**Continuous Integration Environment Provisioning**



**Reference table**

[Analysis 3](#_Toc479361310)

[Intro to project 3](#_Toc479361311)

[What computational methods can be used to solve the problem? 3](#_Toc479361312)

[Who are the stakeholders for the project, what are their requirements for the solution and how will the solution meet their requirements? 5](#_Toc479361313)

[What solutions to similar problems already exist? 6](#_Toc479361314)

[What features must be included in the solution? 10](#_Toc479361315)

[What could the limitations be? 11](#_Toc479361316)

[What will the final solution need to do? 12](#_Toc479361317)

[Hardware requirements? 15](#_Toc479361318)

[Software requirements? 16](#_Toc479361319)

[Design 17](#_Toc479361320)

[Feature break - down 17](#_Toc479361321)

[Usability features: 20](#_Toc479361322)

[Data structures and variables 23](#_Toc479361323)

[Algorithms 25](#_Toc479361324)

[Algorithm comments 27](#_Toc479361325)

[Testing 28](#_Toc479361326)

[Post development tests - USER SUCCESS CRITERIA - NO KNOWLEDGE OF CODE 29](#_Toc479361327)

[Development 31](#_Toc479361328)

[Module 1 manual creation Instance 31](#_Toc479361329)

[Module 2 and Prototype 2 33](#_Toc479361330)

[Module 3 Stop and terminate an instance manually 34](#_Toc479361331)

[Module 4 A new instance Idea 35](#_Toc479361332)

[Module 5 Installing the components of the program -- Jenkins 36](#_Toc479361333)

[Module 6 Installing the components of the program -- Maven & Nexus 37](#_Toc479361334)

[Prototype 8 Installing -- Gerrit 39](#_Toc479361335)

[Testing: Evaluative 43](#_Toc479361336)

[Final Testing Evidence 43](#_Toc479361337)

[Evidence of testing data: 47](#_Toc479361338)

[Usability Testing (supported by evidence below) 49](#_Toc479361339)

[User testimony for usability features: 50](#_Toc479361340)

[Screenshots for usability 51](#_Toc479361341)

[Evaluation 52](#_Toc479361342)

[Evaluate Success Criteria (Reference to evidence of testing data) 52](#_Toc479361343)

[Evaluate Usability Features (refer to user testimony and usability screenshots) 55](#_Toc479361344)

[Limitations & Maintenance 58](#_Toc479361345)

Analysis

## Intro to project

Software development organisations often face a problem of slow application delivery. Usually this happens because the software development process is clunky and not very effective. Developers put the software code into random locations, there are issues with integrating the code coming from different developers, build process takes a lot of time and not reliable, and the generated release artefacts are not properly stored, maintained and versioned.

To tackle these problems, organisations use the method of Continuous Integration which is part of a bigger area of Development Operations (DevOps). Continuous Integration approach uses pre-configured infrastructure environments and specific tooling to ensure the smooth delivery of releases.

Continuous Integration environments are generally very complicated to set up / make sure that its components are working nicely together.

The goal of this project is to create a software application to automate the provisioning and set up of Continuous Integration environment.

## What computational methods can be used to solve the problem?

1. **Thinking abstractly & Visualisation**

There are four main Continuous Integration blocks which will be automatically created and provisioned by my application: **source control - code review - continuous build - release repository**

Release repository

Continuous build

Code review

Source control

Unlike in traditional manual method of provisioning of continuous integration environments, my application provisions an abstract type of the environment with common properties inherited from the Continuous Delivery model with the ability to define and/or override the parameters for the particular setup.

This has been chosen to make the application more generic, suitable for different organisations but using common principles of effective software development and delivery.

Application can deliver one or more basic blocks of environments. Using the abstract model, the application caters full scalability of the environments to suit different requirements in various cases. This provides flexibility for application users including getting more build and computation capacity on demand.

1. **Thinking procedurally / Decomposition**

The end result of running my application is a Continuous Integration environment working as a whole. It does however consist of four main components. Those components (building blocks) work using a pipeline where the output from one component is used as the input for another. Taking into account that otherwise those blocks are fairly independent it makes sense to split the whole application into relevant components, splitting the overall big application into smaller pieces.

1. **Thinking logically**

Like the majority of computer programs, my application widely uses branching (if/else clauses) and loops (for/while). Depending on the configuration parameters, certain steps will or won’t be executed for particular environment. Other steps and procedures will have to be executed more than once to take the desired effect. As the build infrastructure is distributed (I will have multiple build slaves which will compile the code), the same procedure of provisioning the build slave environment will be executed more than once. This is the obvious case for using the loops.

1. **Thinking concurrently**

Software applications are often used to automate inefficient and time consuming manual processes. Similarly, my application automates provisioning of Continuous Integration environment which is traditionally set up and configured manually. In order to speed the execution of my application, processes will be executed in parallel to decrease the total execution time. For example, the modules to provision Git, Jenkins and Nexus will be executed in parallel, concurrently delivering different parts of the desired environment.

1. **Backtracking**

Continuous Integration approach implements the model when the software is developed incrementally. The prototype is put under source control and features are added one by one to ensure that code can be compiled and is functioning at majority of all times, eliminating a ‘big bang’ way of delivering an application (when the program can only work at the very end of development). It is natural that for my application I have chosen the incremental development process too, as it best fits for delivering a working application.

1. **Heuristics**

I use an Agile process for my software development. It consists of multiple iterations starting from a prototype, incrementally adding features to the application. At the end of ever iteration I make sure that the implemented functionality is working and can be tested even though it’s not implementing the entire functionality. Further enhancements can be delivered in future iterations, but having a functioning application pretty much all the time greatly contributes to the overall quality of the application and delivery time.

1. **Pipelining**

My application widely uses pipelining. The entire algorithm uses specially implemented functions which are used in a way that the output of one function is used as an input for another.

## Who are the stakeholders for the project, what are their requirements for the solution and how will the solution meet their requirements?

|  |  |  |  |
| --- | --- | --- | --- |
| **Stakeholder** | **Requirements** | **Usage** | **Success criteria** |
| Software developers | Automation of a repetitious routine tasks such as building and compiling the code | Software developer will use the automated build system provisioned by my application to continuously compile and build code instead of doing this manually | The build environment provisioned by my application dramatically reduces the time required for building the code, freeing up the time for developer to write the code |
| Distributed build system environment | The build procedure utilises multiple build nodes for compiling the code | Build time is significantly reduced by utilising parallel build processes and environments |
| Capability of storing the release artefacts in unified location | The release artefacts are stored in a well-known unified location | Artefacts can be found in release repository. The intermediate libraries are also stored in release repository and are used during the build process |
| Revision and source control for the code | Developers store all the source code under source control using the environment provisioned by my application | Fully functioning source control application providing the ability for developers to commit their code into the source control repository |
| Facilities for code review | Software developers use code review as part of the development process with the tool provisioned by my application | Code can be swiftly reviewed with the help of a tool reducing the effort and providing the workflow for reviews |
| Versioning of release artefacts | Release artefacts are versioned using the common procedure, effectively distinguishing the releases | Software releases can be traced back to the changes in the source code |
| DevOps team | Ability to provision Continuous Integration environment automatically, instead of setting it up using manual and cumbersome processes | DevOps team will execute my application to provision Continuous Integration environment | The provisioning time with my application is significantly reduced compared to the manual approach |
| Release managers | Release artefacts are versioned, providing the ability to combine them into a release | Release managers assemble release artefacts into a release | All the release artefacts are situated in a well-known location and properly versioned |
| QA team | Quick turnaround of bug fixes from bug being identified to being available for re-testing | QA team will get new version deployed into the test environment from the release repository | Compiled/built application is swiftly available in release repository after a code fix being made |
| Production support | A well-defined release artefacts are available for production deployment and rollback | Release artefacts are used for production deployments and rollbacks | Release artefacts are easily available in release repository |

## What solutions to similar problems already exist?

The solutions to create the provisioning of Continuous Integration environment are quite rare on the market. Partially this is because the Continuous Integration approach is relatively new. The other reason is the implementation specifics for the Continuous Integration which depend on the programming language used in the organisation.

Indeed, Java and C# compilation procedures are significantly different and it’s not easy to come up with generic implementation of the unified solution to cover those two platforms.

Quite often companies hire consultants to build a custom implementation for a particular company. All the major companies like Google, Amazon and major banks are using Continuous Integration and Delivery approach, but as for provisioning automation they use proprietary tools developed in-house.

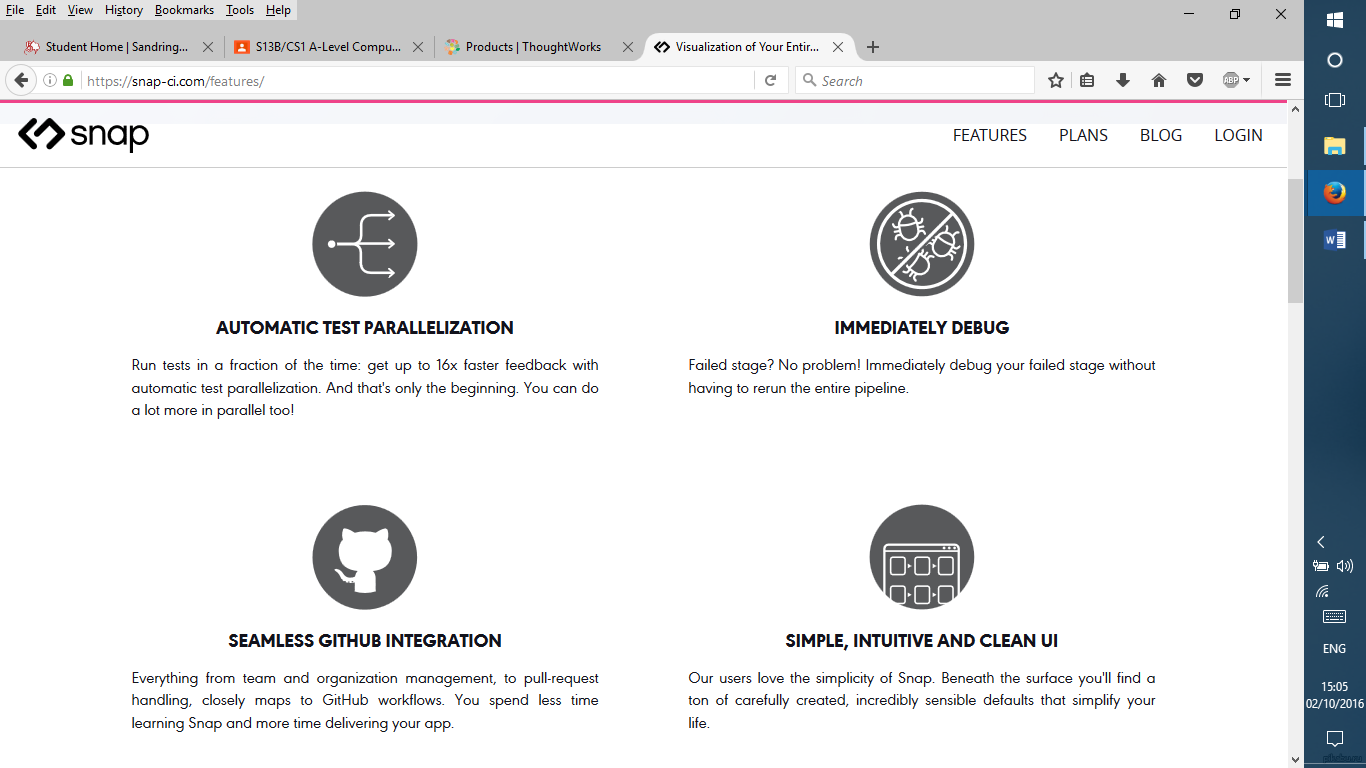
**There are two major ways of developing applications similar to mine.**

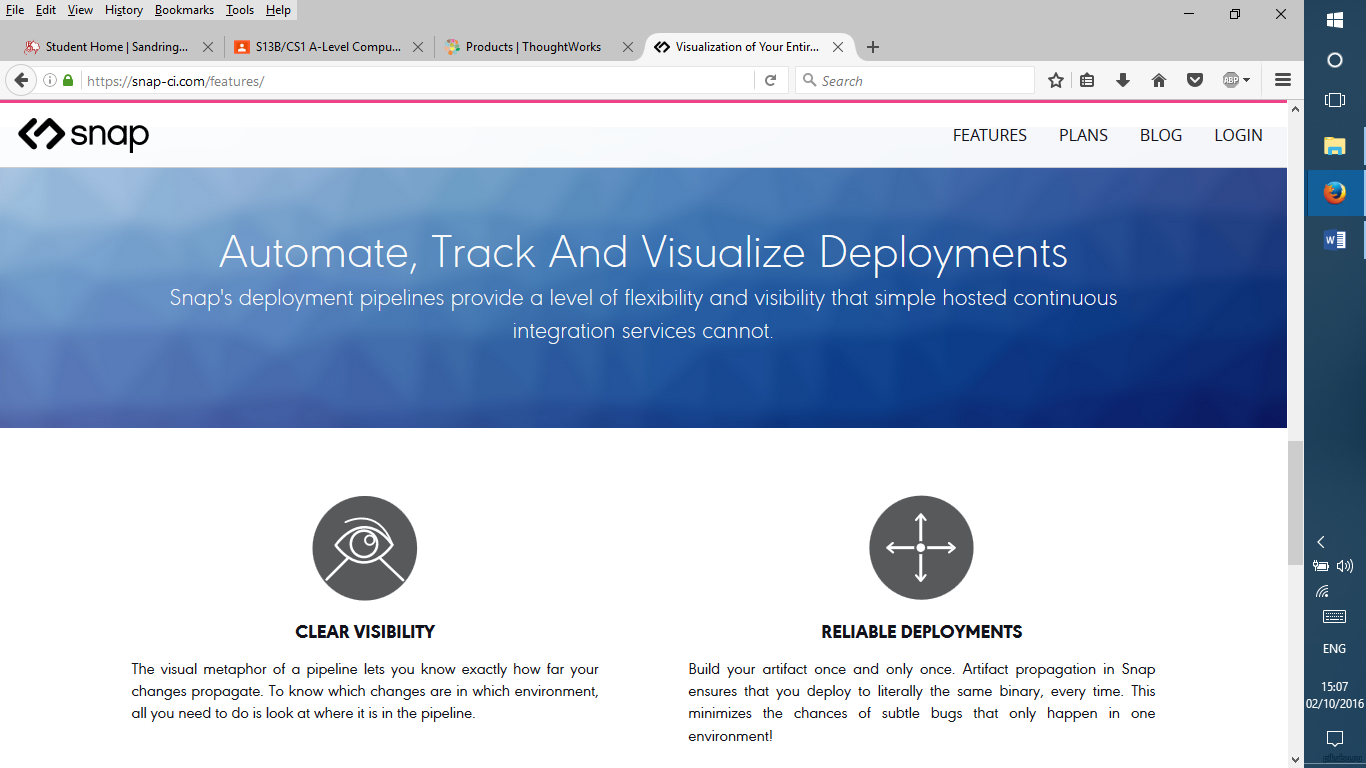
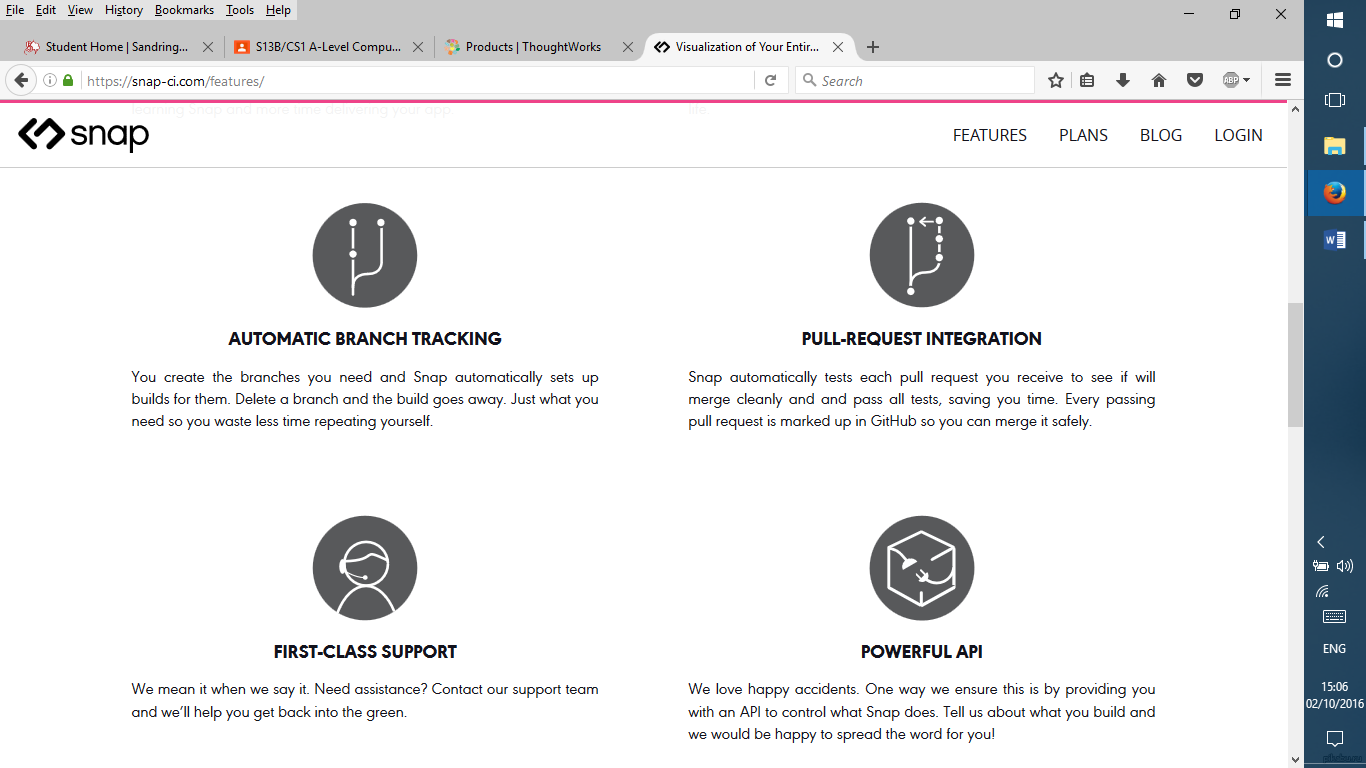
**Option 1**

Companies specialising on continuous delivery approach develop generic applications which can be heavily customised to suit the needs of a wide range of clients. One of the leaders of this market is a company called Thoughtworks which provides a range of applications automating the provisioning of continuous delivery environments. These applications can’t really be used on their own without complicated set up, for this set up the client has to hire Thoughtworks consultants to customise the software application. It can easily cost 5x times for the consultancy fees comparing to the cost of the software application offered.

