system, or version control repository, allows the programmer to see the history of the development of everything that was once added to the repository; allows to create/edit/delete files, making update or reverse, at any moment in time and of any file in the repository, allowing communication between the team members; allows to compare the versions of local files with the files that are on the server, searching by date, or label/code associated to commit in the repository. Does not save only code files, but also binary files, diagrams of database, scripts and configuration files of services and tests. This system is a key part of CI architecture but also it is an essential tool for work between a team regardless of the methodology followed.

To choose a CSV should be considered some important characteristics about the systems: Centralized vs. distributed; the transactional vs. non-transactional; blocking vs. non-blocking files; and free vs. license(Kawalerowicz and Berntson, 2011). According to the book Continuous Integration in .NET(Kawalerowicz and Berntson, 2011), the first characteristic that must be analyzed is the factor cost-benefit, related to the features offered in systems free vs. with license. Another important feature is that the system is centralized or distributed. In a distributed system, the developer has its own version of the code and the commits that performs are local, being considered a private repository. There is a public version in the repository, managed by an administrator, which instructs to join all versions. In other hand, a centralized system is the opposite and all persons working on the same version on the same server. Typically these systems are more appropriate for CI architecture, for being easier to incorporate and require less knowledge about the infrastructure from the repository. In respect to the topic of a system be transactional or not, is no longer a problem of modern systems but is as important as the other aspects, as operations being atomic, as for example, a commit to be done as a whole. This means that if there is any problem during operation, the system will stay in the prior state in the commit attempting, and this is not performed. Finally, a system that allow the blocking files, allows a person to stay with a file to itself, preventing another member to edit this same file. In the non-blocking files it is possible to edit the same file on all occasions, and may lead to conflicts.

Were analyzed three systems that allow the management of version control.

Team Foundation Server (TFS)

The TFS provides a core functionality for development teams, such as project management, monitoring of work items, management of testing cases, build automation, reports, lab management, environment management feedback and version control, blocking the edition of a file for a single user(Blankenship et al., 2013). It is a very complete tool, which provides features for integration with Visual Studio (VS), thus containing a window of version control, and team explorer, which allows to see the features described above. Is a tool designed to any actor of development, as a programmer, tester, architect and project manager. TFS is a system that requires activation key/license.

Subversion + TortoiseSVN

Subversion (Mason, 2010) is a centralized control system (like the TFS). It is a system that is stable and is used in many organizations, is free and open source. The developers work from their environment and only require access to the server in the commit time, allowing more than one person to work on the same file. This tool can detect the information from files version, directories and metadata, i.e., can get to know the history between changes and renaming directories, unlike other systems. Has an atomic commit, i.e., or all content is submitted or the changes are not sent to the server(Mason, 2010). O TortoiseSVN is a client of subversion for Windows that integrates directly in Windows Explorer and allows the normal options found in a repository.

Git + TortoiseGit

Git (Swicegood, 2010) is a distributed tool, i.e., the commit can be done to the local version without the need of connection to the server. The project has its own repository (private) and a common repository for the team (public). Git allows all developers do the normal operations or that there is an administrator to do so. The commits are recognized with an identifier instead of numbers or labels of revisions. In Windows, the users can use the TortoiseGit, which makes the control of version of the GIT, implemented as an interface of Windows commands(2015d). Like Subversion, is a free tool.

Summary

After consideration of the tools, it is necessary to choose one of them. The CSVs presented do not differ greatly between them as regards the main features that the generality of these tools offers. Table 1 shows the summary of characteristics previously specified.

The TFS is the tool used in this moment in the development of the project, it is easy to interact and integrates directly with the VS 2013 (IDE of current development) and in this case the team is already familiar with its operation. In this looming, the TFS was chosen as CSV for the developing architecture.

Table 1. Version control system summary

	TFS	SVN + TortoiseSVN	Git + TortoiseGit
Centralized system	X	X	
Lifecycle management	X		
Visual Studio integration	X		X
File blocking	X		
Transactional	X	X	X
Open Source		X	X
Chosen solution	☑		

The Continuous Integration (CI) Server

A CI server guide the process of CI, verifying changes in the repository and coordinating the process steps. Ideally is a server that runs something predefined on each change of the code and that delivers immediate feedback to the development team (typically has its own feedback mechanism). Normally these servers are driven by a configuration file (script) that specifies the steps to perform during the build process and integration. It is advisable to have a machine that only run the CI server because a process correctly created should have the lowest number of possible dependencies(Kawalerowicz and Berntson, 2011). A compilation may be scheduled, for example, to every hour, or once a day at the same time, among other options.

Were analyzed three systems that behave with CI server.

Team Foundation Server (TFS)

The TFS is more than a system that manage the software life cycle and controls the repository of code, is also a CI server from Microsoft and as such it fits well with .NET technologies. Allows the orchestration of all necessary components for the development process from the development of software, management requirements and project, development of testing and quality assurance(Ehn and Sandstrøm, 2012). The server build can be performed by MSBuild.

CruiseControl.NET (CC.Net)

The CC.Net is a CI server, implemented using .NET, which can run with console or Windows service. This version is based on a version of the tool in Java. The server automates the process of integration by monitor directly the repository, validating the changes of the code updates. The CC.Net can remotely manage the process using a system called CCTray, which can access the server to initialize builds or be notified of the occurrence of a build. The compilation of the server can be performed by MSBuild or Nant.

