



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

“Move to Continuous Integration on the Cloud”

Código: DISCA-221

Proyecto Final de Carrera

ITIG

Autor: Antonio Puche Pérez

Director: Ph. D. Lenin G. Lemus Zúñiga

[Mayo, 2013]

Move to Continuous Integration on the Cloud

This project is entirely my own work and has been submitted to Institute of Technology Tralee in Ireland. The project was carried out during a student exchange between UPV and Institute of Technology Tralee. This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Science (Hons) in Computing with Software Development. Where use has been made of the work of other people it has been fully acknowledged and fully referenced.

Signature:

Antonio Puche Perez

2nd May 2013

Table of Contents

Contenido

0.0 Abstract.....	5
1.0 Introduction	6
2.0 Literature Review	10
2.1 Fundamental practises on continuous integration.....	10
2.1.1 Introduction.....	10
2.1.2 Features of continuous integration.....	12
2.2 Introducing continuous integration.....	13
2.2.1 Benefits of applying principles	14
2.2.2 Concerns about Continuous integration	16
2.2.3 Dealing with continuous integration.....	17
2.2.4 When to implement Continuous Integration?.....	17
2.2.5 Complementing Continuous Integration with other development practices.	17
2.2.6 How long it takes to set up a continuous integration system?	18
2.2.7 Daily habits	18
2.3 Introduction to Cloud Computing.....	21
2.3.1 Cloud computing definition.....	21
2.3.2 Cloud structure.....	23
2.3.3 Deployment Models.....	24
2.3.2 Benefits of cloud computing	24
2.3.3 Security in cloud computing.....	27
2.4 Cloud Architecture	28
2.4.1 Cloud Characterisitcs.....	28
2.4.2 Cloud operative system (KVM).....	28
2.4.3 Infrastructure as a service	31
3.0 Methodology	34
3.1 Introduction	34

3.2 Can the cloud improve the continuous integration system?	34
3.3 Approach.....	35
3.4 Design.....	35
3.4.1 Risk Analysis	36
3.4.2 UML Diagrams	38
3.4.3 Requirements	38
3.4.4 Project Management Plans	39
3.5.1 Work Break Down	42
3.5.2 Prototype.....	42
3.6 Conclusion.....	47
4.0 Implementation	48
4.1 Background	48
4.2 The Build Server	49
4.3 Pre-required cloud instances.....	53
4.4 Altobridge build system	60
4.5 Putting all the parts together.....	69
4.6 Failed implementations and errors	76
4.6.1 Logs corrupted due to multithreading.....	76
4.6.2 Script to start and stop instances from the cloud.....	76
4.6.3 Plugin to start and stop instances form the cloud	81
4.6.4 Issues with the cloud.....	82
4.6.5 Issues with the python system.....	83
5.0 Data Collection	84
6.0 Analysis.....	99
7.0 Conclusions.....	104
Cloud Management	107
Python Build System	108
8.0 References.....	109

0.0 Abstract

Software Industry is in constant change. Since the early 1960's when the software industry expanded and computers were mass-produced the cost and availability of computers began to increase. Before this period, not everyone had an opportunity to own a personal computer due to cost and availability. However, as mass production has seen a revamp of the way we manufacture and distribute our computers, a larger proportion of the population has access to a computer which has had a knock on effect on the way in which software has been produced and developed and has contributed to continual change in the industry. (Kubie, 1994)

As we enter a new era there is a need for new software innovations and as time passes the quality of that software should be improved as our understanding expands. Customers are continually creating demand for improved and more sophisticated software to be developed which is functional and practical to professional and personal lifestyles.

To cater to the requirements of the customer, software developers are becoming sophisticated in their approach in developing more improved and technical software. New techniques are being created to improve the quality of the product and develop advanced software which will be applied to new practises and application tools.

In this research the system of Continuous Integration will be studied to investigate how the production of software by companies can be improved after following a set of practices and tools. However, to meet the requirements of companies which handle multiplatform projects the use of updated technologies, such as Cloud computing technology is necessary.

In conclusion, this research we will try to compile good practices, tools of continuous integration and list the advantages of cloud computing to deal with the very specific requirements of companies. A case will be made to prove that Cloud computing combined with continuous integrations improves the performance of a system by improving the resources management of the system itself.

1.0 Introduction

The technology industry in Ireland has become one of the major international businesses around the world over the last 3 years. The growth in the Information, Communication and Technology sector (ICT) has led to it becoming a key sector in the economy by exporting the third part of the total goods in Ireland, the software industry has helped to raise this though. (Brien, 2010)

Software industry represented the 6% of the total export in the country in 2009. Since then Irish businesses are targeting application development as one of the main sources of providing financial gain over the next few years. That makes application development one of the most important sectors for the development of the economy.

However, even though application development is gaining popularity at the moment strategies need to be developed to ensure that the product is well produced. Project management or simply projects are the upper layer on charge of managing the development of an application. A project defines the purpose and the risk that applications will have and is in this context why reliability and maintainability are hugely important.

It is vital in every software project that the use of common practises to develop good quality code is the first thing to take into account when talking about maintainability and reliability in a software project. When using good practises in the code the “*health*” of the project increases as it makes it more maintainable as bugs are identified easier and can be fixed instantly, ensuring that the project is always in a state of being “*healthy*”. Furthermore, the better the quality of the code will allow the programmer to work more confidently on it. This is because the programmer can trust the version of the software they are working on and will not need to worry that another programmer will add additional feature to the application, they can just keep on developing new features on the top of a working version of the application.

Good software practises come from experienced programmers. When developing in a team environment many members of the same team may have access to the same features and by necessity may need to modify the same files. In this situation, when advice is sought team members can look upon good software practises from experienced experts in the field. In our case the expert from whom we will take advice from will be Martin Fowler.

Martin Fowler is a well-known designer of enterprise applications and he is an expert in object-oriented systems and design patterns. His article about integrate systems and how to get benefit of continuous integration for enterprises is the bases of a lot of the content that is used in related to continuous integration, as well as other articles and books by the same author.

“Integration is a long and unpredictable process” (Fowler, 2006)

In software development the reliability of the source code is top priority for every development team in a company. Typically, the larger the project the more people employed to work on it, resulting in a more reliable version of the software that developers can trust when developing new features. This is when the process of the integration becomes necessary.

For the past few years the process of integrating the software was delayed to the last stage of the software development process. In long term projects with long stages of development the process of integrating all the code could take several months or even a year which can lead to the bigger problem of lack of prediction.

During the integration process all the parts of the application need to come together to investigate their compatibility with each other. As the difference in behaviour of the final application can change, in many different ways that is not obvious to expect. Furthermore, the second thing to do after the problem is identified is to find the cause in the code that provoked the problem and fix it. This could take a long time since there may not be any feedback from that part of the code that caused the issue in the first place, or if it's an error in the code or even an error in the design.

In order to make the integration stage of the software project reliable and predictable the use of continuous integration practises is becoming necessary. With the use of continuous integration practises one more guarantee is added to the project to make sure that the deadlines will be accomplish because you have a "*continual*" feedback of the status of the project.

Feedback of the status of the project is valuable for the management of the project because from feedback managers we can identify how far or how close the project is to completion of the objectives that were set out at the beginning of the project while also providing predicted timelines for the customer to ensure that product will be produced on time.

By applying continuous integration practises a reliable version of the software is available when required by the customer. The customer can literally "*see*" how the project is going and if any requirements or changes are needed they can be made. In this way, there is no need to wait until the end of the long project, for example one year, to check if the software that has been developed to meet the requirements of the customer. During the development process the requirements of the customers may change due to different reasons, this can be an issue if not dealt with as changes occur.

In-short to improve the management of the project and the quality of the application as well as increasing the confidence of the members of the project, the use of continuous integration practises and tools is the best solution. However, the task of carrying out the process of continuous integration might be done fast and easy for small projects where there is no need to use a lot of "*resources*" because the number of branches or the

number of tests to perform is not high. This means that it can be done in a single machine without the need of using several machines, but this is not always the case.

In some scenarios a company can hold many projects at any one given time and for each project there might be a number of products to work on at the same time. Since the number of projects and products may be large, the number of people working on them needs to match the number of features and patches that are being developed and added daily could potential be huge. Furthermore, because of the number of different combinations that need to be performed, the different types of integration depending on what features are desired to add to the product or what releases is wanted to test with what features. In fact, when working with products and features there may be a need to test the same feature in two different products and for that different integration is needed.

Having a single machine which carries out every single integration for every different product combination is not enough because there might be many of them running at the same time and there is a possibility that this would slow down the process. Another issue with having a single server carrying out this task is disaster recovery. As stated earlier, the reliability of the code is an invaluable characteristic but when speaking about hardware there needs to be a characteristic as well in our hardware system. Even if there is an integration process done and ready to work, if a sudden disaster were to happen it would stop all the process of integration and the consequences of that might set back the whole project.

The ideal situation would include having a code that is reliable and that the resources or “*hardware*” where you are deploying your process are reliable as well. Furthermore, the hardware that is going to be used needs to be flexible at the beginning of deployment process as the needs of the process may be small but as soon as the use of continuous integration becomes widely used and the frequency for each new integration there will be a need to configure the hardware resources to carry out it. This will result in a scenario where the resources or “*hardware*” that will be needed to use need to be flexible and scalable so there can be opportunities for the use of more or less resources depending on the numbers integrations that have to be done at a time.

The best technology that currently provides us the appropriate amount of resources that we need at a time is “*Cloud Computing*”. Cloud computing is a new approach to how to configure resources to be able to share in a way that it can be provided on demand through a network and with a minimal effort management. (Technology, 2011)

With this new approach of shared resources what we want to get is the ability for a given integration configuration, from now this will be referred to as “*build*”, to get enough resources from the cloud to run that build to check that the final application is still working and to get the feedback for the status of the application.

As the number of builds grows we will need to run them simultaneously and independently. As each build will be different from each other, there is a need to have a certain amount of resources associated with that build depending on the needs of each one and those resources may be different from the ones that will be assigned to another builds.

At the end there will be approximately 40 to 50 different builds running, in different environments totally independent from each other. This is when one of the main features of cloud computing plays a key role, *“virtualisation”*

By using virtualisation in cloud computing we can set up many different environments that meet the requirements of each different build and associate them with the required resources to those environments to run the configurations. There are some drawbacks that may be found when using cloud computing but probably the most obvious and important one is *“Security and Privacy”*. Since cloud computing is a public service that is hosted somewhere it is possible to find many issues when dealing with customer data protection and privacy of information, such as; personal information of the company, of the products, personal information about clients. However, many of these issues can be fixed by creating a private cloud.

In conclusion, this research will be focused on two main branches, the first one will be on continuous integration about how it is possible to follow good practises to deploy good quality software by automatizing the process itself and the second branch will be about cloud computing and what benefits can be taken from cloud computing to get the integration done and what drawbacks will have to be addressed when working with cloud computing.

The main purpose of this research is to shed light on how cloud computing can be used to improve the quality of the final product in a company trying to automatize some of the steps that are critical for the final release of an application which in this case is the integration step.

2.0 Literature Review

2.1 Fundamental practises on continuous integration.

The founder of www.javaranch.com, Kathy Sierra, said in her blog, “There’s a big difference between saying, ‘Eat an apple a day’ and actually eating the apple. The same thing applies when talking about continuous integration practises and almost all the fundamental practises that are used in projects that have to be used as much as possible to guarantee reliability and stability. To get all the benefits of continuous integration there are a number of fundamental practises that have to be followed to get all the common features available in continuous integration systems.

2.1.1 Introduction

To explain all the benefits that are possible to get from continuous integration we will start with the definition of what it’s a build and provide an example to describe a continuous integration scenario based on a typical implementation.

What is a build?

“A build is much more than a compile (or its dynamic language variations). A build may consist of the compilation, testing, inspection, and deployment—among other things. A build acts as the process for putting source code together and verifying that the software works as a cohesive unit” (Duvall, 2007)

For example, a scenario of continuous integration will describe the process by which a developer commits code to the repository. In a project many people can commit changes to a repository; these changes will start then start the integration process.

The steps in a continuous integration scenario would appear as follows:

1. A developer commits code to the version control repository. Meanwhile, the continuous integration server is looking for changes in the repository.
2. Soon after the commits occurs the server detects those changes and retrieves the latest copy from the repository and executes a build script over them, this script integrates the software.
3. The server generates feedback by emailing build results to specified project members.
4. The server continues looking for new changes in the version control.
5. (Duvall, 2007)

2.1.1.1 Developer Practises

Once the developer has finished adding features or add modification before committing the changes to the version control repository first the developer has to run a private build (with the changes from the rest of the team) to make sure everything is alright and then commit the code to the version control repository.

This step does not affect the continuous integration process because the build doesn't occur until the changes are committed. After the private build is successfully completed you can check your changes to the repository. (Duvall, 2007)

2.1.1.2 Version control Repository

One of the requirements of the continuous integration process is the use of a version control repository. In fact, a version control repository is highly recommended as it does not matter if every project uses continuous integration or not. The purpose of the version control repository is to manage all the changes to the source code using a control access repository. This provides a single source point from where the code is available. This repository allows the user to go back in time and get a different version of the source control. (Duvall, 2007)

The continuous integration process will be executed against the mainline of the repository that is the Head/Trunk. The continuous integration server will look for changes here and if they are founded it will get the latest version.

2.1.1.3 Continuous Integration Server

The continuous integration server is in charge of “firing” the integration builds whenever a change is committed to the version control repository. In this case the integration builds will be fired periodically by time instead of the changes.

The continuous integration server is in charge of getting the source files and run the build script over them. With the use of the server this will be provided with a convenient dashboard where all the results from the build will be displayed allowing continual checks on the status of the entire project. (Duvall, 2007)

2.1.1.4 Build Script

The build script is a single script, or set of scripts, used to compile, test, inspect and deploy software. This can use a build script without implementing a continuous integration system, which is a common practice nowadays but to make the process of continuous integration fully automatic there needs to be a build tool that can automate the software build cycle. (Duvall, 2007)

2.1.1.5 Feedback Mechanism

One of the main purposes of continuous integration is to produce feedback on every integration build to establish if there is a problem with the latest build as soon as possible. By integrating continually you get feedback continually and you can fix the problem quickly, there are many different feedback-mechanisms that can be utilised but one of the simplest is by the email system as a feedback report. (Duvall, 2007)

2.1.1.6 Integration Build Machine

The integration build machine is a separate machine whose sole responsibility is to integrate the software. The integration build machine hosts the continuous integration server and the continuous integration server checks the version control repository.

This part will be different because our “build machine” will be mounted on the cloud so there is a probability the continuous integration server will check for changes and will fire the builds machine on demand. (Duvall, 2007)

2.1.2 Features of continuous integration

Now that we have defined all the practises that there are available we will quote the ones that are **required** to implement continuous integration:

1. A connection to a version control repository
2. A build script
3. Feedback mechanism such an email
4. A process for integrating the source code changes. (Continuous Integration Server)

Once the builds are automated and are triggered by the changes detected in the repository we can start adding new features to our continuous integration system.

Once an automated build is run with every change to the version control system, it is possible to add other features to the continuous integration system. By performing automated and continuous database integration, testing, inspection, deployment, and feedback, the continuous integration system can reduce common risks on the project, thus leading to better confidence and improved communication.

2.1.2.1 Source Code Compilation

One of the most basic and common features of continuous integration is continuous source code compilation. Compilation involves creating executable code from the “human-readable” code.

One point to take into account here is when compiling dynamic programming languages such as Python, PHP or Ruby the compilation environment that needs to be set up might be different, so in this case there will have to be different environments for different programming languages.

2.1.2.2 Database Integration

Some people consider the source code integration and integration of the database as complete separate processes—always performed by different groups. This is unfortunate as the database (if there call to use one on the project) is an integral part of the software application. By using a continuous integrating system, this can ensure the integration of a database through a single source: version control repository.

For instance, when a project member modifies a database script and commits it to the version control system, the same build script that integrates source code will rebuild the database and data as part of the integration build process.

2.1.2.3 Testing

Without automated tests, it is difficult for developers or other project stakeholders to have confidence in software changes. Furthermore, when running different categories of tests from a continuous integration system it will speed up the builds. These categories may include; unit, component, system, load/performance, security, and others.

2.1.2.4 Inspection

Automated code inspections (e.g. static and dynamic analysis) can be used to enhance the quality of the software by enforcing rules. For instance, a project might have a rule that no class may be longer than 300 lines of non-commented code. You can use the continuous integration system to run these rules automatically against a code base.

2.1.2.5 Deployment

With continuous deployment that can deliver working and deployable software at any point in time. This means that we can generate all the software artefacts of the project will have the latest code changes and make it available to a testing environment.

Among other things, the source files from the version control repository must be checked out, a build must be performed, all tests and inspections must successfully executed, the release must be labelled, and the deployment files must be staged.

2.1.2.5 Documentation and Feedback

Another one of the benefits of continuous integration is that it provides all the benefits of documentation without some of the difficulties associated. There are tools that can generate class diagrams based on the committed source code in the version control repository. We will get real-time documentation of source code and project status using continuous integration.

2.2 Introducing continuous integration

“Assumption is the mother of all screw-ups” (Wethern’s Law of Suspended Judgment)

The practise of developing good software is all about following fundamental practises rather than using a particular technology. Experienced programmers highlight that the most important problems to be aware of when developing software is making assumptions. There should be no assumptions that a method will receive the right parameter cannot be made because the method could and most likely will fail. The assumption that developers are following coding and design standards should also be

avoided as the code will become difficult to maintain. There is always a need to test the code and make a review of the code.

When assumptions are made in software development there are added risks. Developing software requires planning for change, continuously observing the results and incrementally course-correcting based on the results. Continuous integration gives the developer the ability of making changes in the code knowing that if the code breaks down there is a possibility to receive immediate feedback. With this feedback developers can course-correct and adjust to change more rapidly.

Continuous integration is a vital task even it is done behind the scenes where the customer cannot see it because it saves time for the developers. The less time developers spend looking for bugs the more time they spend in challenging the task that makes their job interesting. The discipline of keeping the build “in the green” gives developers confidence that the state of the project is positive. Continuous integration is a practise that affects every person on the software development team not only the “build master”. (Duvall, 2007)

2.2.1 Benefits of applying principles

From following the principles and practises that have been pointed out before are the benefits that can be extracted:

2.2.1.1 Reduce Risks

By carrying out continuous integration many times a day can reduce the risk of putting the project in the red by detecting defects while also measuring health of software and reduces the assumptions.

- **Defects are detected and fixed** sooner because the continuous integration integrates and runs tests that inspections several times a day so there is a greater chance that defects are discovered when they are introduced instead of during the late-cycle testing.
- **The health of the software is measurable.** With continuous integration testing and inspection the software product’s health attribute such as complexity can be tracked over time.
- **The reduction of assumption.** By rebuilding and testing software in a clean environment using the same process and scripts on a continual basis to reduce assumptions.

Continuous integration reduces the risk of introducing defects into the code base. Some defects that it helps to mitigate:

- Lack of cohesive, deployable software
- Late defect discovery

- Low-quality software
- Lack of project visibility

(Duvall, 2007)

2.2.1.2 Reduce repetitive processes

The benefit of reducing processes saves time, costs and effort. These processes are all those activities that occur across all projects like code compilation, database integration, testing, inspection, deployment or feedback. With continuous integration ensures that:

- The process runs the same way every time
- The order is followed, i.e. first inspections and then tests.
- The processes will be triggered each time a commit occurs.

2.2.1.3 Generate Deployable Software

With continuous integration to release deployable software at any time is possible. Deployable software is the most tangible asset to the clients and users so it is potentially the most significant benefit that can be obtained from continuous integration. In a project that doesn't follow continuous integration practises, there may be a wait immediately prior to the delivery of integration and testing. This can result in delays due to errors during integration.

2.2.1.4 Improve project visibility

Continuous integration provides the ability to notice trends and make effective decisions that helps to provide the courage to experiment with innovation of new improvements. Projects suffer when there is no real or recent data to support decisions, so at best members of the team can only offer their best guesses. Typically, project members collect this information manually, but this can be a channelling task to collect all the information needed. The result is that not enough sufficient information is gathered. Continuous integration has positive effects on:

- **Effective decision:** A continuous integration system can provide just-in-time information on the recent build status and quality metrics even defect rates and feature completion statuses.
- **Noticing trends:** Since integrations occur frequently the ability to notice trends in build success or failure, overall quality and other project information becomes possible.

2.2.1.5 Greater Product Confidence

With every build, the team knows that tests are run against the software to verify behaviour, that project coding and design standards are met and that the result is a functionally testable product. Continuous integration systems informs when something goes wrong so developers can have more confidence in making changes.



2.2.2 Concerns about Continuous integration

There are a number of reasons or concerns why a development teams to stay away from using continuous integration:

- **Increased overhead in maintaining the continuous integration system.** The need of integrate, test, inspect and deploy exists regardless of whether the system is using continuous integration or not. Managing a robust continuous integration system is better than managing manual processes. Manage the continuous integration system or be controlled by the manual processes. Complicated multiplatform projects are the ones that need continuous integration the most, yet these projects often resist the practice as being “too much extra work”.
- **Too much change.** It appears to have many processes that need to be changed to achieve continuous integration but an incremental approach to continuous integration is more effective. First add builds and tests with lower occurrence, and then increase the frequency as the team members as they become more comfortable with the results.
- **Too many failed builds.** Typically, this occurs when developers are not performing a private build prior to committing their code to the version control repository. A rapid response to failed tests is imperative when using continuous integration because of the frequency of changes.
- **Additional hardware/software costs.** To create an effective continuous integration system separates integration machine when required, which is a nominal expense. It is much more expensive to spend time finding problems later in the development lifecycle.
- **Developers should be performing these activities.** Developers should be performing all these activities but these should be performed more effectively and reliably in a separate environment. Leveraging automated tools can improve the efficiency and frequency of these activities. Additionally, it ensures that these activities are performed in a clean environment, which will reduce assumptions and lead to better decision making.

(Duvall, 2007)

2.2.3 Dealing with continuous integration

Most of the time developers are spending time on automating processes for their users but they forget to automate their own development processes. Sometimes teams believe their automation is enough because they have written some script to automate some steps.

Continuous integration is not just about putting together scripts. To make a process continuous there are some steps that must be followed:

- **Identify.** Identify a process that requires automation. The process may be in the areas of compilation, testing, inspection, deployment and database integration.
- **Build.** Creating a build script makes the automation repeatable and consistent.
- **Share.** By sharing the script through a version control it possible to make it available for other developers to use those scripts. Now the value is being spread consistently across the project.
- **Continuous.** Ensure that the automated process is run with every change applied using a continuous integration server. If the team has the discipline, it can also choose to manually run the build with every change applied to the version control.

A good way to remember this script is to think of the anagram: “**I Build So Consistently**”. Here there is a clear aim for incremental growth in the continuous integration system. With this in mind the team can become more motivated as each new item is added and it can better plan what is needed next based on what’s working so far.

On the other hand, being careful is not a good move to throw every into the continuous integration system at once. Get it to work first, get developers using it and then add other automated processes as needed based on the project risks. (Duvall, 2007)

2.2.4 When to implement Continuous Integration?

The sooner continuous integration is implemented the better. It is difficult to implement continuous integration late in a project, as people will be under pressure and are usually more likely to resist change.

There are different approaches to set up the continuous integration system. Even if the purpose of the continuous integration system is to build on every change it may be better to start running the build on a daily basis to get the practice going in the organisation. People often resist change, and the best approach for an organisation may be to add these automated mechanisms to the process piece by piece. (Duvall, 2007)

2.2.5 Complementing Continuous Integration with other development practices.

Continuous integration is completely compatible with any other development practices such as developer testing, adherence to coding standards, refactoring and small releases. Continuous integration can contribute to the improvement of all these practices:

- **Developer testing.** The tests that developers write to test the code can be automatically executed from the build script. Since the practice of continuous integration is to run all the build every time a change is detected, and the automated tests are part of that build, continuous integration enables automated regression tests to be run on the entire code based on whenever a change is applied to the software.
- **Coding standard adherence.** A coding standard is a set of guidelines that developers must adhere to on a project. On many projects, ensuring adherence is largely a manual process that is performed by a code review. Continuous integration can run a build script to report on adherence to the coding standards by running a suite of automated static analysis tools that inspect the source code against the established standard.
- **Refactoring:** As Fowler states, refactoring is “the process of changing the software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure”. Continuous integration helps with this by running inspection tools that identify potential problem areas at every build.
- **Small releases:** This practise allows testers and users to get working software to use and review as often as required. Continuous integration works very well with this practice because software integration is occurring many times a day and a release is available at any time of the day. Once a continuous integration system is in place a release can be generated with a minimal effort.
- **Collective ownership.** Any developer can work on any part of the software system. This prevents “knowledge silos”, where there is only one person who has knowledge on a particular area of the system. Continuous integration helps by ensuring adherence to coding standards and the running of regression tests on a continual basis. (Duvall, 2007)

2.2.6 How long it takes to set up a continuous integration system?

Implementing a basic continuous integration system for new a project with a few build scripts may take a few hours to set up and configure. As it expands the knowledge of the continuous integration system it will grow with all the tools that are desired to be added. These additional features will need to be added bit by bit.

For a project that is already in progress it can take days, weeks or even months to set it up. It also depends on how many people that are dedicated to the work on the project. Sometimes it is important to move from batch or shell scripts to a build scripting tool such as Ant or managing all of the project’s binary dependencies.

2.2.7 Daily habits

For continuous integration to work effectively on a project, developers must change their typical day-to-day software development habits. The practices that are needed to

integrate continuous integration might need some discipline but they provide all the benefits that have been enumerated at the beginning.

There are seven practices that work well for individuals and teams running continuous integration project:

- Commit code frequently
- Don't commit broken code
- Fix broken builds immediately
- Write automated developer tests
- All tests and inspections must pass
- Run private builds
- Avoid getting broken code

2.2.7.1 Commit Code Frequently

One of the central themes of continuous integration is integrating early and often. Waiting more than one day to commit code to the version control makes integration time-consuming and may prevent developers from being able to use the latest changes. There are two ways to commit more often.

- **Make small changes.** Try not to change many components at once. Choose small tasks, write tests and source code, run the tests and then commit the code to the repository.
- **Commit after each task.** Assuming tasks/work items have been broken up so that they can be finished a few hours, some development shops require developers to commit their code as they complete each task.

It is best to try to avoid everyone on the team committing at the same time every day. Many errors will arise due to the collisions between changes. Very frequent at the end of the day when everybody is ready to leave. The longer the wait, the more difficult the integration will be.

2.2.7.2 Don't commit broken code.

A dangerous assumption on a project is when everybody knows not to commit code that doesn't work to the repository. The solution for that is to have a well-factored build script that compiles and tests the code in a repeatable manner. It's important to also bring to the attention of the team, private builds (which closely resembles the integration build process) before committing code to the version control repository.