Products offered by Thoughtworks (according to their website):

Features and benefits:

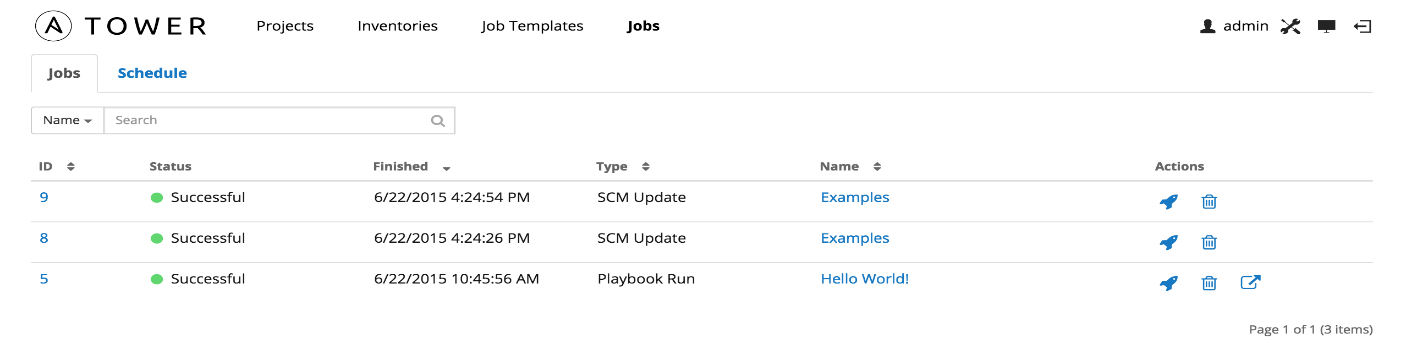




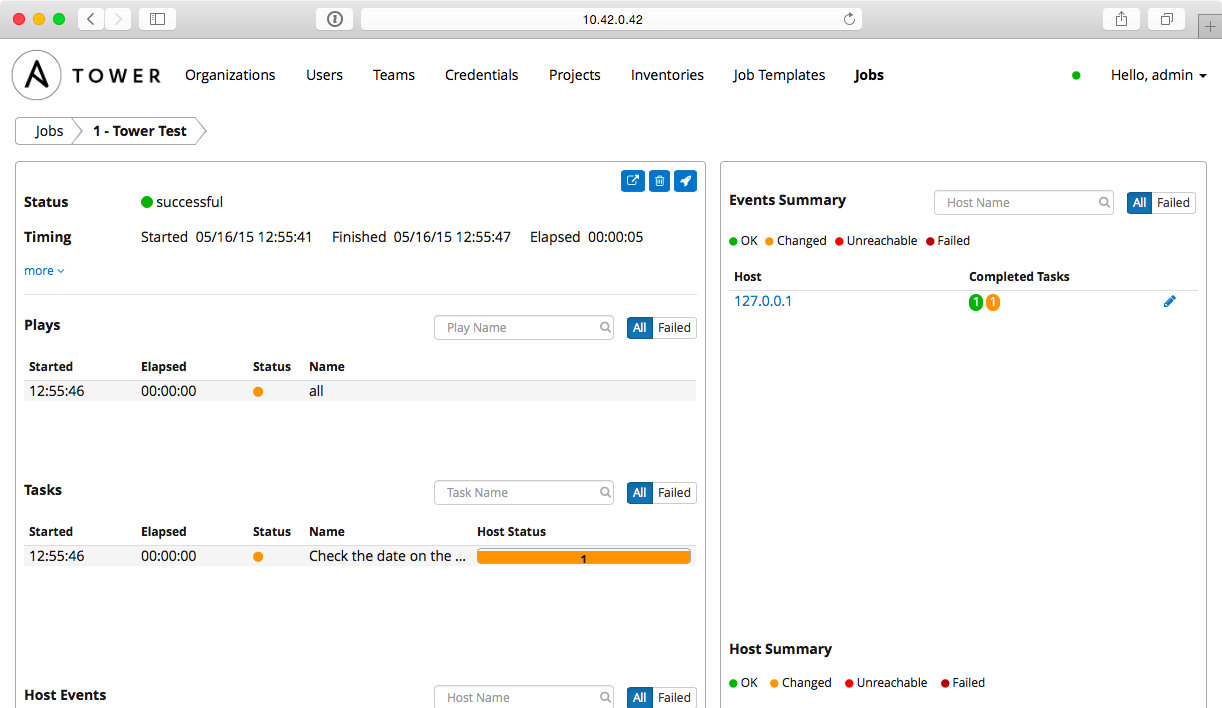
**Sources:**

<https://www.thoughtworks.com/products>

<https://snap-ci.com/features/>

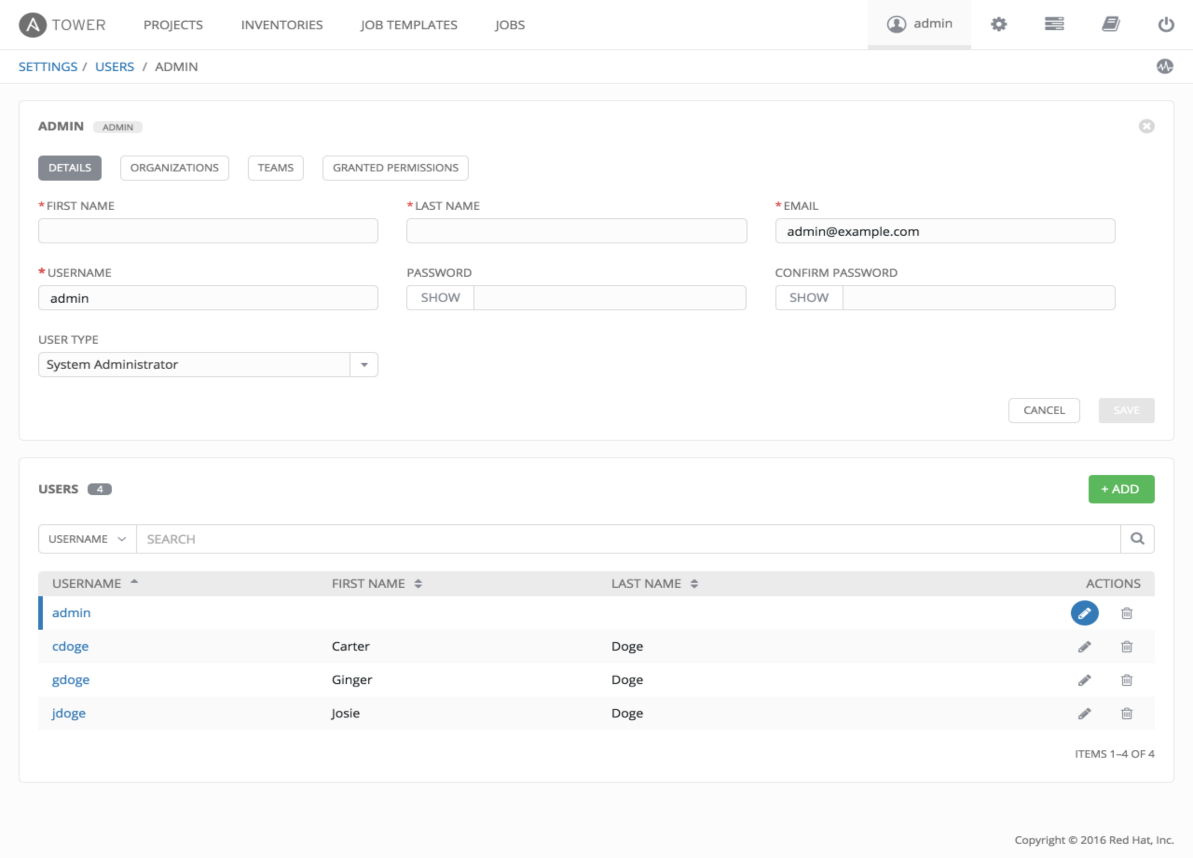
**Option 2**

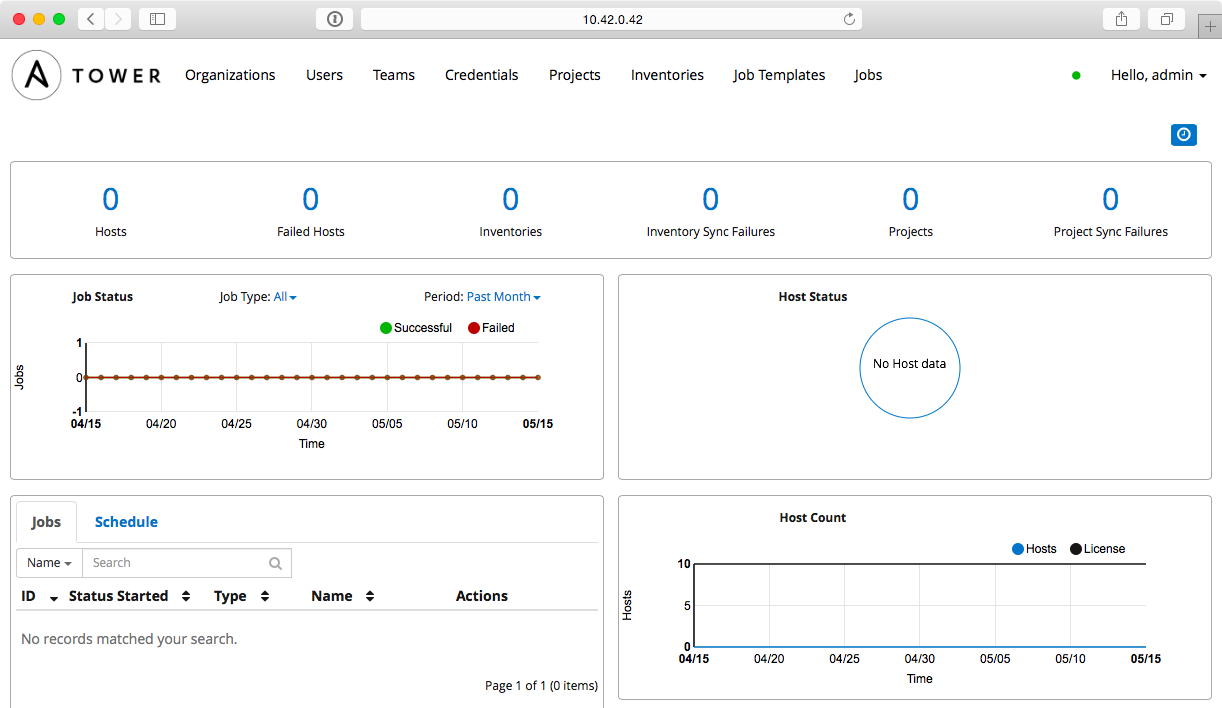
The other software solutions available on the market are coming from the Open Source world.

The applications like Ansible and Docker automate a big chunk of generic activities related to environments provisioning, making Ansible similar to my software application.

At the moment Ansible project has spent a number of years to polish and debug the application making it quite fast and reliable. It has also got relatively decent user interface called Ansible Tower simplifying the customisation for a particular client requirement.

Ansible is considered to be the leader of among tools called “infrastructure as code” and will clearly remain the leader on this market for at least the next few years.



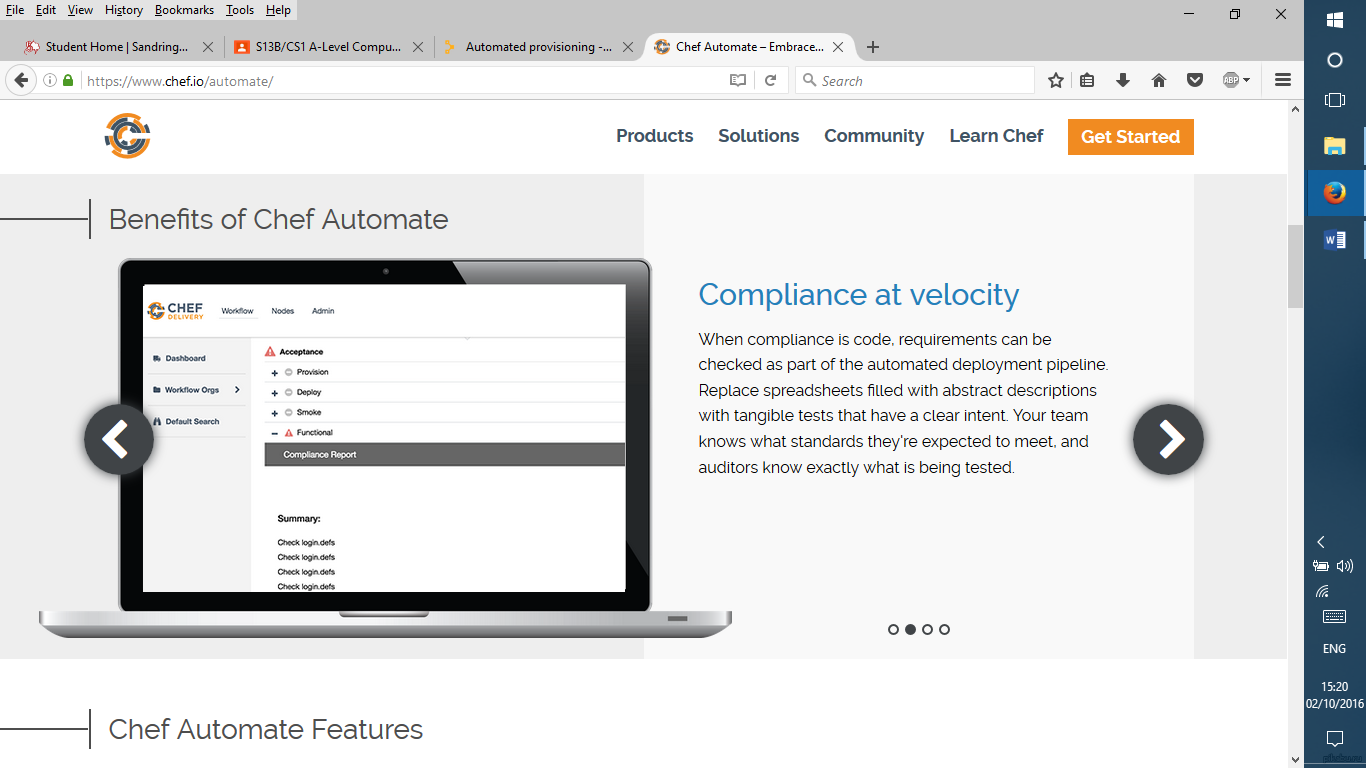
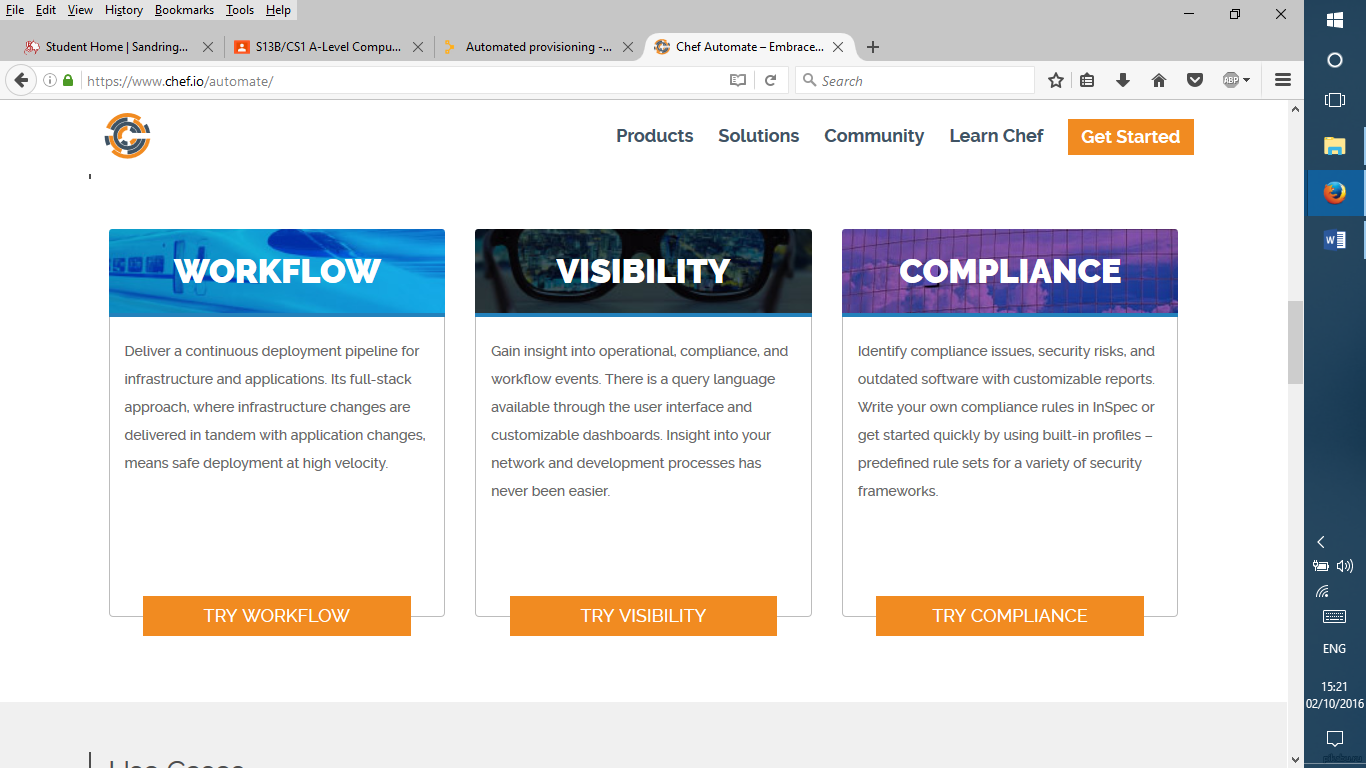


**Sources:**

<http://www.jeffgeerling.com/blog/automating-your-automation-ansible-tower>

<http://docs.ansible.com/ansible-tower/2.2.0/html/userguide/jobs.html>

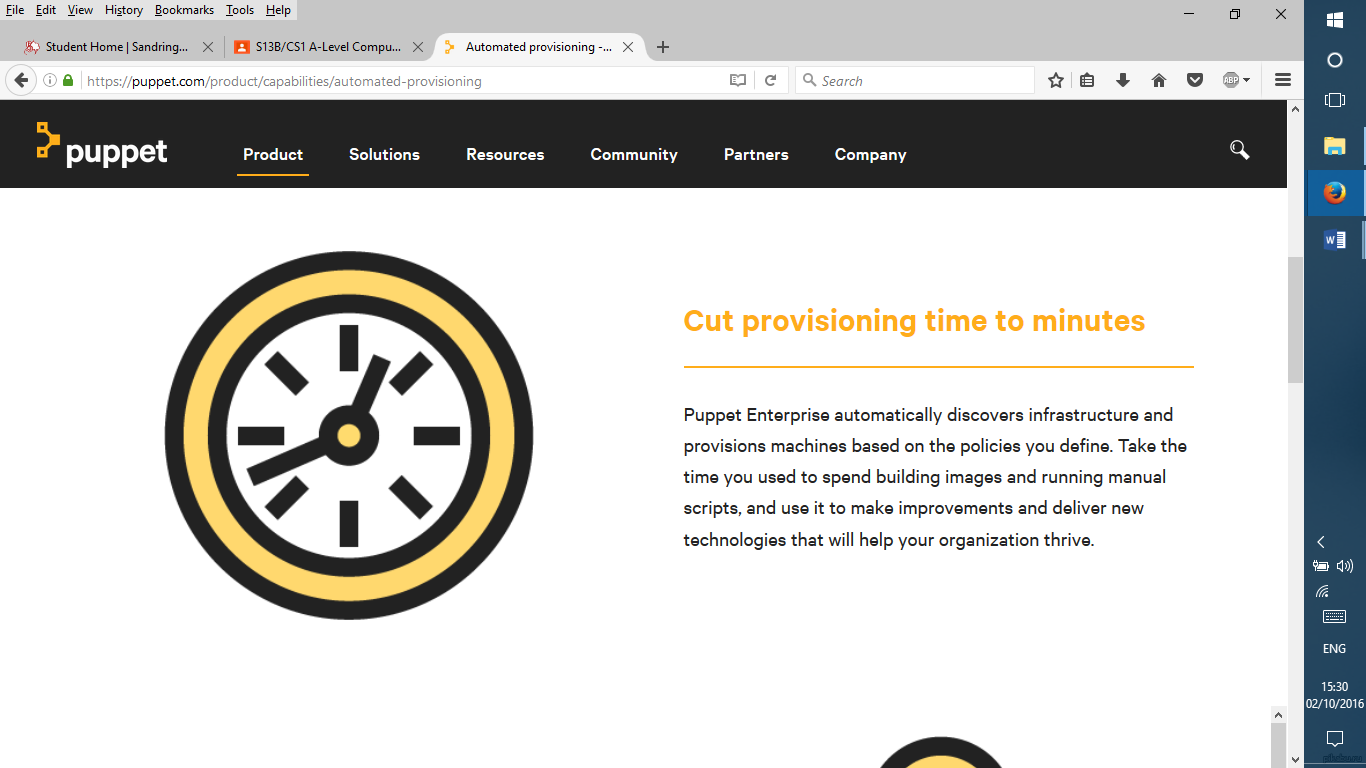
<http://docs.ansible.com/ansible-tower/latest/html/userguide/main_menu.html>