Jenkins/Hudson

Jenkins is a CI server based on Java, used for projects from .NET, Ruby, PHP, Grails, and including Java(Smart, 2011). It is an open source technology, has various plugins to support communication, implementation of testing and integration with other systems, and has a simple configuration based on web interface, taking also the possibility to use the command line to perform the required actions. Jenkins performs the compilation of project via a feature called job, which consists in the execution of a series of tasks(Berg, 2012). The compilation of the server can be performed by MSBuild or NAnt.

Summary

The servers presented note some differences between them with respect to its structure. However, these differences are minimal if we consider the features that are desired to use. Table 2 shows the summary of characteristics previously specified.

Thus, given the previous decision in case study, in relation to the TFS, we opted to keep it as a server for CI.

Build management

To perform the compilation of the code it's possible to choose, among other options, the MSBuild and NAnt. They allow the use of scripts to compile the complete project using only one command. There are systems that use the MSBuild for the code and that use the NAnt to integrate other tools, such as for example for the execution of the unit tests.

MSBuild

MSBuild is an integral part of the .NET Framework and is automatically embedded in VS (button build). The configuration is defined by an XML document. It is limited to the .NET Framework, has some integrated features and it is actively developed(Kawalerowicz and Berntson, 2011). The XML document has the definition of properties and the targets (Debug, release, among others), where each target contains a number of tasks(Blankenship et al., 2013).

NAnt

NAnt is drawn from a Java tool called the Apache Ant. It is an open source tool maintained by a community that can be used on any platform and is not actively developed. It is, like the MSBuild, controlled by a configuration document in XML. It is a good option for developers who dominate Java and which are familiar with Ant(Kawalerowicz and Berntson, 2011).

Summary

MSBuild is more appropriate in a .NET environment, being integrated in VS. As such, was the chosen tool. Table 3 shows the summary of characteristics previously specified.

This solution can, however, be complemented with the NAnt for executing unit tests in other environments, or in specific tasks in which the syntax of the NAnt fits better.

Use of the TFS for software testing

TFS has effective mechanisms for the automation of tests. One of the examples is the MSTest tool. This tool allows the performance of automated tests, view the results of the tests and save them to disk as well in TFS (files .trx). In the direction of automation, and assuming that the testing execution is distributed, it is possible to have recourse to the entities tests "controller" and tests "agents" of TFS.

The test controller is the service responsible for controlling the implementation of tests, i.e., publishes the results of same and coordinates the session of these for the tests agents. The test agent is the service that is

Table 2. CI Server summary

	TFS	CC.Net	Jenkins
Lifecycle management	X		
Visual Studio integration	X		
Documentation	X	X	X
JAVA based		X	X
Open source		X	X
Chosen solution	☑		

Table 3. Build management summary

	MSBuild	NAnt
Open source		X
Cross-platform		X
.NET based	X	
Configuration by XML file	X	X
Visual Studio direct integration	X	
Chosen solution	☑	

always connected to the controller, is installed in a test machine and that allows the tests execution(Rossberg and Olausson, 2012).

The results file generated in the execution, by VS and per type, is analyzed by the tester. In this file are presented information related to the results of tests, as for example, the number of performed tests (864), of which 841 were successful, 20 had failure and 3 were inconclusive. Also shows what agent ran the test and the details associated with some errors that occurred in failed tests.

Use of Cloud Computing for CI

In the book Cloud Computing Bible(Soninsky, 2011), of Barrie Sosinky, the author says that the "cloud" is used for long time. The term cloud computing refers to technology, applications and services that run on a distributed network, using virtualized resources and accessed by Internet standards and protocols.

The term "cloud" implies the definition of two concepts: abstraction and virtualization. Abstraction due to the abstraction layers in the details of the system implementation towards the users and developers, where they do not have to worry about the hardware, with the storage of information or systems administration. Virtualization for the fact that cloud computing virtualizes systems to manage available resources, allowing to distribute scalability, making an agile infrastructure, and also by allow the adjustment of costs to the needs of the cloud costumers.

Cloud computing can be divided in sets of models: deployment and services. Deployment models relate to the location and management of infrastructure, and in accordance with the definition of the NIST, are divided into private, public, hybrid and community cloud. The public cloud infrastructure is available for use to the public, belong to organizations that sell cloud services. The infrastructure of a private cloud is exclusive to the use of an organization and is managed by its own. The hybrid cloud combines multiple types of cloud, forming a unit when they interact among themselves. And finally, the community cloud is a cloud that was organized to serve a common purpose, being integrated in one or several organizations that share common concerns as mission, security guards or policies.

Services models of refer to accessible services from cloud platform and are divided by Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

IaaS provides virtual machines, virtual storage, virtual infrastructures and other hardware components as features that customers can use. Amazon Elastic Compute Cloud (EC2), Eucalyptus and GoGrid are examples of IaaS. PaaS provides virtual machines, operating systems, applications, transactions, services and control structures, managing the cloud infrastructure. Force.com, GoGrid CloudCenter and Google AppEngine are examples of PaaS. And finally, SaaS is a complete operating environment, with applications, management and user interface. Normally it's exposed to the customer by an interface accessed via a web browser, allowing the user to manage his information. SQL Azure GoogleApps, and Oracle On Demand are examples of SaaS(Soninsky, 2011).

Two valuable characteristics in the services of the cloud are scalability and elasticity. These terms mean different things for the cloud, and according to Chip Popoviciu(Popoviciu, 2013), occour in different moments of the life cycle of the business. The scalability of cloud supports long-term strategic needs, i.e., the demand for the services was planned and provided, without the need for major investments in infrastructure. The elasticity supports the short-term technical needs, allowing to dynamically adjust the features depending on the demand for services.