2.2.7.3 Fix Broken Build Immediately

The definition of a broken build could consist of anything that prevents the build from reporting success. Anything from; compilation error, failed test, inspection or database. On continuous integration these problems must be fixed immediately but because each error is discovered incrementally it is likely to be very small. The project culture should convey that fixing a broken build is top project priority.

2.2.7.4 Write automated developer tests

A build should be fully automated in order to run the test for the continuous integration system, and for this the tests should be automated. If a framework is used like JUnit the tests will be able to run automated. (Duvall, 2007)

2.2.7.5 All Tests and Inspections must pass

In a continuous integration environment, 100% of project’s automated tests must pass for the build to pass. There are a few statements that have to be agreed upon by the whole team. Automated tests are as important as the compilation. Everyone accepts that code does not compile and will not work. Therefore, the code that has test errors will not work either. Accepting coded that do not pass the test can lead to lower-quality software.

A bad practice is to comment out a failing test, this kind of practise defeats the purpose of continuous integration. Coverage tools assist in outlining source code that does not have a corresponding test and it is possible to run a code coverage tool as part of an integration build. (Duvall, 2007)

2.2.7.6 Run private builds

Before committing the code and possibly broking the build, developers should emulate integration build on their local workstation IDE after completing their unit tests. This build allows the developer to integrate the new working software with the working software from other developers, obtaining the changes from the repository and building locally with the recent changes. Thus, the code each developer commits has contributed to the greater good, with the code that is less likely to fail on the integration build server. (Duvall, 2007)

2.2.7.7. Avoid getting broken code

When the build is broken, there is no point checking the latest code from the repository. Otherwise, there is a need to must spend time developing a workaround to the error known to have failed the build, so a test can be compiled to test the code. The developers responsible for breaking the code should fix it and commit back again. Sometimes a developer may not have seen the e-mail of a broken build when there is a passive mechanism as a “red light” is necessary. (Duvall, 2007)

2.3 Introduction to Cloud Computing.

“Cloud-computing has revolutionized IT in a way never seen before”.
(Gregory, 2012)

Cloud computing was forecasted as the biggest trend for 2012. It is becoming one of the main attractive of the software industry and is changing the way that software development is being done. Nowadays everybody wants to become part of the phenomena of ‘cloud-computing’ and get all the advantages that this provides. Let's begin by a conversation about what the cloud computing to is and how it is defined.

2.3.1 Cloud computing definition

Cloud computing has been a difficult term to define when talking about cloud computing;

“...the market seems to have come to the conclusion that cloud computing has a lot in common with obscenity – you may not be able to define it, but you’ll know it when you see it”

James Urquhart – The wisdom of clouds

Different experts have different ideas about what cloud computing it is, the National Institute of Standards and Technology last year gave a definition of cloud computing from a technical perspective:

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.”
(Technology, 2011)

This definition is too technical and too difficult to understand even for computing people. A much easier definition can be found on the website “cloud computing search” written by the expert Margaret Rouse and states:

“Cloud computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). The name cloud computing was inspired by the cloud symbol that's often used to represent the Internet in flowcharts and diagrams.”
(Rouse, 2010)

From this definition we can see that the concept of cloud computing is just a way of representing the idea of internet but in a different way. From another sources we can find more critical definition about what cloud computing is and what it can be used for. In the

website www.everymanit.com we get a different definition from the one of the authors of the Website. “Eli the computer guy” comments in one of his many videos that cloud computing is just a term used by “marketers” on his “marketing campaigns” to try to sell new products. This may be a way to get the attention of customers that might be interested in new products.

(www.everymanit.com, 2010)

As example the author mentioned above explains the way Microsoft in December 2010 was selling his new product based on the use of “the cloud” to store everything where you could see a fancy cloud flashing on the screen. From the marketing point of view “cloud computing” is just another way to “get your money out of your pocket” because you will pay for it monthly.

Another approach of cloud computing could come from the need of scalable systems in the IT without the need of the new software or new infrastructure. This definition comes from the Analysts of infoworld:

“Cloud computing comes into focus only when you think about what IT always needs: a way to increase capacity or add capabilities on the fly without investing in new infrastructure, training new personnel, or licensing new software. Cloud computing encompasses any subscription-based or pay-per-use service that, in real time over the Internet, extends IT's existing capabilities.”

(Knorr, 2008)

From the guys of the Economist who have a satire view about the three big companies in the world gave the next definition:

“Much of computing will no longer be done on personal computers in homes and offices, but in the “cloud”: huge data centres housing vast storage systems and hundreds of thousands of servers, the powerful machines that dish up data over the internet. Web-based e-mail, social networking and online games are all examples of what are increasingly called cloud services, and are accessible through browsers, smart-phones or other “client” devices.”

Many definitions can be given to define what is cloud computing, but this is because cloud computing can be used for almost everything in computing. However, before anyone can use all the services that the cloud can bring, first the benefits need to be understood and what can be expected from the cloud.

2.3.2 Cloud structure

The structure of the cloud is divided in three layers which defines the three different services that can be used: Software as a service, Platform as a service and Infrastructure as a service. (Dialogic.com, 2010)

2.3.2.1 Software as a Service (SaS)

This is the top level of the hierarchy and corresponds with the applications that are run and interacted via web browser or host desktops that are accessed through remote clients. With the use of these kinds of applications the user does not need to purchase expensive licenses to run the applications. Instead the cost is charged through a subscription fee. A cloud application eliminates the need of installing and running the application on the customer’s computer and all the cost associated with is is: software maintenance and support.

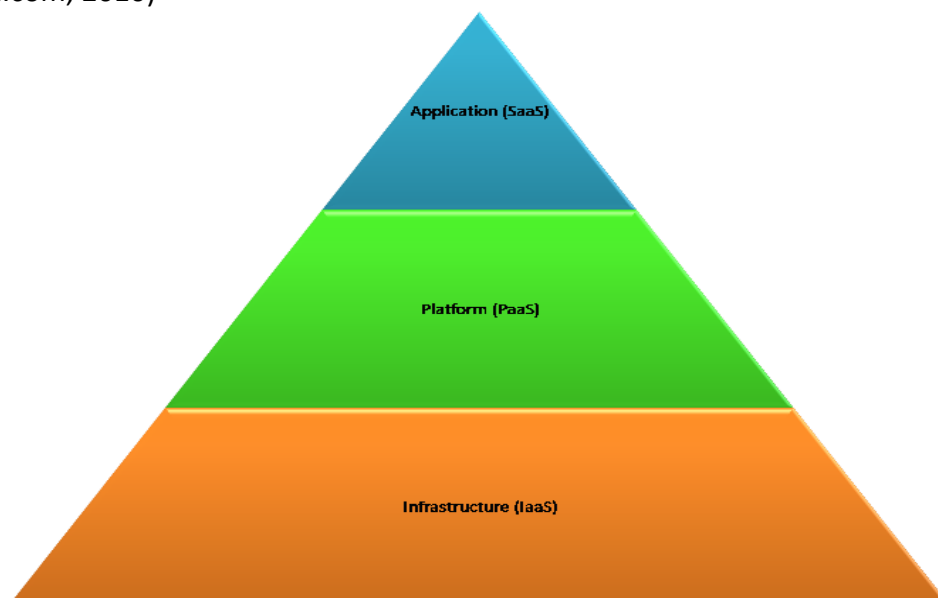
2.3.2.2 Platform as a Service (PaS)

This is the middle level of the hierarchy and provides a computing platform from where to deploy the customer applications. A computing platform dynamically provides provisions and configures to servers as needed to cope with the demand. The operative systems and network access are not managed by the consumer, and there might be constraints as to which applications can be deployed.

2.3.2.3 Infrastructure as a Service (IaS)

This is the base of the hierarchy and corresponds to the delivery of IT infrastructure through virtualisation. Virtualisation allows the split of a single physical piece of hardware into independent, self-governed environments, which can be scaled in terms of CPU, RAM, Disk and others elements. The customer control and manage the systems in terms of the operating systems, applications, storage and network connectivity, but do not control the cloud infrastructure.

(Dialogic.com, 2010)



2.3.3 Deployment Models

The deployment of the cloud infrastructure can differ depending on the requirements of the business. There are four models of deployment that have been identified. Each one of those meets the requirements of the specific services for specific users.

2.3.3.1 Private Cloud

Also referred as ‘corporate’ or ‘internal’ cloud is when all the computing architecture that provides the services is deployed on private networks. Is generally used by large companies and allows administrators to become in-house ‘service providers’ catering the ‘customers’ within the company. However, the maintenance can be done by a third party company on the premises of the company.

2.3.3.2 Public Cloud

Also referred as ‘external’ cloud, this is when the infrastructure of the cloud is available to the public by a service provider on a commercial basis. With this approach the customer can deploy and develop services in the cloud with very little financial outlay compared with the capital that has to be spent to deploy the cloud from scratch.

2.3.3.3 Hybrid Cloud

This is when the infrastructure is a combination of many clouds of any types. The clouds have the ability through their interfaces to allow data and/or applications to be moved from one cloud to another. In this way a company could choose to use a public cloud service for any non-critical service but store its business-critical data within its own data centre.

2.3.3.4 Community Cloud

Community cloud is when the infrastructure is shared among different organisations with similar interests and requirements. In this case the capital expenditure cost is shared among the organisations. The operation of the cloud may be managed by in-house administrators or by a third-party on premises.

2.3.2 Benefits of cloud computing

The number of reasons we can find to move to this kind of model of IT are several so depending on the needs of the companies, it is possible to exploit more the advantages of cloud computing. (Rob Lovell, 2010)

2.3.2.1 Reduction of initial capital expenditure

As a customer of the cloud there is no need for expending money in IT Infrastructure when starting a business. All the machines and applications that a business may need can be provided by moving to the cloud model. With the computing model the expense of the hardware and the software is converted to an operational cost easier to budget month by month.

2.3.2.2 Reduced Administration costs

With the cloud model the IT infrastructure is deployed very quick and managed, maintained, patched and upgraded remotely by the service provider. The technical support is provided round the clock by the provider, avoiding the fact that having IT staff taking care of the IT infrastructure. This means that the business has more time to focus on the tasks that are really important and add business value to the final product avoiding the cost of training and maintaining staff.

2.3.2.3 Improved resource utilization

The fact of combining large amounts of resources into the cloud reduces costs and maximizes the utilisation of the resources by delivering them only when they are needed. With this model a business doesn't have to “over-provision” a service that might not meet their predictions or “under-provision” those which suddenly become popular. Moving all the infrastructure to the cloud can free up time, effort and budget just keeping the business focused on their mission and exploiting the technology for that.

2.3.2.4 Economies of scale

Providers who typically use very large-scale data centres can get much higher efficiency levels and implement multi-tenant architecture that allows them to share resources between many different customers. This model of IT provision allows them to pass on savings to the customers.

2.3.2.5 Scalability on demand.

Scalability allow customers to react quickly to changing IT needs, adding or subtracting capacity and users when required and responding to real requirements rather than projected requirements. Cloud computing follows the utility model, which states that the customer pays for what it consumes so it provides elasticity to the business.

2.3.2.6 Quick and easy implementation

There is no need to purchase hardware, software licenses are the implementation services. The only requirement for the customer is to make an arrangement with the provider, this can be done instantly online.

2.3.2.7 Help Small Businesses to compete

Cloud computing makes it possible for smaller companies to compete with larger ones by 'renting' IT Services. Renting IT services instead of investing in hardware and software makes them much affordable and allows them to invest that money in critical-projects instead.

2.3.2.8 Anywhere access

Cloud based systems allows data an application to be accessed from any place by using internet connection.

2.3.2.9 Disaster Recovery

With virtualisation, the entire server, including the operating system, applications, patches and data is encapsulated into a single software bundle or virtual server. This entire virtual server can be copied or backed up to an offsite data centre and spun up on a virtual host in a matter of minutes. In this way all the infrastructure can be backed up in the cloud with no risk of losing data.

(onlinetech, 2012)

2.3.3 Security in cloud computing

"At the heart of cloud infrastructure is this idea of multi-tenancy and decoupling between specific hardware resources and applications. In the jungle of multi-tenant data, you need to trust the cloud provider that your information will not be exposed."

Datamonitor senior analyst Vuk Trifković.

Good cloud-providers follow strict privacy policies and sophisticated security measures like data encryption. However, not all vendors will offer the same level of security. It is highly recommended for those customers that are concerns with security to research vendor policies before using their services.

Here are some of the main risks identified by the Technology analyst and consulting firm Gartner that the customer should have in mind when hiring services:

2.3.3.1 Privileged user access and Regulatory compliance

Customers of cloud services providers have to be advised that their data can be accessed by third parties working for the provider. That could be an issue when the provider works with many of those third parties because it means that the data of the customers could potentially be accessed by them.

A consideration to take for the customers that are responsible of very high sensitive data is to find out what company is used by the provider and if it is possible to seek an independent audit of their security status. (Binning, 2009)

2.3.3.2 Lack of Standards

Cloud computing is still in its infancy and there are available standards to follow in themes of security. However; IBM, Cisco, SAP, EMC and several other leading technology companies had created an 'Open Cloud Manifesto' calling for more consistent security and monitoring of cloud services. (Software, 2009)

However, these standards can often be restrictive. For this reason many people are still wondering what benefits could be taken from cloud standardisation at this early stage. There are only a few cloud providers who promote the creation of standards before the market is formed. (Trifkovic)

2.3.3.3 Data Location

Possibly the most important issue when talking about security in cloud computing is jurisdiction. Data that might be secure in one country may not be secure in another. One problem that customers might come across is that they don't know where the data is being stored. However, there is a clear intent of harmonise data laws across states but the differences between them are high. For example, in EU the data laws are very strict with privacy but in EU US Patriot Act invest government and other agencies with virtually limitless powers to access information including that belonging to companies.

In fact, European concerns about US privacy laws led to creation of the US Safe Harbor Privacy Principles, which are intended to provide European companies with a degree of insulation from US laws.

2.4 Cloud Architecture

The structure of the cloud is always hidden from ‘normal users’, as everyday users are not concerned about the process of how the programme is really implemented. However, for this research a close approach to the architecture of the cloud is required to know the capabilities of the capacity and power that can be used.

In this chapter we will talk about the specifications of the cloud as: how many computers are made of, what operative systems is install and which platform it will be used to manage the cloud.

2.4.1 Cloud Characterisitcs

The first thing to talk about is, which cloud will be used for the research. Due to the fact that this research is fully entrepraise oriented which means that the target is focused on a concrete company in this case Altobridge Ltd. The cloud that will be used will be the one which altobridge will provide for the experiment and it will be where later on the system will be deployed.

Once this is said, let’s get started talking about the capabilities of the cloud designed for this research. Currently in Altobridge Ltd. there are 3 physical servers, a cloud controller, which also has 'compute' resources and two compute nodes.

- The cloud controller has two quad core CPN (Xeon E5506) for a total of 8 cores and 18G of RAM. The cloud controller also has a 1.2 TB RAID 6.
- The compute nodes both have two quad core CPN (Xeon E5506) for a total of 8 cores and 24G of RAM, these are also equipped with a 300G RAID 1.
- Total power is 24 cores and 66G RAM.

The number of machines that can be spined up is dependant on the resources allocated to each machine and how much "contention" we are willing to live with. i.e. we could exceed the number of physical cores, in allocation but the machines would be services much slower. (Hart, 2012)

This is the physical hardware that the cloud is made of. Now an explanation on how all this software can be glued together and be virtualised to use on demand.

2.4.2 Cloud operative system (KVM)

To cope with all the hardaware resources and operative systems is need to virtualise all the resources. The operative system that will be used is KVM.

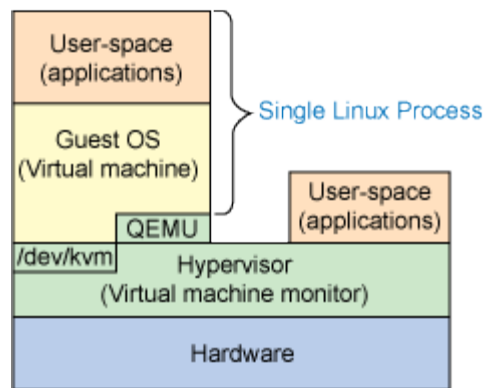
2.4.2.1 KVM

KVM means Kernel-Based Virtual and is becoming one of the most popular open source solution for virtualisation in Linux. The compatible hardware for this is x86 and it contains extensions for Intel (Intel VT) and AMD(AMD-V) processors.

KVM consists of loadable kernel module that provides core virtualization infrastructure and a processor specific module. With this many virtual machines can be fired from unmodified Linux or Windows images. Each virtual machine has private virtualised hardware: network card, disk, graphics adapter, etc.

How KVM takes the Linux kernel and transform it into an hypervisor is simple, it simply loads an additional kernel module for each instance.

The kernel module exports a device called `/dev/kvm` which enables a guest mode of the kernel (in addition to the traditional kernel and user modes). With `/dev/kvm`, a VM has its own address space separate from that of the kernel or any other VM that is running. To summarise, *Devices* in the device tree (`/dev`) are common to all user-space processes but `/dev/kvm` is different in that each process that opens it sees a different map to support isolation of the VMs.



One benefit of turning the Linux kernel itself into a hypervisor is that all the optimisations to the Linux kernel benefit from both sides the hypervisor which is the host operative system and the the guests operative systems. (vsrivercenter, 2012)

In sum the virtual machine is implemented as a regular Linux process, scheduled by the standard Linux scheduler. In fact, each virtual CPU appears as a regular Linux process so the KVM takes all the advantatges of the Linux kernel. The device emulation is handled by a modified version of QEMU which provides an emulated BIOS, PCI Bus, USB Bus and a standard set of devices such as IDE and SCSI disk controllers, network cards, etc.. (Tholeti, 2011)

2.4.2.2 Features of KVM

The next set of features are the most common advantatges that makes KVM one of the best options. The most important of the features are inhereted from the Linux kernel.

Security

KVM comes with the standard Linux security model which provides isolation and resources control. The Linux Kernel uses SELinux (Security-Enhanced Linux) to add mandatory access controls, multi-level and multi-category security and to handle policy enforcement. SELinux provides strict resource isolation and confinement for processes running in the Linux kernel. (Tholeti, 2011)

Memory management

The memory of a virtual machine is stored in the same way that the memory is stored for any other Linux process and can be swapped, backed by large pages for better performance, shared or backed up by a disk file. NUMA (Non-Uniform Memory Access) that is memory design for multiprocessors allows virtual machines to efficiently access large amounts of data. (Tholeti, 2011)

Storage

KVM is able to use any storage supported by Linux to store virtual machines images, that includes: local disks with IDE, SCSI and SATA, Network Attached Storage(NAS) including NFS and SAMBA/CIFS or SAN with support for iCSI and Fibre Channel. As KVM is based on the Linux Kernel it has been proven to be reliable in the infrastructure with support to all leading storage vendors. (Tholeti, 2011)

Live Migration

Provides the ability to move a running virtual machine between physical hosts with no interruption to the service. Live migration is transparent to the user, the virtual machine remains powered on, network connections remain active and user applications continue to run while the virtual machine is relocated to a new physical host. (Tholeti, 2011)

Device Drivers

To deliver high performance I/O for network and block devices KVM supports hybrid virtualization where paravirtualised drivers are installed in the guest operating system to allow virtual machines to use optimized I/O interfaces instead of emulated devices.

The KVM hypervisor uses the VirtIO standard developed by IBM and Red Hat in conjunction with the Linux community for paravirtualized drivers; it is a hypervisor-independent interface for building device drivers allowing the same set of device drivers to be used for multiple hypervisors, allowing for better guest interoperability. (Tholeti, 2011)

Performance and Scalability

KVM supports virtual machines with up to 16 virtual CPU's and 256 GB RAM and host systems with 256 cores and over 1 TB of RAM. It can deliver:

- Up to 95 to 135 percent performance relative to bare metal for real-world enterprise workloads like SAP, Oracle, LAMP, and Microsoft Exchange.
- More than 1 million messages per second and sub-200-microsecond latency in virtual machines running on a standard server.
- The highest consolidation ratios with more than 600 virtual machines running enterprise workloads on a single server.
- That means KVM allows even the most demanding application workloads to be virtualized.

(Tholeti, 2011)

2.4.3 Infrastructure as a service

Once the hardware and the operative system is set up on the cloud the next step is set the bases for the development. Within Altobridge all the infrastructured of the cloud is set up by using OpenStack.

2.4.3.1 Introduction to OpenStack

OpenStack is an Infrastructure as a Service (IaaS) cloud computing project started by Rackspace Cloud and NASA in 2010.

“OpenStack is on a mission: to provide scalable, elastic cloud computing for both public and private clouds, large and small. At the heart of our mission is a pair of basic requirements: clouds must be simple to implement and massively scalable”
(openstack, 2010)

OpenStack defines the layer where all the research will be done and where all our software will be deployed. OpenStack allows the user to manage the cloud by using OpenStack Compute.

“OpenStack Compute gives you a tool to orchestrate a cloud, including running instances, managing networks, and controlling access to the cloud through users and projects. The underlying open source project's name is Nova, and it provides the software that can control an Infrastructure as a Service (IaaS) cloud computing platform. It is similar in scope to Amazon EC2 and Rackspace Cloud Servers. OpenStack Compute does not include any virtualization software; rather it defines drivers that interact with underlying virtualization mechanisms that run on your host operating system, and exposes functionality over a web-based API”
(openstack, 2010)

To better understand the structure of open stack an explanation of the cloud components has to be explained.

2.4.3.2 OpenStack Components

OpenStack divides the cloud in seven core sectors to manage each of the different features:

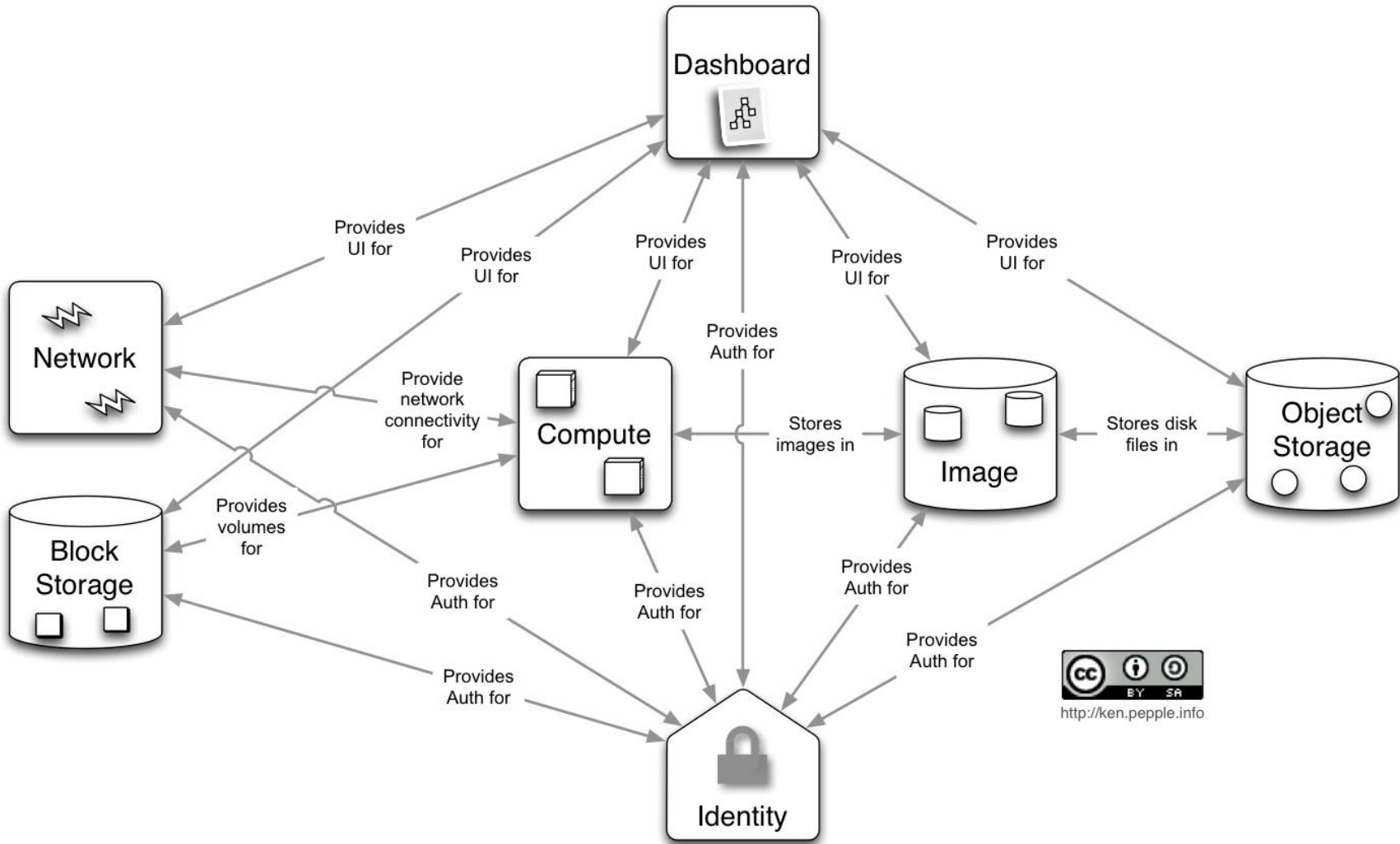
- Object Store (codenamed "Swift") provides object storage. It allows the user to store or retrieve files (but not mount directories like a fileserver).
- Image (codenamed "Glance") provides a catalog and repository for virtual disk images. These disk images are mostly commonly used in OpenStack Compute. While this service is technically optional, any cloud of size will require it.
- Compute (codenamed "Nova") provides virtual servers upon demand
- Dashboard (codenamed "Horizon") provides a modular web-based user interface for all the OpenStack services. With this web GUI, you the user perform most operations on the cloud like launching an instance, assigning IP addresses and setting access controls.
- Identity (codenamed "Keystone") provides authentication and authorization for all the OpenStack services. It also provides a service catalog of services within a particular OpenStack cloud.
- Network (codenamed "Quantum") provides "network connectivity as a service" between interface devices managed by other OpenStack services (most likely Nova). The service works by allowing users to create their own networks and then attach interfaces to them.
- Block Storage (codenamed "Cinder") provides persistent block storage to guest VMs. In the Folsom release, both the nova-volume service and the separate volume service are available.

(openstack, 2010)

These are the main controllers that we will fine in our design to manage the cloud. We will have to deal with them along the integration.

2.4.3.3 Architecture

The OpenStack project as a whole is designed to "deliver(ing) a massively scalable cloud operating system." To achieve this, each of the constituent services are designed to work together to provide a complete Infrastructure as a Service (IaaS). This integration is facilitated through public application programming interfaces (APIs) that each service offers (and in turn can consume). While these APIs allow each of the services to use another service, it also allows an implementer to switch out any service as long as they maintain the API. These are (mostly) the same APIs that are available to end users of the cloud.