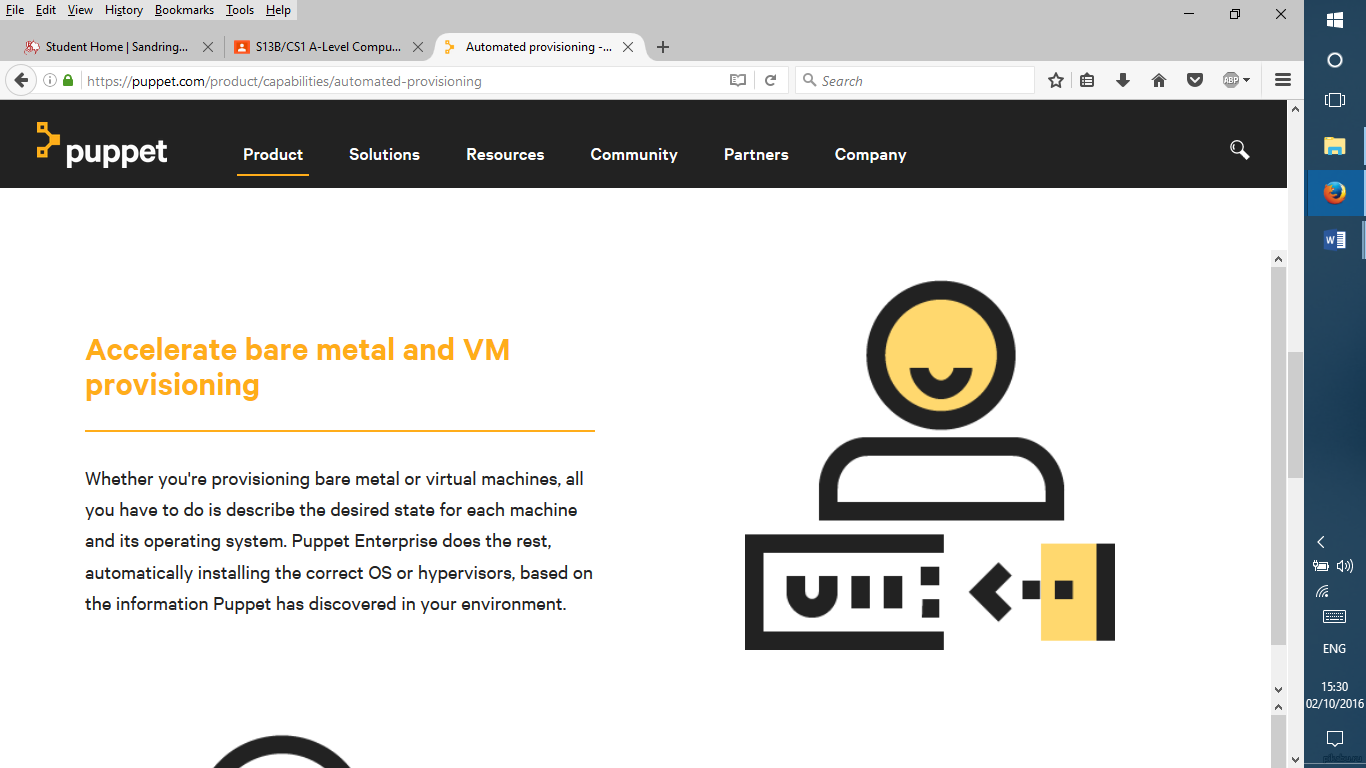
Other players like Puppet or Chef, older existed on the market for a number of years, however they are quickly losing their popularity because of their relative complexity and difficulties of its support.

**Source:**

<https://www.chef.io/automate/>







**Source:**

<https://puppet.com/product/capabilities/automated-provisioning>

* What features could be used in solution?

All the solutions available on the market have different GUI and data structures, however they implement pretty much the same process. Accordingly, in my study I have learned the process used in those applications and will use it in the implementation of my application. There is no libraries or source code coming from those solutions which can be reused in my application. However, my chosen programming language (Perl) has got a number of libraries that I plan to use during the development of my application.

* What is the reason these features?

The reason why I chose to reuse the process implemented in other tools available on the market is because this process became standard in software organisations, partially because of the efforts of companies such as Thoughtworks who have done a lot of work in popularising the concepts of continuous delivery.

## What features must be included in the solution?

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature name / description** | **Why is it essential?** | **Importance rank** | **Justification** |
| Command line interface | Command line interface will be used to invoke the application and therefore is essential | 1 - essential | This is an essential feature as it allows the user to interact with the program and its processes’ |
| GUI (web interface) | Web interface is much more convenient than the basic command line interface, however the application can function properly without it.  Although it is desirable it can’t be considered essential. | 2 – major | This is also another important feature of the solution because it allows for user input in a way that any stakeholder can access it because it’s quite simple. However it isn’t essential as the solution is mainly aimed at developers who are used to CLI’s. |
| Amazon instance provisioned automatically | The continuous integration environment created by my application will use Amazon EC2 as hosting | 2 – major | This is also another important feature as it is responsible for automatic creation of instances which is one of the core features of the solution, however this isn’t as important as some of the prior features and so would only fall under major as there are a lot of essential features excluding this one. |
| Deletion of Amazon instance | Once the Amazon instance is created it can incur the hosting charge. So when the instance is no longer needed it has to be deleted.  However, the deletion of an instance is a very simple task that can be done manually from Amazon EC2 console so there is no justification to include this feature to a basic functionality of my application as this feature has got an effective workaround. | 3 – minor | This is also an important part of the project. The deletion of an instance after being used is quite a significant feature, however it isn’t major or essential because it’s mainly done for efficiency of closing the instance whereas most of the more important jobs that will need to be done just after the creation (i.e. installation of key core features.) |
| Functionality to automatically provision the following applications:   1. Git 2. Maria Db 3. Gerrit 4. Jenkins 5. Jenkins slave 6. Maven 7. Nexus | Essential features covering the core functionality:  Source Control  Backend DB  Code Reviews  Continuous Build  Distributed Builds  Java compilation  Release repository | 1 – essential  1 – essential  1 – essential  1 – essential  1 – essential  1 – essential  1 – essential | This subsection of features includes all the core functionalities that the solution should have. These are all marked essential because they provide the basis of the working Continuous Integration environment and so are a top priority. For example the feature of source control enables the user to make continuous additions of code with the help of the release repository which is a core function of the environment. |

## What could the limitations be?

The solution my application implements does not cover all the possible clients, organisation, and processes which might exist in those organisations. The scope covers only Linux platform and Java software development projects. It doesn’t cover any processes using Microsoft products like C#. Similarly, it doesn’t cover any C++ code development either on Windows or Linux. Some projects which use scripting languages like Python, Perl or Unix shell scripts do not require compilation and therefore would not benefit that much from using my application.

My application would greatly benefit from having a user friendly web interface. However, this task is resource consuming, so because of the resources / time constraint I have decided to not implement it in my first release. The goal of the first release for my application is to implement the core functionality and create the platform that is capable of adding new models, including the GUI, at a later time.

The main limitations are caused by limited time available for this project. To a significant extent skills and experience is also a limitation because I have limited experience with Perl and implementation therefore will take longer. I have chosen to use open source tools for my application because commercial tools and versions are not available to me because of budget restrictions.

The main impact of these limitations is a relatively slow development because of limited skills. It might cause some of the non-essential features to be dropped in order to meet the deadline.

The unavailability of the budget rules out the use of commercial tools / libraries, therefore restricting the functionality to open source libraries, potentially impacting the timeframe required for this project.

## What will the final solution need to do?

The final solution is a software application that automates provisioning of different Continuous Integration modules and services so that they work well together. The output of one module is used as an input for the next module, widely utilising the pipeline approach.



The main modules are:

* Git
* Gerrit
* Jenkins
* Maven
* Nexus

Visually the continuous delivery pipeline is represented on this diagram.

In terms of what the solution needs to do the table below explains what the program should do and labels it based on the priority of feature to the end solution. There are three levels of priority 1 - essential, 2 - major and 3 - minor.

**Here is a brief explanation of why they are labeled so:**

1 - Essential - the feature that the is being discussed is crucial for the program to work, hence without the script from which the process is started doesn’t produce the desired result of a base (high priority process). An example of this could be the ‘connectivity to EC2 instance’ - this is labeled as essential as it's a base process that must be completed in order to gain memory space for the environment from AWS (amazon web services). This will then allow the components of the Continuous Integration environment to be installed and configured for stakeholder usage.

2 - Major - the feature that is being discussed isn't crucial to the creation of the Continuous Integration environment, however its priority is still high which would suggest that it still needs to be completed ideally for the environment to be correctly tested or made automatic to fulfil the stakeholder requirements. The example for this could be the automated creation of a master GIT repository where Github projects would be committed to by developers. This is a very valuable part of the Continuous Integration environment because it eases the developers job and can decrease the time required to complete the process of commitment of source code to the main repository. However this feature isn't essential meaning it can be done manually by the developer if this requires to be done.

3 - Minor - the feature that is being discussed isn't crucial to the creation of the Continuous integration environment and also isn't a major necessity of the stakeholder groups, however they can benefit from this feature in process of using the integration environment. This feature priority that should be completed if all other priority 1 and priority 2 features have been successfully implemented or the stakeholder has recognised / decided to change part of the solution requirement during the development phase of the program. An example of this priority group is the automatic creation of an instance in AWS, this feature which creates a new instance with full configuration from a script and then when required to do so can also terminate it is a ‘handy’ feature to have by the stakeholder, however isn't required by the main program and therefore should only be done as a prototype feature / least priority feature.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature name** | **Description** | **Priority** | **Justification** |
| **Command line interface and parameters handling**  1 Parsing and passing command line parameters to main application | Basic functionality to pass parameters from user to the application | 1  Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 2 Reading and handling configuration files from file system | Ability to read and use common configuration parameters which tend to stay the same for majority of the application runs | 1  Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| **Network and connectivity**  1 Connectivity to Amazon EC2 environment | Ability to connect to the target environment where the applications will be deployed | 1  Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 2 Create Amazon instance programmatically using Amazon API | Ability to automatically provision the servers on demand | 3  Minor | This feature is only a minor because it isn’t a core feature of the environment and isn’t required for it to function |
| **Code to provision: Git** | | | |
| 1 Automate downloading of Git package from repository, available in the internet | The way of obtaining the binary package for source control system, essential for the solution | 1  Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 2 Functionality to deploy and install Git app | Implementation of automated install of the source control system | 1- Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 3 Automated creation of Git repository | Creation of test / demo repository | 2  Major | This feature would be classed as major because it is an important aspect of the environment, however it isn’t important enough to be the core functionality of the environment. |
| **Code to provision: Maria DB** | | | |
| 1. Automate download of installation package from public repository | Obtaining the installation package from the internet automatically | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 2 Automate installation of Maria DB | Installation of database | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| **Code to provision: Gerrit** | | | |
| 1 Automate download of Gerrit | Obtaining the installation package from the internet automatically | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 2. Create a database in Maria DB to be used by Gerrit | Creation of data storage for Gerrit app | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 3.Create accounts for Git/Gerrit in Maria DB | Setting up a test / demo / admin account for Git / Gerrit | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 4.Create a test project in Gerrit | Demo project in Gerrit | 3 Minor | This feature is labelled as minor because it isn’t required by the solution to complete basic functionality. |
| **Code to provision: Jenkins** | | | |
| 1 Automate downloading of the package | Obtaining the installation package from the internet automatically | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 2 Programmatically set up a Jenkins master instance | Set up of continuous build application | 1 essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 3  Create a test / demo project on Jenkins master | Test / demo project to visualise the results of set up | 3 Minor | Justification behind having this feature as minor is that it’s not necessary for the program to work. |
| **Code to provision: Jenkins slave** | | | |
| 1  Automate installation of Jenkins slave | Provision Jenkins slave to implement distributed build functionality | 2 Major | This feature is classed as major because it contains some crucial functionality to the program, however isn’t essential because the functionality isn’t core to the solution. |
| 2  Connect Jenkins slave to Jenkins master | Implementation of connectivity between master and slave instances of Jenkins | 2 Major | This feature is classed as major because it contains some crucial functionality to the program, however isn’t essential because the functionality isn’t core to the solution. |
| **Code to provision: Maven** | | | |
| 1.Automate Maven installation | a. Install Java build tool so that it can be used in a continuous build process | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 3.Update custom Maven configuration to match environment | Change the generic Maven configuration files to match the environment | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| **Code to provision: Nexus** | | | |
| 1 Automate the download of free Nexus version from Sonatype repository | Getting the Nexus binaries from the vender | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 2.Automate Nexus installation | programmatically install the release repository app | 1 Essential | The justification behind having this feature is that it is essential to the solution and is a core feature that must be implemented for the solution to work. |
| 3 Automate creation of test / demo Nexus repository | Setting up a demo release repository | 3 Minor | Justification behind having this feature as minor is that it’s not necessary for the program to work. |

## Hardware requirements?

Continuous Integration applications are installed on Linux virtual machines in Amazon EC2 environments. Each of the virtual machines (servers) should have at least one CPU and 512MB of RAM. Nexus Repository requires 5 GB of storage whereas the rest of apps require 3 GB of storage.

## Software requirements?

Software versions used for this projects will be the latest release versions of products. To browse the web interface for Gerrit, Jenkins and Nexus I will use Chrome (latest release version).

**Sources:**

GIT <http://gitolite.com/server-sizing.html>

Gerrit <https://gerrit-review.googlesource.com/Documentation/dev-design.html>

Jenkins <https://jenkins.io/doc/book/hardware-recommendations/>

Nexus <https://support.sonatype.com/hc/en-us/articles/213464208-Sonatype-Nexus-System-Requirements>

MariaDB <https://mariadb.com/kb/en/mariadb/mariadb-hardware-requirements/>

**What is the criteria for success?**

1. All of the priority 1 - Essential features are implemented and there are no high priority defects for these features.
2. It is possible to create a new Source Control repository in the environment created by my application.
3. Jenkins application is up and running and it is possible to access it using web interface.
4. It is possible to access the repository in Nexus using web interface.

**What specific areas will you need to meets the requirements?**

Ensure that Continuous Integration pipeline is working in parts related to:

* Source Control
* Build
* Release Repository

by creating a test project in the environment created by my application.

**How to measure that it was success or failure?**

The success will be measured by the following tests:

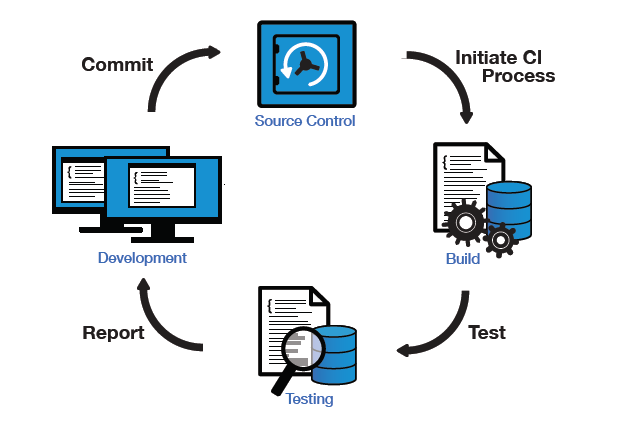
* As a developer it is possible to connect to the Source Control repository
* It is possible to commit code to the Git repository
* Each commit triggers off a Jenkins build
* The artefacts from Jenkins build get uploaded to Nexus

The operations above are typical steps carried out by the developing team (key stakeholder for the app).

Verifying that developers can perform these tasks makes sure that the entire solution of my software application is successful.

## Design

**Simplification of problem:**

Setting up Continuous Integration manually is a long and tiresome process; hence developers prefer to automate the process in order to be more efficient. The process is illustrated to the right:

1. Developers commit code to source control app

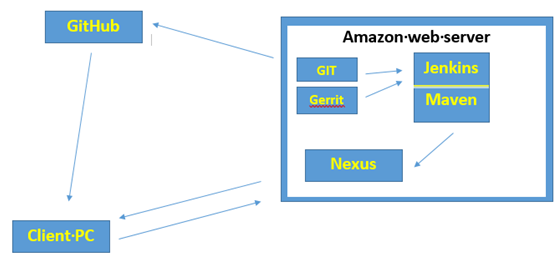
1b. Checks on code

1. Changes are sent to build app
2. Completed code saved to storage
3. Tested
4. Notification of success or failure

**Relation to my problem:**

My project aims to provide a means of simplifying the process. The main aim is to make all of this automatic, so the program must firstly create the workspace (i.e. all tools required for CI) to fulfil its success criteria. These tools include Git, Gerrit, Jenkins, Maven and Nexus. The next most important part that is very useful to developers is portability and so AWS (amazon web servers) are used to make it easily accessible and also usable cross-platform.