OpenNebula

In the SIJ structure it is used the OpenNebula tool for the execution of software testing. This platform provides a simple solution, but flexible and rich in features for the management of the resources associated with the cloud(2015c). The OpenNebula works as a wrapper, i.e., integrates any component and/or any infrastructure system or other software, aggregating and centralizing the existing operations. Was developed to which these integrations were simple to perform and use. Thus, the result is a modular system that can implement a variety of architectures and can interact with multiple services. It is considered an Infrastructure as a Service (IaaS), given that provides management of images and context, storage, network, monitoring and calendaring, virtualization, management of users and roles(Aranda, 2013). The OpenNebula Sunstone is a graphical user interface that works as the center of operations of OpenNebula and is used by administrators and users, simplifying the typical operations of management as managing users, servers or virtual machines(2014).

Most of the features of the OpenNebula are available through services and API's, cataloged internally as cloud interfaces and system interfaces. The first is to develop tools for the end user, with a high level of

abstraction of cloud features. This interface, even simpler and nearest conceptually of the needs in this case study, was discontinued. The second sets out all OpenNebula functionalities and it is used to adapt and adjust the behavior of OpenNebula in the desired infrastructure(2015a). The system interfaces provide low-level API's. These APIs for integration are the XML-RPC and OpenNebula Cloud API (OCA). OpenNebula also offers interface drivers, which allow interaction between the OpenNebula and the remaining infrastructure. The areas covered by the drivers are storage, virtualization, monitoring, authorization and networking.

XML-RPC API

The XML-RPC interface is the primary interface of OpenNebula and sets out all the features of the same. This interface is used, for example, to develop specialized libraries for cloud applications or for the cases in which the developer needs a low-level interface to the core of OpenNebula. Allows the control and management of any feature of the cloud, including VMs, using actions, such as for example poweroff, suspend, resume, restart, among others.

Ruby/Java OpenNebula Cloud API (OCA)

OCA is available in Ruby and Java libraries, and allows a simple integration, in these languages, with the core of OpenNebula. It is composed of a set of libraries that facilitate communication with the interface XML-RPC(2015b). This API must be used if it is to develop an IaaS tool that needs total access to the features of the OpenNebula(2015a).

OneFlow API

The RESTful OneFlow API is a service to manage, create, control and monitor applications with multiple layers or services compounds for interlinked VMs. All data are sent/received by JSON(2015a).

Summary

The XML-RPC and OCA API's are based on the same structure, differentiating on language of communication, while the OneFlow is based on HTTP requests by JSON. In this way, the API chosen is the XML-RPC because is simple to be used and provide the desired features (Table 4).

Table 4. OpenNebula APIs summary

	XML-RPC	OCA	OneFlow
Simple	X		X
Low level	X		
Key features	X	X	X
Intermediate layer		X	
RESTful service			X
Chosen solution	☑		

SOLUTION

The use of the practice of CI and cloud computing for automation and distributed execution of tests is essential for the optimization of the development of a system. The tests allow ensuring that the system operates as expected when adding a new functionality. I.e., when a new feature is added, the system operation must remain correct. The use of cloud computing allows the improvement of the tests runtime (since they run in parallel), and the cloud could become elastic. This means that OpenNebula can dynamically adapt to the needs imposed by the tester, and thus be able to add and/or remove VMs that perform the tests more quickly, and schedule the test executions to obtain results when desired.

The design and implementation of an architecture involves the study and modeling of components that are necessary for its preparation, and the analysis of use cases, data models, workflows and sequence diagrams, to better understand how to pair the intended structure.

In this section it is possible to observe the design and implementation of an architecture that enables the build automation and tests runs, allowing the optimization of the cloud elasticity. It is performed the analysis and modeling of several components that help to understand the procedures in the configuration service of automatic build configuration. This service, also described as "Script", has the intention of doing all the management of VMs and test execution. It was studied the library that facilitates the communication between

the architecture and OpenNebula and finally will be explained the Web application "Dashboard" that assists the tester for the configuration of automatic build.

The architecture of CI to the SIJ

The structure of the architecture designed can be distinguished in three phases: Build configuration, CI Structure and Communication with the cloud.

Build Configuration

To support the tester in the setup for build configuration was developed a web application that behaves as the dashboard. The component in the diagram that represents the build configuration can be seen in Figure 2. The web application communicates with a SQL Server database to store the data defined in the same database.

CI Structure

In Figure 2 is presented the normal structure of CI. There is the involvement of programers who work on a development environment, where they put the changes in CSV, while the CI server demand by changes in this repository. The CI server runs a script that contains information about the build configuration, integrates the database, performs tests and makes the software deploy to the cloud.

Communication with the cloud

The script is a vital part of the architecture because consumes the database records that contain information about the build configuration (Figure 3). Performs the compilation and makes the system deploy, communicating with the cloud to manage the VMs that will perform the tests remotely.