(openstack, 2010)

3.0 Methodology

3.1 Introduction

For the past few years the number of projects that are being undertaken by the Altobridge has increased by a larger proportion. The number of different branches for each project and all the different features are being developed continually. This makes the process of deploying the final application difficult and unpredictable.

The question that was brought to the table: *what would be necessary to create a “system” that automatically “builds” (compile, test, reviews and deploys) our software and give us a short-term “feedback” about the state of the project?*

With feedback the process of gathering information about the status of the final application to know if the quality of the application is enough, if it satisfies the customer needs and at the same time the tester can start the process of “finding bugs” and behavioural tests to make sure that the application is doing what it was created to achieve.

The initial response was optimistic “yes, that would be possible” but to support this process there may be a need for extra resources as there are so many elements to work on. The first thing that came up was to hire the services of a third party cloud provider which would provide all resources that were needed. Although this idea was good and would allow Altobridge to focus on daily tasks and not have to deal with all the drawbacks under the implementation, one of the policies of the company is ensure that there was risk of personal information of the customer being exploited. A solution needs to be found while also ensuring that personal information is kept internally protected.

Another policy that it has to be adhered to is the use of “Free Software” based solutions. As Altobridge support small communities and avail of free software to reduce costs and bring the value back to the custom, this research needed to follow this trend. The main goal for the research was to find a way to automatize the section of the software development that integrates all the application together. While ensuring to utilise “free software” solutions that benefit “cloud computing resources” and carry this out by keeping all sensitive information under private security policies and exploring all the benefits of cloud computing.

3.2 Can the cloud improve the continuous integration system?

The cloud as a trend is something designed to improve how resources are used in systems in the modern world and by improving the resources of a system we can improve the performance of the whole system.

The aim of this research was to demonstrate how powerful resource management of the cloud can be and to demonstrate the advantages of utilising cloud technology.

The research will investigate a build with a continuous integration system and it will be literally “plugged” to the cloud. The cloud will manage all the resources needed to do the work and it will come back with the results. The data that will be used to measure the effectiveness in terms of the amount time that the cloud takes to accomplish the task and the resources involved in the cloud to accomplish.

3.3 Approach

The approach will question whether the cloud solution for the integration system is better than the current solution that is currently being run by measuring the times that it takes to run the builds and comparing the times when using the cloud-based model. Once the cloud model is finished it will be able to take more measurements such as; resources used or time consumed.

By comparing the time consumed between the builds done with the old system and the builds done with the new system which will be measured by using log files. In Altobridge there are log files of all the builds that have been done in the past. It is possible to extract from this data the times of the builds and compare them with the logs that are used. The tools used for this project will be all open source tools.

The tools planned to use for this project are:

1. Jenkins – The main one
2. JClouds – A Java library to interact with the cloud
3. Groovy – Scripting language used to interact with the server
4. Maven – To build plugins in case they are need
5. Ant – To build the projects itself
6. Shell Scripting – For possible instances.

The main tool for the configuration of the server will be Jenkins as is open-source and very reliable. The original idea was to work with TeamCity as there is a free-version that can be used but the problem with that it was that is not supported by a number of agents and is narrowed to three. So TeamCity was delimited as it didn’t meet the demands of this project.

3.4 Design

The following includes some design documents for the upcoming implementation section of the project.

3.4.1 Risk Analysis

The practice chosen to evaluate the possible risks in this research has been Qualitative risk analysis. With this technology the risks are being prioritised based on their probability and impact of occurrence.

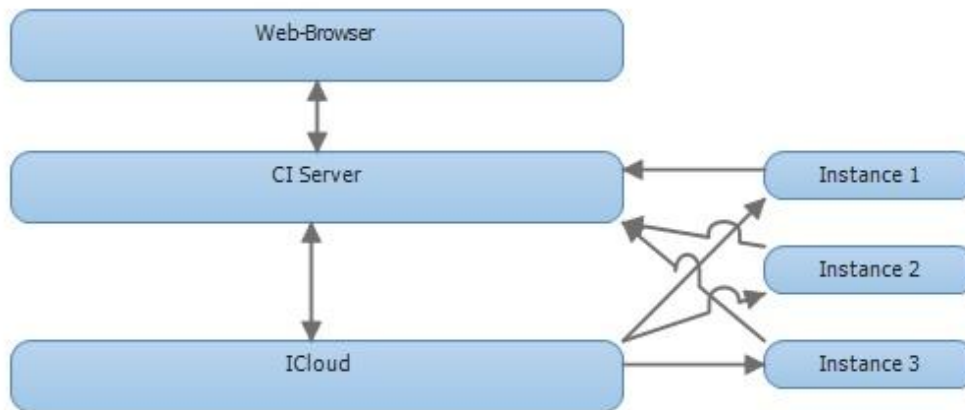
By following the next table we can identify quickly the risks that have the highest probability of impact to the project so alternative solutions can be found and estimated dates for delivering of the project.

Risk Source	Probability			Impact			Result	Impact areas			Risk Response
	Low	Med	High	Low	Med	High	Prob x Impact	Cost	Schedule	Perform	
Lack of previous experience		4			6		24		X	X	Increase research and formation
Server incompatibilities	1					8	8	X	X	X	Change any of the application
Breadown in Communication between designer and developer		6			6		36	X	X	X	Communication Plan
Lack of Cloud Resources	3				7		21	X	X	X	Optimize cloud management
Language platform incompatibilities	2				5		10	X	X	X	Change platform language

3.4.2 UML Diagrams

This is a rough diagram about what entities will be running and how they will interact with each other. The continuous integration server will do the work of running the builds that will gather information but is not able to manage the cloud. The problem is the start, instances in the cloud and most like to start the right instance for the right job. The most challenging part of the implementation is matching the jobs and build machines by using the cloud because the server has no logic so the need is for a third application involved to connect the gap between them.

From the web-browser it is possible to communication with the continuous integration server. The continuous integration server will communicate with the ICloud which machines to spin up. The ICloud will start the machines required for the job that has to be done and will return the control to the continuous integration Server.



3.4.3 Requirements

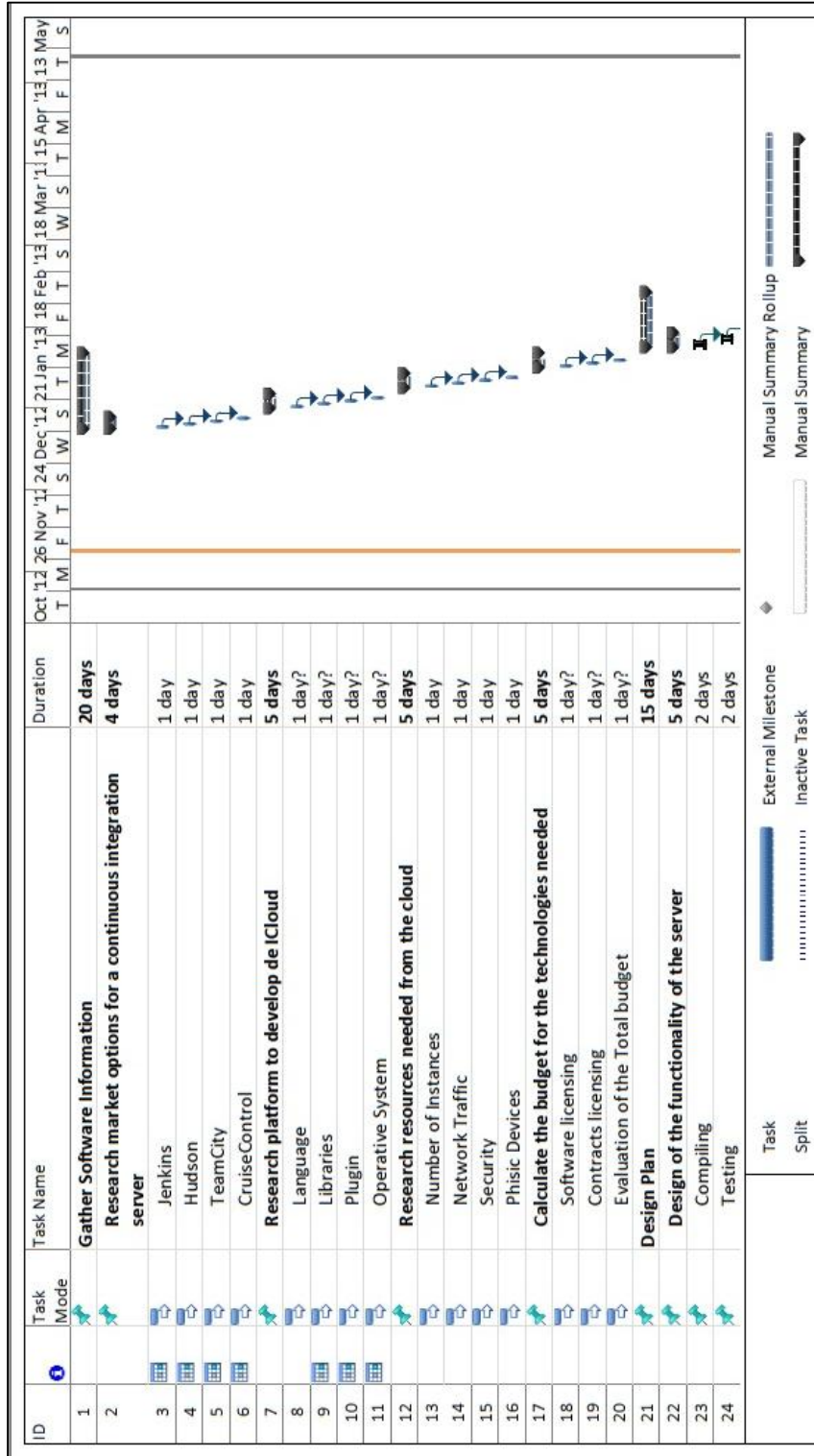
These are some of the requirements for the final system:

1. To improve the quality of the final product
2. To provide a central system to manage all the different build
3. To use the resources of a company in an efficient way by cloud management
4. To make the detection of errors faster
5. To set a number of practises that helps when developing in software
6. To reduce the risk of the overall project
7. To provide frequent feedbacks with the state of the product

3.4.4 Project Management Plans

The tool chosen to schedule the project has been the Gant Chart. Gantt charts provide a standard format for displaying project schedule information by listing project activities and their corresponding start and finish dates in a calendar format. Use a Gantt chart for planning and tracking schedule information.

“Moving to continuous integration on the cloud”



3.5.1 Work Break Down

Here we have the list of tasks that we've plan for the project:

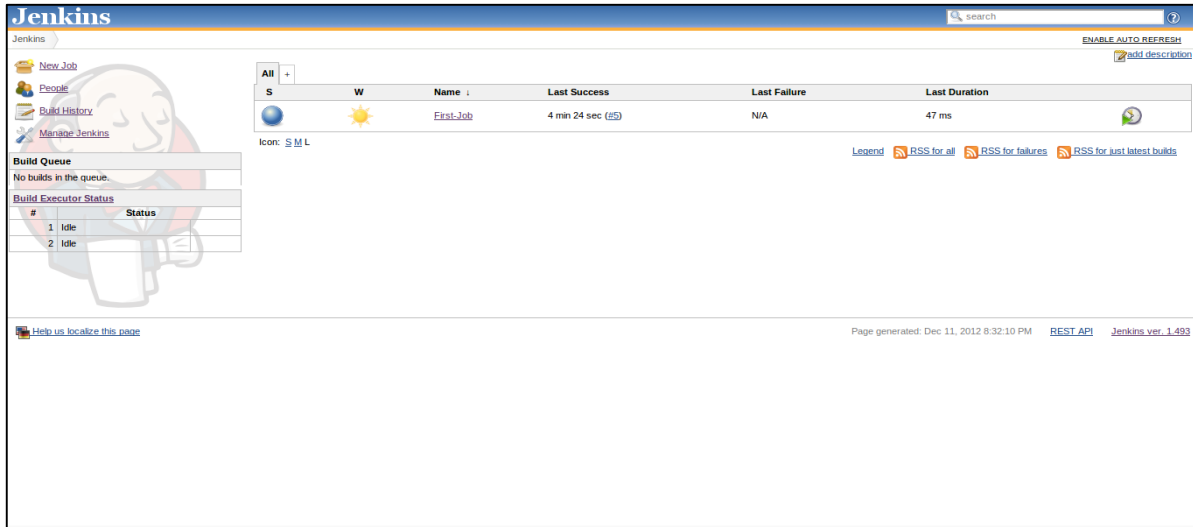
1. Gather Software Information:
 - a. Research market options for a continuous integration server
 - i. Jenkins, Hudson, TeamCity, Cruise Control
 - b. Research platform to develop de ICloud.
 - i. Language, Plugin, Libraries,
 - c. Research resources needed from the cloud.
 - i. Number of instances, access, processors, memory available.
 - d. Calculate the budget for the technologies needed.
 - i. Putting everything together and see how much it costs.
2. Plan design:
 - a. Design of the implementation of the server
 - i. Define functions of the server
 - b. Design of the ICloud
 - i. Classes, interfaces, communication with the server.
 - c. Design of the organization in the cloud
 - i. How will instances be managed, how much will be needed and how many can be run at a time.
3. Plan Implementation:
 - a. Server Configuration
 - i. Server Configuration, build configuration, mail configuration
 - b. Implementation of the ICloud.
 - i. Hard-code the classes require for the manger
 - ii. Unit Tests
 - c. Create test-cases
 - i. Functional tests
 - ii. Integration Test
4. Integration and Deployment
 - a. Run the unit-test and behavioural test to make sure everything is working.
 - b. Create the executable.
 - c. Support with any issue that can come up.

3.5.2 Prototype

The prototype is based on Jenkins Server so for the first approach a build has been done in a local machine to get in contact with the server. Here is how it works:

“Moving to continuous integration on the cloud”

In the middle of the screen the state of the build can be seen.



The screenshot shows the Jenkins web interface. At the top, there is a search bar and a 'Jenkins' logo. Below the search bar, there are navigation links: 'New Job', 'People', 'Build History', and 'Manage Jenkins'. The main content area features a table with the following columns: 'All +', 'S', 'W', 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The table contains one row for a build named 'First-Job' with a status of 'S' (Success), a duration of '4 min 24 sec (#5)', and 'N/A' for the last failure. Below the table, there are links for 'Icon: S M L', 'Legend', 'RSS for all', 'RSS for failures', and 'RSS for just latest builds'. On the left side, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (a table with 2 idle executors). At the bottom, there is a footer with 'Page generated: Dec 11, 2012 8:32:10 PM', 'BEST API', and 'Jenkins ver. 1.493'.

The build is just a build done in the experimental local machine.



This is a close-up screenshot of the Jenkins build table. It shows the following data:

All +	S	W	Name	Last Success	Last Failure	Last Duration
			First-Job	4 min 24 sec (#5)	N/A	47 ms

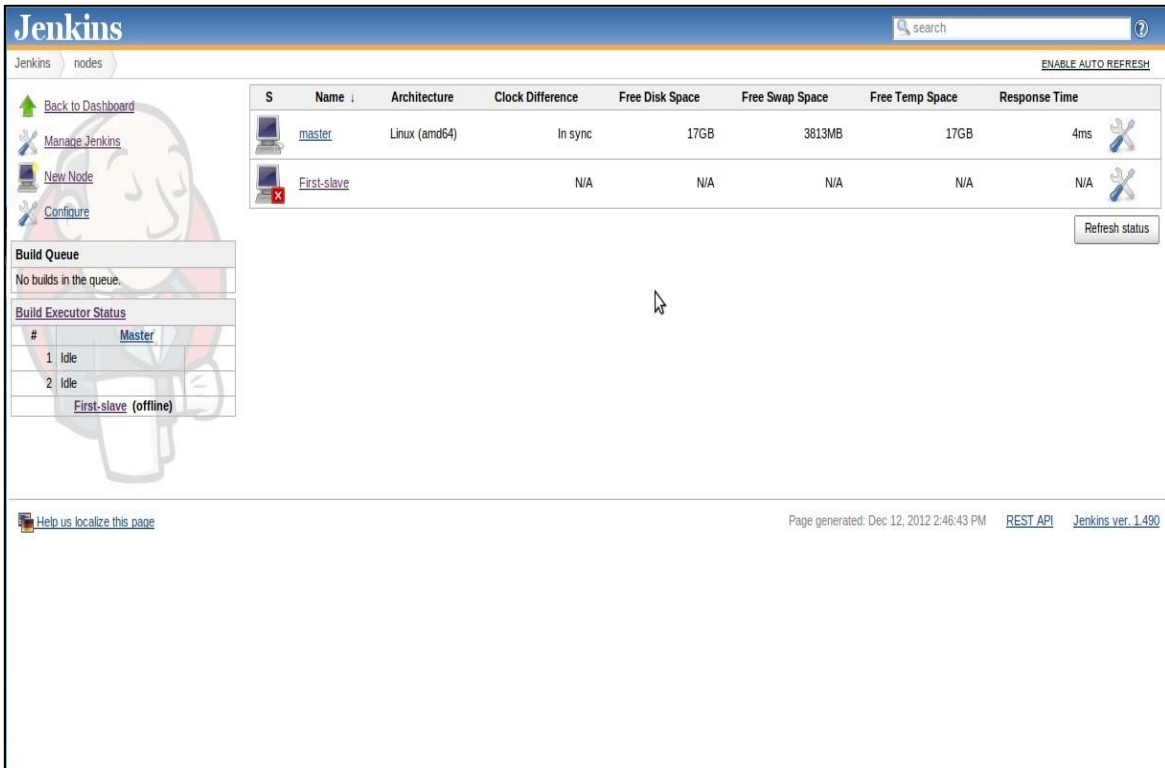
Below the table, there are links for 'Icon: S M L', 'Legend', 'RSS for all', and 'RSS for failures'.

This screen reflect the status of the last build if it was successful or not and the duration of the build. The duration of the build is the simplest data that is going to probe the system implemented in this project which is better than the old system.

With the measurements of the new system and the logs from the old system can make comparisons that can be made of times to see which is faster and at the same time take advantage of having all the builds centralised on a single server.

The local machine exists already and is a physical one. The next step is to create virtual machines and deploy a slave. A slave is an application of Jenkins that runs in the build machine and it has no logic it just follows the instruction of the master.

“Moving to continuous integration on the cloud”



The screenshot shows the Jenkins web interface for managing nodes. The main content area displays a table of nodes with the following data:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Linux (amd64)	In sync	17GB	3813MB	17GB	4ms
	First-slave		N/A	N/A	N/A	N/A	N/A

Below the table, there is a "Refresh status" button. To the left, there are navigation links: "Back to Dashboard", "Manage Jenkins", "New Node", and "Configure". Below these, there are sections for "Build Queue" (showing "No builds in the queue.") and "Build Executor Status" (showing two executors: "1 idle" and "2 idle", with "First-slave (offline)" listed below).

At the bottom of the page, there is a footer with the text "Page generated: Dec 12, 2012 2:46:43 PM" and links for "REST API" and "Jenkins ver. 1.490".

In this first example both of the local machines are represented but the next step is to upload them to the cloud.

“Moving to continuous integration on the cloud”

Altobridge Ltd has its own cloud and is managed by OpenStack. It looks like this:

The screenshot displays the OpenStack 'Images & Snapshots' dashboard. The top right corner shows the user is logged in as 'builder' with links for 'Settings' and 'Sign Out'. The main content area is titled 'Images' and features a table of available images. Each row in the table includes a checkbox, the image name, type, status, public status, format, and a 'Launch' button. The sidebar on the left contains navigation links for 'Project', 'CURRENT PROJECT builders', 'Manage Compute', 'Overview', 'Instances', 'Volumes', 'Images & Snapshots', and 'Access & Security'. At the bottom of the table, it indicates 'Displaying 14 items'.

<input type="checkbox"/>	Image Name	Type	Status	Public	Format	Actions
<input type="checkbox"/>	Ubuntu 12.04 Server	Image	Active	Yes	QCOW2	Launch
<input type="checkbox"/>	CentOS 5.2 (MSS)	Image	Active	Yes	QCOW2	Launch
<input type="checkbox"/>	CentOS 5.2 (EMS)	Image	Active	Yes	QCOW2	Launch
<input type="checkbox"/>	sBSC-Conf	Image	Active	Yes	RAW	Launch
<input type="checkbox"/>	omcR-Conf	Image	Active	Yes	RAW	Launch
<input type="checkbox"/>	RHEL4.8	Image	Active	Yes	RAW	Launch
<input type="checkbox"/>	IPA-withSSHD	Image	Active	Yes	RAW	Launch
<input type="checkbox"/>	Windows 2012 Server	Image	Active	Yes	RAW	Launch
<input type="checkbox"/>	sBSC-noConf	Image	Active	Yes	RAW	Launch
<input type="checkbox"/>	omcR-noConf	Image	Active	Yes	RAW	Launch
<input type="checkbox"/>	ubuntu-10.04-server-uec-amd64	Image	Active	Yes	AMI	Launch
<input type="checkbox"/>	ubuntu-12.04-server-cloudimg-amd64	Image	Active	Yes	AMI	Launch
<input type="checkbox"/>	tylinux	Image	Active	Yes	AMI	Launch
<input type="checkbox"/>	cirros-0.3.0-x86_64-uec	Image	Active	Yes	AMI	Launch

This tab contains all the possible images to load into the instances.

“Moving to continuous integration on the cloud”

The following image below demonstrates the two instances that have been utilised since the start by using the prototype and testing.

One is the master and the other is the slave, it doesn't matter which is which as both can play either role.

Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
<input type="checkbox"/> cbl	[REDACTED]	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/> cbl	[REDACTED]	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot

Displaying 2 items

The initial idea was to have a number of instances, roughly 40 or 50, in the cloud for the builds. The reason why there was a need for so many instances is because Altobridge has a number of products and they all have different requirements depending on the customer that they are targeting. Due to the nature of the product of the software that is developed it was done in C and a very low level.

The use of common flags in GCC is for improving the performance of the compilation and avoiding errors with make file. In general each operative system has its own set of tags or speaking properly for each compiler in each operating system the flags are different. To build a number of project in a number of operative systems is necessary a number of NxN different combinations to build. This makes the process of building very specific for each build and this is the reason for having so many different instances in the cloud.
(Staplin, 2005) & (Fundation, 2004)

3.6 Conclusion

To summarise, the number of specific “build machines” that are needed grows exponentially, this is where the cloud plays a role. It was discovered that the cloud is not un-limited and has its own limits so the implementation of the server in the cloud and should be even more specific.

The way the instances will be used is as follows:

1. An instance will be created.
2. The instance will be configured.
3. A “SNAPSHOT” will be taken.
 - a. Snapshot means that we will store all the info of the instance that was going one in that moment in a database as a file.
4. The snapshot will be stored for future builds.
5. Once a job comes along, an instance is generated from the Snapshot.
6. The instance does the job.
7. Once the instance is finished another Snapshot is taken.
8. Then the instance is terminated.
 - a. Terminated means that all the resources are free to be used by another instance or process and disappears.
9. The snapshot remains stored for next uses.

All the logic of the creation of an instance has to be created somewhere. There are two possibilities:

1. To create a plugin for Jenkins that does all that process each time.
2. To create an external application that will listen for Jenkins requests.

The research of both approaches should be done to see how complicate and which advantages can we get.

4.0 Implementation

The main goal of the research is to compare how much profit can be made for using a Continuous Integration System in combination with a Private Cloud to build the software that is going to be released to the customers. To show the implementation approach taken in the research an explanation of the background needs to be done.

4.1 Background

The target of the research is to deliver to Altobridge a system that allows the company to release more efficient software continuously. The foundation of the company has been focused on the development of hardware packages to deploy mobile network architectures. The projects on Altobridge were mostly based on C programming language. The reason of using C is the software that runs on top of that hardware needs to be as efficient as possible, since that software is not going to be running in any other hardware a language that allows exposure to most of the hardware is required. Some aspects of the C programming language allow more control over Memory Usage or Input/Output in the device than other language so when dealing with very specific hardware those characteristics become extremely necessary.

Recently, Altobridge is trying to move away from the hardware development and getting deep into software development. In fact, what Altobridge is trying to do is to focus the income of the company in Software Licensing instead of Hardware Selling. To accomplish the new objectives of moving to software licensing a system that allows to build all the projects of the company in a single product is required. The business model in Altobridge is about splitting different parts of the product in isolated projects. When a new version of the product has to be released to the customer many projects have to be combined (built) into a single product. When combining these projects a set of integration tests to make sure everything works fine are necessary.

The objective of the continuous integration system is to make all the process of releasing the software to the customer independent from a single type of machine, so the software can run in every machine of the same type, and make it more continuous so there is no need to wait until the last stage of the release to come back and fix possible errors that might arise. In order to keep a track of the steps of building a continuous integration system the process of setting up the continuous integration system will be divided in stages.

4.2 The Build Server

The first thing we required is a Server or a machine from where to deploy the tools needed to set up the system. This server will be the central point of all the builds and the whole configuration. Anything that has to be changed or improved will be done from this server.

Since Altobridge has recently installed a private cloud it is possible to create a virtual server¹ in the cloud and manage it via ssh. The main reason of the virtual server to be in the Cloud is due to the entire network configuration it is associated with hosts a new server. By creating a virtual server in the cloud we save a lot work dealing with network configuration and it is easier to communicate with the rest of the instances that will make the work easier. Another advantage is that the instances are all behind the firewall of the company so all the information is secured under the security policies of the company. In other words no connection can pass through the firewall without the knowledge of the network administrator.

The private cloud in Altobridge is configured by using OpenStack. OpenStack is an open software tool that allows the company to manage the cloud. There are two approaches to cloud management; either from the command line by using a client application called nova or by using the dashboard. The management that is necessary to use the instances from the scripts will be the nova client. The control from the dashboard is very useful when testing because the connections to the instances can be checked on real time but to manage the connections from Jenkins we need to use the nova client.