**The basics functions that the program must perform:**

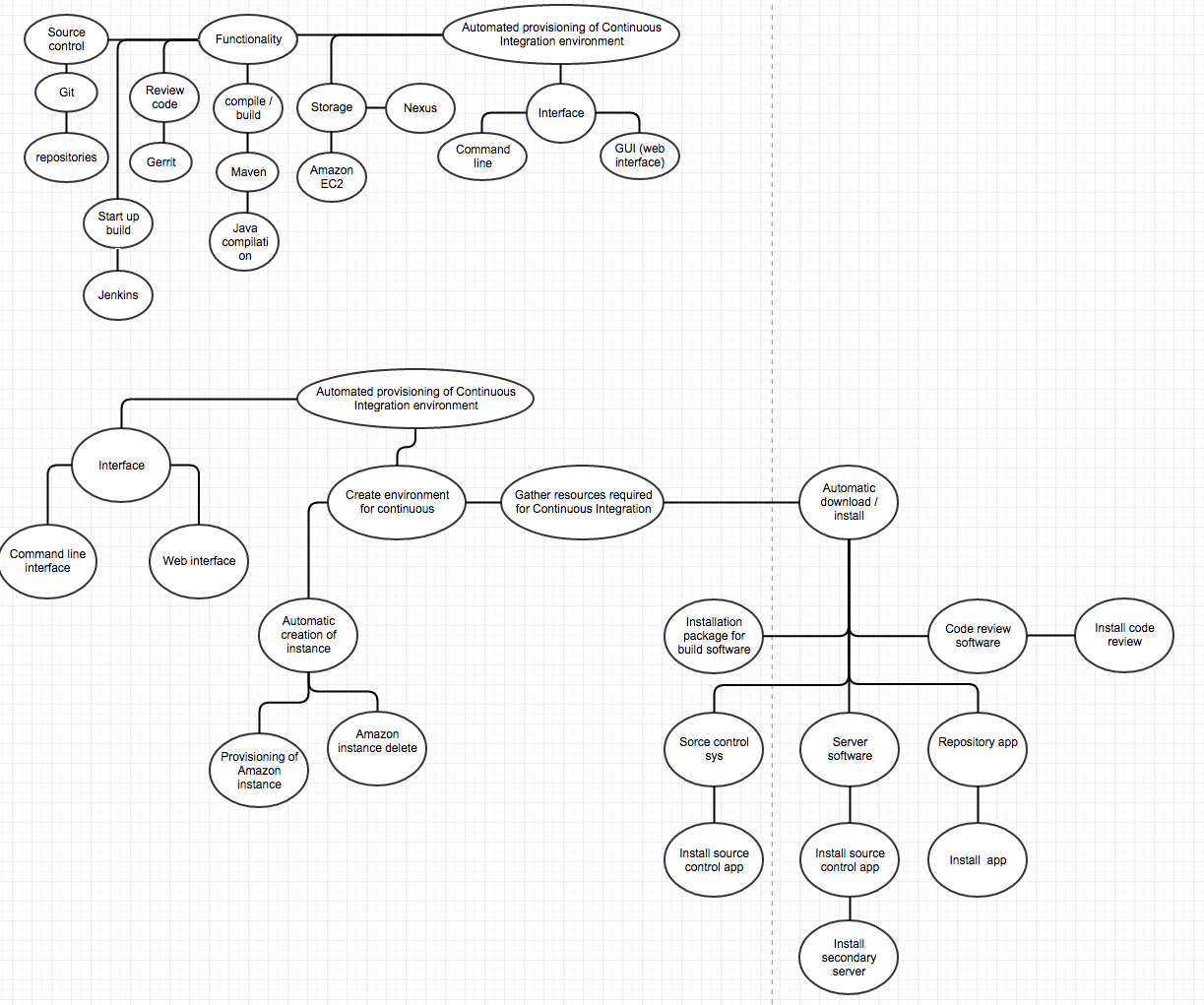


1. Create Amazon instance, delete amazon instance
2. Download GIT, Gerrit, Jenkins, Maven and Nexus packages and install
3. Terminal interface to access, use and exit program
4. Web interface to monitor code changes (some basic buttons for some of the components)

## Feature break - down

|  |  |
| --- | --- |
| **Feature / problem** | **Simplified explanation** |
| 1. Create instance automatically | To create Amazon instance it requires several inputs that allow the program to create the instance. So firstly the ID and access key followed and several other parameters that should be pre - set by into the program. After this a request has to be sent to AWS for them to allocate an IP address and associate the address with the account of the person using the program. This is secondary objective as the program must firstly function and then this feature can be added to make the program easily reusable. |
| 1. Delete instance automatically | To delete the Amazon, instance the procedure is the opposite. Disassociate address and release IP address. This could potentially be very useful as it would save time as if done manually several of these steps would have to be done on the AWS website and that would be time consuming. This is secondary objective as the program must firstly function and then this feature can be added to make the program easily reusable. |
| Backup if auto create / delete doesn’t work | If creating and deleting the instance automatically doesn’t work and there is a problem with getting the correct code to make it function the way it should i.e. with commands to create and delete from script, then manually create one instance that would host all of the functionality of program and would run indefinitely with the same parameters – IP, instance number and region. |
| 1. Terminal interface | The terminal would include the basic interface that will allow the user to interact with the program. Because the job of the Continuous Integration environment is to provide a base environment for the developer to operate in. This interface would provide a relatively simple way to access the program and would allow the user to run different commands when all of the default components are there. This is a basic feature that must be included in the solution as it allows the user to input and receive output from the program. |
| 1. Web interface | The web interface is also necessary because it allows the user to see a very simple screen with all of the key features on it. This makes it easier for the user to interact with the processes that are going on. For example, there could be a button for compiling code rather than having to type out a command to do so. This is a secondary objective however is still very important for the stakeholders i.e. in the code review and compilation stages. For example, both Jenkins and Gerrit would use a web interface to allow the user to input their specific details. |
| 1. Automatically download / install components: 2. Git 3. Gerrit 4. Jenkins 5. Maven 6. Nexus | Source Control Code Reviews Continuous Build Distributed Builds Java compilation  Release repository  The components listed in the left column must be downloaded, installed and then set up in order for the user to be able get the materials they require in order for the environment to function correctly.  Git should be downloaded as package, then unpacked and installed into the instance that is being used. This will allow for source control to be available to the potential user i.e. Git repositories could be made and data from major sites such as GITHUB could be downloaded to a local repo in order to then work with that data. Also release several repositories can be made in order to satisfy the requirement of a Release repository that the developer may require. Thus satisfying the continuous build requirement as code can be committed every time the user requires it.  Gerrit should also be downloaded and configured so that it is ready for use by the developer. So the relative connection should be established either manually or by script so that the instance can create an instance of the code review website that would allow a potential developer to be able to check their code that is to be committed to the main repository.  Jenkins should be installed similarly to the others and also made that there isn’t a need to log in every time the user opens the instance of the program. Jenkins however should be able to be opened in a web based format much like Gerrit and should have a different port to enable a simultaneous program run of both Gerrit and Jenkins to make it easier for the user – no need to close the Gerrit down when getting ready for using Jenkins for example.  Maven should be downloaded and installed also in an area where it can be easily accessed and therefore can be used by the user. This is because it is used during the java compilation process and therefore when the user is required to assign several parts manually i.e. stating what maven version is being used and where the files that required are being stored so that Jenkins can use them.  Nexus should also be made available should the user require to download the releases or specific parts to be tested or released to a main server. This might be useful for the user because they could need to send specific batches of code / program to a tester and then so that they don’t have to release the entire code for privacy or another reason that wouldn’t allow them to disclose all of their code. | |
|  |  |

The components are suitably sized because they have been split into major components that are crucial to create the final solution to the problem that is being solved. For example, installation and configuration of crucial components is crucial to make the program work as without it the main body of the program wouldn’t function and therefore the solution wouldn’t work, the size is also suitable because it bundles together similar bits of features that require a similar approach to apply them to the program and therefore should be simpler to program as a similar approach is used. Also the subproblems that are made to allow for each of the components to be of a similar difficulty and size making the program more spread out. This will make it easier to code as each components significance can be simply understood.



As mentioned before these sub sections of code as can be seen illustrated in the flowchart above all have a crucial meaning to the end solution to the problem. All of them carry a unique value that makes them important to the end product of the code that is to be written.

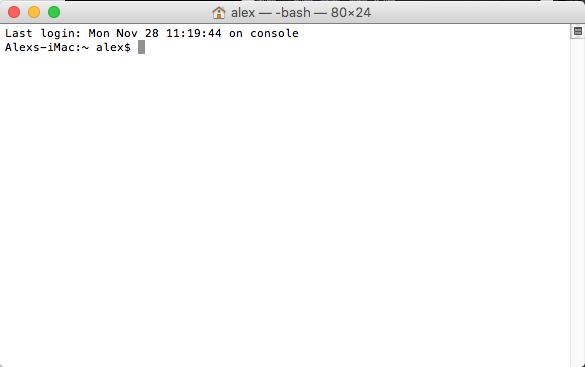
Firstly, GIT, the main point behind source control which allows creation of repositories and finding reusable code. This is crucial for the solution as it’s a ‘Continuous Integration environment’ for java based code, thus it would primarily require actual code that it can use to carry out operations on such as push, pull and commit.

The next component is Gerrit, this is the main tool behind code review which is very important for stakeholders as code needs to be correct and functional before being publically released. With Gerrit this is made possible giving the stakeholders which are mainly developer based a chance to review their code for errors and make some tests before it is sent to its next destination.

Another crucial component for this is Jenkins which is what is used to compile the code that is made / checked / copied with the prior two components. Jenkins allows for this code to start compiling in a user friendly manner. Working closely with Jenkins is maven and Nexus which provide the compilation and storage parts of the process. So Jenkins is similar to a template that allows other programs to be integrated into it in order for it complete its functions. Maven would be program that does the actual ‘build’ which is crucial as the program wouldn’t run before it has been compiled. And then After all of the compilation has finished Nexus has to be installed as an option for the developers to download the completed version if it is required.

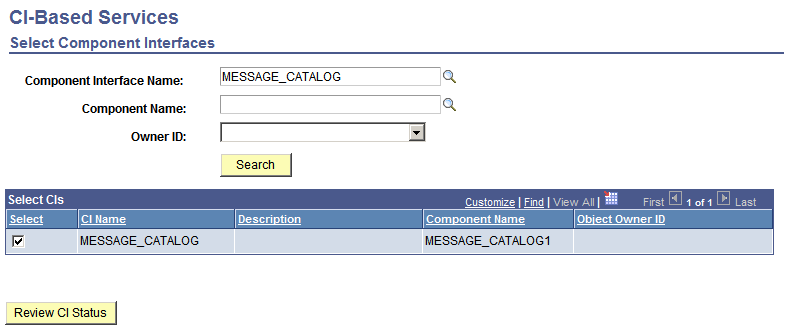
## Usability features:

**Early ideas for the design of interfaces**



**The Terminal:**

As mentioned previously there will be a terminal that will allow the user to interact with the processes that happen when the program is running, before being started and when closing it. Each part of the cycle has it’s own purpose. For example, at the start the program is going to have connect to AWS and create an instance, the end would have to delete and release the instance and the middle bit would have to take user input about what the program has to do. The terminal is a quite a simple way of doing this allows to check for errors and where they have occurred.

The terminal is also a crucial part of the program as it is the main thing that allows for the user to communicate with the program should a mistake / error occur. It also gives the freedom for the user to deviate from the program should they require a different outcome. This is very important as not all stakeholders may want to go with the standard procedure that is made by the initial developer of the program which is why certain options should always be available to satisfy the needs of the user. Also this might be important as the first time set up of the program may require some manual input to allow for it to be usable by the stakeholder i.e. the program may prompt for specific user details such as what ’port’ to use for displaying Jenkins and gerrit in the web interface.

**Web interface:**

Web interface is a less necessary component of my solution. The use of GUI instead of commands from the terminal makes it much easier to complete actions for the user because things like buttons replace the traditional commands that would be used in the terminal.

If I was to implement the interface, I would have to have a several web pages that are connected to each other. So for example, a main page that allows you to access all of the components of program and pages where the main and working directories can be located all of these pages would need to have buttons, ticks and search’s.

Main page

Sys checks

Status

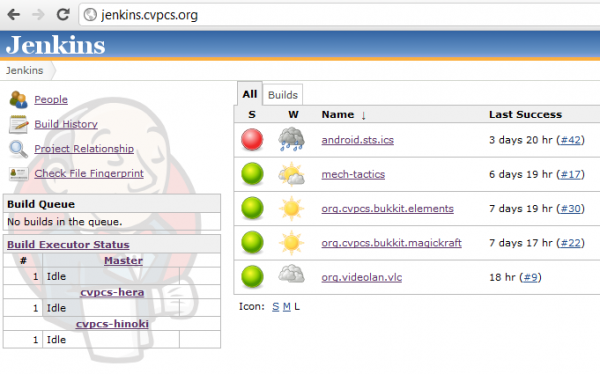
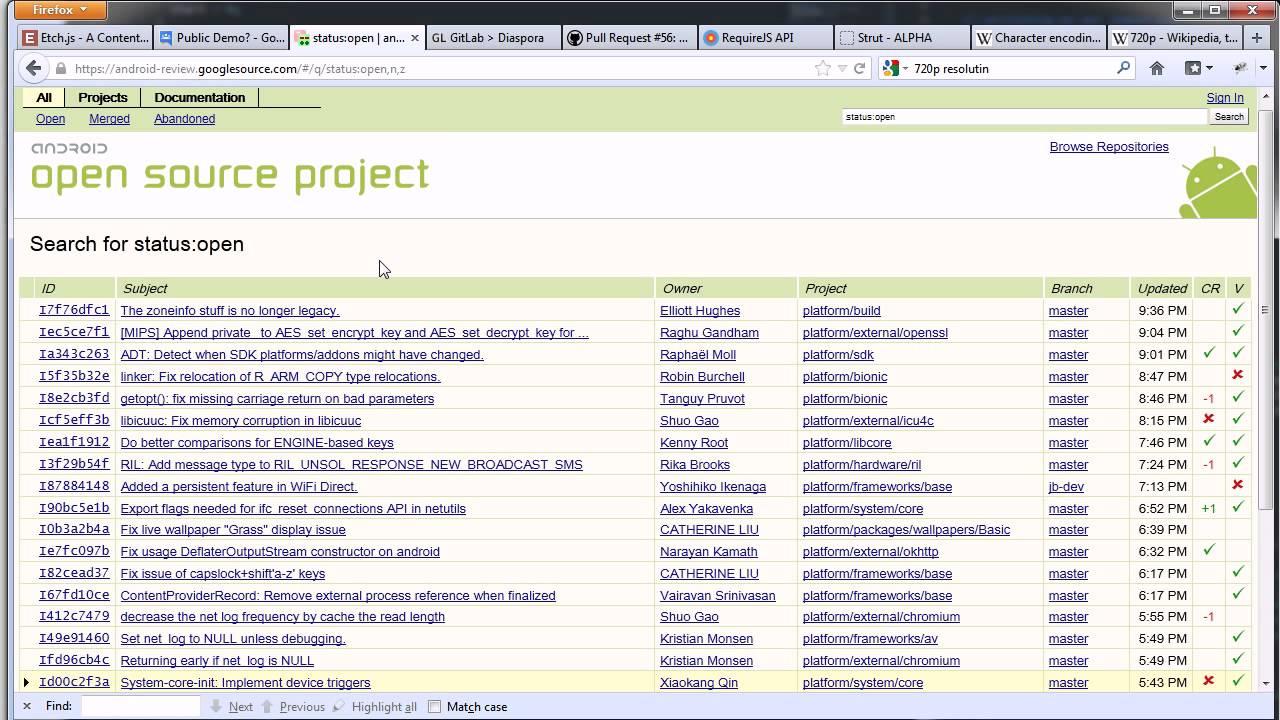
My web interface should always have a bar that would allow the user to go back to the main page if they should decide that there is a need to go back and restart the process or if they had made a mistake and would have like to go back.

Code review

Choose a directory

Commit code and build

However, if this isn’t implemented due to problems with the main code and its usability, then the standard can be used by opening the Gerrit and Jenkins websites on a port of the machine that is being used to display their functions and be able to use them this way.



**Intro**

Usability of the program is key factor that needs to be considered in my solution. The way stakeholders such as developers interact with the program is very important as it could be the reason between the program being used or not. The main factors that can affect the usability of a program are navigation and visual clarity, consistency and flexibility, user feedback and error prevention and also efficiency. These factors play a vital role in how a stakeholder views the solution to the problem as these features make the solution more user friendly and therefore could potentially fulfil a range of tasks if modified and so would be an asset to the stakeholder.

**Navigation:**

Navigation is one of the main factors that affects how the user views the completed solution. Whenever it’s the path to a certain folder that needs to be accessed or the way through a website to the destination that the user in looking for, navigation tries to make it the easiest to be able to get to the right destination which would allow for user satisfaction depending on how simple it is to fulfil a task that needs to be done. Here it might be difficult for a stakeholder to navigate through the Jenkins site as it requires some knowledge of what needs to be done before it can be used properly, so a solution to this could be to create a mini – guide to instruct where the user is required to make changes in the configuration which may allow for simpler navigation to this point. An example of this factor being involved would be in the website specifically made for the purpose of simplifying the process that the user has to go through to get a job started on with Continuous Integration Environment. This would include links or a specified port connection that would allow the user to access the key features of the program - including Jenkins and maven builds, GitHub and Gerrit code review which would allow the user to utilise the time spent with the program giving them an advantage over the traditional manual ways of completing a build project.

**Visual clarity:**

Closely related to navigation is visual clarity – this is a very important feature because it allows the user to find the correct item they are looking for and also to then proceed with their task quickly leaving them satisfied with how easy it is to find the item they are looking for. This is could be crucial to a stakeholder as depending on what the solution looks visually the stakeholder will most likely consider if they want to use it or not i.e. if the terminal window is full with the outputs from the program it might be difficult for the user to understand what has been done and what hasn’t. To address this simple line spacing between each completed process could allow for more clarity. Examples of visual clarity within my project would include firstly the main website, GUI provided by Gerrit and also the Command line interface. Starting with the website it is very important for the to be able to navigate through it - this is one of the most important ideas that a programmer must understand because should it not be implemented the program wouldn't be effective in its usage. To do so the programmer must make the interface simple and easy to understand, much like the first drafts of my web interface I have made it very easy to understand and with only the crucial functionality that would allow the stakeholder to accomplish their task. Secondly the GUI provided by Gerrit and Jenkins has been used because it is the easiest in terms of interfaces and makes the tasks of a stakeholder easier to complete, although a few flaws in that the user will have to know what they are doing in order to customise and utilise the GUI to its full potential. The other visual clarity feature is the CLI, this feature is also very important to the user as most developers and programmers prefer the command line as opposed a GUI (in a business environment) as it is easier to obtain materials and so on as well as running different actions that would have been difficult to do in a GUI. From my side making the CLI visually friendly is important because if left alone it will output line after line of difficult to understand text which is all clumped together making it very difficult to understand. So the solution for this is adding things like spaces and new lines (similar to /n) after each line of code / command has been executed and also having a heading and a small explanation for these lines. For example code for ‘instance creation’ could be outputted to the user as:

**‘ Instance creation status : started ‘**

**‘ Instance creation status : complete’**

**‘ Instance creation status : incomplete : 3 errors detected’**

The implication of such features would make it much easier for the user to understand the problem with the data that they have put in. Also the program use would be much more simpler for them as they would have an idea of where they have gone wrong. Also this gives the visual clarity that makes it easier to understand what has happened after the program was run.

**Consistency:**

Consistency is another very important factor with usability for a program. This is because if the solution is only half right the stakeholders wouldn’t be satisfied. Therefore, it is ideal that the solution would consistently be good throughout itself. This is very important also in my project as the consistency of certain features such as visual clarity being the same in every part of the program where it can be implemented, error messages being outputted with easy to understand comments (like ‘unable to connect to AWS server: action: check instance connection’) which would make it much easier for the developer / stakeholder. All of these features must be implemented throughout the program to make it consistent. As it wouldn't be a good practice to just have them in one part of the program and have nothing in the remaining part. Also another important point to consistency is the program being totally dependant on Amazon Web Services in order maintain the instance which in some cases could make it more difficult to use if AWS suddenly crashed - the instance would also get turned off which is a significant difficulty with using AWS.

**Flexibility:**

Flexibility is also another very important usability feature. This is because the user may want to change something that they dislike or require a deviation from the original code which would fulfil a more specific requirement that the stakeholder may have. In my case this is also another very important feature that would have to be implemented - for example if a part of the program did not work as intended a replacement piece of code would have to be made in order to do the same job. Hence it would be a good idea to make the program easily customisable and versatile to changes that the stakeholders may have. An example of what will be done in my program would be that the instance connection process can be changed (i.e although automatic connection and creation to instance is a very interesting feature to the environment it can be replaced with a manually created instance and connections via SSH in order to compensate the feature which would allow it to do the same job as the prior.