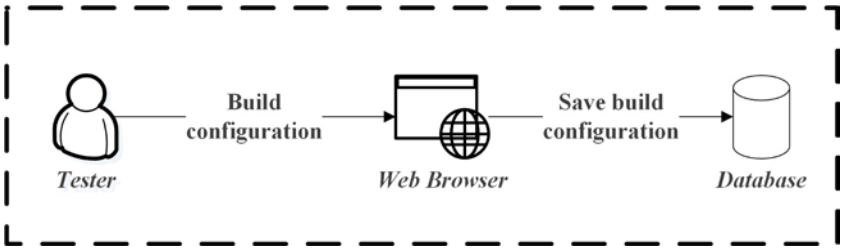

Figure 2. Architecture diagram – Build configuration

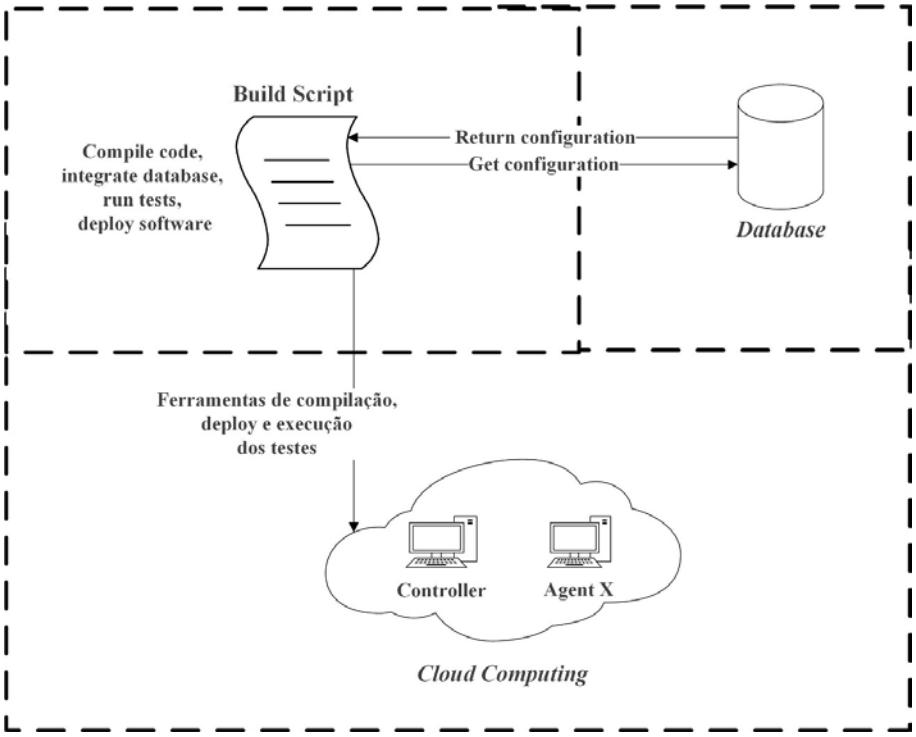

Figure 3. Architecture diagram – Script

After defining the necessary components, a CI proposal is presented, taking into account the objectives pursued for the SIJ. Since the system is based on .NET platform, was selected for the editor of development the VS2013, with the CSV and CI server the TFS. The build, deploy and the test execution are run by the tools MSBuild, MSDeploy, and MSTest, respectively. For the distribution of the tests by the VMs are used the test controller and tests agents of the VS. The settings and management of VMs that are running the tests are carried out through a web application "Dashboard" where the data is stored in a database. The architecture can be seen in Figure 4.

Structuring the Configuration Service

The service that performs the necessary settings for the automation of the process of test execution in the SIJ system, has several functions, such as the communication with the VMs, communication with the database, deploy all the necessary files to remote execution of tests for the same machines, implementation of remote testing and feedback communication to the tester.

The service is structured into modules, where each one meets a distinctive role. There is a module that manages the build configuration, a module that has all the available methods for the VMs management (instantiation, details and actions), a module that manages the OS configurations of each VM that is hosted in