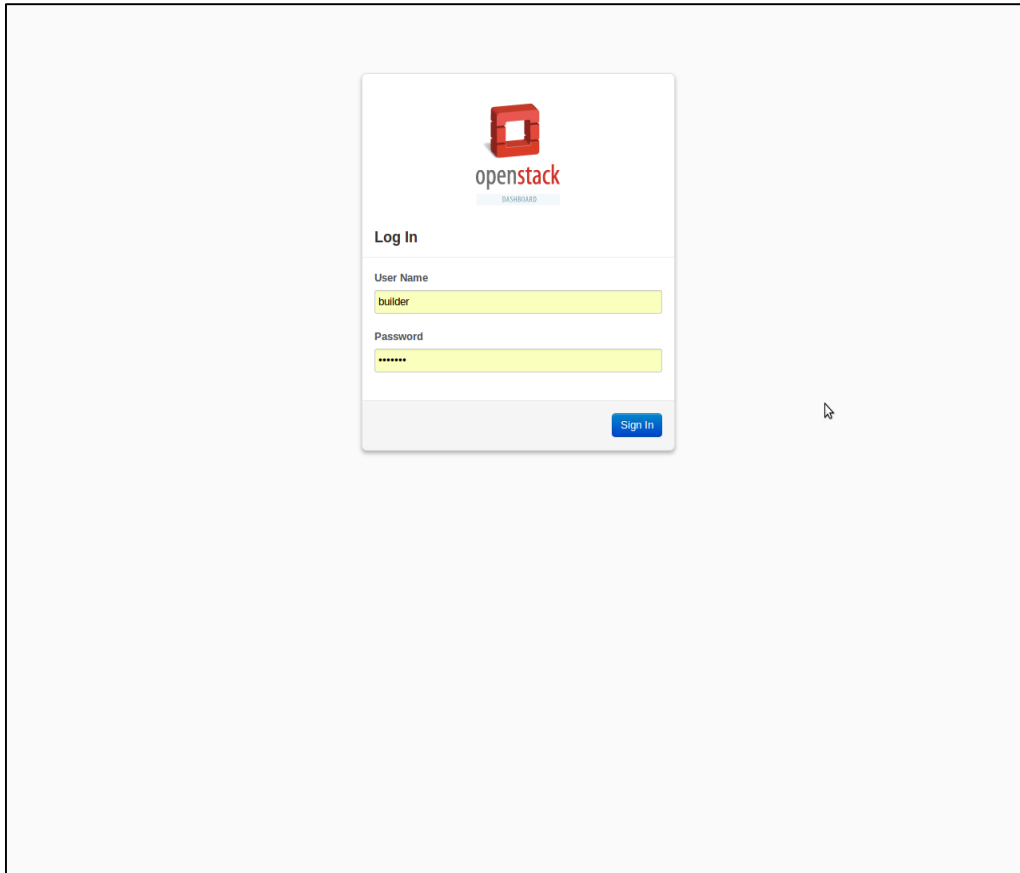
Jenkins is the application that will be used to manage all the builds. Jenkins contains a GUI made in Java from where the builds management is done. From Jenkin's GUI is possible for the builders to order builds and stack them in queue.

The first step will then be created to instance by putting it running. To do this we will do the following step:

¹ The word 'virtual' means that the server is not physically in a box. The server is a virtual machine hosted in the cloud as any other virtual machine.

“Moving to continuous integration on the cloud”

1. A log is required to use the cloud so the administrator of the cloud has to provide an account to be able of creating instances. For the purpose of this research the user ‘Builder’ has been created and it will be used for that purpose.



The image shows a screenshot of the OpenStack Dashboard login interface. At the top center, there is the OpenStack logo (a red square with a white 'O') and the text 'openstack' in a sans-serif font, with 'DASHBOARD' in smaller letters below it. Below the logo, the heading 'Log In' is displayed. Underneath, there are two input fields: 'User Name' with the value 'builder' and 'Password' with a masked password of seven asterisks. A blue 'Sign In' button is located at the bottom right of the form area. The entire form is set against a light gray background.

“Moving to continuous integration on the cloud”

2. Access to the tab called ‘Instances’ to get an overview of all the instances that are running. Click on launch instance in the top right corner of the dashboard.

Instances

Logged in as: builder [Settings](#) [Sign Out](#)

[Launch Instance](#) [Terminate Instances](#)

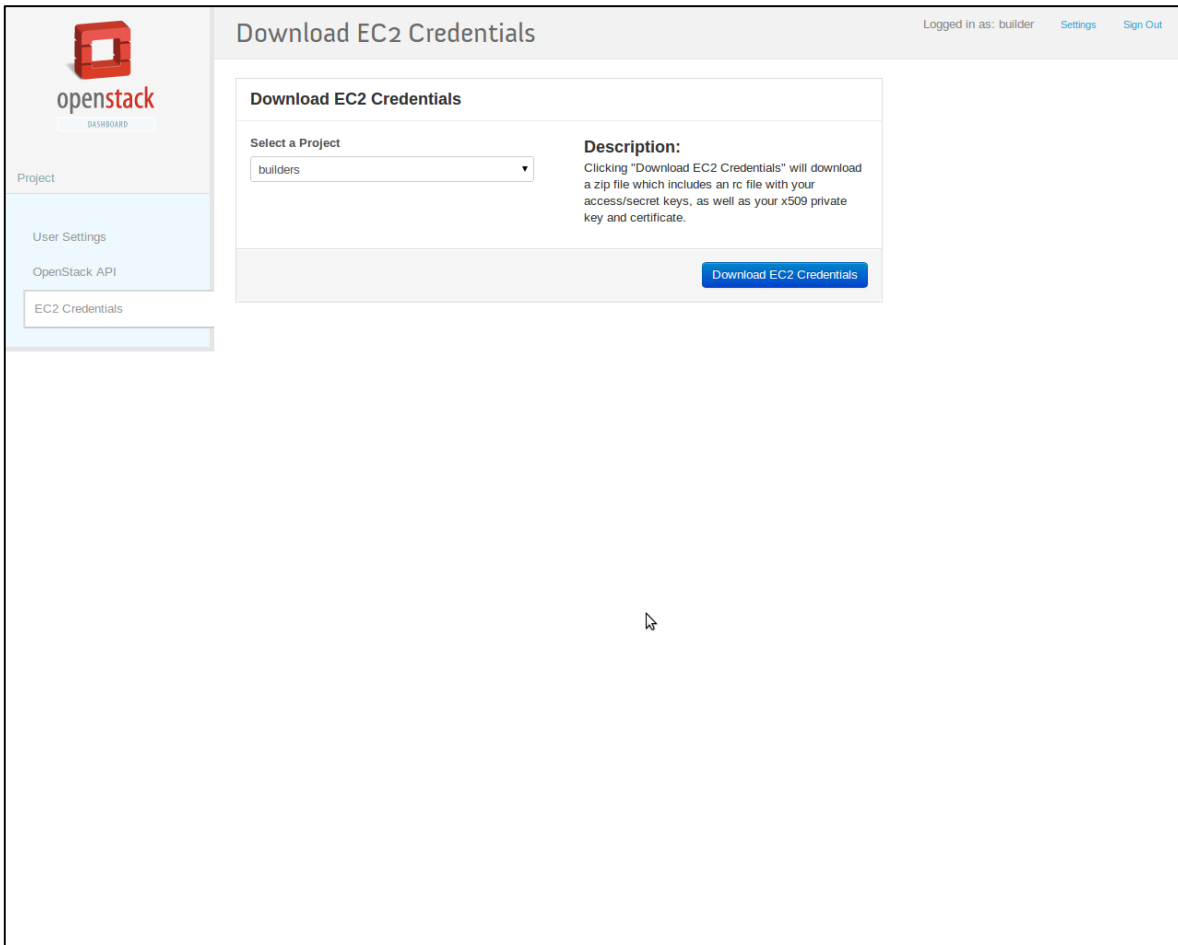
<input type="checkbox"/>	Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
<input type="checkbox"/>	ubuntu10.04-date2.0-altoPod-ABC-build	172.16.5.36	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/>	ubuntu10.04-date2.0-altoPod-RDM-build	172.16.5.35	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/>	ubuntu10.04-date2.0-altoPod-HNBA-build	172.16.5.34	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/>	ubuntu10.04_template	172.16.5.27	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/>	Artifactory	172.16.5.22	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/>	Jenkins	172.16.5.18	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/>	abrogan-run	172.16.5.26	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot
<input type="checkbox"/>	abrogan-dev	172.16.5.24	m1.small 2GB RAM 1 VCPU 20GB Disk	builder	Active	None	Running	Create Snapshot

Displaying 8 items

cc.altobridge.com/nova/instances/

“Moving to continuous integration on the cloud”

3. Specify the characteristics of the instance to start. For this research, the following steps were followed;
 - 3.1. Name: Jenkins (name of the application server)
 - 3.2. OS: Ubuntu 12.04 cloudimg (The OS that contains all the network information to run within the intranet of the company)
 - 3.3. Flavour: The amount of RAM, processor and hard disk that will contain.
 - 3.4. Security: Builder Keypair



4. Wait until the instance gets ready for the job.

The next step is to install Jenkins on the instance we have already created.

1. To manage the instance it's necessary to use ssh to log and manage it from the command line. A keypair called 'Builder' has been created for this purpose to be able to ssh the instance as 'Builder'
2. With the keypair we will log in the install by using ssh.
ssh -i builder_rsa Ubuntu@172.16.5.21
3. Then it is necessary to install Jenkins as an ubuntu package
<https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Ubuntu>
It is pretty straight forward.
4. Note: For this experiment Jenkins will be installed in a different instance from where the code is so it won't be downloaded anywhere else here. This is the instance that will hold Jenkins and will have the ip = 172.16.5.18

From there we will ssh another instances to run the builds.

1. The second step is to create a second instance which will hold the controller of the cloud and the builds. Since the build system is place is quite unique (because it has been made specially for this company) we will have to put all the classes and packages that we will need
2. Then the new instance was created on Ubuntu with Keypairs and everything (see the step before)
3. Once this is done it will set the controller to be able to build one product: An Altopod
 - 4.1. In the home folder /home/ubuntu we check out the trackers code
svn co https://repository.altobridge.com/svn/amg/trunk/trackers
 - 4.1.1. OS: Ubuntu 12.04 clouding (The OS that contains all the network information to run within the intranet of the company)
 - 4.1.2. Flavour: The amount of RAM, processor and hard disk that will contain.
 - 4.1.3. Security: Builder Keypair

4.3 Pre-required cloud instances

The Altobridge build system is an unique build system and in the sense that there has been created by the company. The system has been created, set up and developed over the past two years. It is not possible to try to change it because it would take too much time. Instead of changing the system the procedure will be to run the system in every instance so when the controller jumps to one of the instances to run a job it will think that is still the same machine and not in any other one. The only difference will be to execute the commands with an ssh in front.

Then to make sure that the build controller thinks that is all the time in the same machine it is necessary to create one instance per project and set it up in the way that the build

controller is expecting to find it. The way that the instance is set depends on the project that is going to be built, some project might require Java but some other might not.

Steps that have been followed to put all the instances on place:

1. Install svn
`sudo apt-get install subversion`
2. checkout the folder with the code
`svn co https://repository.altobridge.com/svn/amg/trunk/trackers`
3. Agree to accept permanently you credentials
p
4. Give manually the builder credentials
`--username=builder --password=bP1R4FaP`
5. Accept store unencrypted
6. Accept store unencrypted.
7. Then Jump into the builder folder
`cd trackers/ie-build-1/guest/builder/`
8. The we make the current folder the home folder
`export HOME=$PWD`
9. Then we set up the environment to make sure everything is in place and svn doesn't ask for credentials
`source bashrc`
10. If it still ask for permission then we have to move the folder '.subversion' to the current home
`cp -R /home/ubuntu/.subversion/ .`
11. Then It will say that the packages are not installed, so first we go to
`cd trackers/ie-build-3/root`
12. Then modify the script adding this two lines at the beginning of the main method
`apt-get update`
`apt-get upgrade`
And then comment out the line of mount_disks
`# mount_disks`
13. Change permissions to make it executable
`chmod +x install.sh`
14. then Execute the script with sudo
`sudo ./install.sh`

15. Wait until everything is fetched and downloaded (Around 20 min or 60 min)
A weekend after...
16. Autoconf package is missing so install it
`sudo apt-get install autoconf`
17. altoPod 2.0/RC 18 doesnt exis yet so the one to build will be RC 17.
`cd /home/ubuntu/trackers/ie-build-1/guest/builder/;export
HOME=$PWD;bin/release altoPod 2.0/RC17 ;`
18. Solve all the issue that will arise
19. Install this
20. `wget http://get.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.7.3.tar.gz`
21. `tar xzf qt-everywhere-opensource-src-4.7.3.tar.gz`
22. `cd qt-everywhere-opensource-src-4.7.3`
23. `sed -i -e 73s/6/30/ src/network/access/qhttpnetworkconnection.cpp`
24. `apt-get install libxfixes-dev`
25. `apt-get install xvfb`
26. `apt-get install xfonts-100dpi xfonts-75dpi xfonts-scalable xfonts-cyrillic libgl1-mesa-dri x11-xkb-utils`
27. `apt-get install uuid-dev`
28. `apt-get install curl libcurl3 libcurl3-dev`
29. `apt-get build-dep qt4-qmake`
30. `./configure -fast -opensource -no-sql-mysql -no-sql-sqlite -no-qt3support -no-xmlpatterns -no-multimedia -no-audio-backend -n`
31. `#`
32. `# Warning - make took over 3 hours last time I tried it`
33. `#`
34. `make`
35. `make install`
36. `cd ..`
37. `rm -rf qt-everywhere-opensource-src-4.7.3`
38. Remember to Export `java_home`
39. `vim /home/ubuntu/.profile`
`export JAVA_HOME=/usr/lib/jvm/java-7-oracle`
40. check typing the comman `env`
`env`
41. Extra Steps

42. install manually qmake
sudo apt-get install qt4-qmake
43. allow people to copy into usr folder
sudo chmod ugo+w -R /usr/share/snmp/mibs/
44. install snmp
sudo apt-get install snmp
45. install dev libs
sudo apt-get install libsnmp-dev
46. sudo apt-get install libace-5.6.3
47. sudo apt-get install libace-dev
48. sudo chmod ugo+w /usr/share/snmp
49. sudo apt-get install libsctp-dev
50. sudo apt-get install libpopt-dev
51. Installing Replify package
52. cd /mnt/source/svn/
53. svn co <https://repository.altobridge.com/svn/platforms>
54. cd /mnt/source/svn/platforms/3G-platform/src/repo/lucid/pool/alto/replify
55. dpkg -i *.deb
56. # apt-get -f install gave error at this point, complaining about
57. # E: The package erlang-base-hipe needs to be reinstalled, but I can't find an archive for it.
58. # So
59. dpkg -i erlang-base-hipe_15.b.1-dfsg-3ubuntu1_amd64.deb #
60. # Try again
61. apt-get -f install
62. apt-get install gcc libpam-modules libpam0g-dev g++ unzip dpkg-dev
63. Checking out platforms
64. #It may take hours so instead of this we will mount platform as external drive
svn co <https://repository.altobridge.com/svn/platforms>

Installing HNBA in instance 172.16.5.34

65. First Check out the code
66. Remove the folder on /mnt/source
67. Installing missing packets
68. Autoconf
sudo apt-get install -y autoconf

69. Strange thing

```
sudo apt-get -f install
```

70. sudo apt-get install libxml2-dev

71. sudo apt-get install libsqlite3-dev

72. sudo apt-get install ncurses-dev

73. mount ie-build-1 into the instance

74. sudo mkdir -p /mnt/source/svn/platforms

75. edit sudo vim /etc/fstab

76. Add this line at the end

```
172.16.1.45:/mnt/source/svn/platforms /mnt/source/svn/platforms nfs  
ro,noatime,rsize=8192,wsiz=8192,nosuid,soft 0 0
```

77. sudo mount /mnt/source/svn/platforms

78. Now platforms is mounted

79. Check out Code

80. sudo mkdir -p /mnt/source/svn/date

81. sudo chown ubuntu /mnt/source/svn/date -R

82. cd /mnt/source/svn/date

83. svn co -N <https://repository.altobridge.com/svn/date/releases/>

84. cd releases/

85. svn co -N <https://repository.altobridge.com/svn/date/releases/2.0>

86. cd 2.0

87. svn co <https://repository.altobridge.com/svn/date/releases/2.0/RC17>

88. Now we have the code on place

89. Installing Headers Stuff

```
90. sudo rsync -av antonio.puche@ie-build-3:/usr/local/ossasn1.compile_and_build  
/usr/local/ossasn1
```

```
91. sudo mv /usr/local/ossasn1/ossasn1.compile_and_build/* /usr/local/ossasn1/
```

92. Useful commands

93. apt-file search sqlite3.h

94. End building at 09:21 on Thursday, 2013-04-11 (build took 5 minutes and 25.61 seconds)

Installing RDM in instance 172.16.5.35

95. cd /home/ubuntu/trackers/ie-build-3/root

96. vim install.sh (add -y to update and upgrade then chang -qqy to -y to see the trace)

97. mount ie-build-1 into the instance
98. svn up
99. chmod +x install.sh
 - a. nohup sudo ./install.sh &
 - b. Check Out Code
100. Start Debugging and installing packages as needed
101. export HOME=\$PWD * This one might not be necessary * only with release script
102. python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py altoPod RDM 2.0/RC17
103. If there is no PYTHONPATH set then
source bin/which_python
104. if OSError: [Errno 2] No such file or directory
105. sudo mkdir -p /mnt/releases/building/building/ubuntu-10.04/2.0/RC17/log/
106. sudo chown ubuntu /mnt/releases -R
107. Install autoconf.
108. If apt-get is fucked up then
sudo apt-get -f -y install
109. If requires java set JAVA_HOME (th e6-jdk is already installed)
110. vim /mnt/releases/building/building/ubuntu-10.04/2.0/RC17/log/current/build.sh.log
to debug errors tagged as 'build errors'
111. try builds yourself by doing for example
bash build.sh rdm
112. Install jdk 1.7 for RDM
113. Change permission in snmp/mibs
114. sudo chmod 777 /usr/share/snmp/mibs/ -R
115. sudo chmod 777 /usr/share/snmp/
116. # End building at 09:11 on Thursday, 2013-04-11 (build took 28.16 seconds)

Installing ABC in instance 172.16.5.36

117. do the exact first 6 steps in RDM installation
118. Start Debugging and installing packages as needed
119. cd /home/ubuntu/trackers/ie-build-1/guest/builder/
120. source bashrc
121. source bin/which_python

122. python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py altoPod ABC 2.0/RC17
123. OSError: [Errno 2] No such file or directory
124. sudo apt-get -f install
125. Install missing packages
126. sudo mkdir -p /mnt/releases/building/building/ubuntu-10.04/2.0/RC17/log/
127. sudo chown ubuntu /mnt/releases -R
128. Installing necessary packages
129. epam.c:2:22: error: pam_appl.h: No such file or directory
130. apt-file search pam_appl.h
131. sudo apt-get install libpam0g-dev
132. mkdir: cannot create directory `ebin': File exists
133. sudo apt-get install unzip
134. # End building at 10:24 on Thursday, 2013-04-11 (build took 12 minutes and 11.73 seconds)
135. ll /mnt/source/svn/date/releases/2.0/RC17/dataopt/replify/release
136. rm /mnt/source/svn/date/releases/2.0/RC17/dataopt/replify/release
137. just in case it says that built_package() is empty

Installing GAP in instance 172.16.5.32

138. do the exact 6 steps in RDM installation
139. install autoconf
140. Start debugging
141. same 3 steps as before
142. sudo mkdir -p /mnt/releases/date
143. sudo chown ubuntu /mnt/releases -R
144. python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py altoPod GAP 2.0/RC17
145. OS Error
146. # End building at 10:59 on Thursday, 2013-04-11 (build took 7.23 seconds)

Code Changes

147. Change the release script to comment out the line
148. @@ -8,7 +8,7 @@
149. # For each server we know which packages it needs
150. -known_altoPod_packages="HNBA RDM ABC

151. +known_altoPod_packages="HNBAC"
152. The code has to be changed in three different parts (date.py, releaseble.py)
153. Still doesnt work scp does not appear
154. You have to modify
155. release_server.sh
156. date.py in 2 places
157. Steps to build the projects in instances
158. find out which project, now ABC
159. create a new one that inherences from that one but addin cloud, ABCCloud
160. change the parameters
161. Follow up with the change in code on Altobridge build system explained down there

4.4 Altobridge build system

The build system that is used in Altobridge has been created in the company for packing their own software purposes. With this system Altobridge is able to package the software into a Linux package and deploy it to the user as a single packet. The name of the system is Trackers.

One of the advantages of this system is that it allows the user to build different version of Linux by mounting disk in different drives which are all located in the same disk. It is a feature taken from Linux and this build system exploits it a lot.

One of the disadvantages is that is very difficult to modify. For each change that has to be done when building a project there is a need to test that the project is building. This build system has been developed in python by a single engineer so for each different error that comes up there is a need to contact that engineer and query him about the error.

Most of the errors have been debugged with the support of the Altobridge engineers who support this project and who this project is made for. All the build system is a set of python classes where all the required operations are contained to create folders, move files, copy files and ssh instances.

What needs to be done is to modify those classes to be able to build the same projects in different instances and then copy the result back to the initial instance. In that way the

main python script will run as if it were in a single machine and it won't know that the builds are actually being done in different machines.

To put into context, the python script fire processes that ssh instances and then it copies back the result. To put the python system in place the first thing that needs to be done is to check all the code for the python classes, the python project can be found in the Altobridge intranet under the name of 'trackers'. The code needs to be check out in order to start the implementation:

```
svn co http://trackers.altobridge.com/svn/date/trackers
```

In more detail: This command will check out all the code necessary to build projects for Altobridge. Is not known how many classes are contained in this package but for this experiment only a number of classes will be reviewed.

To be able to modify the way that projects are being built the first thing to do is to modify the class that holds that project. In this python system a hierarchy of classes is contained to hold all the different projects and variations.

In the folder 'projects' all the variations of the classes for all the possible projects can be found, so the first thing will be to take a look at the class *date.py* contained in the folder projects. That class contains all the different projects for the branch date.

The class that will be reviewed *trackers/python/site-packages/altobridge/building/projects/date.py* where all the projects can be found.

The product that will be built for this experiment will be an Altopod:

```
package_name = 'altoPod'
```

```
def __init__(self):  
    DateServer.__init__(self, 'HNBA RDM ABC GAP')
```

The product Altopod is comprised of four different projects: HNBA, RDM, ABC and GAP. These four projects are independent of each other but they might depend in other packages. For example, we have RDM that needs to build NEM first and with NEM then builds RDM.

The list of packages that can be built are found in this class as well as under the name packages:

```
def _projects_with_packages():
    """A list of project objects which produce packages"""
+   return [CDM(), RDM(), HNBG(), HNBA(), OMC_3G(), ABC(), GuiAltoPod(),
NetworkElementManagement(), HNBAC(), RDMC(), ABCC(), GuiAltoPodCloud()]
```

Here at the end of the line the projects that are going to be created have to be added, the projects that have been created for this experiment are HNBAC, RDMC, ABCC and GuiAltoPodCloud. The necessary modification to address is the new projects which extend from the previous ones that already exist.

Note that, the instance where the build will take place has to be the exact same one as the build from where this script is being run from. If the controller machine is Ubuntu 10.04 the other machine where the script will do ssh has to be Ubuntu 10.04 and has to have the python system (trackers) installed in order to make the build successful. In this way the ‘Trackers’ will not know if the build is taking place on same machine or on a different device.

The next step is to add the new classes to build the new projects. The new projects will be built in the same way as previously done but in the cloud the new classes will be inherited from the old ones. All the necessary steps and the methods that execute the compile command as *makefile* or *mvn compile* will be overridden to do the exact same thing on the instance instead.

In the HNBA project the class that represents this project is called HNBA, in order to override this class and create the new one a new class will need to be created with the name of HNBAC. The ‘C’ stands for cloud.

The two methods inside the class HNBA that have to be changed in order not to alter the behaviour of the build system are *build_command_line* and *move_package*. The first one needs to do the ssh to the instance and the second needs to do scp to retrieve the resulting package from the instance.

The code looks like this:

```
class HNBAC(HNBA):
    name='Home Node B Access Cloud'
    instance_ip = '172.16.5.34'

    def build_command_line(self):
    return '%s "cd %s;%s" ' % ( ssh_to(self.instance_ip), self.source.path ,
HNBA.build_command_line(self))
```



```
def move_package(self):
    self.build_completed = True
    self.package = linux.package(
        self.spec.package_dir(),
        self.package_name,
        self.source.build,
        self.source.release
    )
    #if command_line.options.quick:
    self.package.path = self.package.path.make_file_exist()
    #else:
        #self.package.path.assert_exists()
        log_sh.run_command('%s%s %s' % ( scp_here_from(self.instance_ip),
self.package.path, self.package.path))
        self.package.moveTo(self.release.project_being_released.path_to_building)
```

The first command does the ssh to the instance and runs the command to compile the project. For HNBA is a make file but for other projects might be Java compile or Maven compile.

The command is embedded in the class so with this implementation there is no need to know which command to run on the machine. The only requirement is to guarantee that the machine is the same one that the controller uses, otherwise a lot of errors with paths will arise.

The two methods that does the ssh and the scp are:

```
def ssh_to(ip):
    return 'ssh -i /home/ubuntu/builder/builder_rsa -o "StrictHostKeyChecking no"
ubuntu@%s' % ip

def scp_here_from(ip):
    return 'scp -i /home/ubuntu/builder/builder_rsa -o "StrictHostKeyChecking no"
ubuntu@%s:' % ip
```

In the first command ssh is done to the instance with the option ‘*No host key checking*’ to avoid being asked for host permission. In the second command the result package is copied back to the controller instance.

With this implementation we have just included a new project in the system. Now the project has to be executed. To execute this project the project has to be marked as ‘*releasable*’. This step is necessary because a shell script will be used to run the build and



it will run all the projects marked as releasable, if the project is not marked then it won't build. That script is called 'release_server.sh'.

The way to add the project to a list of release projects is modifying the file called releaseable.py. That file can be found in 'trackers/python/site-packages/altobridge/releasing/releaseable.py'. The next line has to be modified like this:

```
def _date_all_packages():  
    packages = ['GAP', 'ABC', 'HNBA', 'HNBG', 'RDM', 'CDM', 'OM3', 'NEM', 'HNBAC',  
               'RDMC', 'ABCC', 'GAPC']
```

The other projects that require modifications are: RDM, ABC and GAP. The procedure to modify them will be the same. A new class will be created extending from the old one but modifying the build methods to execute them in the cloud.

Let's see ABCC:

```
class ABCC(ABC):  
    name = 'Accelerating Byte Cacher Cloud'  
    instance_ip = '172.16.5.36'  
  
    def build_command_line(self):  
        return '%s "cd %s;%s" ' % ( ssh_to(self.instance_ip), self.source.path ,  
ABC.build_command_line(self))  
  
    def clean_command_line(self):  
        return '%s "cd %s;%s" ' % ( ssh_to(self.instance_ip), self.source.path ,  
ABC.clean_command_line(self))  
  
    def post_build(self):  
        self.path_to_built_packages().make_directory_exist()  
        log_sh.run_command('%s%s %s' % ( scp_here_from(self.instance_ip),  
self.path_to_built_packages()/'*.deb', self.path_to_built_packages()))  
        ABC.post_build(self)
```

The definition of *build_command_line* is the same as in the last example (HNBA) but in this build there is a 'clean' command that has to be run. The way to do this is by creating a *clean_command_line* which runs a clean command before the compilation is done, the way the methods are modified is by adding to the clean command a 'ssh' string before with the ip of the instance so that command will be executed in an instance.