**User feedback:**

User feedback is also another aspect that has to be considered when creating a solution and also then managing the finished solution. This is very important as the user / stakeholder opinion of the solution can help to improve some errors / recommendations that they may have to improve the solution which would result in making the solution better than what it was prior. Good management of user feedback is very important as it can help to improve some of the sections where the developing person may have not considered a certain aspect that might have been crucial in order for the solution to work. User feedback in my solution would be considered by research before the project had began and also during the development process feedback from the user would be collected in order to change what they may not like.

**Error prevention**

Error prevention is also another important aspect of the usability of a program. Obviously the solution will have to be tested for errors, however to check for these errors testers / stakeholders can be used if the program is fitted with the correct sections to return where the error is and what has happened incorrectly. Also from a stakeholder’s point of view errors shouldn’t be happening in a finished solution as it would have meant that the solution isn’t finished completely. This is a significant feature that must be included in the solution. Although completely taking out errors isn't a very easy task the coded solution will have notifications to the user where the program has started to go wrong which would give them an insight into where the code / user input had gone wrong. This would make it significantly easier for the user to locate and prevent errors.

**Efficiency:**

Efficiency is also another important aspect of a solution. This is because stakeholders generally would like to make it as easily and quickly as possible because that may only be part of what they are working on. Therefore, the solution must be made so that the user would be able to use it easily and be able to finish with a faster time than just manually creating the same solution. Efficiency is a factor that is defined by the previous factors. This is because they make up the main functionality of the program and therefore if done correctly make the the program more efficient. For example the error handling / locationing feature would allow the user to minimise time searching for problems with the code and also things like visual clarity would allow the user to save time finding things buttons and features that would make it faster to access key parts of the program and reduce time for builds and key processes.

## Data structures and variables

Creating and setting the correct variable and data structure names is very important in any solution. Data structures should be justified as unrelated structures will only slow the workings of the solution. A ‘strange’ naming policy may also work against the developer as they may forget what the variable was and what its function was. It is best to name the variables in accordance to the situation for example naming a function ‘potatoes’ might not be a good idea as nobody would understand what it is supposed to do.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variable name | Data type | Description | Validation | Justification of validation |
| For creating new instance automatically and deleting it | | | |  |
| ec2 | Scalar | Holds amazon library for Ec2 instances. Holds access key id, secret id, version and region. | Validation can be made through the instance opening | The validation is enough to allow the user to know that the variable has worked as it has a direct link to the instance opening – the instance won’t start if variable fails. |
| instance | Scalar | Holds image id, instance type and amount of instances | Validation can be made through the instance opening with the correct parameters | The validation is enough to allow the user to know that the variable has worked as it has a direct link to the instance opening – the instance won’t start if variable fails. |
| running\_instance | Scalar | Holds data to describe instance | Validation no required as is not an input | No validation can be done as this is a pre - allocated variable – done by the instance when it is created. |
| instance\_id | Scalar | Contains instance scalar | Validation can be done through correct creation of instance | No validation can be done as this is a pre - allocated variable – done by the instance when it is created. |
| allocation\_id | Scalar | Contains the command to allocate vpc address | Validation cannot be done as is not an input | No validation can be done as this is a pre - allocated variable – done by the instance when it is created. |
| ip | Scalar | Contains command to allocate address | Validation cannot be done as is not an input | No validation can be done as this is a pre - allocated variable – done by the instance when it is created. |
| For main code | | | |  |
| Command | Scalar | Contains ‘shift’  shift() is a built in Perl subroutine that takes an array as an argument, then returns and deletes the first item in that array. | No validation as is not an input | No validation can be done as this variable is specific to Perl and is a built in function. |
| Host | Scalar | Holds the host ip address | Correct instance connected to means that this is valid | The validation is enough to allow the user to know that the variable has worked as it has a direct link to the instance opening – the instance won’t start if variable fails. |
| Pem\_file | Scalar | Holds the .pem file which includes the  Public certificate, Intermediate Certificate  Root certificate, Private key | Correct connection to instance means that this has worked | The validation is enough to allow the user to know that the variable has worked as it has a direct link to the instance opening – the instance won’t start if variable fails. |
| Ssh\_cmd\_prefix | Scalar | Holds the default host username that allows connection | Correct instance connection is the validation for this | The validation is enough to allow the user to know that the variable has worked as it has a direct link to the instance opening – the instance won’t start if variable fails. |
| Remote\_command | Scalar | Holds the command being executed and the instance where its being executed | Correct instance connection and then correct command displayed | The validation is enough to allow the user to know that the variable has worked as it has a direct link to the instance opening – the instance won’t start if variable fails. |
| Remote\_exec | function | Function that allows all of the above data to be stored in itself which allows for the function to be called when a new line of code is written. | Correct execution of commands and connection to instance as validation. | The validation is enough to allow the user to know that the variable has worked as it has a direct link to the instance – the command won’t execute if function fails. |

## Algorithms

**Creating instance**

# (Use ec2 libarary – Net::Amazon::EC2)

My ec2 = new instance

input access key id, secret access key, version, region

# initiate instance

My instance = start instance

use image ID, min, max instance count, instance type

# allow time for instance to be created

program pause

output ‘instance has been created’

Output ‘binding ip address’

Associate address (instance id => variable1, allocation id => variable2) or stop report error

Output “instance creation complete”

**Terminate instance**

Output ‘press enter to terminate instance’

Terminate instance

Use instance id => variable or die “couldn’t terminate instance”

Disassociate instance

Use public Ip or stop report error

Release instance

Use PublicIp => $ip, AllocationId => $allocation\_id

**Main program**

**Remote command**

# creates function that connects to instance and completes command

Create function ‘remote command’

Use host\_address, ssh id, .pem location,

Return command, output and user or die “couldn’t complete function remote command”

**Installing Components**

#install Jenkins

Download repository from source

Import Jenkins

Install Jenkins

Change permissions for library => jenkins to root

Jenkins start or return“Jenkins couldn’t start”

#install maven

Download from repository

Start maven or return “maven couldn’t be started”

# install Nexus

Download from repository

Start nexus or return “nexus couldn’t start”

# Install Git

Download git from repository

Install git

Create key pair

Output ‘key pair’

Connect key pair to git instance

# Gerrit

Create user gerrit

Make file for web interface

Download gerrit

Install and start or return “gerrit not started”

**Test for Git**

#test git is working

Clone repo from github to local machine or return “git isn’t operating correctly”

## Algorithm comments

|  |  |  |
| --- | --- | --- |
| **Algorithm name** | **Comments** | **Justification - decision** |
| Creating instance | The algorithm shows how to code the creation of a new instance from script. The sub tasks in that include input of key data that specifies what parameters the instance will take. The next step will then take the type of instance, specific image that it will be based on and the number that needs to be created. There is then a pause to allow instance to be made and validated by AWS EC2 servers. Once done the instance will bind a unique ip address to the instance so that it can be connected to directly. | The justification behind having this as one module is that there are quite a few steps that are need to be done before the instance correctly connects, which if not done correctly may lead to the instance not being connected to and the following pieces of code not being executed. |
| Terminate instance | The next algorithm shows how to code the termination / deletion of an instance that has been created. Sub tasks include creating a prompt for the user to press ‘ENTER’ when they want the instance to terminate, use the EC2 library functions terminate, disassociate and release to delete the instance, disassociate IP address and then delete the instance from EC2 servers. | The justification behind this being placed as a key algorithm is that it is quite complex. Also in the three library functions are required to be done in that order, with the consequence being that the instance will not disassociate if the termination has not been completed. |
| Remote command | The next algorithm shows how to make a function that remotely accesses EC2 instance that has already been created and then passes commands from a pre made script. The algorithm is made for a Perl based script and uses SSH as a means of communication to the instance. | This algorithm is suitably allocated because this forms the basis of the machine connecting to instance and allowing the user to have control over the instances functionality. Thus meaning that it is a key module / part of the whole process, which would require it to be analysed in detail for the script to work. |
| Install components    Jenkins  GIT  Gerrit  Nexus  Maven | This algorithm includes the ideas behind installing and starting key components of the Continuous Integration environment. These include the downloading of Jenkins, Git, Gerrit, Nexus and Maven, which will then be installed into the instance. The next subtask then needs to make sure that the the installed components are working. | This algorithm is also very important as it shows the potential developer the key components of the CI environment that allow the user to gain full functionality from them. This is suitably sized because it is an important part of the coded solution and to some extent the largest. |

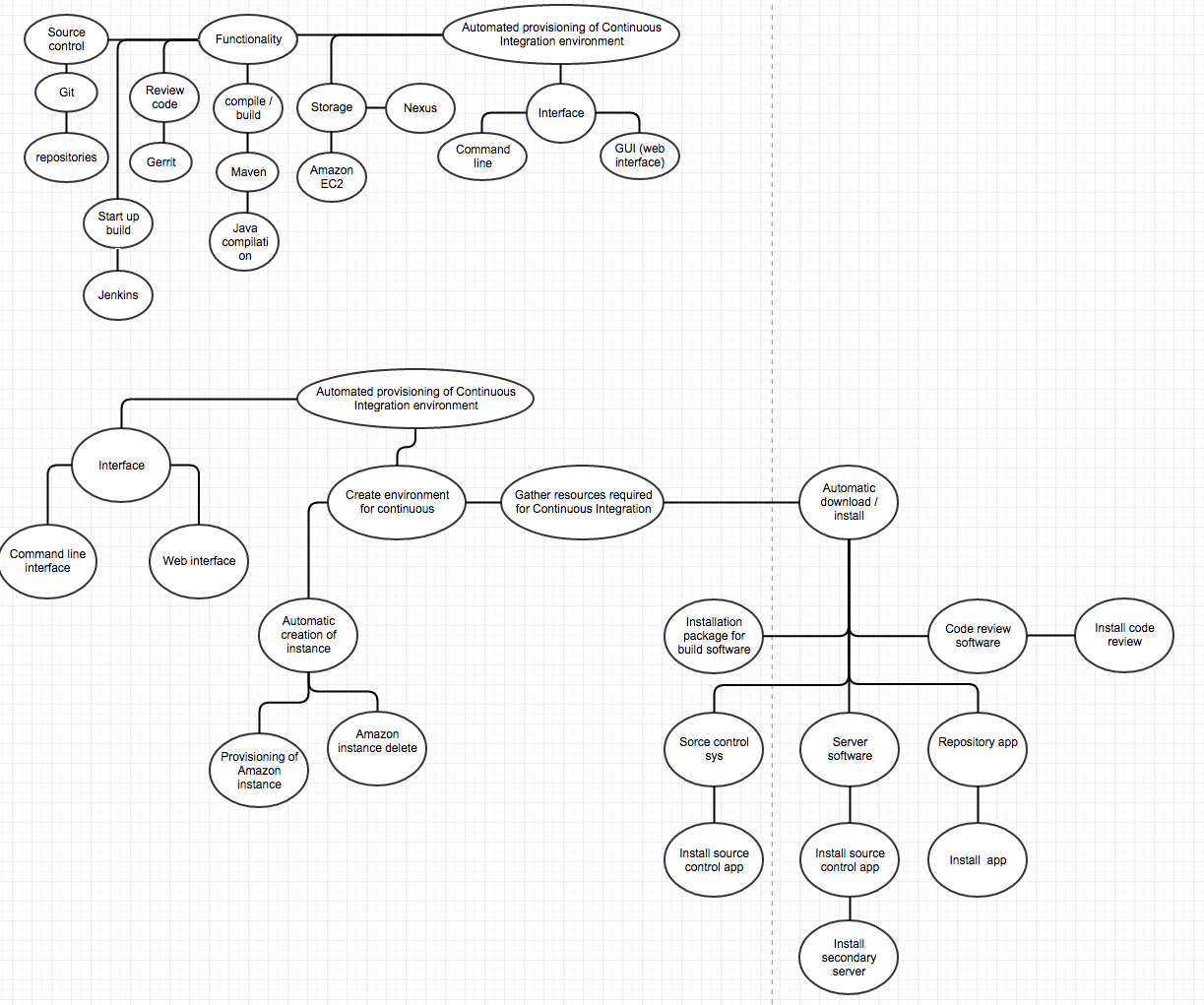
## Testing

|  |  |  |  |
| --- | --- | --- | --- |
| **What testing?** | **How testing?** | **Good / Bad Boundary** | **Why testing / justification** |
| Instance creation | Creation of instance manually. | If instance is created result is successful. | This is crucial to be tested as it gives an insight as to what steps are required to make an automation of the instance. |
| Script set-up connection to AWS (Script that connects to instance via SSH and output a test result) | Result is good if instance is connected to and outputs some results | This is an important test point because it would show that the instance can successfully be created from a script. |
| Script that starts instance and binds IP | Good if IP is bound to instance automatically | This test is important because as it shows that the instance is correctly bound to an Ip address |
| Instance termination | AWS website to terminate instance (Manual input) | Good if instance terminated or instance not terminated | This is important to test as it gives an insight to what steps will be required to terminate instance from script |
| Small script to terminate instance  Script to terminate instance | Good if instance terminated automatically | This is very important as it shows that the code written works without problems.If a different approach will be required in terms of terminating or not terminating at all |
| Disassociation of unique IP address | Script to disassociate address | If script disassociated the Ip address then good | This test is important as it shows that the instance can be successfully deleted if required to be done |
| Script to connect to instance (static instance) | Script to test connectivity and then output data | If script can connect and install Jenkins it's good if not bad | This test is important as it shows that the instance can be connected to when in a static form (manually). Where the IP never changes. |
| Script to make a local version of Jenkins that runs with local host | Script that installs Jenkins and then creates a local host at port 8080 | If Jenkins webpage can be opened at port 8080 | This is an important test as it shows that Jenkins can be used on the users device and does not require the internet version to run (could benefit non internet cases) |
| Install Maven from script | Script that downloads and installs maven | If maven is downloaded and installed, then good | This is important because a core program is being installed into the environment. Also if it doesn't work the environment can be used. |
| Install Nexus from script | Script that downloads and installs Nexus  Script to set up Nexus | If Nexus is downloaded and installed, then good | This is important because a core program is being installed into the environment. Also if it doesn't work the environment can be used. |
| Install Git from script | Script that downloads and installs Git  Script to set up Git | If Git is downloaded and installed, then good | This is important because a core program is being installed into the environment. Also if it doesn't work the environment can be used. |
| Download and install gerrit | Download gerrit and create review site  Script to download gerrit | If gerrit is downloaded then its good | This is important because a core program is being installed into the environment. Also if it doesn't work the environment can be used. |

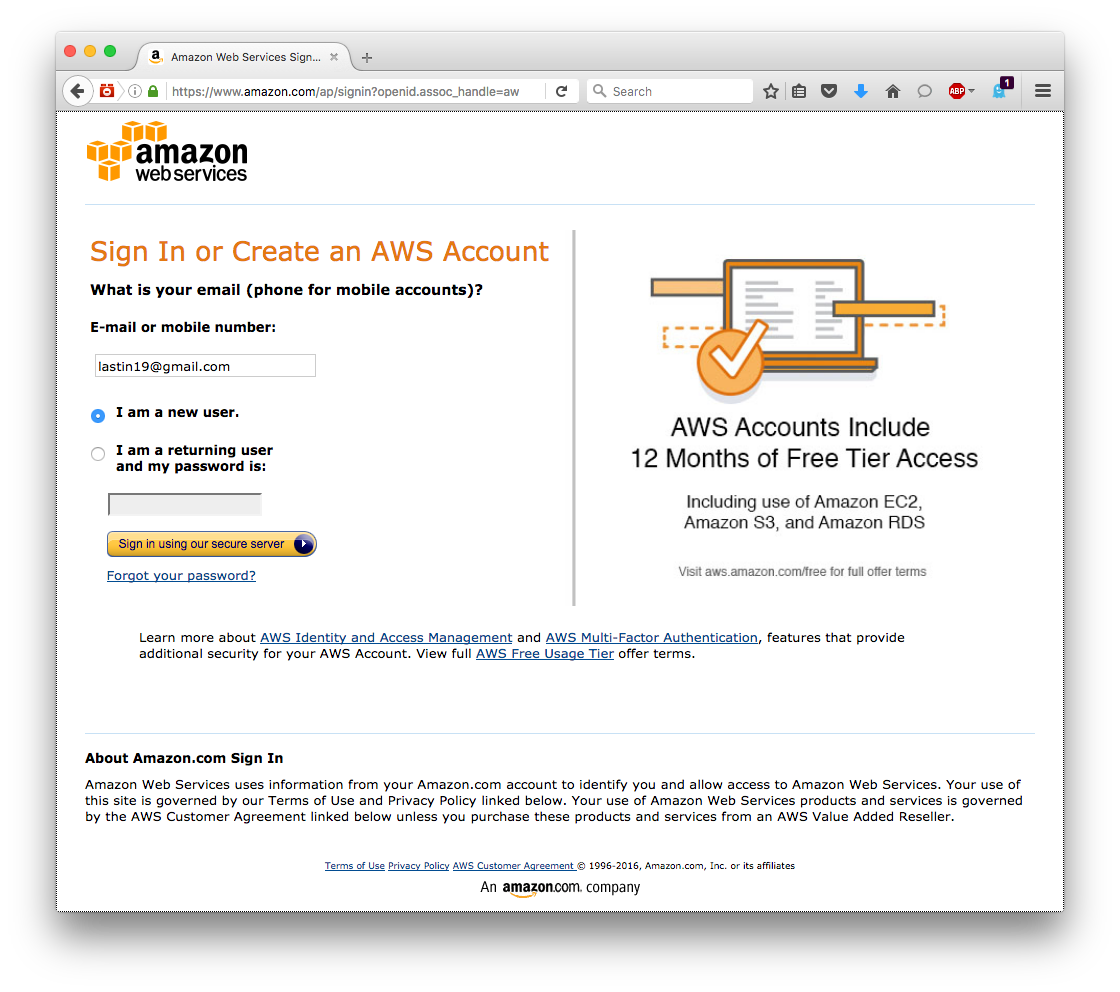
## Post development tests - USER SUCCESS CRITERIA - NO KNOWLEDGE OF CODE