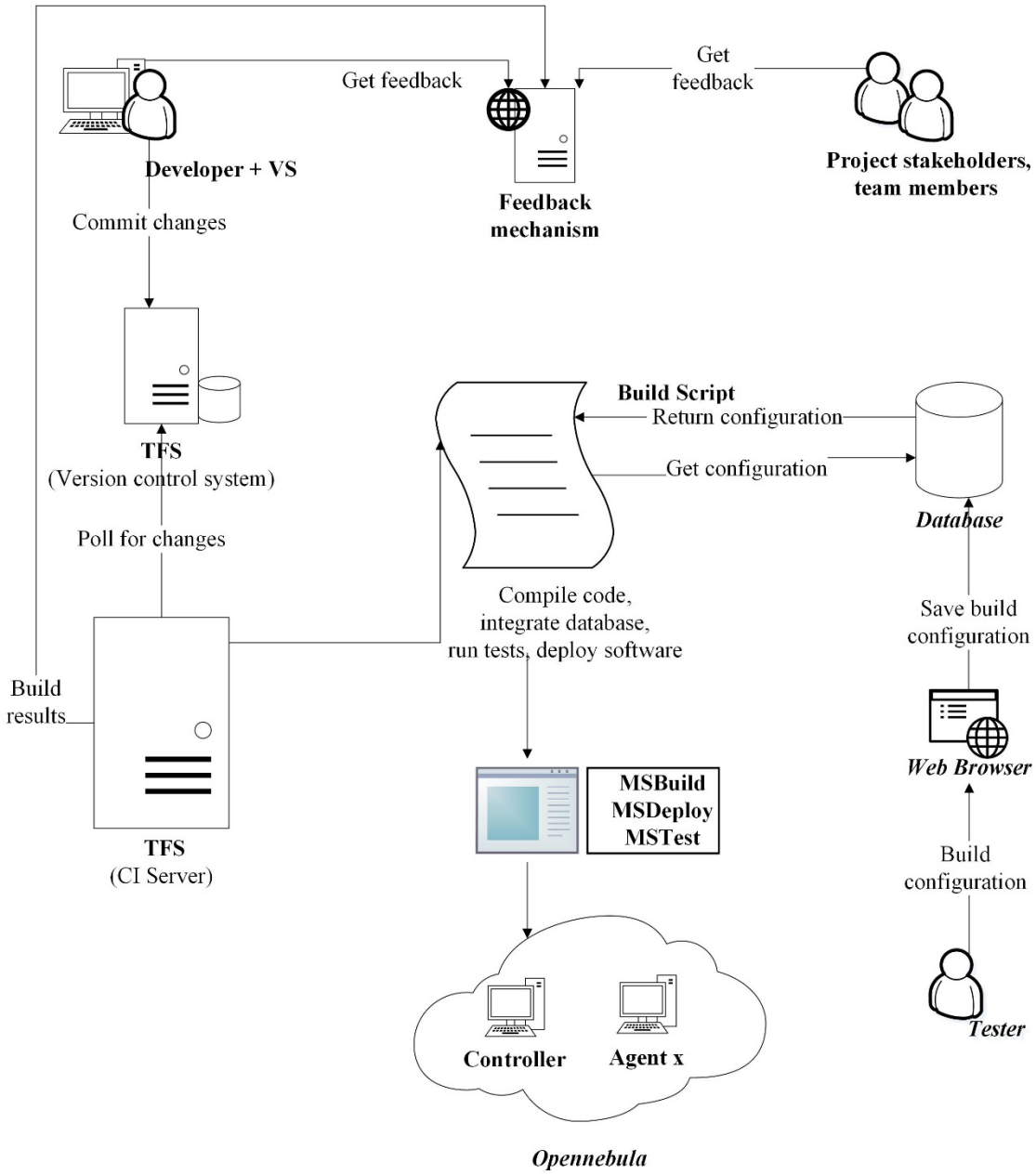

Figure 4. CI architecture to the SIJ

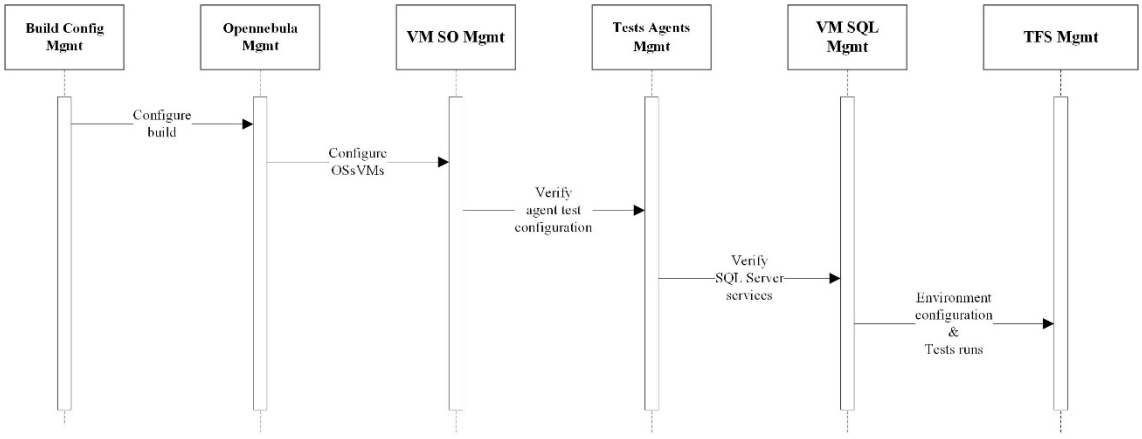

Figure 5. Sequence diagram – Communication between modules

the cloud, a module that manages the files relating to the test agent for each VM, a module that manages the SQL services of each VM, and finally a module that manages the TFS for tests execution and dissemination of results. The diagram of the modules communication the can be seen in Figure 5.

The module of build configuration manages the data compilation and communicates with the module of OpenNebula management, where there is an analysis if the VMs will be created or recovered, checking their status in the cloud (whether or not they are running). This module communicates with the module of management of Operating Systems (OS) of the VMs for their configuration (insertion in the domain network or firewall configuration). When this module ends, he communicates with the module that manages the processes of tests agents to validate the status. When these processes are validated, the communication with the SQL services module is established to validate if these are running, and finally, is communicated to the module of TFS management that can prepare the environments of VMs, run the tests and report the results. In the end, the VMs are powered off (or deleted) and the process ends.

Build flow

The VS has a tool called Team Foundation Build (TFBuild) that allows the code validation and of each build, with the possibility of schedule one or more builds. The TFBuild allows the creation and the management of processes that compile and test the application automatically, increasing the rigor and quality of builds and even support CI strategies(2016). It is possible to define the build process that contains information on the project that will be compiled and what actions will be held in the compilation, which tests will be executed, and how often will be performed. I.e., if it is desirable to perform this process day to day, or at every code integration with the repository, among others. In addition, it is possible to associate scripts that execute the same build process, such as execution of powershell files.

Based on this architecture, and taking into account the use of the API to manage the VMs, it will be possible to establish a direct connection between a task of code validation (started manually or scheduled, according to a given frequency of commit, a specific frequency temporal, etc.) and the existence of VMs in enough quantity for the test execution, in the smallest possible space of time. As soon as the tests ending, these VMs will be discarded, releasing the resources of the private cloud infrastructure. The proposal for automated workflow in cloud computing scenarios is presented Figure 6.

In component highlighted by a shredded rectangle, the tester decides to schedule an automatic build in VS (either for the moment, each integration, all days at the same time, among others.), and begins to establish connection with the build controller of the TFS. This controller define the team project collection and the team Project.