The last method *post_build* creates the folder where the result package is going to be. The method is now the same as it was but instead of getting the package from the current machine a scp command is done to retrieve from the instance where the package has been built.

As a reminder the build system thinks that everything is being built in one machine so it does not know anything about the instances so everything has to be exactly the same on all of the machines.

The next project to review is RDM.

```
class RDMC(RDM):
    name='Remote Data Manager Cloud'
    instance_ip = '172.16.5.35'

    def build_command_line(self):
        return '%s "cd %s;%s" ' % ( ssh_to(self.instance_ip), self.source.path ,
RDM.build_command_line(self))

    def clean_command_line(self):
        return '%s "cd %s;%s" ' % ( ssh_to(self.instance_ip), self.source.path ,
RDM.clean_command_line(self))

    def move_results(self):
        path_to_source_build = makepath(self.source.path / 'build')
        try:
            path_to_source_build.make_directory_exist()
            log_sh.run_command('%s%s %s' % (
scp_here_from(self.instance_ip), path_to_source_build/'*.deb', path_to_source_build))
            package = path_to_source_build.files(self.package_glob())[0]
        except IndexError:
            raise BuildError(self, 'No results found at %s/*.deb' %
path_to_source_build)
        package.mv(self.path_to_package())

    def set_java_home(self):
        pass
```

This project requires Java to be built and checks if Java home is set correctly. The problem with this is that the check will need to be done before jumping into the instance, by the controller, otherwise the build will give an error saying that Java was not installed. To get over this error the method that sets Java home has to be overwritten to do nothing, that is *pass*.



The methods *clean_command_line* and *build_command_line* are the same as in past implementations. The method *move_results* is a bit different because the path to the build has to be made before the files have to be copied in and the final package can be moved to a different location.

Finally the last project GuiAltoPod. Is the simplest one of the four and the changes are the same as in ABC project.

```
class GuiAltoPodCloud(GuiAltoPod):
    name = 'Gui Alto Pod Cloud'
    instance_ip = '172.16.5.22'

    def build_command_line(self):
        return '%s "cd %s;%s" ' % ( ssh_to(self.instance_ip), self.source.path ,
GuiAltoPod.build_command_line(self))

    def clean_command_line(self):
        return '%s "cd %s;%s" ' % ( ssh_to(self.instance_ip), self.source.path ,
GuiAltoPod.clean_command_line(self))

    def post_build(self):
        self.path_to_built_packages().make_directory_exist()
        log_sh.run_command('%s%s %s' % ( scp_here_from(self.instance_ip),
self.path_to_built_packages()/'*.deb', self.path_to_built_packages()))
        GuiAltoPod.post_build(self)
```

Once included, there is another place where the name of the project has to be added to be able to execute that file of the release script that will be used to trigger the build.

That file can be found on

```
'/mnt/source/svn/date/releases/2.0/RC17/release_scripts/release_server.sh'
```

The way to use this script is by using the next command:

```
bash release_server.sh altoPod 2.0/RC17 -m
```

Bash is the Linux shell, *release_server.sh* is the script itself and the option are *altoPod*, which is the product that will be built for this experiment, and the version of that product *2.0/RC17*. The last option means '*muddy*' it is saying that build should not update the files on the tracker and that release should not be able to deliver to the customer. Is like a test, this mode will be used to ensure that the script is working properly.

By using this command the altopod will be packaged and ready to be deployed on a machine. However, this script builds the product package by package, in other words, it builds project per project sequentially even if the build jobs are done in the cloud. The purpose of this research is to see how fast the builds can be done by using the cloud so this script will be modified to run all the jobs in the cloud on parallel.

For that purpose the next lines will be modified:

For each server we know which packages it needs

```
-known_altoPod_packages="HNBA RDM ABC GAP"  
+known_altoPod_packages="HNBAC RDMC ABCC GAPC"
```

In this line the products included are the same as the old ones but with a ‘C’ at the end specifying that the project will be built in the cloud.

The next code to modify is the loop that builds the project. In the original version of the script the builds were done sequentially but now the script will be modified to do the jobs in parallel.

```
build_server ()  
{  
    pid_array=()  
    for release_package in $release_packages  
    method_call &  
    pid_array+=($!)  
        echo "${pid_array[@]}"  
    done  
    for pid in $pid_array  
    do  
  
        echo "My pid is ($pid)"  
        wait $pid  
        rc=$?  
        if [ $rc -ne 0 ] ; then  
            exit $rc  
        fi  
    done  
}
```

The method *build_server* creates the Altopod. Inside this method we have *method_call* which builds each of the project of the server (altopod). To make it independent and able

to run in parallel the part of the program that was building the project was extracted and encapsulated in a method called *method_call*. To run this method in parallel the only thing remaining is to add an ‘&’ at the end of the line to run it in the background. In this way all four processes will be built as independent processes (background processes).

After the four jobs have started running another loop has to be included to make sure that the build method waits until all four projects are built. PID stands for Process Id and basically is a number assigned to a process to check if the process is still running or not and identify it.

Once all four projects are built we need to check if any of them have returned an error message and if so then exit scripts with the same error code in the process.

Due to the parallel nature of the current procedure there might be errors with paths and folders. In particular there are some folders that have to be created to perform the build. If those folders are already created the build will raise an error and it will not continue. To avoid this error, which is not really an error because the folder already dose exist, a try catch statement has to be made to catch the situation and to say that everything is fine and the program should continue working. The file to modify is the next one:

/home/ubuntu/trackers/python/site-packages/altobridge/shell/script_logger.py

The modification to do will be the next one:

```
path_to_old_linkee = path_to_old_linker.realpath()
path_to_new_linkee = path_to_new_logs / processes.identification_directory()
move_aside(path_to_new_linker)
try:
    path_to_new_linkee.make_directory_exist()
    path_to_new_linkee.symlink(path_to_new_linker)
except OSError:
    pass
for path_to_old_item in path_to_old_linkee.walkfiles():
    relative_path = path_to_old_linkee.relpathto(path_to_old_item)
    path_to_new_item = path_to_new_linkee / relative_path
    path_to_new_item.write_text(path_to_old_item.text(), append=True)
    path_to_old_item.try_remove()
    path_to_old_linkee.try_remove()
    try:
        path_to_old_linker.remove()
    except OSError:
        pass
```

```
self._set_current_directory(path_to_new_linker)
```

4.5 Putting all the parts together

Once the Controller Build Instance (the instance from where the build is triggered) is ready and does the job, now the build has to be triggered from Jenkins.

To do that we must first change the user Jenkins because it is already installed and we need the permission from the admin's owns so the easiest way is to run the builds as *ubuntu user* instead of *jenkins user*.

To do that the next steps have to be followed:

On the instance where *jenkins* is installed go to the file `/etc/default/jenkins` and change the next line:

```
# user id to be invoked as (otherwise will run as root; not wise!)  
-JENKINS_USER=jenkins  
+JENKINS_USER=ubuntu
```

Then change the permission of the following folders:

```
chown -R ubuntu /var/log/jenkins  
chown -R ubuntu /var/lib/jenkins  
chown -R ubuntu /var/run/jenkins  
chown -R ubuntu /var/cache/jenkins
```

Then restart the Jenkins server

```
sudo service jenkins restart
```

Once Jenkins is restarted there it is possible to start configuring jobs from the GUI on the webserver.

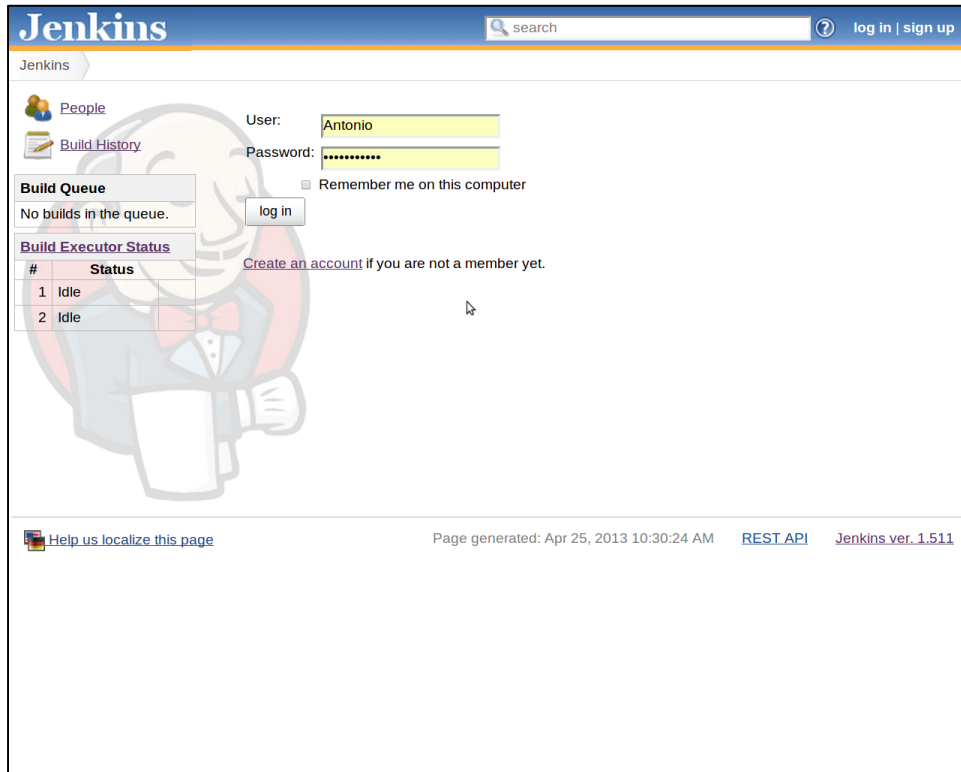
A user has been created with admin permissions to perform jobs in Jenkins and that user is called Antonio.

The first thing to do is to access the login page of Jenkins and type the credentials, the instances are set up in the private network of Altobridge so we will just type the private id and it will work for this experiment. Just typing in the browser the url:

<http://172.16.5.18:8080>

It should work:

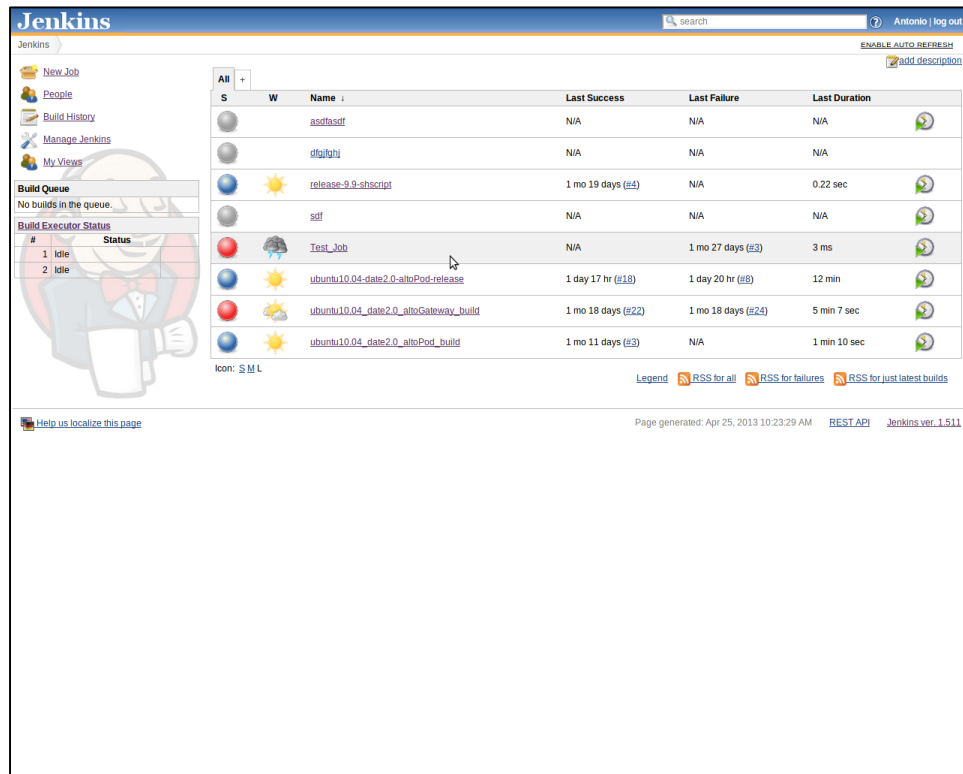
“Moving to continuous integration on the cloud”



The creation of the user is not relevant for this experiment the only requirement is to get a user with administrator rights and everything will work fine.

“Moving to continuous integration on the cloud”

In the next step will be to access to the dashboard to star configuring a job:



The screenshot shows the Jenkins dashboard interface. At the top, there's a search bar and user information (Antonio | log out). Below the search bar, there's a navigation menu with options like 'New Job', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. The main content area displays a table of builds. The table has columns for 'S', 'W', 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The build 'ubuntu10.04-date2.0-altoPod-release' is highlighted in blue. Below the table, there are links for 'Legend', 'RSS for all', 'RSS for failures', and 'RSS for just latest builds'. At the bottom, there's a footer with 'Page generated: Apr 25, 2013 10:23:29 AM', 'REST API', and 'Jenkins ver. 1.511'.

S	W	Name	Last Success	Last Failure	Last Duration
		asdfasdf	N/A	N/A	N/A
		dfjghj	N/A	N/A	N/A
		release-9.9-shscript	1 mo 19 days (#4)	N/A	0.22 sec
		sdf	N/A	N/A	N/A
		Test_Job	N/A	1 mo 27 days (#3)	3 ms
		ubuntu10.04-date2.0-altoPod-release	1 day 17 hr (#18)	1 day 20 hr (#8)	12 min
		ubuntu10.04_date2.0_altoGateway_build	1 mo 18 days (#22)	1 mo 18 days (#24)	5 min 7 sec
		ubuntu10.04_date2.0_altoPod_build	1 mo 11 days (#3)	N/A	1 min 10 sec

The dashboard shows up all the information about the builds.

For this experiment only the job on charge of building an Altopod is prepared, up and running. The other jobs are mostly tests.

From the dashboard the icon with weather images shows the status of the build. The status of the build is calculated by Jenkins taking into account the number of times that the build has broken down in the last 5 or 10 builds. Depending of the number of failures the icon will be more rainy and cloudy or it will be very sunny if all the builds are fine.

For Jenkins a successful build is a process or command that returns 0. If at the end of the script the last command returns 0 then the build will be successful for Jenkins. Otherwise the command could return as being failed.

The build that works is *ubuntu10.04-date2.0-altoPod-release* to see the implementation of that build the only requirement is to click on the name and see all the data for that job.

“Moving to continuous integration on the cloud”

Jenkins ubuntu10.04-date2.0-altoPod-release Antonio | log out

Back to Dashboard

Project ubuntu10.04-date2.0-altoPod-release

The release script is used to build all 4 projects in AltoPod in paralel. This is RC17. HNBAC RDMC ABCC GAPC

edit description
Disable Project

Workspace
Recent Changes

Build History (trend)

#	Time
#18	Apr 23, 2013 4:51:38 PM
#17	Apr 23, 2013 4:29:38 PM
#16	Apr 23, 2013 4:16:16 PM
#15	Apr 23, 2013 4:02:01 PM
#14	Apr 23, 2013 3:45:20 PM
#13	Apr 23, 2013 3:43:12 PM
#12	Apr 23, 2013 3:40:48 PM
#11	Apr 23, 2013 3:37:11 PM
#10	Apr 23, 2013 3:35:14 PM
#9	Apr 23, 2013 3:03:28 PM
#8	Apr 23, 2013 2:15:09 PM
#7	Apr 19, 2013 3:45:58 PM
#6	Apr 19, 2013 3:39:30 PM
#5	Apr 19, 2013 3:37:03 PM
#4	Apr 19, 2013 3:36:02 PM
#3	Apr 19, 2013 3:32:11 PM
#2	Apr 19, 2013 3:31:11 PM
#1	Apr 19, 2013 3:30:05 PM

Permalinks

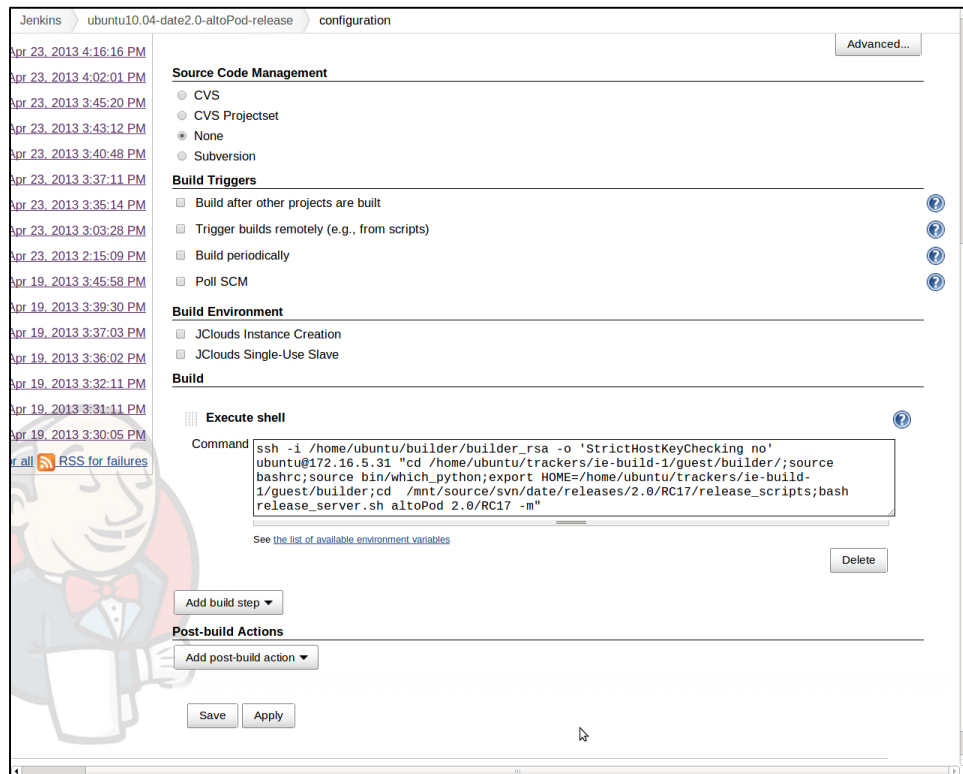
- Last build (#18), 1 day 17 hr ago
- Last stable build (#18), 1 day 17 hr ago
- Last successful build (#18), 1 day 17 hr ago
- Last failed build (#9), 1 day 20 hr ago
- Last unsuccessful build (#8), 1 day 20 hr ago

Help us localize this page

Page generated: Apr 25, 2013 10:23:57 AM REST API Jenkins ver. 1.511

Here are all the times that the build has been triggered which are shown on the left. All the times of these builds are measured by Jenkins. First of all it is necessary to see the commands that make the job. Just by clicking on *configure on the top left* and scrolling down its visible the shell script that is run by this job.

“Moving to continuous integration on the cloud”



Under the header **Build** in bold letters it says *Execute shell*, that part executes the shell script that will run the job. It is basically an ssh command that runs the build in a different machine. Here it can be seen:

```
ssh -i /home/ubuntu/builder/builder_rsa -o 'StrictHostKeyChecking no'  
ubuntu@172.16.5.31 "cd /home/ubuntu/trackers/ie-build-1/guest/builder;;source  
bashrc;source bin/which_python;export HOME=/home/ubuntu/trackers/ie-build-  
1/guest/builder;cd /mnt/source/svn/date/releases/2.0/RC17/release_scripts;bash  
release_server.sh altoPod 2.0/RC17 -m"
```

This is an ssh command that runs all the necessary shell commands to trigger the build in the build Controller.

To see if the build is complete successfully Jenkins holds an option called *Console Output* which shows up all the outputs from the commands that have been run in the cloud. To see them it is only necessary to click on one of the builds on the left and click on *Console Output*. Here there is an example of the last build that was run:

“Moving to continuous integration on the cloud”

```
Jenkins ubuntu10.04-date2.0-altopod-release #18
Back to Project
Status
Changes
Console Output
View as plain text
Edit Build Information
Delete Build
Previous Build

Console Output

Started by user Antonio Puche
Building in workspace /var/lib/jenkins/jobs/ubuntu10.04-date2.0-altopod-release/workspace
[workspace] $ /bin/sh -xe /tmp/hudson5869392654906665.sh
+ ssh -i /home/ubuntu/builder/builder_rsa -o StrictHostKeyChecking=no ubuntu@172.16.5.31 cd /home/ubuntu/trackers/site-build-1/guest/builder; source bashrc;source bin/which python;export HOME=/home/ubuntu/trackers/site-build-1/guest/builder;cd /mnt/source/svn/date/releases/2.0/RCL7/release_scripts;bash release_server.sh altopod 2.0/RCL7 -n 820
820 821
820 821 822
820 821 822 823
Using issue 19231
svn up /home/ubuntu/trackers/python/site-packages : Using issue 19231
svn up /home/ubuntu/trackers/python/site-packages : Using issue 19231
svn up /home/ubuntu/trackers/python/site-packages : Using issue 19231
svn: Working copy '/home/ubuntu/trackers/python/site-packages' locked
svn: run 'svn cleanup' to remove locks (type 'svn help cleanup' for details)
svn: run 'svn cleanup' to remove locks (type 'svn help cleanup' for details)
svn: Working copy '/home/ubuntu/trackers/python/site-packages' locked
svn: run 'svn cleanup' to remove locks (type 'svn help cleanup' for details)
It looks like you ran this on Ubuntu 10.04.4 LTS in \:
$ /usr/bin/python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py --muddy 2.0/RCL7 RDMC altopod

# Start building on Ubuntu 10.04 at 16:56 on Tuesday, 2013-04-23
Building RDMC: Source: /mnt/source/svn/date/releases/2.0/RCL7, destination: /mnt/releases/date
It looks like you ran this on Ubuntu 10.04.4 LTS in \:
$ /usr/bin/python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py --muddy 2.0/RCL7 ABCC altopod

# Start building on Ubuntu 10.04 at 16:56 on Tuesday, 2013-04-23
Building ABCC: Source: /mnt/source/svn/date/releases/2.0/RCL7, destination: /mnt/releases/date
It looks like you ran this on Ubuntu 10.04.4 LTS in \:
$ /usr/bin/python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py --muddy 2.0/RCL7 GAPC altopod

# Start building on Ubuntu 10.04 at 16:56 on Tuesday, 2013-04-23
Building GAPC: Source: /mnt/source/svn/date/releases/2.0/RCL7, destination: /mnt/releases/date
# ----> Remote Data Manager Cloud
# ----> Accelerating Byte Cacher Cloud
# ----> Gui Alto Pod Cloud
U /home/ubuntu/trackers/python/site-packages/roundup/mailgw.py
Updated to revision 35095.
It looks like you ran this on Ubuntu 10.04.4 LTS in \:
$ /usr/bin/python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py --muddy 2.0/RCL7 HNBAC altopod

# Start building on Ubuntu 10.04 at 16:57 on Tuesday, 2013-04-23
Building HNBAC: Source: /mnt/source/svn/date/releases/2.0/RCL7, destination: /mnt/releases/date
# ----> Home Node B Access Cloud
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/RDM/NEH-2.0.RCL7-aad64.deb
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/GAP/GAPC-2.0.RCL7-aad64.deb

Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 16:57 on Tuesday, 2013-04-23 (build took 10.32 seconds)
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/NEH-2.0.RCL7-aad64.deb
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/altorgv-2.0.RCL7-aad64.deb

Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 16:57 on Tuesday, 2013-04-23 (build took 10.32 seconds)
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/NEH-2.0.RCL7-aad64.deb
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/altorgv-2.0.RCL7-aad64.deb

Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 16:57 on Tuesday, 2013-04-23 (build took 10.32 seconds)
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/NEH-2.0.RCL7-aad64.deb
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/altorgv-2.0.RCL7-aad64.deb

Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 16:58 on Tuesday, 2013-04-23 (build took 1 minutes and 2.95 seconds)
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/RDM/rde-2.0.RCL7-Ubuntu.LTS.aad64.deb

Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 17:08 on Tuesday, 2013-04-23 (build took 11 minutes and 53.32 seconds)
tar: on: Cannot stat: No such file or directory
tar: ubuntu10.altobridge.com: Cannot stat: No such file or directory
tar: from: Cannot stat: No such file or directory
tar: 2.0/RCL7: Cannot stat: No such file or directory
tar: Exiting with failure status due to previous errors
/mnt/releases/date/ubuntu-10.04/2.0/RCL7/altopod-2.0.RCL7.tar
V----- 0/0 0 2013-04-23 17:08 built--Volume Header--
-rwxr-xr-x ubuntu/ubuntu 4709 2013-04-23 16:56 install_altopod
Finished: SUCCESS
```

```
Jenkins ubuntu10.04-date2.0-altopod-release #18

# Start building on Ubuntu 10.04 at 16:56 on Tuesday, 2013-04-23
Building GAPC: Source: /mnt/source/svn/date/releases/2.0/RCL7, destination: /mnt/releases/date
# ----> Remote Data Manager Cloud
# ----> Accelerating Byte Cacher Cloud
# ----> Gui Alto Pod Cloud
U /home/ubuntu/trackers/python/site-packages/roundup/mailgw.py
Updated to revision 35095.
It looks like you ran this on Ubuntu 10.04.4 LTS in \:
$ /usr/bin/python /home/ubuntu/trackers/python/site-packages/altobridge/building/build.py --muddy 2.0/RCL7 HNBAC altopod

# Start building on Ubuntu 10.04 at 16:57 on Tuesday, 2013-04-23
Building HNBAC: Source: /mnt/source/svn/date/releases/2.0/RCL7, destination: /mnt/releases/date
# ----> Home Node B Access Cloud
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/RDM/NEH-2.0.RCL7-aad64.deb
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/GAP/GAPC-2.0.RCL7-aad64.deb

Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 16:57 on Tuesday, 2013-04-23 (build took 10.32 seconds)
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/NEH-2.0.RCL7-aad64.deb
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/HNBA/altorgv-2.0.RCL7-aad64.deb

Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 16:58 on Tuesday, 2013-04-23 (build took 1 minutes and 2.95 seconds)
# SUCCESS: /mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/RDM/rde-2.0.RCL7-Ubuntu.LTS.aad64.deb

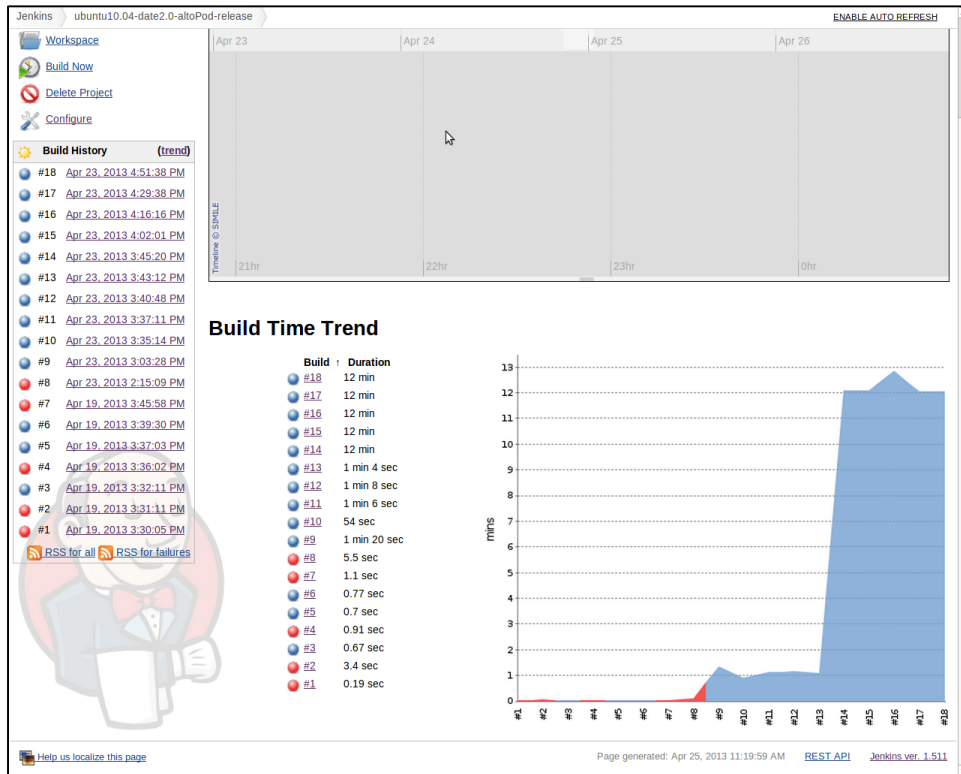
Build logs are at:
/mnt/releases/building/building/ubuntu-10.04/2.0/RCL7/log/current/
Tarballs:
Returning 0 in line 77
# End building at 17:08 on Tuesday, 2013-04-23 (build took 11 minutes and 53.32 seconds)
tar: on: Cannot stat: No such file or directory
tar: ubuntu10.altobridge.com: Cannot stat: No such file or directory
tar: from: Cannot stat: No such file or directory
tar: 2.0/RCL7: Cannot stat: No such file or directory
tar: Exiting with failure status due to previous errors
/mnt/releases/date/ubuntu-10.04/2.0/RCL7/altopod-2.0.RCL7.tar
V----- 0/0 0 2013-04-23 17:08 built--Volume Header--
-rwxr-xr-x ubuntu/ubuntu 4709 2013-04-23 16:56 install_altopod
Finished: SUCCESS

Help us localize this page
Page generated: Apr 25, 2013 10:25:18 AM BEST API Jenkins ver 1.511
```

From here all the results of the build can be seen as if it were on the console itself.