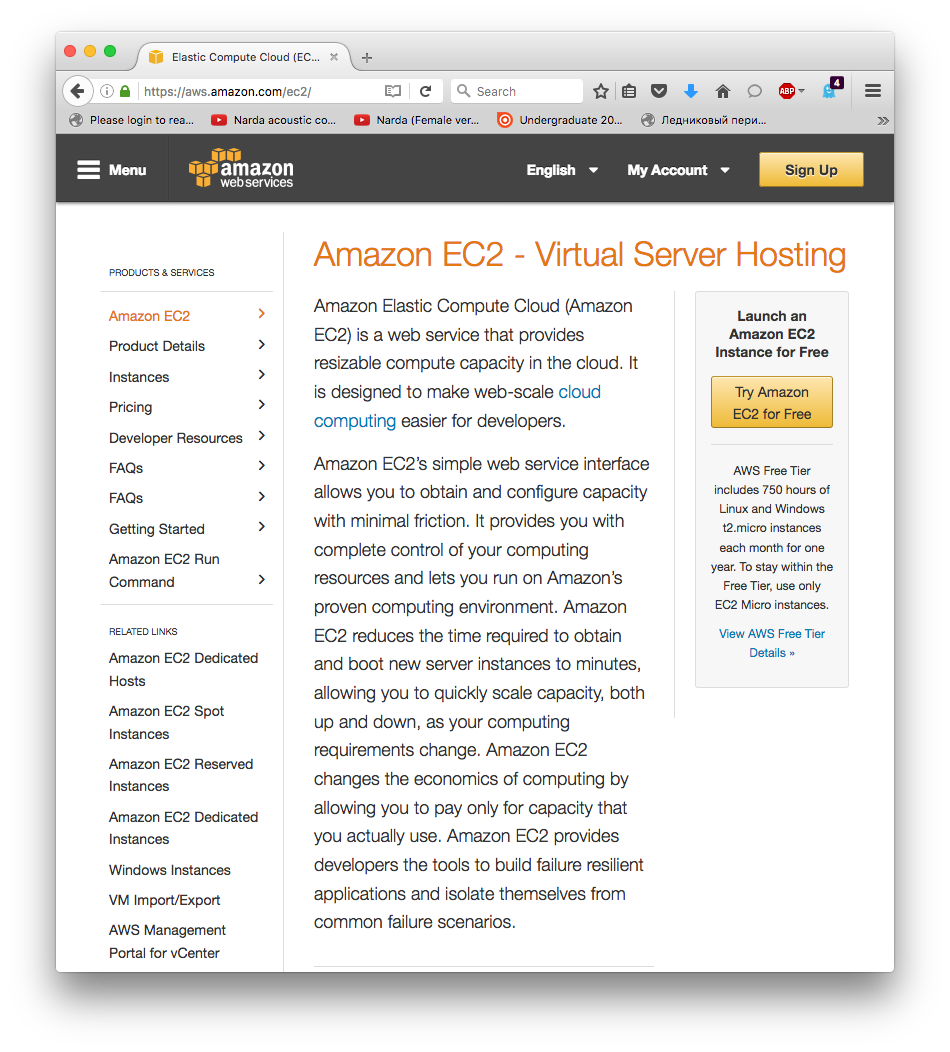
|  |  |  |
| --- | --- | --- |
| **What / how testing?** | **Why testing?** | **Justification** |
| Command line interface – allows the user to input and receive output from the instance / program | Testing to see if the command line input is working make changes to the contents of instance. Also to check if the environment has the right content and is ready to be used. | Justification for user testing for the CLI is essential because they may have some requirements in terms of what they want to make it more customisable to their needs. |
| Network connectivity – If allows user to open a webpage from the instance | Testing to see if the instance is able to connect to the internet and can also make a web interface of Jenkins and Gerrit as they are crucial features of the solution. | Network connectivity should also be checked by the user as specific connections could have been blocked and would require some changing of parameters in order for the solution to work. |
| Main components of solution installed and ready for usage    Git  Gerrit  Jenkins  Maven  Nexux | Testing to see if Git is installed on the instance. This is so that it is known that Git ready and installed in the instance with further testing in the area of Gits functionality afterwards.    Testing to see if Git can clone a repository from the web to make a local one on the current machine. This is so that the Git can be tested for its functionality that the user is going to require from the solution.    Testing to see if Gerrit is installed on the instance. This is also to test if the download for gerrit has succeeded and that further testing can be proceeded with.    Testing to see if Gerrit website is running. This is to test network connectivity to and from the instance and also to see if this is working correctly.    Testing to see if Jenkins is installed. Check to see if Jenkins package is installed and is ready for testing the usability of Jenkins web instance.    Testing to see if Jenkins local instance is running. This test is to see if network connectivity is correctly configured and also so that the Jenkins local website is up and is ready to run.    Testing to see if Maven is installed on instance. This is to check if the component is ready to be used by Jenkins to build projects.    Testing to see if nexus is installed. This is to see if the component is ready to be used. | Firstly the testing of the component being correctly installed, also the reason behind having the user test is that they should be able to make sure that the component is there and is ready to be used.    User testing of core functionality is essential to find out what the further user requirements could be implemented.      User testing of core functionality is essential to find out what the further user requirements could be implemented.    User testing of core functionality is essential to find out what the further user requirements could be implemented.    User testing of core functionality is essential to find out what the further user requirements could be implemented.    User testing of core functionality is essential to find out what the further user requirements could be implemented.      User testing of core functionality is essential to find out what the further user requirements could be implemented.    User testing of core functionality is essential to find out what the further user requirements could be implemented. |

## Development

**Introduction:**

Here I will go through the modular design of my solution. Each module will have a representation of where it fits on the diagram .

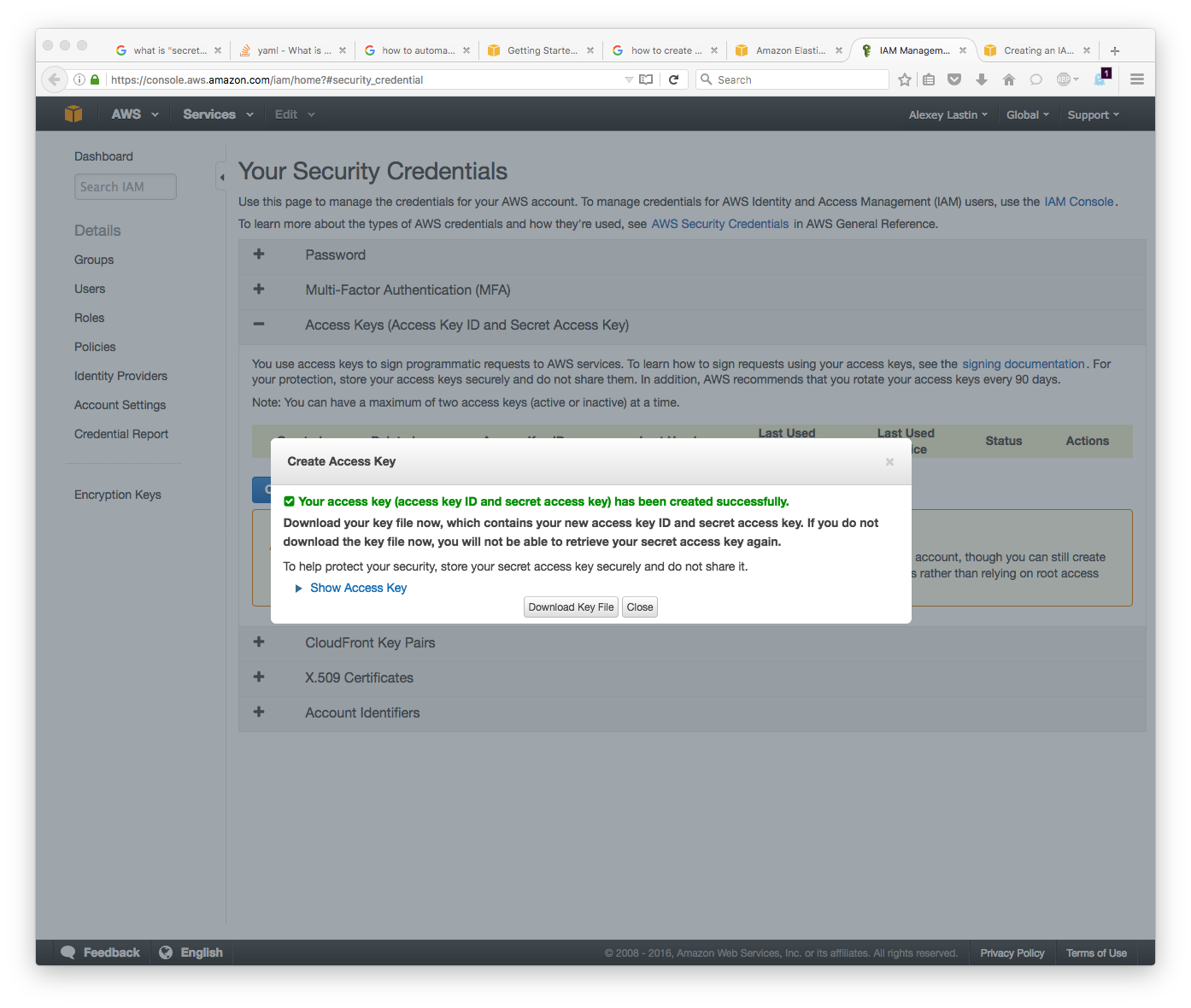




## Module 1 manual creation Instance

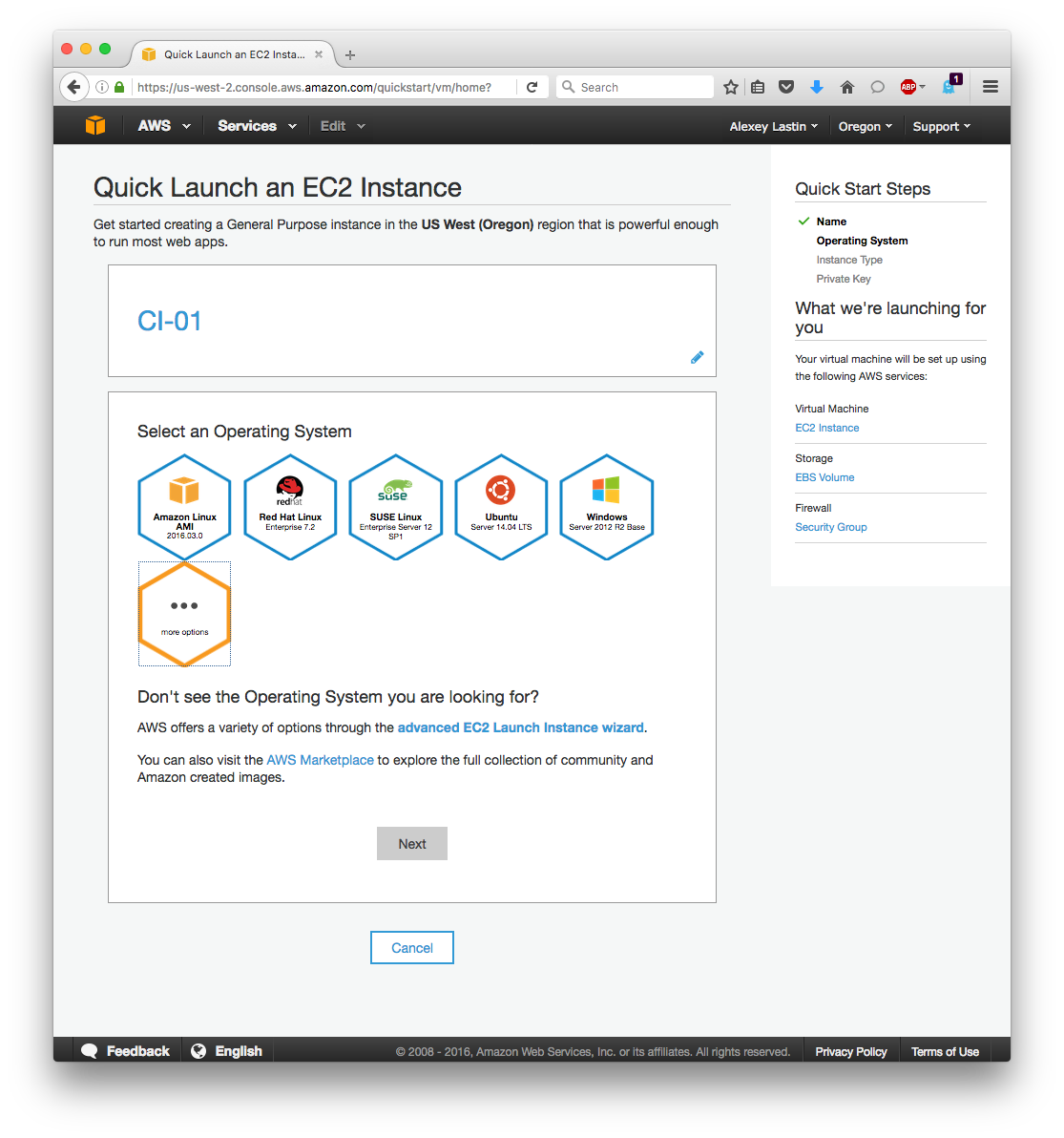
The development process started with creating an account with Amazon web services. This is a web hosting service that allows the user to run various operating systems to make use of their capabilities.

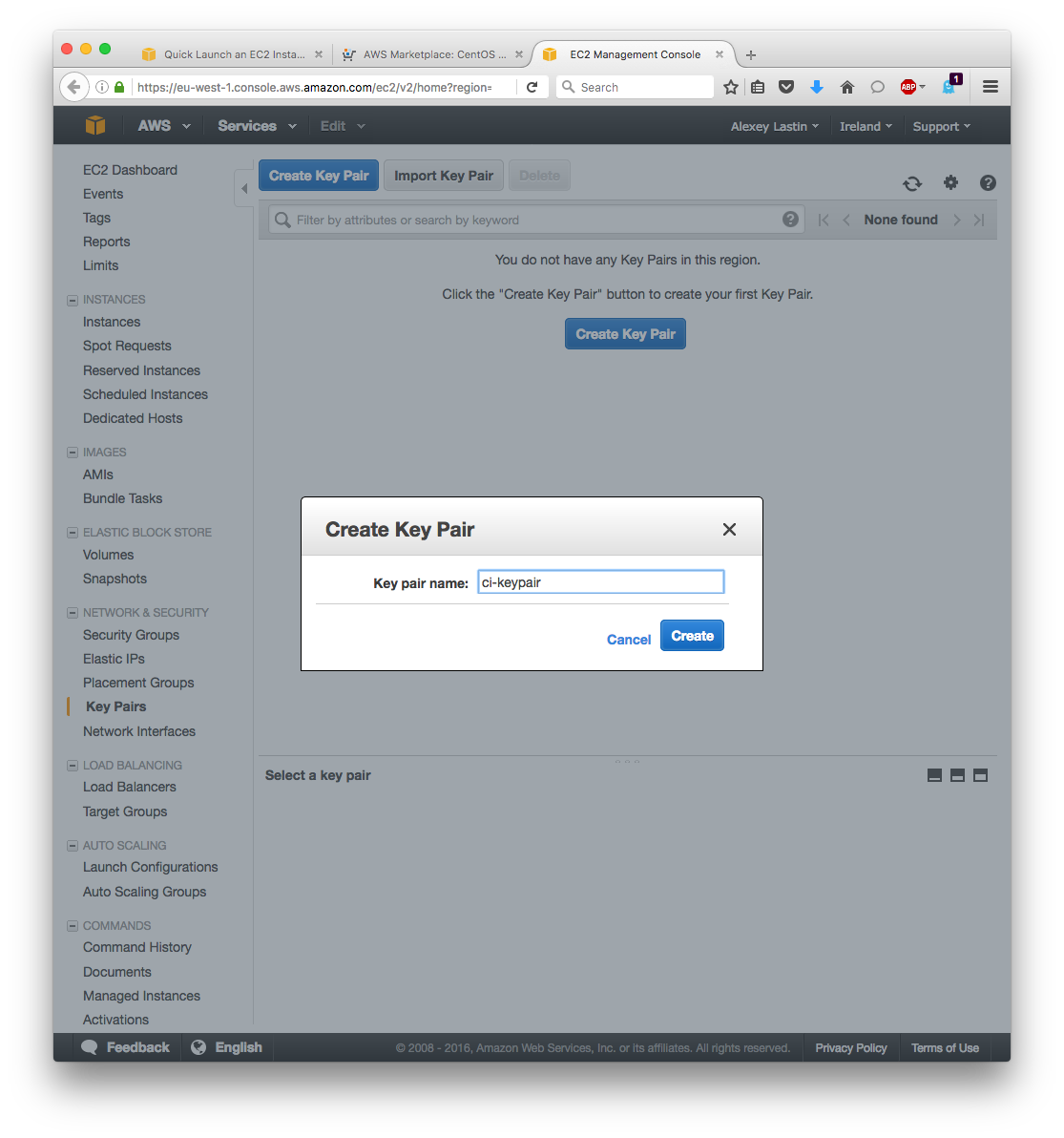
After registering at Amazon Web Services the first thing that needed to be done is to create an instance manually to understand what subcommands I would need to code so that the instance could automatically start and terminate.

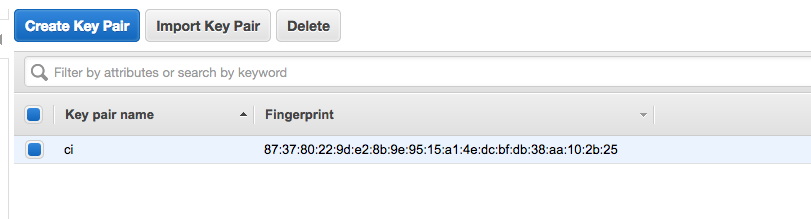
So firstly I had to launch the instance. On the dashboard there was a button called launch instance and so after pressing this button you were put through a few pages of configuration that allowed you to create an instance. You then had to configure the details of the instance’s security group (again from the dashboard). Also the instance required a key pair that had been set up during the launch of the instance.

In my case I used a standard Linux AMI provided by Amazon because it was simple and efficient. I had to specify the operating system, which I chose to be Linux because I could utilise many open source programs. I would then select the region ‘eu west’ because it was the closest region available and instance type ‘t1 micro’ as it easily accessible.

Then I had to create a key pair so that I could access my instance. I then would have to add these key pairs (public and private key) to the security groups and allow outbound traffic so that the instances could use the web. This had been quite successful so the next stage would be to create something similar in a script format.



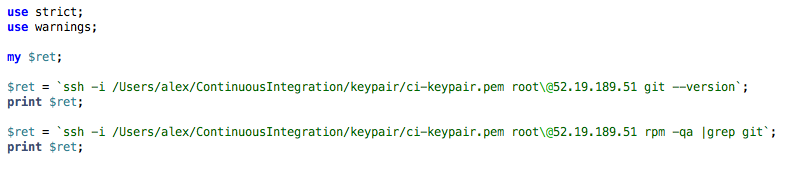




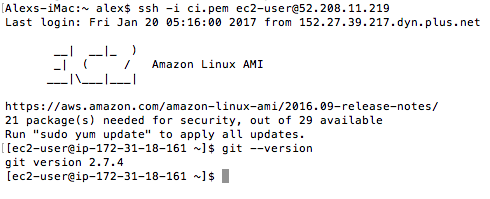
**Prototype related to module 1**

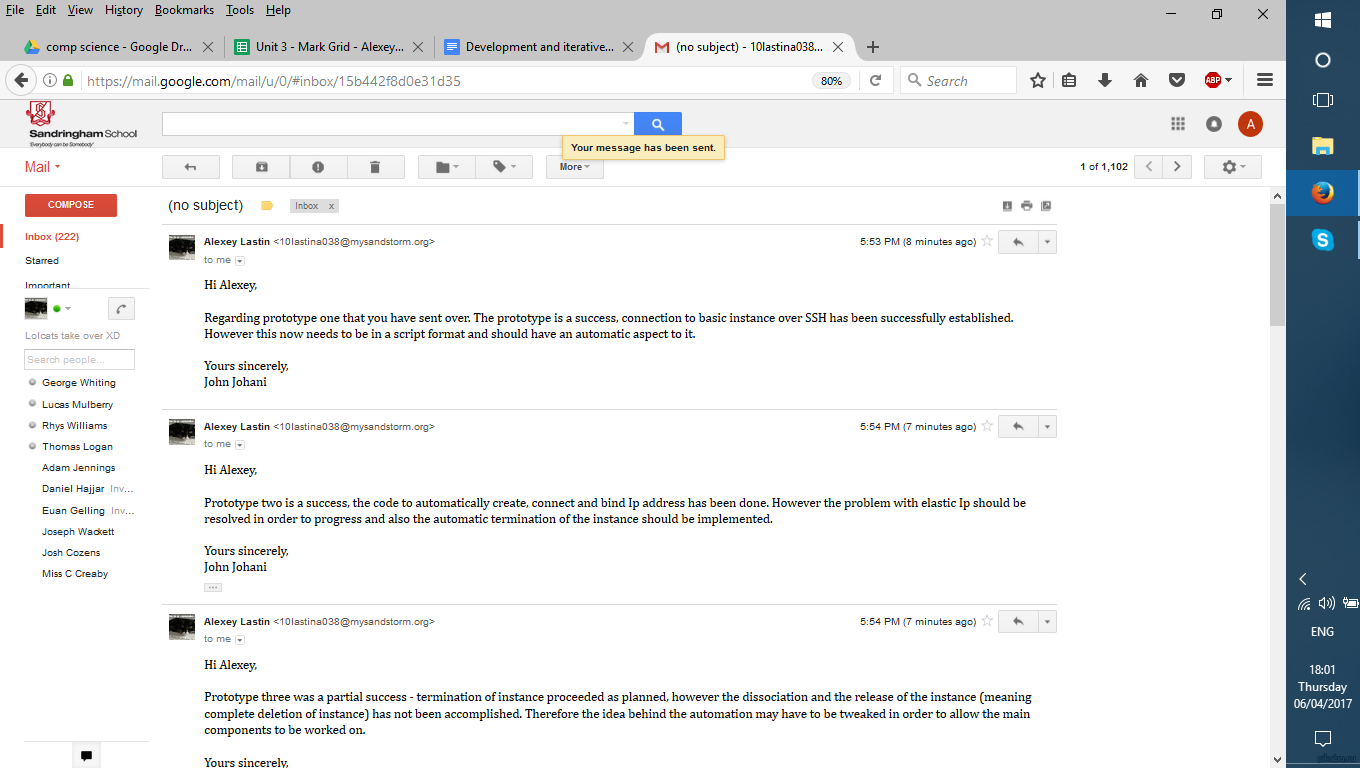
**Wanted outcome -- Use script to connect to instance and then output a result**

The next stage of the development would be to make script that when run would connect to the instance and then output a result to allow for the user to understand that they have connected.