If there is no connection with the build controller, then the test team is notified and the automatic build will be canceled.

On the other hand, when there is a connection with the build controller, the process follows to a stage where all projects within the solution of SIJ are compiled, represented by the highlighted component in the dotted rectangle. If the build is performed with success, is invoked a post-build event that will run a sub process. The sub process will invoke a command-line file (extension .bat). The definition of the post-build event (that runs if the build is successful) is defined as shown in the following representation:


```
<PropertyGroup>
   <PostBuildEvent> call C:\Folder\file.bat </PostBuildEvent>
</PropertyGroup>
```

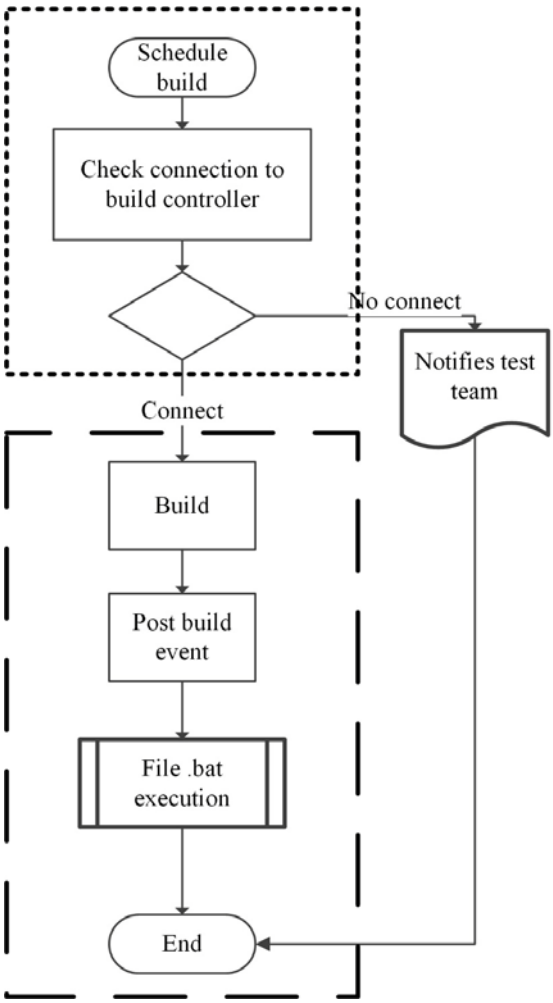

Figure 6. Build flow

XML-RPC.NET library by Charles Cook

As an essential part of the new proposal, it is intended the integration between the CI server and OpenNebula, accomplishing a dynamic and elastic management of VMs. This integration is possible by using a library.NET (2011) that encapsulates the access to XML-RPC API of OpenNebula. Several possibilities were analyzed and the library of Charles Cook is better suited to the use of core features of OpenNebula. This library allows the execution of requests to OpenNebula for managing the VMs. It is proposed the creation of a service that consume the library and that is invoked in the hour of the code validation.

To perform the communication with the XML-RPC.NET API, the programmer simply access to one of the interfaces that the library offers (or create an interface), choose the methods to be accessed and define what is the address of the cloud XML-RPC. In the following illustration is possible to see the interface that establish the address of the OpenNebula XML-RPC, and the method that allows the communication with each VM. The method in question allows the allocation of a new VM in the cloud. In the same figure it is possible to check that the library of Charles Cook offers information about which method will be invoked (description).


```
namespace OpenNebulaHosting.OpenNebulaAdapter
{
   [XmlRpcUrl("http://cloudpt2.clients.ua.pt:2633/RPC2")]
   public interface XmlRpcVirtualMachineManagement : IXmlRpcProxy
   {
      [XmlRpcMethod("one.vm.allocate")]
   Array oneVirtualMachineAllocate(string sessionSHA, string atributeValueTemplate);

      /*Description: Allocates a new virtual machine in OpenNebula.

         IN    String The session string.
         IN    String A string containing the template for the vm. Syntax can be the usual
"attribute=value" or XML.
         OUT Boolean  true or false whenever is successful or not
         OUT Int/String    The allocated resource ID / The error string.
         OUT Int    Error code.*/
   }
}
```

Web application to support the build configuration

The tester can choose to add all required settings for the implementation of automatic build manually or can choose to set them in the web application. The manual configuration implies the edition of an auxiliary file that is in development solution and that will be read only if nothing is inserted on the database in relation to the VMs. If the tester opts for web interface, achieves a simple and clearer vision of what is essential to configure. By the browser it is possible to perform simple actions for data definition and visualization. The interface was designed to be simple and practical, serving as a support of automatic build, defined by the tester, and was developed in the framework of development MVC 5 (ASP.NET) with use of Bootstrat for user interface treatment. This web application communicates with the OpenNebula for reading the status of each VM and performance data from the cloud hosts, and communicates with a SQL Server database, where are stored all the necessary information and which will be included the information that the user wishes to.

RESULTS

To prove that the implemented structure has an impact on the performance of OpenNebula, the test team made an analysis to the time of execution of the service that allows the configuration of the automatic build and studied the impact that it has on the structure previously implemented.

Was also performed a monitoring of the values of Sunstone, i.e., an analysis of the interface that holds information about VMs and the hosts of the cloud OpenNebula. It was verified the performance of other network computers, controlling and comparing the values of their behavior at the time that the remote testing of SIJ were happening, and at the moment in which the tests were not running and the tests agents were disconnected.