“Moving to continuous integration on the cloud”

Another feature of Jenkins is a graph that shows up the times of the last builds. This graph makes it easier to see which are taking longer and which builds might be stuck due to a very long build procedure.

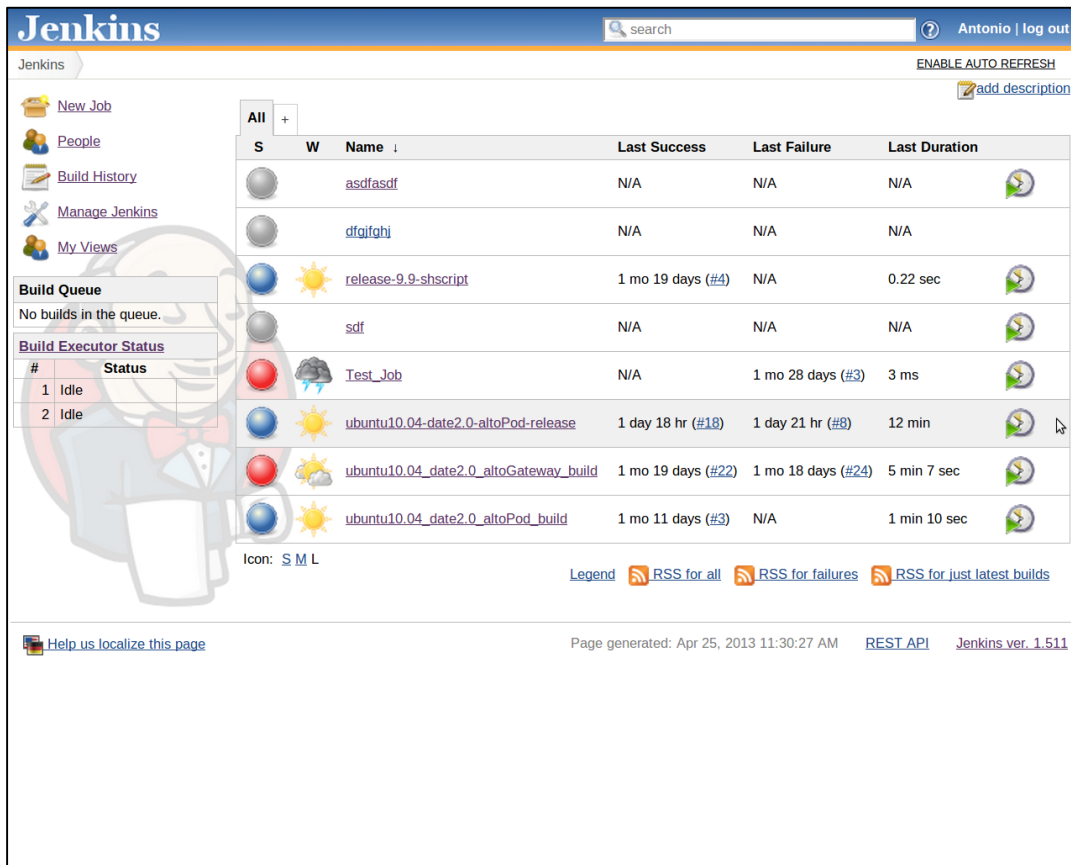


This feature is useful to manage the time that the builds take.

The initial idea was to run a build every hour or every time a developer commits a code but putting this into practise became very difficult to implement on the build system and to ensure it worked for the experiment. After some revaluation the builds were then set to trigger when clicking on the button (which is an improvement from the old system where it has to be triggered by shell commands).

To run a build for an Altopod the only thing remaining is to click on the button on the right side of the name of the build name and the build will be done. It is part of the requirements which were to run builds just with a button and would be improved the times which is a necessary step for continuous integration.

“Moving to continuous integration on the cloud”



The screenshot shows the Jenkins web interface. At the top, there's a search bar and the user name 'Antonio | log out'. Below the search bar, there's a navigation menu with options like 'New Job', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. The main content area displays a table of builds with columns for 'S' (Success), 'W' (Warning), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The table lists several builds, including 'release-9.9-shscript', 'Test_Job', and 'ubuntu10.04_date2.0_altoPod-release'. A 'Build Queue' section on the left indicates 'No builds in the queue'. Below the queue, there's a 'Build Executor Status' table showing two executors in an 'Idle' state. At the bottom, there's a footer with 'Page generated: Apr 25, 2013 11:30:27 AM' and 'Jenkins ver. 1.511'.

S	W	Name	Last Success	Last Failure	Last Duration
●		asdfasdf	N/A	N/A	N/A
●		dfgdfghj	N/A	N/A	N/A
●	☀	release-9.9-shscript	1 mo 19 days (#4)	N/A	0.22 sec
●		sdf	N/A	N/A	N/A
●	☁	Test_Job	N/A	1 mo 28 days (#3)	3 ms
●	☀	ubuntu10.04_date2.0_altoPod-release	1 day 18 hr (#18)	1 day 21 hr (#8)	12 min
●	☀	ubuntu10.04_date2.0_altoGateway_build	1 mo 19 days (#22)	1 mo 18 days (#24)	5 min 7 sec
●	☀	ubuntu10.04_date2.0_altoPod_build	1 mo 11 days (#3)	N/A	1 min 10 sec

From here the system is ready to build an Altopod. If it is successful it will be ready to package it and send it to test or if the tests are included in the build will be just a matter of releasing it to the user.

4.6 Failed implementations and errors

4.6.1 Logs corrupted due to multithreading

There might be some errors due to the parallel way of doing the builds. The main problem is found when writing logs because all the instances are writing logs in the same file so the files might become corrupted. The logs are still readable but not trustworthy because the order in which the commands are executed may change every time with every build so it is not possible to trust them.

4.6.2 Script to start and stop instances from the cloud

This implementation could be considered as a second approach. With this approach instead of having instances running all the time, an improvement was made to the resources of the cloud by using snapshots. This approach is better because it uses the resources of the cloud wisely but the performance is impacted. The main problem with

this instance is if they are finally started they may be slow. During implementation it usually took 15 minutes to start and another 15 minutes to stop the instance plus the time to delete the instance that was around a minute. So in total there was a +31 min to add to the build time. That approach, even if it worked, was not feasible because it was making the process very slow.

At the very beginning of the experiment during the research of the capabilities of the cloud and because of the high requirements of the build system it was obvious that the cloud won't handle the number of instances that were needed to be run at the time so the solution was to create a manager script on shell script that performs instance management. The instance management to perform was the next one:

1. The script will find a particular snapshot in the database.
2. Then Jenkins will ssh the instance in particular and will the run the build as is doing now.
3. After that another script will take a snapshot of that instance and it will store with the same name as the old one.
4. Then the old one will be deleted.
5. The new snapshot is ready to use.

The instances are stored in the database by using unique id made of a mix between letters are numbers but when starting and stopping the names of the instances have to have a relation with the job they are doing so the programmers we can identify which one is running and see if it's correct. the naming convention we use was to put the first 2 letters of the job we want to run, example:

The job *ubuntu10.04-date2.0-altoPod-release* will be named as *ub10da20PodBuild* since *alto* is not relevant because all the products of Altobridge are named as alto something only the second word of the name of the product will be used.

Basically the procedure to follow was to run one script to start the right instance then let Jenkins work and then run another script to stop and store the instance.

Here is the script to start instances:

```
#!/bin/bash
keypair_user=builder
openrc_conf=/home/ubuntu/builder/openrc.sh
echo "Configuring Nova Client $openrc_conf"
sudo route add -host 172.16.4.11 gw 172.16.5.17
```

```
source $openrc_conf
echo 'Checking that nova commands can be executed'
nova list

instance_name=$1
build_steps=$2
if [ -z $instance_name ]; then
    echo 'The name of the instance can not be empty'
    exit 1
elif [ -z $build_steps ]; then
    echo 'The build steps can not be empty'
    exit 1
else
    echo "The instance name is $instance_name"
    instance_id=$(nova image-list | awk -F'|' '/$instance_name/ { print \$2 }')
    if [ -z $instance_id ]; then
        echo 'The Instance Id has not been found by the name given'
        nova image-list
        exit 2
    else
        echo "The Instance Id has been found! Id:$instance_id"
        nova boot $instance_name --image $instance_id --flavor 1 --key_name
        $keypair_user
        booting=true
        while [ "$booting" = "true" ]; do
            state=$(nova list | awk -F'|' '/$instance_name/ { print \$4 }' | tr -d ' ')
            if [ "$state" = "ACTIVE" ]; then
                booting=false
            elif [ "$state" = "BUILD" ]; then
                echo "The state of the instance is $state and $booting. Going to Sleep
                for a minute."
                sleep 60
            else
                echo "More than one instance found for that name $instance_name
                $state"
                nova list
                exit 3
            fi
        done
        instance_ip=$(nova list | awk -F'|' '/$instance_name/ { print \$5 }' | cut -d '=' -f2 )
        echo "Getting instance ip $instance_ip"
        ssh-keygen -f "/home/ubuntu/.ssh/known_hosts" -R $instance_ip
        ssh -i /home/ubuntu/builder/builder_rsa -o 'StrictHostKeyChecking no'
        ubuntu@$instance_ip "$build_steps"
        exit 0
    fi
done
```

```
    fi  
fi
```

Here is the script to stop the instances:

```
#!/bin/bash  
  
keypair_user=builder  
openrc_conf=/home/ubuntu/builder/openrc.sh  
echo "Configuring Nova Client $openrc_conf"  
source $openrc_conf  
echo 'Checking that nova commands can be executed'  
nova list  
  
instance_name=$1  
if [ -z $instance_name ]; then  
    echo 'The name of the instance can not be empty'  
    exit 1  
else  
    echo "The instance name is $instance_name"  
    instance_id=$(nova list | awk -F'|' '/$instance_name/ { print \$2 }')  
    if [ -z $instance_id ]; then  
        echo "The Instance Id has not been found by the name given $instance_name"  
        nova image-list  
        exit 2  
    else  
        image_id=$(nova image-list | awk -F'|' '/$instance_name/ { print \$2 }')  
        echo "Deleting old image $image_id"  
        nova image-delete $image_id  
        nova image-create $instance_id $instance_name  
        saving=true  
        while [ "$saving" = "true" ]; do  
            state=$(nova image-list | awk -F'|' '/$instance_name/ { print \$4 }' | tr -d ' ' )  
            if [ "$state" = "ACTIVE" ]; then  
                saving=false  
            else  
                echo "The state of the instance is $state and $saving"  
                sleep 60  
            fi  
        done  
    fi  
fi  
fi
```

An issue problem with the implementation was the snapshots took too long to start and stop.

At the beginning of the instance management implementation the times that the instances took was around a minute, but as the experiment started and the number of instances started to grow the time the snapshot took increased. The highest increase was when Trackers was installed to allow the building, the amount of hard drive needed to slow the time it took the instances to start.

It was not sure if it was for a lack of resources on the cloud or if the network was is not prepared to handle the network traffic but the result was the it took on average fifteen minutes for the snapshot to start.

At the same time it was measuring when there were stops in the instances. To snapshot an instance and store it on the database usually took around 15 minutes. After all the instance management, the time that the builds took was 30 minutes longer in each instance. This was clearly not an improvement. After that it was not possible anymore to start and stop instances. It took 2 weeks "to resolve" the problem and the solution was to reset the controller of the cloud so it had to delete from the instances and start from scratch again. After that the error came back so it was obvious that the cloud was not stable enough to manage snapshots.

Each time the controller is reset the snapshots also have to be reset. Once there is an error with the snapshot, the solution is to delete snapshot and start again. It is not possible to repeat all the work each time as something goes wrong with this approach. Now it is possible to start instances, to take a snapshot and stop the instance but it is not possible to start that snapshot anymore. That issue was open for longer time and the manager of the cloud could not find a solution so the implementation through snapshot management was not possible maybe in the future when the cloud comes more stable this will be a possible option.

Making the assumption that the cloud with snapshot management worked smoothly with the implementation, but the results were not hopeful. The builds were taking longer due to the time of start and stopping instances. The whole point of the build system was to improve the speed of the build so it can be done more often. If a snapshot takes 15 minutes to start and another 15 minutes to be stored the extra time added to the build of each individual project would be +30 minutes for each build. Those results were not favourable at all. So depending on the duration of the build it may or may not be worth it. If it is the case where the builds takes more than an hour it would be worthwhile to use

snapshot management but for the rest of the builds where the build time is not over 20 minutes or so then is not worth it because it will take twice the normal time to build.

4.6.3 Plugin to start and stop instances form the cloud

Another approach taken was to manage the instances from a Jenkins plugin which was written in Java to manage the creating and initialisation of instances in the cloud. The problem with this approach was that the client used to send commands to the cloud didn't seem to be compatible with the plugin. The error was the credentials of the cloud were lost when connection to the cloud through the nova client.

Some attempts to get support were made by writing questions in blogs of developers who encountered the same issues but the plugin has been created by a company so the support was not available on free basis. This basically means that the plugin can be used but there is no support. So the guy in the forum couldn't really help the research.

This approach was dropped due to the lack of support. Here is the link of the blog. <http://www.tikalk.com/alm/managing-private-dev-cloud-using-openstack-chef-jenkins>

“Moving to continuous integration on the cloud”

The screenshot shows the Jenkins configuration page for a Cloud (JClouds) provider. The configuration includes the following fields:

- Profile:** JCloud-OpenStack
- Provider Name:** nova
- End Point URL:** http://172.16.4.11:5000/v2.0/
- Max. No. of Instances:** 60
- Retention Time:** 30
- Identity:** builders:builder
- Credential:** [Redacted]
- RSA Private Key:** -----BEGIN RSA PRIVATE KEY-----
MIIIEogIBA... [Redacted] ...-----END RSA PRIVATE KEY-----
- Public Key:** ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQ... [Redacted] ...

At the bottom of the configuration, there are buttons for "Save" and "Apply", and an "Advanced..." link.

4.6.4 Issues with the cloud

The main issue with the cloud in Altobridge is that it is a private cloud. Most of the issues with this are the traffic congestion and cloud management.

Another problem with the cloud is that it needs maintenance and it needs people assigned to working on the cloud as if a database.

But the main issue was with the management script to start and stop instances. Some of the snapshots were not possible to be started and the only error message that could be seen was 'Error'. It didn't give any further information on the message it would just not start the instance. Sometimes the solution of the manager was to manipulate the table with the instances in the database of the cloud and delete them by using SQL statements.

Another problem found was not being able to log into the cloud due to an error in the login page. There was no feedback from the instance just an error message with python trace back that was sent to the cloud administrator to fix it.

4.6.5 Issues with the python system

The python system is a build system that has been made by Altobridge. The main purpose of this system is to create Debian packages which will be sold as products on top of the hardware that Altobridge sells as well.

The python system compiles all the projects that are part of a product and package them into an Ubuntu version so that when you install it on the box all the projects are there installed and ready to use.

The system is very good in the way that it allows the package projects automatically so by running the command from the command lines allow the user to release a new version of the software. However, is not as simple as it seemed at the beginning, actually it can be very complicated. It is made of hundreds of classes which interact with each other, creates the files directories and structure needed to create the Debian package.

This system was the real challenge as there were a number of issues with the errors that need to be debugged with the help of J Alan Brogan (The designer of the system) and Martin Grealish (Developer in the system). A large number of changes there were in the code would make the system work. The main problem was that the system was not created with parallel building on mind. It was completely set out to work in the opposite way, the purpose was to do things one by one and make sure only one process is running at a time so when the development started a number of issue arose and they were difficult to deal with.

There is not enough time to change a huge build system as Trackers so the approach taken was to fix only the main errors that block the builds happening, one by one until the build is successful.

5.0 Data Collection

To analyse the time of the builds I got access to the logs folder where all the logs of the last builds are stored. To obtain the times from there I ran the next command:

```
ssh user@ie-build-3 'grep "build took" /mnt/releases/building/logs/*.log' > times.txt
```

This command filters the lines that says ‘build took’ that where the time that the build took can be found and put them in a file called times.txt.

The result was a file filled with hundreds of lines with the final time of the build. The next step is to pass them to a spreadsheet where they can be analysed.

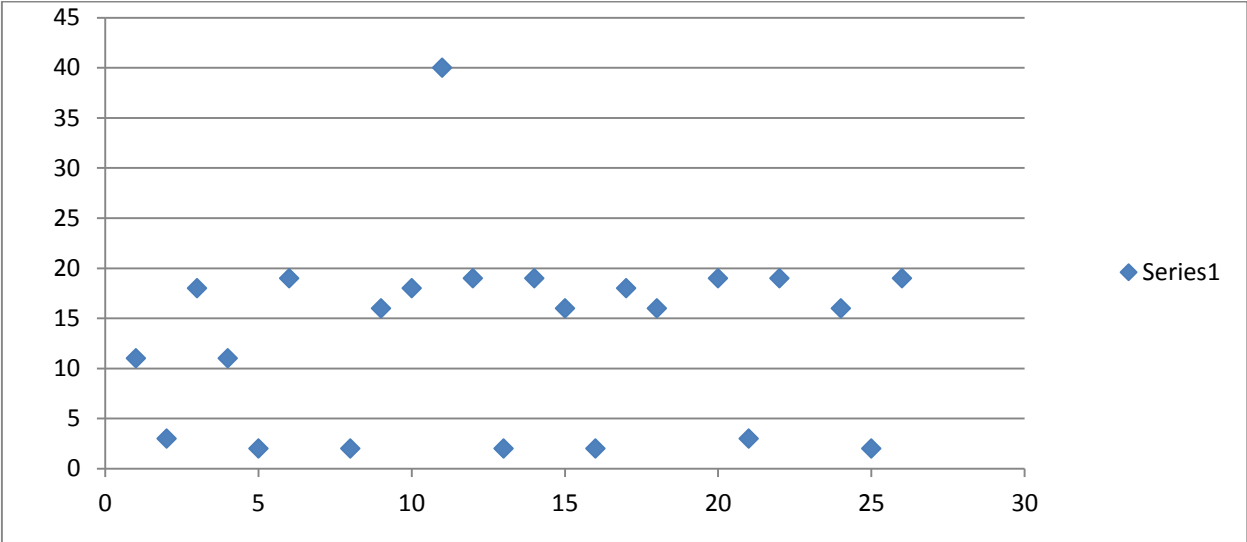
DATE-altoPod-1.0.RC37-Ubuntu10.04.build.log:	at	11:12	took	11	minutes	21.17	seconds)
DATE-altoPod-1.0.RC37-Ubuntu10.04.build.log:	at	11:16	took	3	minutes	0.68	seconds)
DATE-altoPod-1.0.RC37-Ubuntu10.04.build.log:	at	11:35	took	18	minutes	56.13	seconds)
DATE-altoPod-1.0.RC38-Ubuntu10.04.build.log:	at	15:14	took	11	minutes	5.55	seconds)
DATE-altoPod-1.0.RC38-Ubuntu10.04.build.log:	at	15:17	took	2	minutes	51.3	seconds)
DATE-altoPod-1.0.RC38-Ubuntu10.04.build.log:	at	15:36	took	19	minutes	1.35	seconds)
DATE-altoPod-2.0.RC13-Ubuntu10.04.build.log:	at	10:26	took			39.91	seconds)
DATE-altoPod-2.0.RC13-Ubuntu10.04.build.log:	at	10:29	took	2	minutes	20.93	seconds)
DATE-altoPod-2.0.RC13-Ubuntu10.04.build.log:	at	10:46	took	16	minutes	24.08	seconds)
DATE-altoPod-2.0.RC13-Ubuntu10.04.build.log:	at	11:06	took	18	minutes	27.26	seconds)
DATE-altoPod-2.0.RC14-Ubuntu10.04.build.log:	at	16:04	took	40	minutes	31.23	seconds)
DATE-altoPod-2.0.RC34-Ubuntu10.04.build.log:	at	17:32	took	19	minutes	6.16	seconds)
DATE-altoPod-2.0.RC34-Ubuntu10.04.build.log:	at	17:35	took	2	minutes	57.52	seconds)
DATE-altoPod-2.0.RC34-Ubuntu10.04.build.log:	at	17:55	took	19	minutes	36.83	seconds)
DATE-altoPod-2.0.RC5-Ubuntu10.04.build.log:	at	16:11	took	16	minutes	12.72	seconds)
DATE-altoPod-2.0.RC5-Ubuntu10.04.build.log:	at	16:15	took	2	minutes	48.61	seconds)
DATE-altoPod-2.0.RC5-Ubuntu10.04.build.log:	at	16:34	took	18	minutes	50.76	seconds)
DATE-altoPod-2.0.RC6-Ubuntu10.04.build.log:	at	18:36	took	16	minutes	29.99	seconds)
DATE-altoPod-2.0.RC6-Ubuntu10.04.build.log:	at	18:38	took			59.48	seconds)
DATE-altoPod-2.0.RC7-Ubuntu10.04.build.log:	at	14:46	took	19	minutes	24.99	seconds)
DATE-altoPod-2.0.RC7-Ubuntu10.04.build.log:	at	14:49	took	3	minutes	0.13	seconds)
DATE-altoPod-2.0.RC7-Ubuntu10.04.build.log:	at	15:09	took	19	minutes	46.12	seconds)
DATE-altoPod-2.0.RC7-Ubuntu10.04.build.log:	at	15:10	took			32.14	seconds)
DATE-altoPod-2.0.RC8-Ubuntu10.04.build.log:	at	03:43	took	16	minutes	15.07	seconds)
DATE-altoPod-2.0.RC8-Ubuntu10.04.build.log:	at	03:46	took	2	minutes	57.3	seconds)
DATE-altoPod-2.0.RC8-Ubuntu10.04.build.log:	at	04:06	took	19	minutes	54.81	seconds)

“Moving to continuous integration on the cloud”

DATE-altoPod-2.0.RC8-Ubuntu10.04.build.log:	at 04:07	took	34.91	seconds)
---	----------	------	-------	----------

Once all the data is formatted in an excel file the information that is relevant for this research will be selected. The product that has been built for this experiment is called Altopod so the times to look at are the times of the last builds of the Altopod product:

Next step is to generate a graph based on those times:

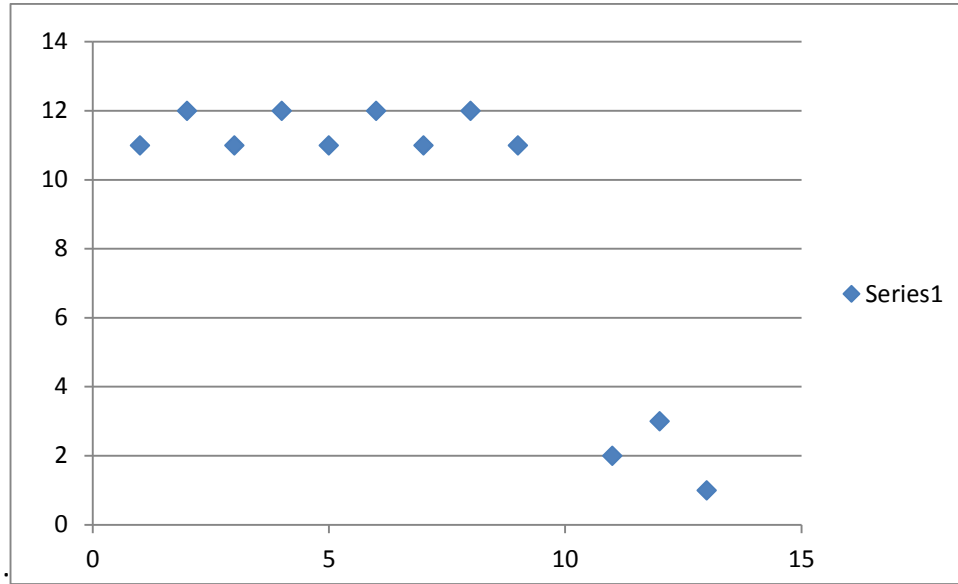


The next data will be the times of the build done on Jenkins. These times we fetched from the Jenkins GUI straight away and put them in excel in appropriate format.