Here I have used SSH to connect to the instance via the IP and key pair saving it into a scalar (variable) and then printing it out onto the local machine. As can be seen from the output (I loaded downloaded Git to see if it would work) - the instance checks for the program in itself and then outputs that the program is there and the version is 2.7.4.

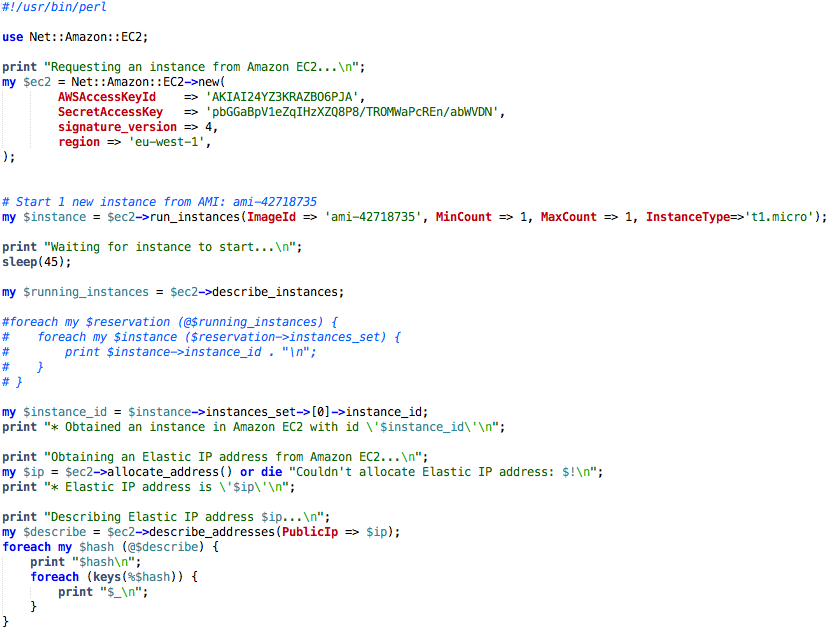
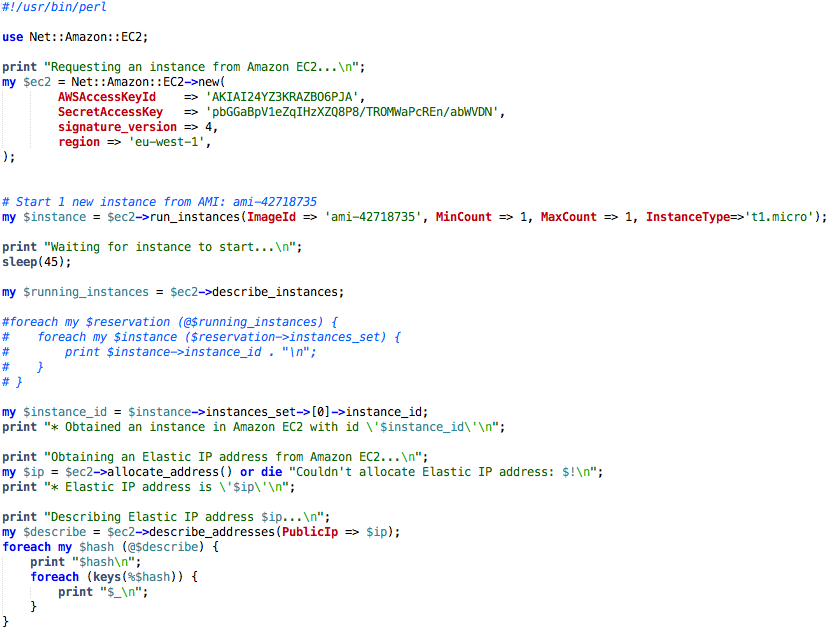




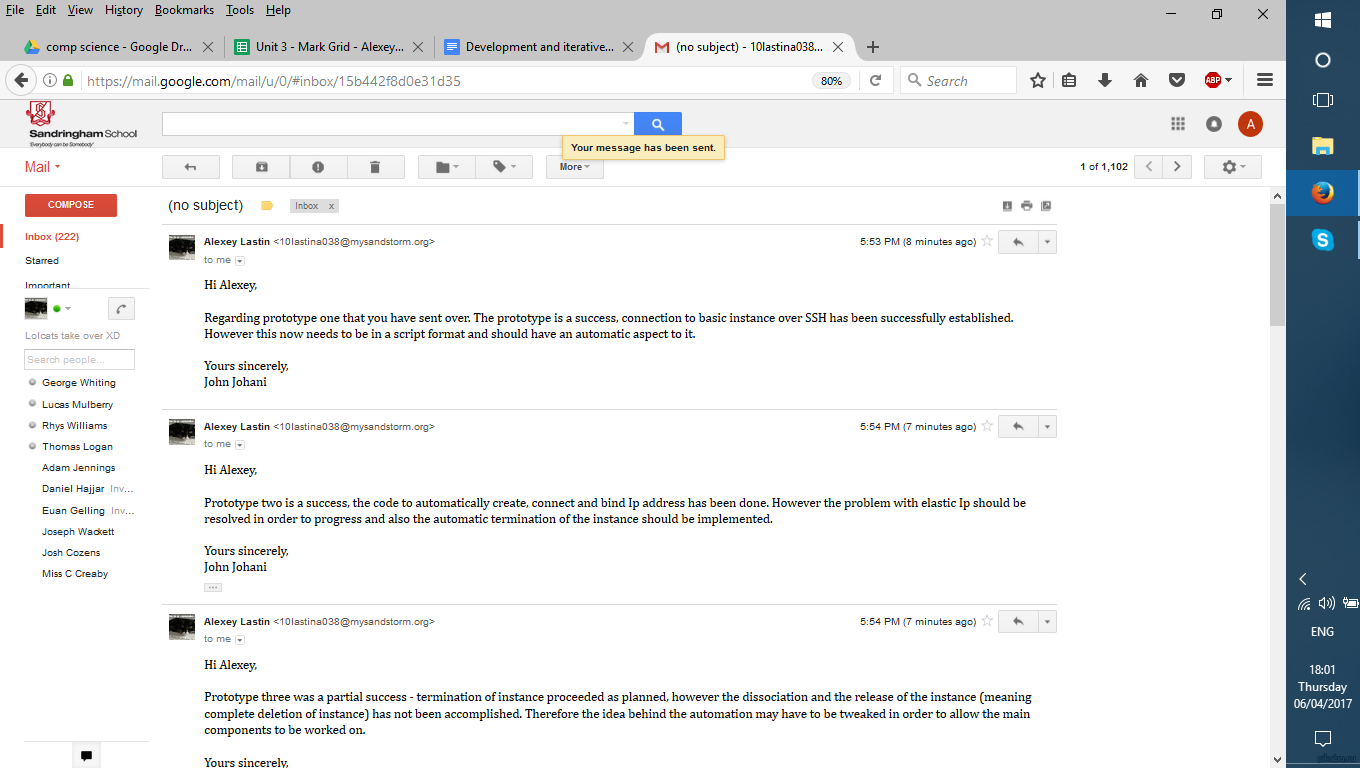
## Module 2 and Prototype 2

**Wanted outcome --Create instance automatically and output a result**

Using a Perl library from the internet (Net::Amazon::EC2) I was able to create some code that would create an instance (define its parameters) and then create an ID for it and associate an address so that I would be able to connect to it just by referring to the ID and IP of the instance. This caused some problems because the first mini iterations of code that I had made didn’t want to create the correct instance and would otherwise be left uncreated because of factors such as the computer trying to associate to an address when the instance wasn’t created (I solved this by giving a time in order to let the instance start) and also the region had to specifically found out on the EC2 website as it wouldn’t let me connect from some of the regions.



However I ran into a problem with the last bit of this piece of code because the elastic IP that is needed in order for the instance to be successfully shut down wasn’t saving as a variable and therefore I could only output it as a string.



## Module 3 Stop and terminate an instance manuallyDisk partition:Users:alex:Desktop:Screen Shot 2017-01-20 at 10.25.44.png

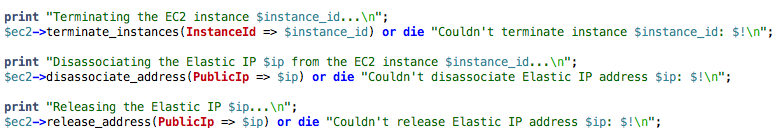
With the creation of the instance now complete it was time to think of a way to close the instance so that a new one could be opened afterwards with the idea being that you could type in a command and the instance should shut down afterwards.

To do this I would require again some knowledge of what the processes are to delete an instance. As it turned out there were two things that needed to be done:

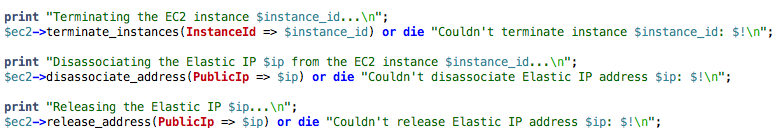
1. Terminate the instance
2. Release the instance

This meant that there would have to be at least two commands in the script that do these things in order to delete the instance.

**Prototype related to module 3**

**Wanted outcome Automatically delete instance** 

The first test that I did on this was to check if I could terminate the instance. This was a success and so I went onto the next stage of deleting the instance.



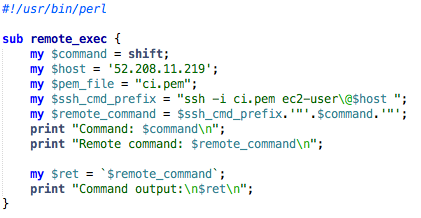
However at this point a there was a problem. Because the instance’s elastic IP address hadn’t saved as a variable

and I wasn’t able find a way of fixing this the instance didn’t want to release after the initial termination part was complete. And so after searching through any materials that could be found I concluded that this wouldn’t work resulting in a partial failure - only one of the two components had worked. Consequently there was a need for a new way of having an instance that would serve the same or a similar purpose (storing the components and then executing them).

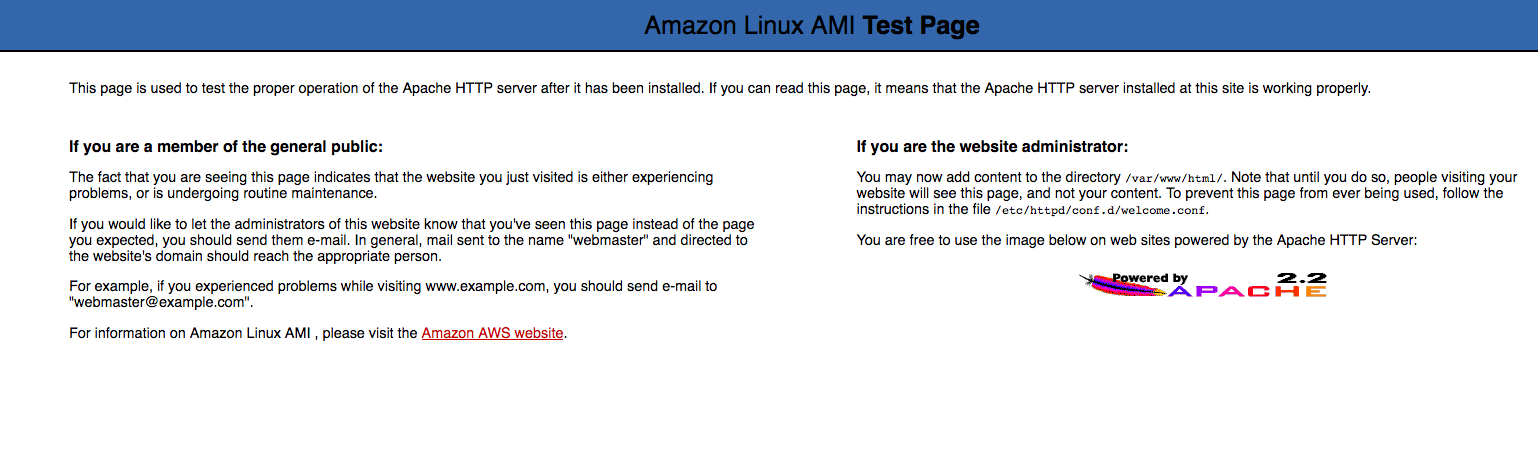
## Module 4 A new instance Idea

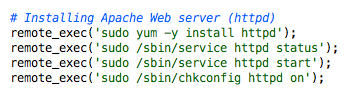
This meant that a new idea about how to use the instances would have to be used, hence deciding to go for the simple yet most reliable idea - I decided to create an instance manually and then leave it until it was no longer needed. Which of course brought some downsides such having to change the code every time someone needed to delete the instance, however unlike the prior instance creator - it worked and so that was a success.

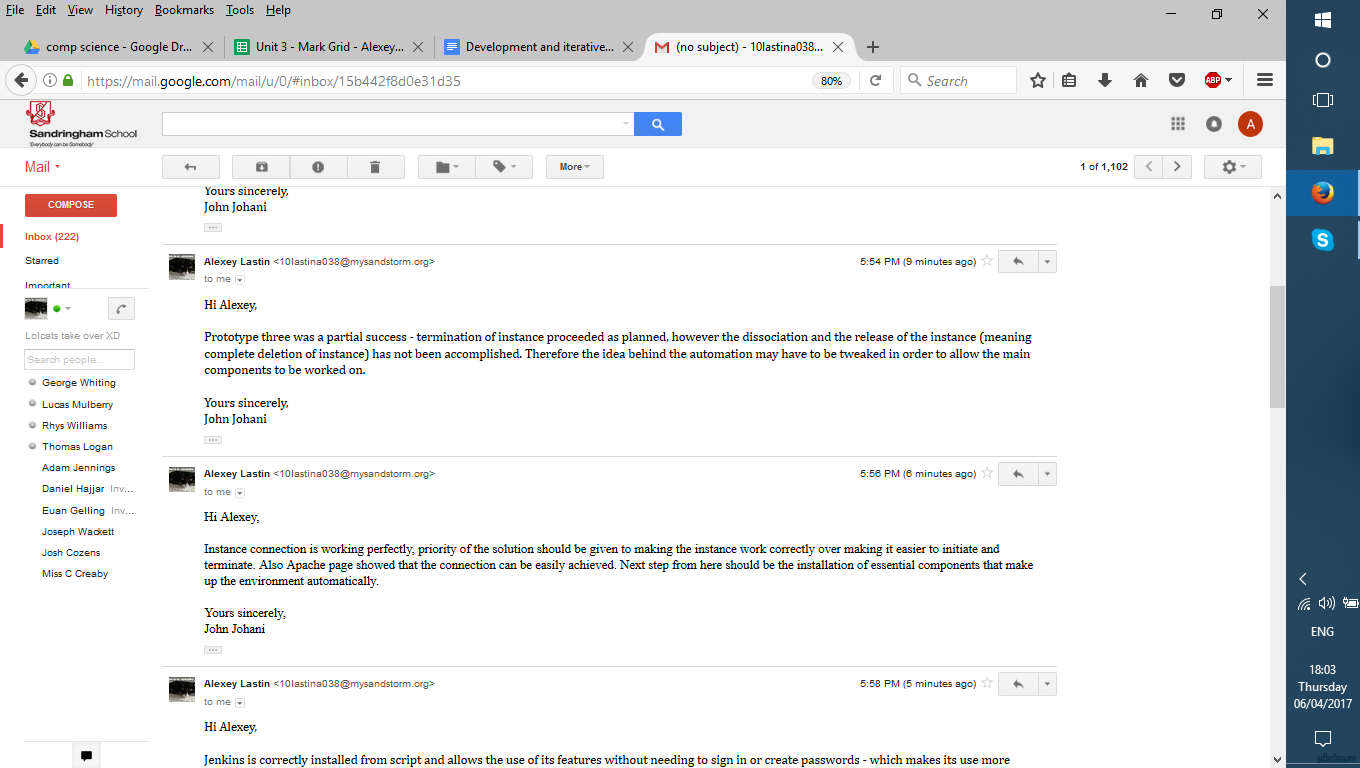
**Prototype related to module 4**

This piece of code is a function that takes several parameters in order to connect to the instance. And so every time ‘remote command’ gets called the computer will connect via the SSH to the instance and execute that order.