Runtime in previous implementation

The old architecture used for the execution of remote testing of SIJ had a manual configuration, which was static and managed only by the tester. This structure prepared manually 14 VMs that had the responsibility of performing the tests. For the preparation of test environments of these VMs, the test team has created some powershell files that allowed the mechanization of some steps to perform the machines configuration. I.e., the tester ran manually the powershell files that published the database files, services and interface in the VMs, and in this way it was possible to prepare the environments for testing. The powershell files had all their data in a static way, since what VMs would run the tests, which SIJ version or database would be used for the implementation, among other aspects. After executing the powershell files was possible, via VS, start the tests execution.

This process, although partially automated, was heavy and little adaptable to changes on the variables with which the test team need to work. The runtime of the powershell files, which permitted the preparation of VMs, was approximately 25 minutes, and the runtime of database tests (850 tests) was 40 minutes. After this

process was necessary to consult the file with the result of the run to verify them, or see the VS feedback, which provided a summary of the results.

It should be noticed that despite the average time of tests execution of 65 minutes be a reasonable value, VMs were still connected in the cloud, consuming resources of the hosts, when actually were not being necessary for any execution.

Runtime for the configuration service

The architecture implemented and studied in this document presents a configuration service to automatic build that enables the tests execution, as well as the preparation of all environments and also the management of the cloud resources. This configuration has associated a total of 14 VMs that act as tests agents. The tester can also opt to choose a configuration where defines, in the moment, how many VMs will run the tests.

To assess the impact of the architecture implemented on the process of automatic build, were analyzed 14 VMs that already existed in the cloud (considering as static configuration) and were created new 14 VMs (considering as dynamic configuration), with the same characteristics of the VMs that already existed.

The static configuration assumes that VMs can be connected and disconnected. There are two distinct states for disconnect the VMs: poweroff and undeployed. When a VM is in the poweroff state, their configuration files and disks stay in the host, occupying space on the same. In the undeployed state the disks are transferred and stored in datastore of OpenNebula, freeing space of the host.

The time evaluated covers all the intermediate steps performed by the configuration service, since the creation or connection of the VMs, configuration and verification of services, execution of tests and elimination or disconnection of the VMs. To simplify the process, were only performed the database tests (850 tests).

The times are compared, represented in Table 5 relatively to the scenarios:

- Scenario 1: 14 static VMs, disconnected at poweroff;
- Scenario 2: 14 static VMs, disconnected at undeployed;
- Scenario 3: 14 dynamic VMs, created at the time.

As can be observed in the previous table, the time is slightly more expensive in relation to the creation and elimination of VMs, and times of disconnecting, whether staying at state poweroff or undeployed, are very close, considering that the best scenario is where the VMs are static and disconnected in undeployed state.

Analyzing in detail this scenario, it is possible to split the runtime of the configuration service by "Initial configuration", "Tests execution" and "Final configuration", as shown in Table 6.

The values shown are approximations, in minutes, for better understanding of the problem. The initial setup time is 34 minutes and concerns the startup of the VMs and the configuring of the environments, i.e. the starting of VMs, checking the connection to the domain network, the verification of the implementation of SQL services and processes of Test Agents of VS, and even the publication of the files required for the execution of the tests (Windows services, database and interfaces). The final configuration refers to the disposal of results, such as the treatment of the results file and sending email to the person that is responsible for the execution, and even to the action of disconnect all VMs, and it takes 2 minutes. The runtime of the execution of the tests is 43 minutes.

Comparison of current architecture with the previous

The structure used previously for the tests execution performed the whole process in 65 minutes, using 25 minutes in the preparation of the environments and 40 minutes for the execution of tests. In the current structure, the time takes more 15 minutes to perform the previous structure.

Table 5. Runtime of general configuration

Type	Runtime
Scenario 1	1 hour and 21 minutes (approx.)
Scenario 2	1 hour and 19 minutes (approx.)
Scenario 3	1 hour and 51 minutes (approx.)

Table 6. Runtime splitted d by phases of configuration service

Tipo	Runtime
Initial configuration	34 minutes
Tests execution	43 minutes
Final configuration	2 minutes
	1 hour e 19 minutes (approx.)

Although the time of the previous run had better values, current execution needs more time precisely for connect and disconnect the VMs that will run the tests, enabling the management and configuration of the VMs. In the end the gain is greater because it allows the release of resources in the hosts, giving the possibility to other users to consuming the resources they need.

It is noted that with the currently structure, it is possible to define how many VMs will run the tests. If the tester want to increase the number of VMs, it is possible that the total time of execution decrease, by distribution of tests for more VMs, allowing the test team to observe the results more quickly and at the same time keep the structure adaptable to the needs of the moment.

Performance of other VMs of the cloud

The monitoring of the levels of Sunstone, on VMs and the hosts of the cloud, and the monitoring of the performance of other computers on the network, by monitoring and comparison of the values of their behavior at the time that the remote testing of SIJ are running, and at the moment in which the tests are not running and the tests agents are disconnected, was described and proven in the dissertation on which this article is involved.

Of the data addressed in the dissertation it is possible to check that when the VMs are connect all the time, even when the tests are not running, affects negatively the behavior of other network computers as well take up unnecessary resources of cloud hosts.

When the VMs are disconnected, the values of other computers in the cloud, and the space freed of the hosts, improve, allowing the cloud resources to have a better distribution to other users who require the same resources.