“Moving to continuous integration on the cloud”

Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2013	11	minutes	50	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2014	12	minutes	10	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2015	11	minutes	40	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2016	12	minutes	15	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2017	11	minutes	55	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2018	12	minutes	3	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2019	11	minutes	35	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2020	12	minutes	15	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2021	11	minutes	34	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2022			56	seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2023	2	minutes		seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2024	3	minutes		seconds
Jenkins/DATE-altoPod-2.0.RC17-Ubuntu10.04.build.log:	at	24/24/2025	1	minutes		seconds

A graph generated from these times:



Another relevant times for this experiment are:

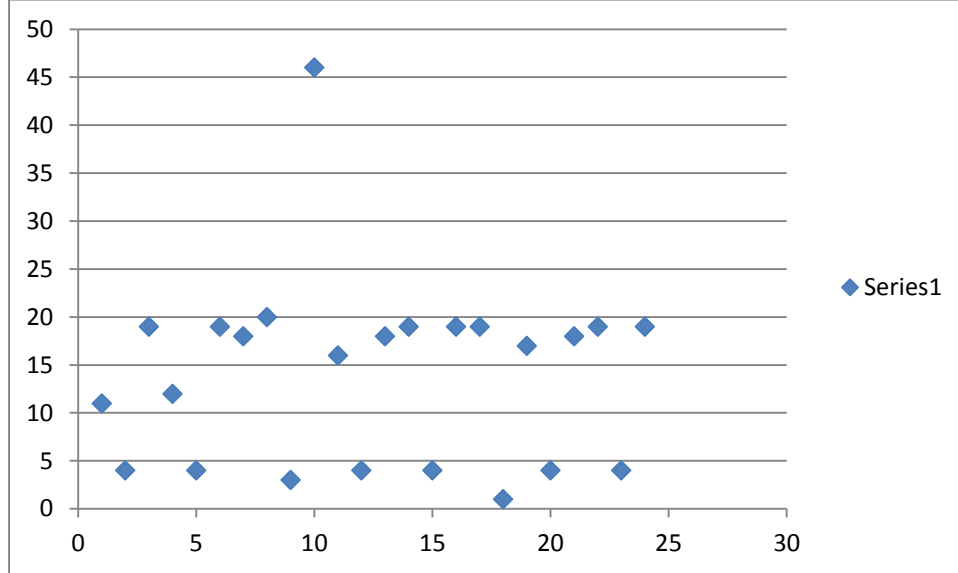
To build the product AltoGateway.

“Moving to continuous integration on the cloud”

DATE-altoGateway-1.0.RC37-Ubuntu10.04.build.log:#	at	11:52	took	11	minutes	44,89	seconds)
DATE-altoGateway-1.0.RC37-Ubuntu10.04.build.log:#	at	11:57	took	4	minutes	41,21	seconds)
DATE-altoGateway-1.0.RC37-Ubuntu10.04.build.log:#	at	12:17	took	19	minutes	53,11	seconds)
DATE-altoGateway-1.0.RC38-Ubuntu10.04.build.log:#	at	15:56	took	12	minutes	3,15	seconds)
DATE-altoGateway-1.0.RC38-Ubuntu10.04.build.log:#	at	16:00	took	4	minutes	24,92	seconds)
DATE-altoGateway-1.0.RC38-Ubuntu10.04.build.log:#	at	16:20	took	19	minutes	51,25	seconds)
DatE-altoGateway-2.0.RC13-Ubuntu10.04.build.log:#	at	11:49	took	18	minutes	47,26	seconds)
DatE-altoGateway-2.0.RC13-Ubuntu10.04.build.log:#	at	12:18	took	20	minutes	39,24	seconds)
DatE-altoGateway-2.0.RC13-Ubuntu10.04.build.log:#	at	12:25	took	3	minutes	51,19	seconds)
DATE-altoGateway-2.0.RC14-Ubuntu10.04.build.log:#	at	17:40	took	46	minutes	10,8	seconds)
DATE-altoGateway-2.0.RC34-Ubuntu10.04.build.log:#	at	16:43	took	16	minutes	54,42	seconds)
DATE-altoGateway-2.0.RC34-Ubuntu10.04.build.log:#	at	16:48	took	4	minutes	17,68	seconds)
DATE-altoGateway-2.0.RC34-Ubuntu10.04.build.log:#	at	17:07	took	18	minutes	45,84	seconds)
DATE-altoGateway-2.0.RC5-Ubuntu10.04.build.log:#	at	17:00	took	19	minutes	47,1	seconds)
DATE-altoGateway-2.0.RC5-Ubuntu10.04.build.log:#	at	17:05	took	4	minutes	25,83	seconds)
DATE-altoGateway-2.0.RC5-Ubuntu10.04.build.log:#	at	17:25	took	19	minutes	34,4	seconds)
DATE-altoGateway-2.0.RC6-Ubuntu10.04.build.log:#	at	22:27	took	19	minutes	38,08	seconds)
DATE-altoGateway-2.0.RC6-Ubuntu10.04.build.log:#	at	22:29	took	1	minutes	57,83	seconds)
DATE-altoGateway-2.0.RC7-Ubuntu10.04.build.log:#	at	13:59	took	17	minutes	4,46	seconds)
DATE-altoGateway-2.0.RC7-Ubuntu10.04.build.log:#	at	14:03	took	4	minutes	22,44	seconds)
DATE-altoGateway-2.0.RC7-Ubuntu10.04.build.log:#	at	14:22	took	18	minutes	27,71	seconds)
DATE-altoGateway-2.0.RC8-Ubuntu10.04.build.log:#	at	04:36	took	19	minutes	48,47	seconds)

“Moving to continuous integration on the cloud”

DATE-altoGateway-2.0.RC8- Ubuntu10.04.build.log:#	at 04:41	took 4	minutes	35,55	seconds)
DATE-altoGateway-2.0.RC8- Ubuntu10.04.build.log:#	at 05:01	took 19	minutes	47,49	seconds)



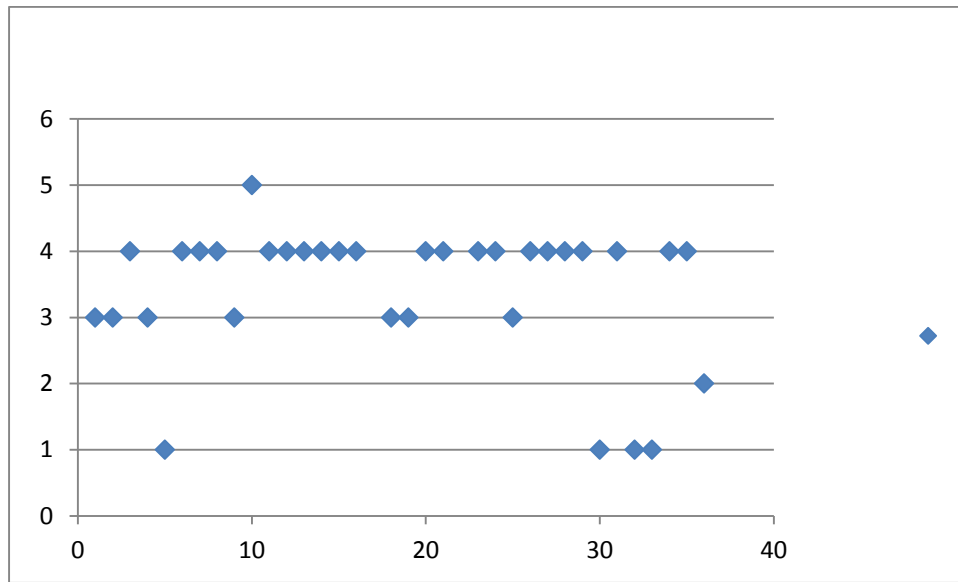
The project CDM

CDM-1.0.RC13-Ubuntu10.04.build.log:#	at 17:47	took	9	seconds)
CDM-1.0.RC14-Ubuntu10.04.build.log:#	at 15:14	took 3	minutes	
CDM-1.0.RC15-Ubuntu10.04.build.log:#	at 23:32	took 3	minutes	
CDM-1.0.RC16- Ubuntu10.04.build.log:#	at 11:36	took 4	minutes	
CDM-1.0.RC17- Ubuntu10.04.build.log:#	at 17:39	took 3	minutes	
CDM-1.0.RC18- Ubuntu10.04.build.log:#	at 02:00	took 1	minutes	
CDM-1.0.RC19- Ubuntu10.04.build.log:#	at 14:51	took 4	minutes	
CDM-1.0.RC20- Ubuntu10.04.build.log:#	at 17:53	took 4	minutes	
CDM-1.0.RC21- Ubuntu10.04.build.log:#	at 10:56	took 4	minutes	
CDM-1.0.RC22- Ubuntu10.04.build.log:#	at 18:54	took 3	minutes	
CDM-1.0.RC23- Ubuntu10.04.build.log:#	at 17:27	took 5	minutes	21,23 seconds)

“Moving to continuous integration on the cloud”

CDM-1.0.RC24- Ubuntu10.04.build.log:#	at	13:55	took	4	minutes	12,02	seconds)
CDM-1.0.RC25- Ubuntu10.04.build.log:#	at	21:26	took	4	minutes	8,37	seconds)
CDM-1.0.RC26- Ubuntu10.04.build.log:#	at	23:36	took	4	minutes	11,29	seconds)
CDM-1.0.RC27- Ubuntu10.04.build.log:#	at	11:29	took	4	minutes	14,6	seconds)
CDM-1.0.RC28- Ubuntu10.04.build.log:#	at	16:36	took	4	minutes	10,34	seconds)
CDM-1.0.RC29- Ubuntu10.04.build.log:#	at	13:34	took	4	minutes	12,09	seconds)
CDM-1.0.RC30- Ubuntu10.04.build.log:#	at	14:40	took			8,75	seconds)
CDM-1.0.RC31- Ubuntu10.04.build.log:#	at	18:52	took	3	minutes	37,96	seconds)
CDM-1.0.RC32- Ubuntu10.04.build.log:#	at	16:20	took	3	minutes	36,03	seconds)
CDM-1.0.RC33- Ubuntu10.04.build.log:#	at	06:52	took	4	minutes	20,92	seconds)
CDM-1.0.RC34- Ubuntu10.04.build.log:#	at	16:55	took	4	minutes	12,89	seconds)
CDM-1.0.RC34- Ubuntu12.04.build.log:#	at	15:43	took			34,22	seconds)
CDM-1.0.RC37- Ubuntu10.04.build.log:#	at	11:57	took	4	minutes	41,21	seconds)
CDM-1.0.RC38- Ubuntu10.04.build.log:#	at	16:00	took	4	minutes	24,92	seconds)
CDM-2.0.RC17- Ubuntu10.04.build.log:#	at	21:17	took	3	minutes	58,23	seconds)
CDM-2.0.RC17- Ubuntu12.04.build.log:#	at	10:42	took	4	minutes	27,67	seconds)
CDM-2.0.RC1-Ubuntu10.04.build.log:#	at	17:31	took	4	minutes	9,93	seconds)
CDM-2.0.RC3-Ubuntu10.04.build.log:#	at	15:39	took	4	minutes	21,38	seconds)
CDM-2.0.RC4-Ubuntu10.04.build.log:#	at	16:48	took	4	minutes	17,68	seconds)
CDM-2.0.RC4-Ubuntu12.04.build.log:#	at	16:24	took	1	minutes	23,3	seconds)
CDM-2.0.RC5-Ubuntu10.04.build.log:#	at	17:05	took	4	minutes	25,83	seconds)
CDM-2.0.RC5-Ubuntu12.04.build.log:#	at	12:21	took	1	minutes	40,56	seconds)
CDM-2.0.RC6-Ubuntu10.04.build.log:#	at	22:29	took	1	minutes	57,83	seconds)
CDM-2.0.RC7-Ubuntu10.04.build.log:#	at	14:03	took	4	minutes	22,44	seconds)
CDM-2.0.RC8-Ubuntu10.04.build.log:#	at	04:41	took	4	minutes	35,55	seconds)
CDM-2.0.RC8-Ubuntu12.04.build.log:#	at	13:28	took	2	minutes	0,99	seconds)

“Moving to continuous integration on the cloud”



HNBG

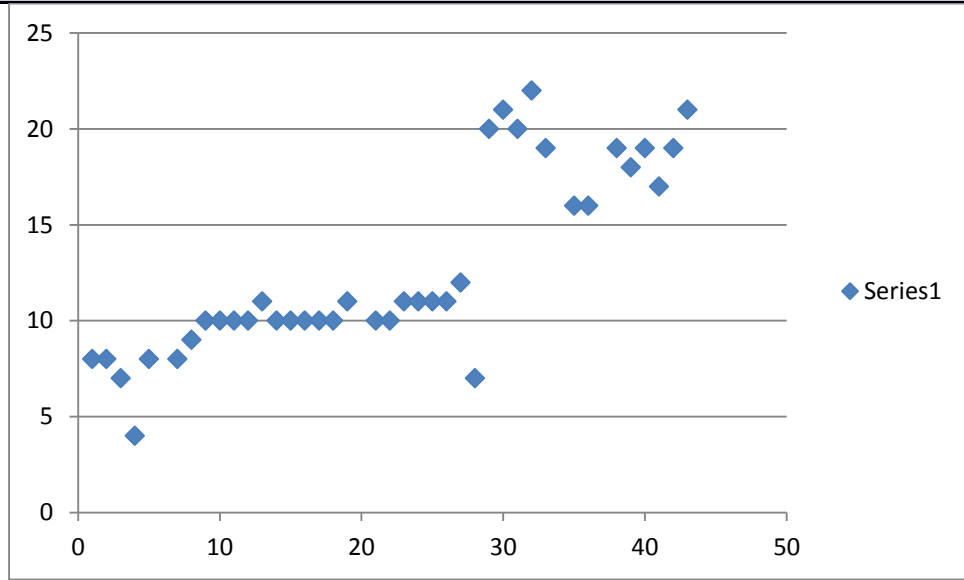
HNBG-1.0.RC10-	Debian10.04.build.log:#	at 08:54	took 7	minutes
HNBG-1.0.RC11-	Debian10.04.build.log:#	at 13:13	took 4	minutes
HNBG-1.0.RC12-	Debian10.04.build.log:#	at 10:05	took 8	minutes
HNBG-1.0.RC12-	Debian10.04.quick.dirty.log:#	at 10:36	took	29 seconds)
HNBG-1.0.RC12-	Ubuntu10.04.build.log:#	at 10:47	took 8	minutes
HNBG-1.0.RC15-	Ubuntu10.04.build.log:#	at 00:02	took 9	minutes
HNBG-1.0.RC16-	Ubuntu10.04.build.log:#	at 10:21	took 10	minutes
HNBG-1.0.RC17-	Ubuntu10.04.build.log:#	at 16:45	took 10	minutes
HNBG-1.0.RC18-	Ubuntu10.04.build.log:#	at 02:42	took 10	minutes
HNBG-1.0.RC21-	Ubuntu10.04.build.log:#	at 13:10	took 10	minutes
HNBG-1.0.RC22-	Ubuntu10.04.build.log:#	at 14:44	took 11	minutes
HNBG-1.0.RC23-	Ubuntu10.04.build.log:#	at 16:49	took 10	minutes 31,82 seconds)
HNBG-1.0.RC24-	Ubuntu10.04.build.log:#	at 14:29	took 10	minutes 33,59 seconds)

“Moving to continuous integration on the cloud”

HNBG-1.0.RC25- Ubuntu10.04.build.log:#	at 22:50	took 10 minutes 48,37 seconds)
HNBG-1.0.RC26- Ubuntu10.04.build.log:#	at 00:02	took 10 minutes 35,69 seconds)
HNBG-1.0.RC27- Ubuntu10.04.build.log:#	at 17:27	took 10 minutes 30,84 seconds)
HNBG-1.0.RC28- Ubuntu10.04.build.log:#	at 17:08	took 11 minutes 22,85 seconds)
HNBG-1.0.RC29- Ubuntu10.04.build.log:#	at 13:44	took 21,13 seconds)
HNBG-1.0.RC30- Ubuntu10.04.build.log:#	at 14:45	took 10 minutes 55,33 seconds)
HNBG-1.0.RC31- Ubuntu10.04.build.log:#	at 18:48	took 10 minutes 54,01 seconds)
HNBG-1.0.RC32- Ubuntu10.04.build.log:#	at 16:16	took 11 minutes 45,51 seconds)
HNBG-1.0.RC33- Ubuntu10.04.build.log:#	at 06:47	took 11 minutes 47,7 seconds)
HNBG-1.0.RC34- Ubuntu10.04.build.log:#	at 16:51	took 11 minutes 37,19 seconds)
HNBG-1.0.RC37- Ubuntu10.04.build.log:#	at 11:52	took 11 minutes 44,89 seconds)
HNBG-1.0.RC38- Ubuntu10.04.build.log:#	at 15:56	took 12 minutes 3,15 seconds)
HNBG-1.0.RC9-Debian10.04.build.log:#	at 12:04	took 7 minutes
HNBG-2.0.RC14- Ubuntu10.04.build.log:#	at 16:49	took 20 minutes 13,8 seconds)
HNBG-2.0.RC14- Ubuntu12.04.build.log:#	at 16:47	took 21 minutes 39,15 seconds)
HNBG-2.0.RC17- Ubuntu10.04.build.log:#	at 21:13	took 20 minutes 4,06 seconds)
HNBG-2.0.RC17- Ubuntu12.04.build.log:#	at 10:37	took 22 minutes 2,79 seconds)
HNBG-2.0.RC1- Ubuntu10.04.build.log:#	at 17:26	took 19 minutes 4,61 seconds)
HNBG-2.0.RC2- Ubuntu10.04.build.log:#	at 14:44	took 42,27 seconds)
HNBG-2.0.RC3- Ubuntu10.04.build.log:#	at 15:35	took 16 minutes 54,19 seconds)
HNBG-2.0.RC4- Ubuntu10.04.build.log:#	at 16:43	took 16 minutes 54,42 seconds)
HNBG-2.0.RC4- Ubuntu12.04.build.log:#	at 12:12	took 0,2 seconds)
HNBG-2.0.RC5- Ubuntu10.04.build.log:#	at 17:00	took 19 minutes 47,1 seconds)

“Moving to continuous integration on the cloud”

Ubuntu10.04.build.log:#									
HNBG-2.0.RC5-									
Ubuntu12.04.build.log:#	at	12:20	took	18	minutes	51,47	seconds)		
HNBG-2.0.RC6-									
Ubuntu10.04.build.log:#	at	22:27	took	19	minutes	38,08	seconds)		
HNBG-2.0.RC7-									
Ubuntu10.04.build.log:#	at	13:59	took	17	minutes	4,46	seconds)		
HNBG-2.0.RC8-									
Ubuntu10.04.build.log:#	at	04:36	took	19	minutes	48,47	seconds)		
HNBG-2.0.RC8-									
Ubuntu12.04.build.log:#	at	13:26	took	21	minutes	45,31	seconds)		

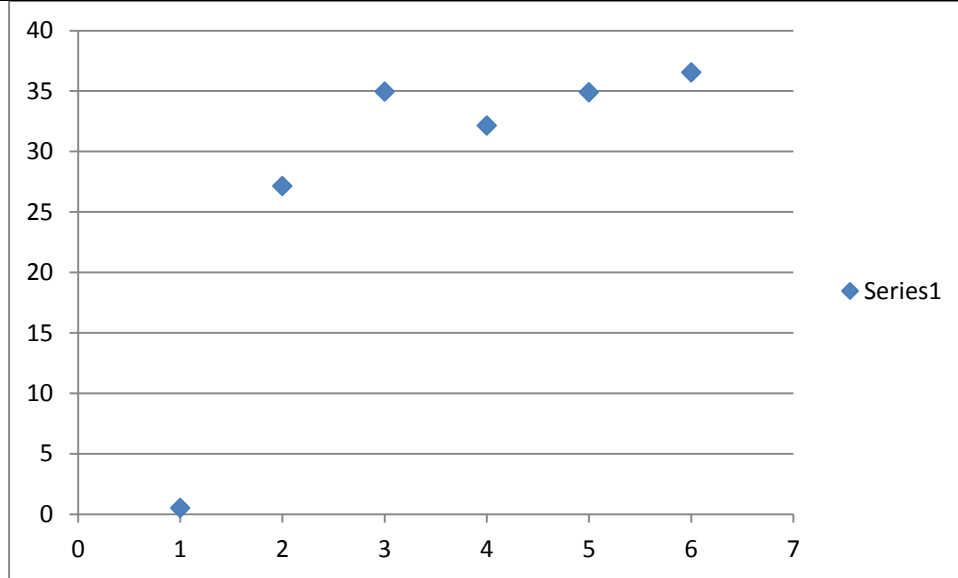


There is a third project called ABC that is used for Altopod and Altogateway. ABC with the cloud implementation takes between 11 and 12 minutes to finish the build. Those times were taken manually while testing the system.

GAP-2.0.RC14-Ubuntu12.04.build.log:#	a	09:4		seconds
	t	9	took	0,52)
GAP-2.0.RC17-Ubuntu10.04.build.log:#	a	19:0		27,1 seconds
	t	8	took	6)
GAP-2.0.RC17-Ubuntu12.04.build.log:#	a	13:3		34,9 seconds
	t	2	took	7)
GAP-2.0.RC7-Ubuntu10.04.build.log:#	a	15:1		32,1 seconds
	t	0	took	4)
GAP-2.0.RC8-Ubuntu10.04.build.log:#	a	04:0		34,9 seconds
	t	7	took	1)

“Moving to continuous integration on the cloud”

GAP-2.0.RC8-Ubuntu12.04.build.log:#	a	15:5	36,5	seconds
	t	1	took	4

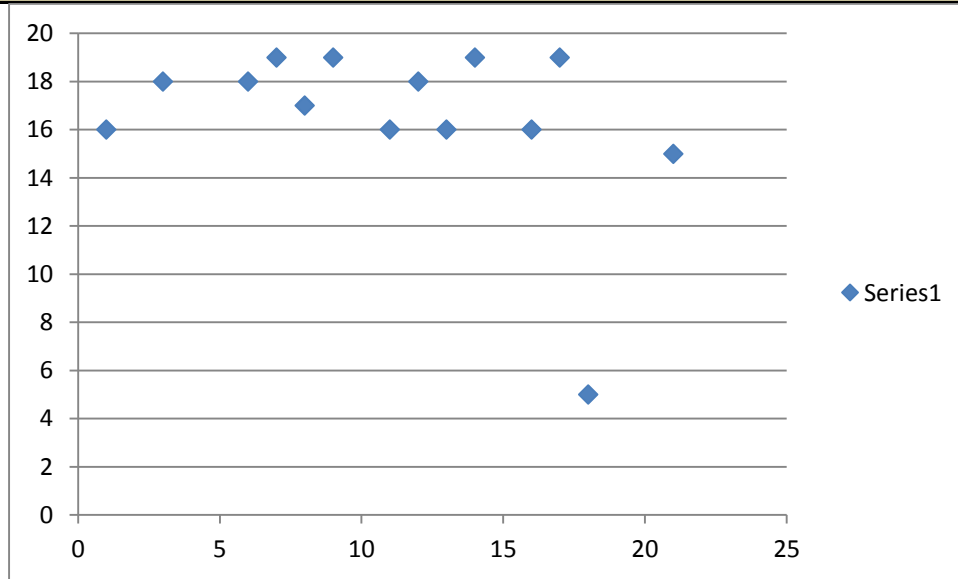


HNBA

HNBA-2.0.RC11-Ubuntu10.04.build.log:#	at	07:25	took	16	minutes	16,69	seconds)
HNBA-2.0.RC13-Ubuntu10.04.build.log:#	at	13:23	took			0,15	seconds)
HNBA-2.0.RC14-Ubuntu10.04.build.log:#	at	15:44	took	18	minutes	25,68	seconds)
HNBA-2.0.RC14-Ubuntu12.04.build.log:#	at	09:49	took			1,87	seconds)
HNBA-2.0.RC15-Ubuntu10.04.build.log:#	at	13:15	took			39,17	seconds)
HNBA-2.0.RC16-Ubuntu10.04.build.log:#	at	16:20	took	18	minutes	45,84	seconds)
HNBA-2.0.RC17-Ubuntu10.04.build.log:#	at	18:46	took	19	minutes	5,58	seconds)
HNBA-2.0.RC1-Ubuntu10.04.build.log:#	at	17:53	took	17	minutes	31,18	seconds)
HNBA-2.0.RC4-Ubuntu10.04.build.log:#	at	17:32	took	19	minutes	6,16	seconds)
HNBA-2.0.RC4-Ubuntu12.04.quick.dirty.log:#	at	17:32	took			41,48	seconds)
HNBA-2.0.RC5-Ubuntu10.04.build.log:#	at	16:11	took	16	minutes	12,72	seconds)
HNBA-2.0.RC5-Ubuntu12.04.build.log:#	at	12:46	took	18	minutes	17,72	seconds)

“Moving to continuous integration on the cloud”

HNBA-2.0.RC6- Ubuntu10.04.build.log:#	at 18:36	took 16 minutes 29,99 seconds)
HNBA-2.0.RC7- Ubuntu10.04.build.log:#	at 14:46	took 19 minutes 24,99 seconds)
HNBA-2.0.RC7- Ubuntu12.04.build.log:#	at 18:13	took 0,54 seconds)
HNBA-2.0.RC8- Ubuntu10.04.build.log:#	at 03:43	took 16 minutes 15,07 seconds)
HNBA-2.0.RC8- Ubuntu12.04.build.log:#	at 15:28	took 19 minutes 41,08 seconds)
HNBA-2.0.RC8- Ubuntu12.04.dirty.log:#	at 17:31	took 5 minutes 6,72 seconds)
HNBA-2.0.RC8- Ubuntu12.04.quick.dirty.log:#	at 17:19	took 40,84 seconds)
HNBA-2.0.RC8- Ubuntu12.04.quick.log:#	at 17:34	took 49,2 seconds)
HNBA-2.0.RC9- Ubuntu10.04.build.log:#	at 06:07	took 15 minutes 58,31 seconds)



RDM

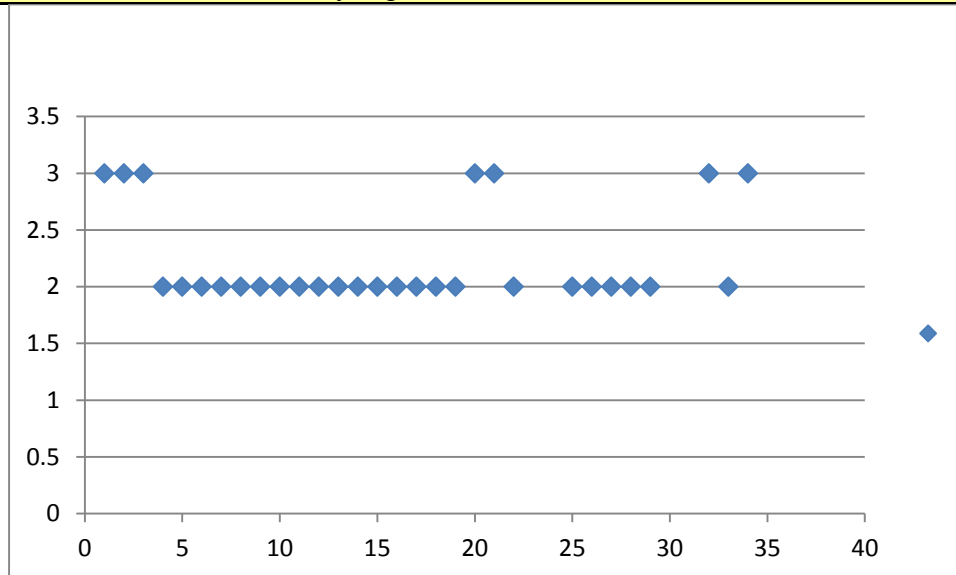
RDM-0.0.RC0- Ubuntu10.04.quick.dirty.log:#	a 18:0	seconds
	t 2	took 0,08)
RDM-1.0.RC14- Ubuntu10.04.build.log:#	a 15:1	minute
	t 7	took 3 s
RDM-1.0.RC15- Ubuntu10.04.build.log:#	a 23:4	minute
	t 0	took 3 s
RDM-1.0.RC16- Ubuntu10.04.build.log:#	a 11:4	minute
	t 5	took 3 s

“Moving to continuous integration on the cloud”

RDM-1.0.RC17-	a	17:4			minute		
Ubuntu10.04.build.log:#	t	5	took	2	s		
RDM-1.0.RC18-	a	02:2			minute		
Ubuntu10.04.build.log:#	t	0	took	2	s		
RDM-1.0.RC20-	a	17:5			minute		
Ubuntu10.04.build.log:#	t	5	took	2	s		
RDM-1.0.RC21-	a	11:0			minute		
Ubuntu10.04.build.log:#	t	1	took	2	s		
RDM-1.0.RC22-	a	18:5			minute		
Ubuntu10.04.build.log:#	t	7	took	2	s		
RDM-1.0.RC23-	a	16:2			minute		seconds
Ubuntu10.04.build.log:#	t	8	took	2	s	2,77)
RDM-1.0.RC24-	a	14:0			minute	13,6	seconds
Ubuntu10.04.build.log:#	t	3	took	2	s	5)
RDM-1.0.RC25-	a	21:2			minute		seconds
Ubuntu10.04.build.log:#	t	9	took	2	s	4,19)
RDM-1.0.RC26-	a	23:3			minute		seconds
Ubuntu10.04.build.log:#	t	9	took	2	s	2,56)
RDM-1.0.RC27-	a	11:3			minute		seconds
Ubuntu10.04.build.log:#	t	4	took	2	s	0,96)
RDM-1.0.RC28-	a	16:4			minute		seconds
Ubuntu10.04.build.log:#	t	1	took	2	s	0,4)
RDM-1.0.RC29-	a	13:4			minute		seconds
Ubuntu10.04.build.log:#	t	2	took	2	s	2,01)
RDM-1.0.RC30-	a	15:0			minute	38,0	seconds
Ubuntu10.04.build.log:#	t	1	took	2	s	6)
RDM-1.0.RC31-	a	18:2			minute	32,5	seconds
Ubuntu10.04.build.log:#	t	8	took	2	s	8)
RDM-1.0.RC33-	a	07:3			minute	53,4	seconds
Ubuntu10.04.build.log:#	t	7	took	2	s	3)
RDM-1.0.RC34-	a	17:1			minute	14,7	seconds
Ubuntu10.04.build.log:#	t	7	took	2	s	5)
RDM-1.0.RC36-	a	18:2			minute		seconds
Ubuntu10.04.build.log:#	t	5	took	3	s	4,37)
RDM-1.0.RC37-	a	11:1			minute		seconds
Ubuntu10.04.build.log:#	t	6	took	3	s	0,68)
RDM-1.0.RC38-	a	15:1			minute		seconds
Ubuntu10.04.build.log:#	t	7	took	2	s	51,3)
RDM-1.0-	a	13:1				14,2	seconds
Ubuntu10.04.quick.dirty.log:#	t	2	took			8)
RDM-2.0.RC14-	a	09:4					seconds
Ubuntu12.04.build.log:#	t	9	took			0,62)
RDM-2.0.RC17-	a	18:4			minute	34,8	seconds
Ubuntu10.04.build.log:#	t	8	took	2	s	8)

“Moving to continuous integration on the cloud”

RDM-2.0.RC17- Ubuntu12.04.build.log:#	a	13:1		minute	54,9	seconds
	t	2	took	2 s	5)
	a	17:5		minute	43,7	seconds
RDM-2.0.RC1-Ubuntu10.04.build.log:#	t	6	took	2 s	5)
	a	17:3		minute	57,5	seconds
RDM-2.0.RC4-Ubuntu10.04.build.log:#	t	5	took	2 s	2)
	a	16:1		minute	48,6	seconds
RDM-2.0.RC5-Ubuntu10.04.build.log:#	t	5	took	2 s	1)
	a	12:4			27,9	seconds
RDM-2.0.RC5-Ubuntu12.04.build.log:#	t	6	took		6)
	a	18:3			59,4	seconds
RDM-2.0.RC6-Ubuntu10.04.build.log:#	t	8	took		8)
	a	14:4		minute		seconds
RDM-2.0.RC7-Ubuntu10.04.build.log:#	t	9	took	3 s	0,13)
	a	03:4		minute		seconds
RDM-2.0.RC8-Ubuntu10.04.build.log:#	t	6	took	2 s	57,3)
	a	15:3		minute	16,7	seconds
RDM-2.0.RC8-Ubuntu12.04.build.log:#	t	1	took	3 s	8)
	a	17:3			47,0	seconds
RDM-2.0.RC8-Ubuntu12.04.dirty.log:#	t	2	took		7)



6.0 Analysis

The data collected from this project was primarily concerned with the Altopod project. After implementation on Jenkins, the product can be built on Jenkins is an Altopod.

The comparison of data was done between the Altopod built, the python system and the altopod built with the Jenkins system. A comparison of the two approaches can be done as well taking a look with the snapshot implementation and the non-snapshot implementation.

To understand the time data of the builds first it is necessary to know how many projects is an altopod made to know how much time it needs to be build.

First of all, an Altopod is made of four projects which are HNBA, RDM, ABC and GAP. Previous, to the implementation of those four builds need to be built independently to make sure they work effectively. During that independent build the time for each of the projects were:

For HNBA were around 2 minutes. For RDM, 2 minutes. For GAP around 30 seconds and finally for ABC 12 minutes. Once these 4 building were safely being built then it was possible to run all four together with the release script. The time with the release script was 12 minutes.

After analysing the results, the first conclusion that can be made is that the builds take the time of the longest build to be made. In the case of Atopod the longest jobs was the ABC which was 12 minutes.

When looking at the logs of the last builds on Altopod the time that it takes on average is 19 minutes.

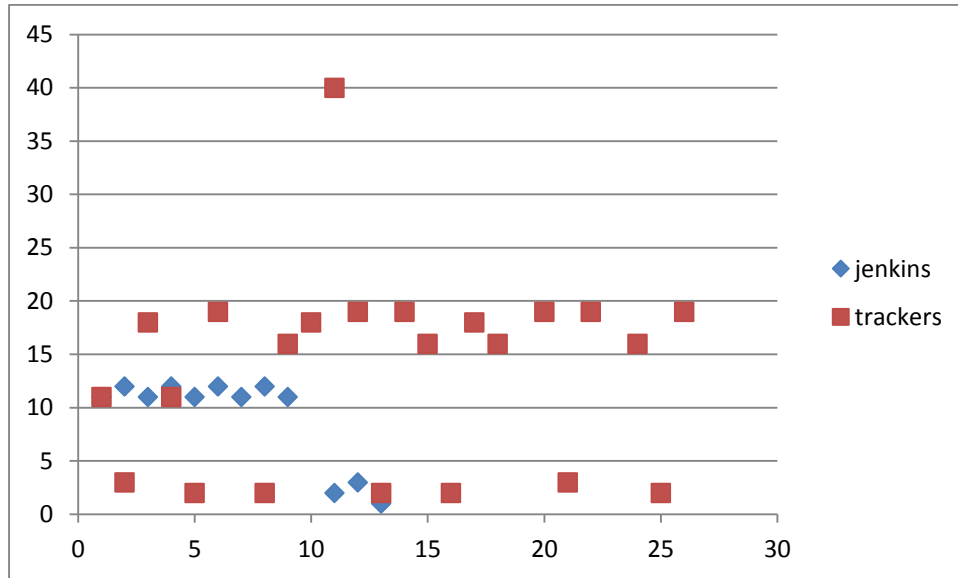
The results can be seen in the next graphs:

In the second graph the values that are boundaries and are being removed due to the lack of relevance. The results indicated that a build which takes a lot less time to work compared to another means that it failed for some unknown reason. If a build takes longer to achieve than the build could get stuck on a script and it needs to stopped or killed.

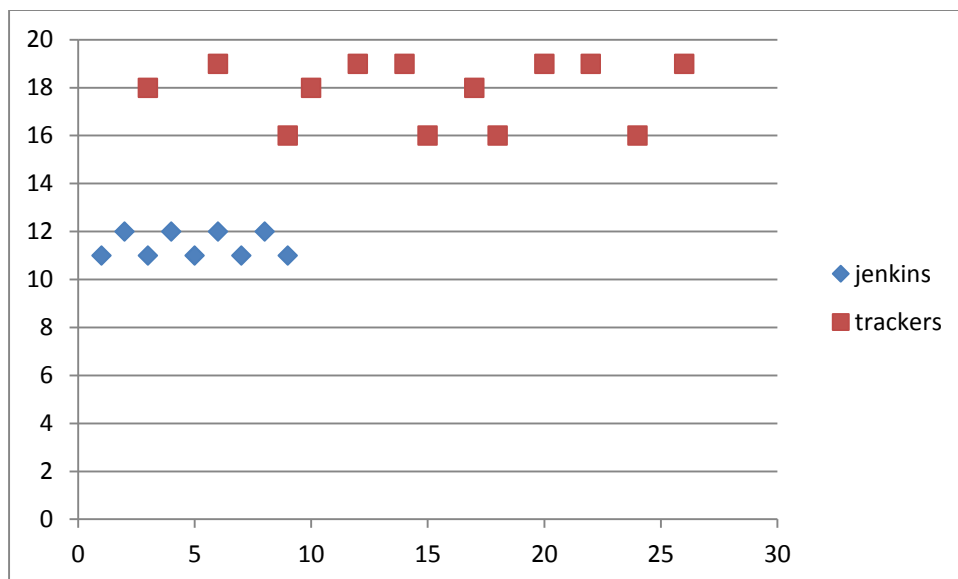
“Moving to continuous integration on the cloud”

Typically after executing a build many times means the more repetitive values that are generated so it can be assumed that the values are valid. The boundaries are considered wrong values and some strange values in the middle will be considered wrong as well. At the end it was decided to keep the two most common values of both graphics, which is demonstrated in the second graphic.

Both Together



Data without boundary values



“Moving to continuous integration on the cloud”

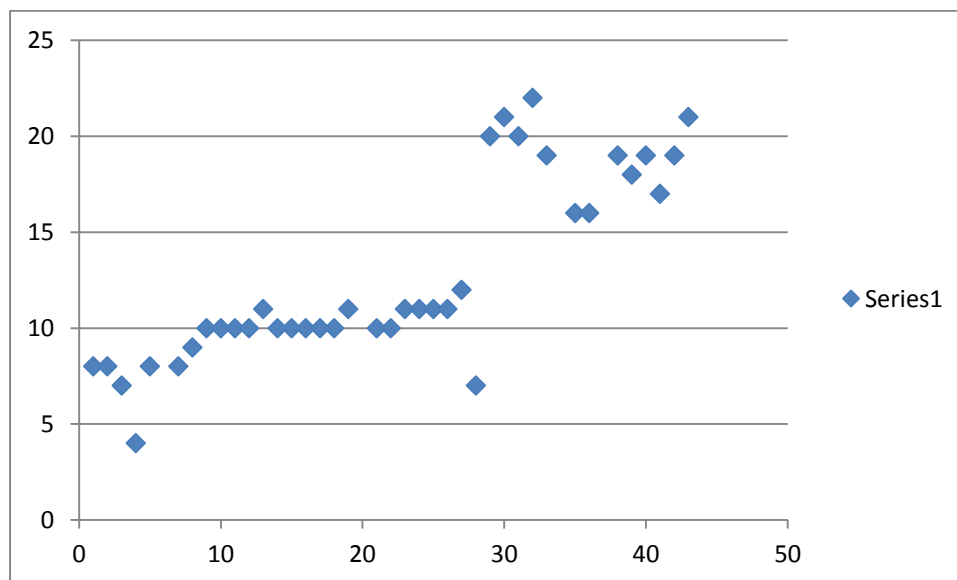
In the graphics it demonstrates how the times in the Jenkins are shorter than the times of the python build system in Altobridge. Furthermore, the times in Jenkins are more constant and stable. The difference between times is shorter due to the fact that the jobs are done in different machines. In the normal build system all the builds are done on one machine so the amount of time that the machine takes can change depending on what processes are running at that moment or if that machine is being used for another task or job. The machine can be more sensitive to other processes because it will need as much resources as it can get.

In the cloud the machines are created to do that specific job so that it can be sure that the machines are 100% available to run the builds. There are no third processes involved in making the build slower.

The other results gathered from the logs are the times from other projects that are built in other product, for example, the Altogateway.

The Altogateway product is made of three projects that are similar to the ones that are part of the Altopod, those projects are HNBG, CDM and ABC.

One mention has to be done with HNBG, in the times token from the logs there is a difference between HNBG 1.0 and HBNG 2.0.



On the graphic the times that HNBG takes are segregated in two. The first part takes an average around every 10 minutes and the second part takes on average around every 20 minutes. After analysing these results it was clear that there was a difference, but the

“Moving to continuous integration on the cloud”

difference here is on the version. On HNBG 1.0 the time to build was 10 minutes but in HNBG 2.0 the time to build was increased to 20 minutes. The times from 1.0 to 2.0 has increased on 10 minutes so it has now doubled from what it was. The cause is that they included a new feature that required a lot added libraries to include in the project. When doing the command 'makefile' it takes longer.

“Moving to continuous integration on the cloud”

The times for those projects on average are:

For HNBBG 1.0 10 minutes or HNBBG 2.0 20 minutes. For CDM 4 minutes and for ABC 12 minutes.

If the same experiment would be done with an Altogateway which build times are higher than altopod times the improvement of the time it would be bigger.

With the Altopod experiment the final time of the build will be equals to the time of the longest project to build. Then on Altogateway with the times extracted from the log the next conclusion can be made:

The necessary projects to build an AltoGateway are: HNBBG 1.0 + CDM + ABC.

The time that will take on trackers will be: 10 Min + 4 Min + 12 Min so 26 Min.

The time that will take in the cloud will be: 10 Min or 4 Min or 12 Min so 12 Min.

It is a saving of the 55% of the time when building an Altogateway.

If the version is HNBBG 2.0 the times will be as it follows:

The necessary projects to build an AltoGateway are = HNBBG 2.0 + CDM + ABC

The time that will take on trackers will be: 20 Min + 4 Min + 12 Min so 36 Min

The time that will take in the cloud will be: 20 Min or 4 Min or 12 Min so 20 Min

It is a saving of the 45% of the time when building an Altogateway.

The implementation of the AltoGateway product was not possible due to the lack of time but if the same implementation would be done to the Altogateway we could get a substantial increase the time saved while on projects.

7.0 Conclusions

The initial question was to find out how the cloud can improve the continuous integration system in the company. The assumption was that the cloud was necessary to speed up the builds so that it would be possible to build more often. At the end it was a matter of implementing a continuous integration system in a company that needs the cloud.

The cloud was a requirement and not simply a technology to use. The problem faced at the beginning was how to build all the products of a company in a system. This system existed already; however, the problem grew until it become more about how to build all these projects in different Ubuntu versions and on parallel so the process of building could become fast enough to implement the practise of continuous integration.

Some of the requirements stated at the beginning were:

1. To provide a central system to manage all the different build
2. To use the resources of a company in an efficient way by cloud management
3. To make the detection of errors faster
4. To set a number of practises that helps when developing in software
5. To reduce the risk of the overall project
6. To provide frequent feedbacks about the state of the product

The first requirement was: *To provide a central system to manage all the different build.*

This requirement was accomplish by using the Jenkins interface. Jenkins has been integrated with the python system so both work together. Now Jenkins seems more intuitive and easy to use because it displays all the build information in the dashboard. It offers other metrics as the state of the build, which means if the last code committed broke the build or not, it can incorporate code metrics as a number of lines or bugs found and fixed.

In this experiment the central system was a success for managing build, it is more intuitive than the python system so this requirement was accomplish.

The second requirement: *To use the resources of a company in an efficient way by cloud management:*

This requirement was not accomplished due to external causes. The requirement of integrating continuous integration with *Trackers* was to handle around 50 different



instances with 50 different configurations which may involve six different operating system versions. Due to limitation in the cloud it was not possible to get 50 instances running at the same time. Instead, the implementation of the snapshots was done to deal with all those configurations so it was possible to switch off the instances that are not necessary and store them in a database. However, that was not possible either. The first attempt to start snapshots was not good enough, and then it was not possible to start instances after the snapshot was taken. The consequence of such as action was that if an instance was stopped it wouldn't be possible to start that instance again, so all the work done up to that point was lost and the controller of the cloud had to be reset.

The second attempt after the controller was reset was more hopeful. It was possible to start and stop instances from snapshots but the times were very slow. It took 15 minutes on average to start an instance and another 15 minutes to store it and delete it. With these times it was not feasible to go on with continuous integration because the main requirement of speed up builds was not accomplished.

After two weeks the cloud stopped working again, it was not possible to start instances from snapshots. For some companies to use the amazon cloud and pay for usage might seem expensive but it saves a lot of work on maintenance and network traffic. To take a few computers and install the cloud controller on it is easy but to try to implement your own cloud and try to get the same performance as in amazon without paying a price is not realistic.

First of all there are many things that affect the cloud and can slow down the times of the instances. One of them was the network traffic, some days due to testing operation or simply overuse of the network it was very hard to manage the instances. Even using ssh to the instances was a lengthy process and was hard to manage them. The cloud also needed maintenance as every time the space in the cloud has to be defragmented and fragmented again. During those times some running instances might not work and it might not be possible to start new ones.

The research would indicate that it would be more beneficial and more cost efficient to buy three physical machines and configure them on the network then set them as build machines. This solution would be simpler, faster and cheaper and there is no need for cloud maintenance. Cloud technology should be implemented within larger companies with very specific requirements, high workloads and very efficient networks. For small companies the benefit of using cloud technology does not add up when consideration is given to benefit of implementing and maintaining their own private cloud. It would be



“Moving to continuous integration on the cloud”

cheaper to hire instances in amazon then the company would not need to deal with maintenance and network traffic issues.

The third requirement: *To make the detection of errors faster.*

Since the only product that has been build has been the Altopod it is difficult to measure if it is efficient or not. The more frequently it is used the easier it would be to spot an error or bug. There was not enough time to measure this requirement because the system has to be working and monitored over a few months to see if there is really an improvement in errors spotted.

The fourth requirement: *To set a number of practises that helps when developing in software.*

This requirement was challenging. Good practice would suggest that the builder needs to know each project and how the built being developed to have a true understanding of what is happening. In practice it was found that a lot of the time developer were concentrating on their project and didn't have any awareness of the status on other projects their colleagues were working on. Once the system is implemented and running there should be a meeting with the developers to show them how the new system will work and why new practises would apply to everyone, otherwise there is a risk that the same problem will arise again and the builder will be the only person who knows the status of the system.

The fifth requirement: *To reduce the risk of the overall project.*

This is a consequence of continuous integration but by building faster and continuously the project gets stronger, more reliable and stable. The build of the product is configured so the product can be built any time and anybody can check the status a see how it looks. The consequence of this is that there could be a case where is not sure how to measure the stability of a project by the number of builds. Jenkins does this by accounting the number of failed builds of the last 20 builds. If the percentage of good builds is high then is stable, otherwise is not stable.

The sixth requirement: *To provide frequent feedback.*

The requirement was very difficult to implement, many challenges were faced when trying to find a solution. In the end, due to time constraints this requirement was not fulfilled.

Cloud Management

The research would indicate that implementing a private cloud within a small company is very challenging and time consuming and serious consideration should be given whether it is a feasible option. If there were a lot of resources (machines) set in place, along with a



decided team of developers assigned to deal with the management and maintenance of the cloud as well improvements with the network to address traffic within the system., then it could be an option. However, the amount of resources needed to get the cloud properly working is not worth all the investment for a small company. It is more cost effective and less challenging to avail of an account such as Amazon and pay for it.

Python Build System

After experience of working with the python system it would be recommended to use open source tolls that achieve the same objective with less effort. The python system is a great idea in theory but there are too many challenges and it ends up being too complicated to maintain. The problem is not with the build system, the build system is a very good system but the idea of having your own system to build your own projects is not practical. The problem is that the complexity of the system is such that for any change that needs to be done the system needs to know, quite in-depth, all the classes and what will be the results. The problem with creating *private* software for your own purposes is that only the people who created know exactly how it works. If that person decides to leave the company it is very difficult hard to get someone with the same knowledge and familiarity with that system to carry on the work in the same fashion.

A strong recommendation would be to use open source tools with great support such as Maven to build projects. The reason the system was created was to isolate the process of developing the product and make it independent from the process of building and packaging the product. So it would be easier to deploy to the customer.

The developers only know about their own projects so the responsibility is placed on the builder to make sure they work when the projects are packaged together. One suggestion is that it would be easier for everyone involved if the projects were built on a common platform such as Maven so the task of the builder would be considerably simpler. Furthermore, systems such as Maven can be easily integrated with other tools and subversion or Jenkins can be integrated at a later stage when new methodologies on continuous integration make it easily adopted.

8.0 References

- www.everymanit.com*. (2010, 12 17). Retrieved 12 1, 2012, from *www.everymanit.com*:
<http://www.everymanit.com/2010/12/18/introduction-to-cloud-computing/>
- Binning, D. (2009, 4 24). *www.computerweekly.com*. Retrieved 12 1, 2012, from *www.computerweekly.com*:
<http://www.computerweekly.com/news/2240089111/Top-five-cloud-computing-security-issues#2>
- Brien, A. O. (2010). *Career Portal*. Retrieved 10 20, 2012, from Career Portal:
http://www.careersportal.ie/sectors/sector_org_answer.php?client_id=42&group_name=Questions+about+the+sector&sq_group_id=1#.UIK_sobYH3w
- Dialogic.com. (2010). *Introduction to cloud computing*. Montreal: Dialogic Corporation.
- Duvall, P. M. (2007). *Continuous Integration: Improving Software Quality and Reducing Risks*. Crawfordsville, Indiana, US: Addison–Wesley.
- Fowler, M. (2006, 05 01). *www.martinfowler.com*. Retrieved 10 14, 2012, from *www.martinfowler.com*:
<http://martinfowler.com/articles/continuousIntegration.html>
- Fundation, F. S. (2004). <http://gcc.gnu.org>. Retrieved 12 12, 2012, from <http://gcc.gnu.org>: <http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>
- Gregory. (2012, 9 28). <http://www.cloudtweaks.com>. Retrieved 12 1, 2012, from <http://www.cloudtweaks.com>: <http://www.cloudtweaks.com/2012/09/top-5-cloud-computing-trends-of-the-future/>
- Hart, S. (2012). *Information about Altobridge Cloud*. Tralee: Altobridge Ltd.
- Knorr, E. (2008, 7 4). *www.infoworld.com*. Retrieved 12 1, 2012, from *www.infoworld.com*: <http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031>
- Kubie, E. C. (1994). *Recollections of the first software company*.
- onlinetech. (2012). *www.onlinetech.com*. Retrieved 12 1, 2012, from *www.onlinetech.com*: <http://www.onlinetech.com/resources/e-tips/disaster-recovery/benefits-of-disaster-recovery-in-cloud-computing>

- openstack. (2010). <http://www.openstack.org/>. Retrieved 12 08, 2012, from <http://www.openstack.org/>: <http://docs.openstack.org/trunk/openstack-compute/admin/content/what-is-openstack.html>
- Perks, M. (2006, August 10). *IBM*. Retrieved 10 20, 2012, from IBM: http://www.ibm.com/developerworks/websphere/library/techarticles/0306_perks/perks2.html
- Rob Lovell, C. (2010). *www.thinkgrid.com*. Retrieved 12 1, 2012, from www.thinkgrid.com: <http://www.thinkgrid.com/cloud-computing.aspx>
- Rouse, M. (2010, December). *searchcloudcomputing.techtarget.com*. Retrieved 12 1, 2012, from searchcloudcomputing.techtarget.com: <http://searchcloudcomputing.techtarget.com/definition/cloud-computing>
- Software, F. (2009, Spring). <http://www.opencloudmanifesto.org/>. Retrieved 12 1, 2012, from <http://www.opencloudmanifesto.org/>: <http://www.opencloudmanifesto.org/opencloudmanifesto1.htm>
- Staplin, G. P. (2005, 8 31). <http://user.xmission.com/>. Retrieved 12 12, 2012, from <http://user.xmission.com/>: http://user.xmission.com/~georgeps/documentation/tutorials/compilation_and_makefiles.html
- Techonology, N. I. (2011). *The NIST Definition of Cloud Computing*. Gaithersburg: U.S Department of Commerce.
- Tholeti, B. P. (2011, September 23). *www.ibm.com*. Retrieved 12 08, 2012, from www.ibm.com: <http://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare-kvm/index.html>
- Trifkovic, V. (n.d.). <http://www.linkedin.com/in/vtrifkovic>. Retrieved 12 1, 2012, from <http://www.linkedin.com/in/vtrifkovic>.
- vservercenter. (2012). *www.vservercenter.com*. Retrieved 12 08, 2012, from www.vservercenter.com: <http://www.vservercenter.com/kvm-hypervisor>

“Moving to continuous integration on the cloud”