CONCLUSIONS

An Information System can become very complex as it is developed and used, and when the process of their development and testing is an automated process, developers are with more time to develop/correct system features. The use of continuous integration allows precisely the automation of this process. The development architecture that the SIJ presents today is not a solution for CI.

Thus, it was proposed the development of architecture for automating the process of development of the same. The structure presented was also conceived to be a model architecture, i.e., easily adapts to another language or tool, both of build and test. The automation of build and tests deploy will allow that the tests execution ran automatically, preferably in a night period, in virtual machines. The aim was to obtain a full control over the VMs, and also to have the possibility to allocate all possible cloud resources only during the execution of the tests, freeing the resources when it is over, and taking advantage of the concept of elastic computing associated to cloud computing. With the use of monitoring tools to determine and to analyze the use of the proposed solution, it is possible to verify that the new structure is making a difference in relation to the overload of OpenNebula.

According to the dissertation study was possible to create test scenarios for configuration of automatic build and prove that the best scenario to be adopted is using a strategy where the VMs are disconnected, in the undeployed state, not consuming resources of the hosts, and only be used during the execution of the tests.

REFERENCES

Aranda, D. M. (19 de november de 2013). *An Introduction to Cloud Computing with OpenNebula*. Obtido em 10 de february de 2015, de SlideShare: http://www.slideshare.net/opennebula/tecni-28445050

Berg, A., 2012. Preface. In A.M. Berg, *Jenkins Continuous Integration Cookbook*. Packt Publishing Ltd.

Blankenship, E., Woodward, M., Holliday, G. and Keller, B., 2012. *Professional Team Foundation Server 2012*. John Wiley & Sons.

Brazeta, J.A., 2014. Testes de software no sistema de informação da justiça cabo-verdiana. In J. A. Brazeta, *Testes de Software no Sistema de Informação da Justiça Cabo-Verdiana* (pp. 2-3;9-10;44-45;57).

Coelho Vigario, J., Teixeira, C. and Sousa Pinto, J., 2015, June. Continuous integration using cloud computing. In *Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on* (pp. 1-7). IEEE.

Cook, C., 2011. XML-RPC for .NET. *www. xml-rpc. net*.

CruiseControl.NET. (s.d.). *About CruiseControl.NET (abbr. CCNet)*. (CruiseControl.NET) Obtido em 19 de january de 2015, de CruiseControl.NET: http://www.cruisecontrolnet.org/projects/ccnet/wiki/About

Duvall, P.M., Matyas, S. and Glover, A., 2007. *Continuous integration: improving software quality and reducing risk*. Pearson Education.

Ehn, J. and Sandstrøm, T., 2012. So, what is Team Foundation Server 2012? In J. Ehn, & T. Sandstrøm, *Team Foundation Server 2012 Stater* (pp. 1-2). Birmingham: Packt Publishing.

Fowler, M. and Foemmel, M., 2006. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, p.122.

Kawalerowicz, M. and Berntson, C., 2011. *Continuous integration in. NET*. Manning.

Mason, M., 2010. *Pragmatic Guide to Subversion* (pp. 9-12; 18-19). Pragmatic Programmers, LLC.

Microsoft., 2016. *Build the application.* (Microsoft) Obtido em 3 de february de 2016, de Microsoft: https://msdn.microsoft.com/en-us/library/ms181709(v=vs.120).aspx

Popoviciu, C., 2013. *How do cloud elasticity and cloud scalability differ.* (TechTarget) Obtido em september de 2015, de http://searchtelecom.techtarget.com/answer/How-do-cloud-elasticity-and-cloud-scalability-differ

Project, OpenNebula., 2014. *OpenNebula Sunstone: The Cloud Operations Center.* (OpenNebula Project) Obtido em 15 de november de 2014, de OpenNebula Project: http://docs.opennebula.org/4.12/administration/sunstone_gui/sunstone.html

Project, OpenNebula., 2015. *An Overview of OpenNebula.* (OpenNebula) Obtido em 05 de january de 2015, de OpenNebula Project: http://docs.opennebula.org/4.8/design_and_installation/building_your_cloud/intro.html

Project, OpenNebula., 2015. *Integration.* (OpenNebula) Obtido em 23 de january de 2015, de OpenNebula Project: http://docs.opennebula.org/4.8/integration/getting_started/introapis.html

Project, OpenNebula., 2015. *OpenNebula 4.10 Design and Installation Guide.* Em O. Project, *OpenNebula 4.10 Design and Installation Guide, Release 4.10.2* (p. 1;5;26).

Rossberg, J. and Olausson, M., 2012. *Pro Application Lifecycle Management with Visual Studio 2012*. Apress.

Sandler, C., Badgett, T. andMyers, G. J., 2012. *The art of software testing.* New Jersey: JohnWiley & Sons, Inc.

Smart, J.F., 2011. *Jenkins: the definitive guide.* " O'Reilly Media, Inc.".

Sosinsky, B., 2011. Cloud Computing Bible (2011). *Indianapolis, Indiana.*

Swicegood, T., 2010. Introduction. In *Pragmatic Guide to Git* (pp. 9-16). Pragmatic Programmers, LLC.

TortoiseGit., 2015. *TortoiseGit - The coolest Interface to Git Version Control.* (TortoiseGit) Obtido em 19 de january de 2015, de TortoiseGit - Windows Shell Interface to Git: https://code.google.com/p/tortoisegit/

Vigário, J. C., Teixeira, C. and Sousa Pinto, J. (2015). Continuous integration using cloud computing. *Information Systems and Technologies (CISTI), 2015 10th Iberian Conference.* Aveiro: IEEE.