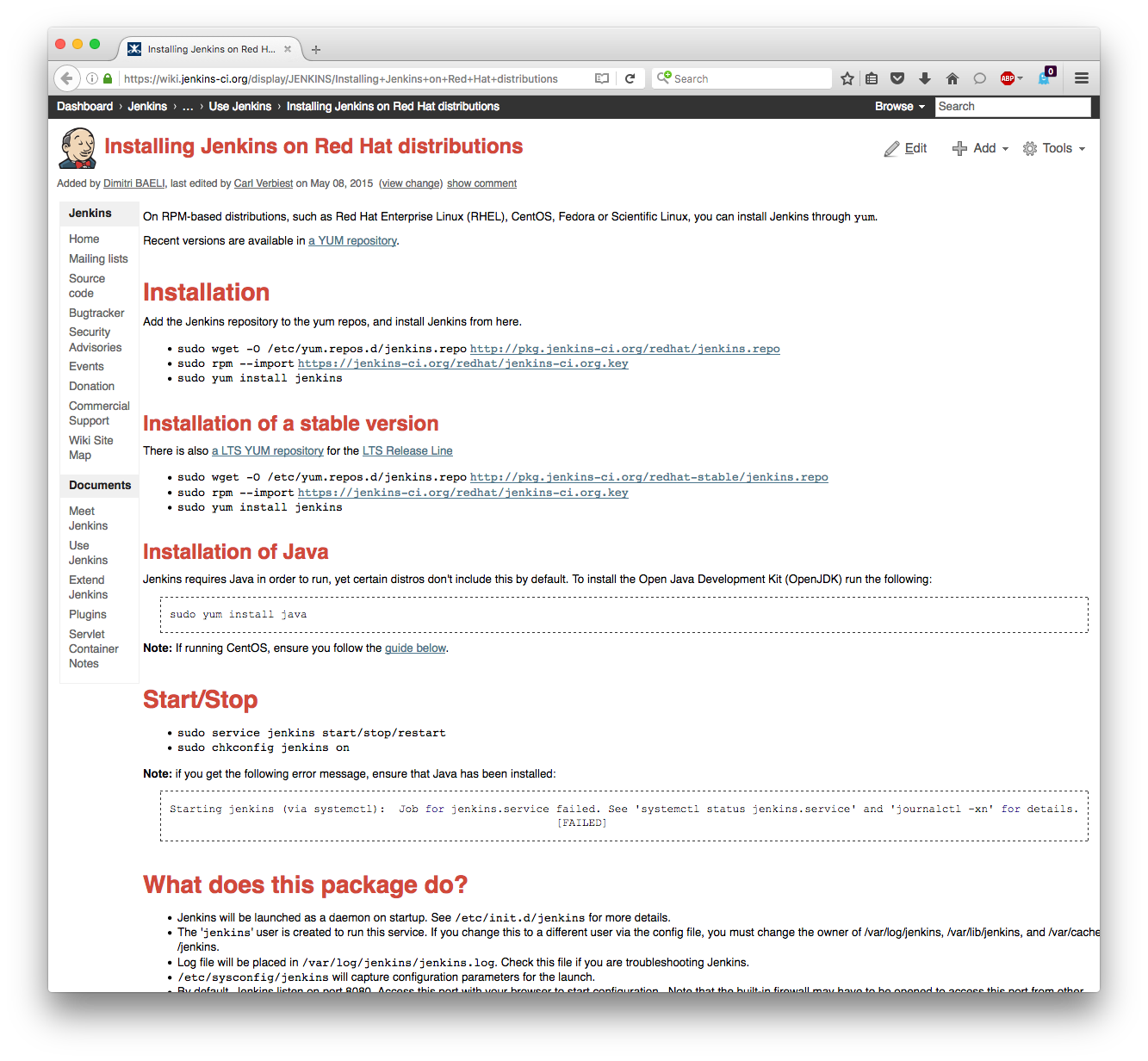
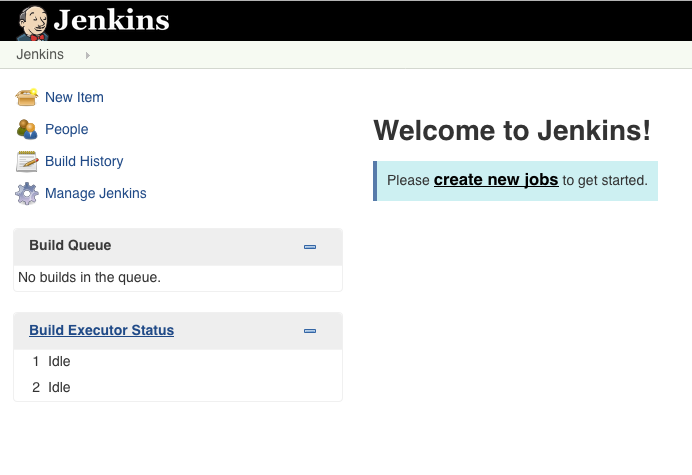
To test out If this has been working correctly I made created an Apache web server. Through this I was able to understand that the Internet was working fine and that there shouldn’t be any problems with using the instance to create servers.



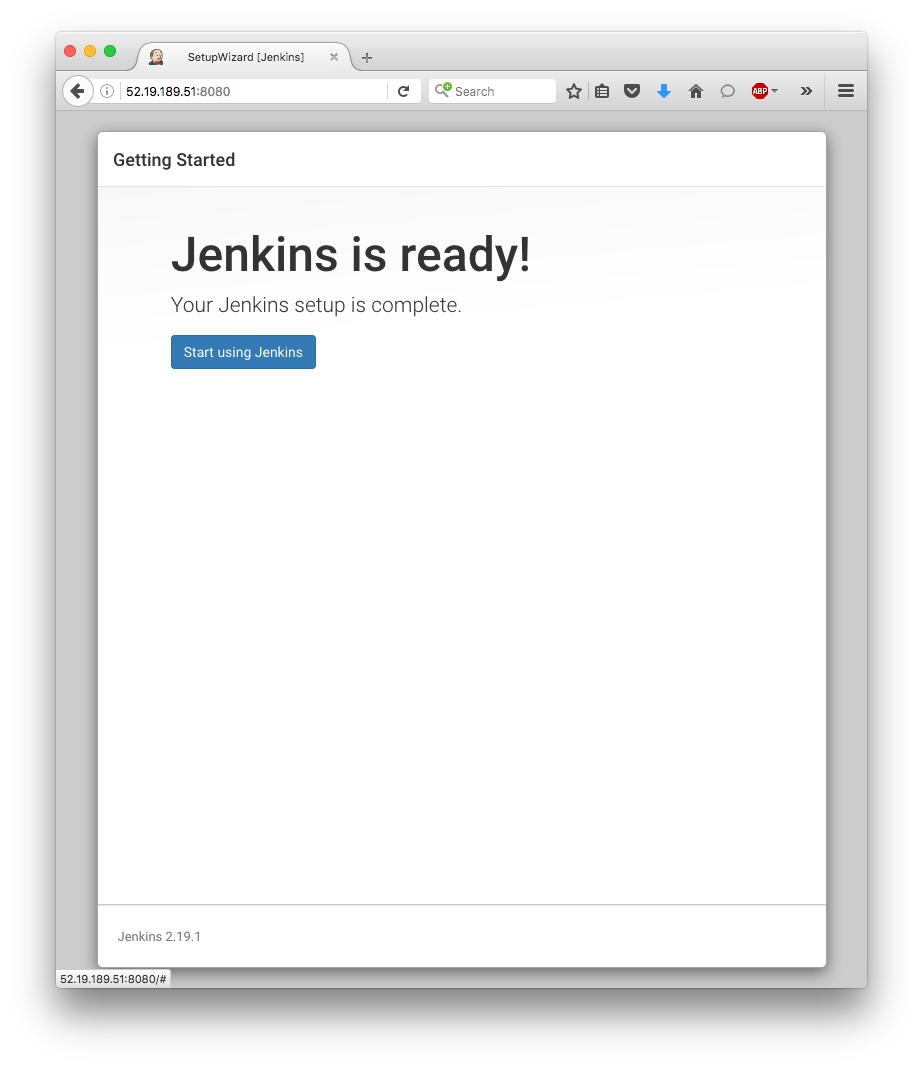


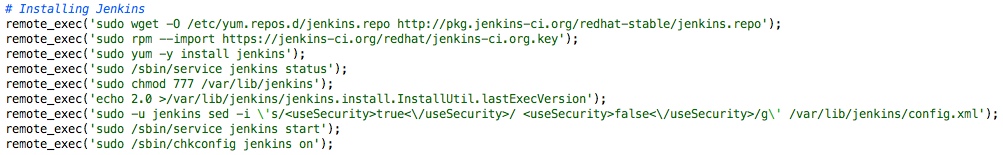


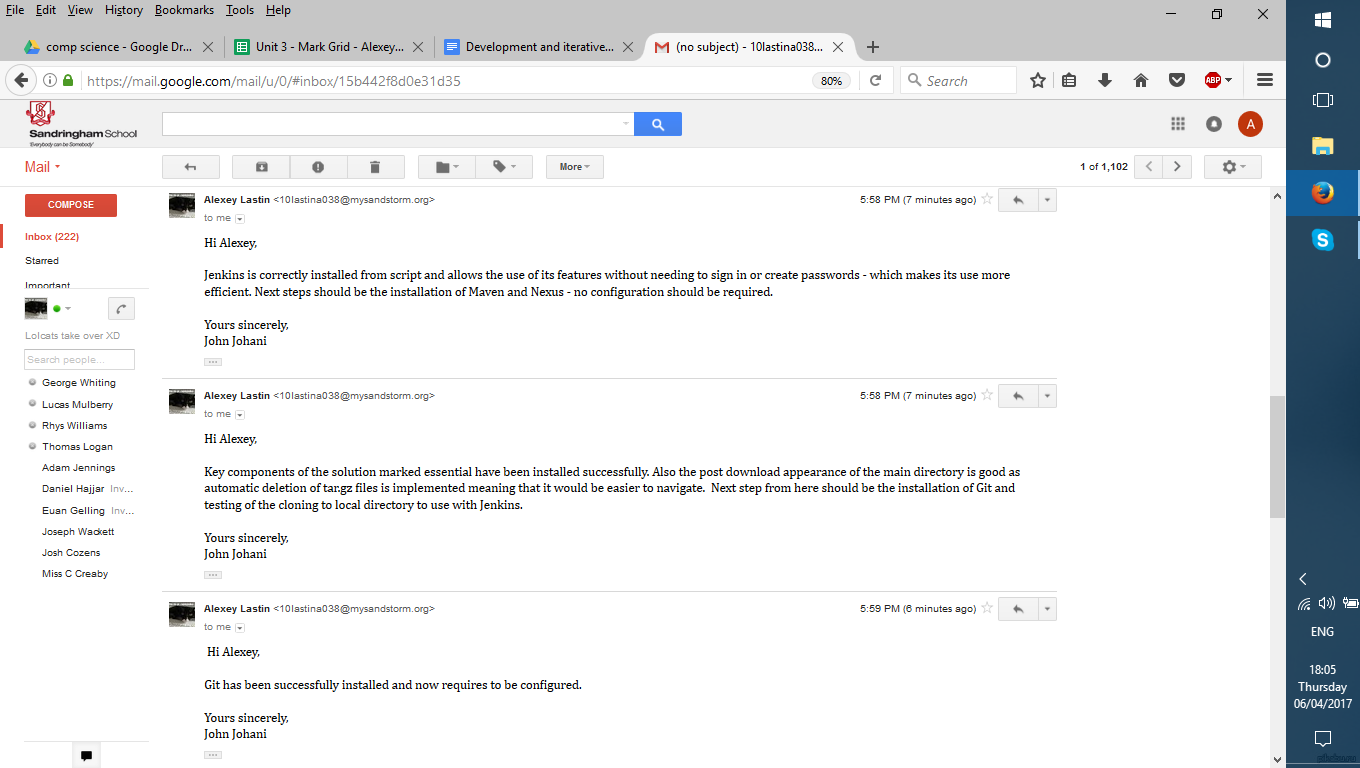
## Module 5 Installing the components of the program -- JenkinsDisk partition:Users:alex:Desktop:untitled folder:Screen Shot 2016-10-15 at 21.51.58.png

The first main code written in this was the code for downloading and installing Jenkins. So using a Perl command to connect to my instance I was able to download the Jenkins files from a repository and then using the rpm and yum commands to install Jenkins onto my instance. After installation Jenkins asked for what plugins to be installed. Having chosen the standard package, as the two main things that were important in that are the Maven and GitHub plugins that would be used by a developer when building a project. Having downloaded and restarted Jenkins I noticed that it wanted a password every time the application was opened. This was the result of one file, which contained security protocols for the Internet not to open an instance of Jenkins. To solve this I edited the file and set it to ‘true’ so that I would be able to start Jenkins.

**Prototype 5 Jenkins test for works**

To check that this part of the code was working correctly I was able to a link to Jenkins which would allow a potential stakeholder to establish connection faster as a result reducing the time and effort required for a potential stakeholder to access Jenkins.

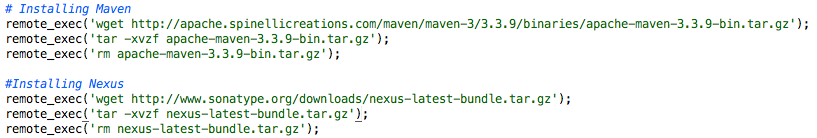




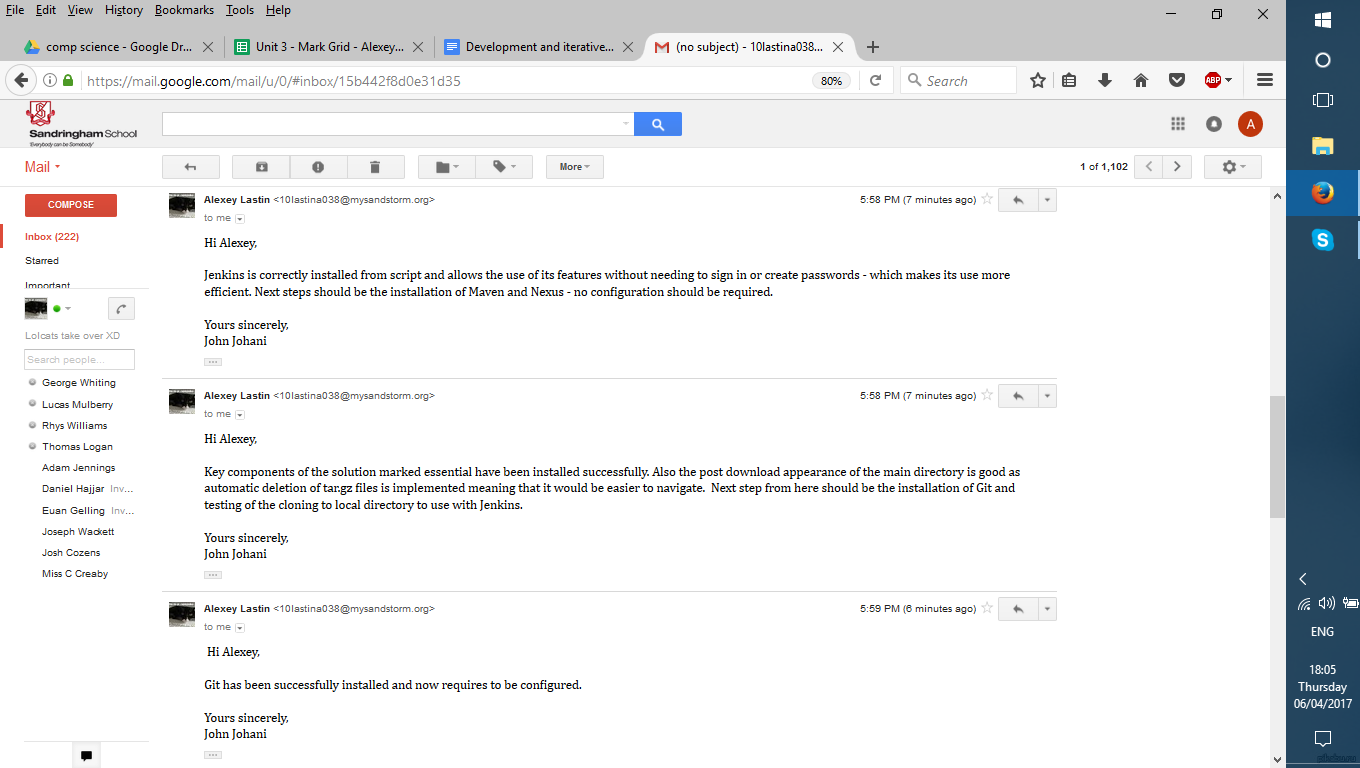
## Module 6 Installing the components of the program -- Maven & Nexus

The next step was to download & install Maven and Nexus. The steps required for both of them to be installed were very similar where I downloaded them from an online repository, unzipped them and then removed the files that were no longer necessary. However to test them I would need to install Git and Gerrit as I won't be able to build a project if I don’t have the necessary files from a GitHub project for example.

**Prototype 6 Installing -- Maven and Nexus**



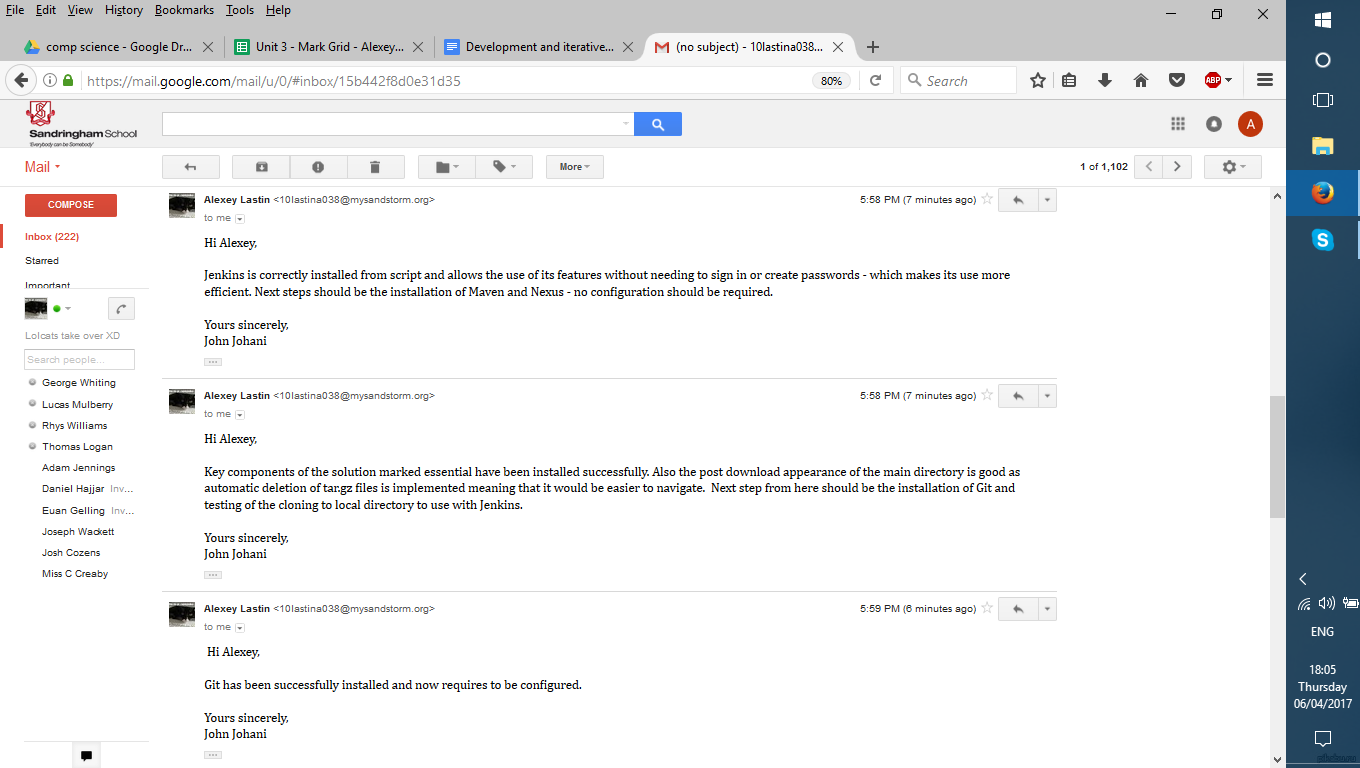
Here is the prototype of the installation of Maven and Nexus. The prototype was successful and has been able to install ‘apache-maven-3.3.9’ and ‘nexus-2.14.2-01’.

Disk partition:Users:alex:Desktop:Screen Shot 2017-01-20 at 13.32.08.png

**Prototype 7 Installing -- GIT**

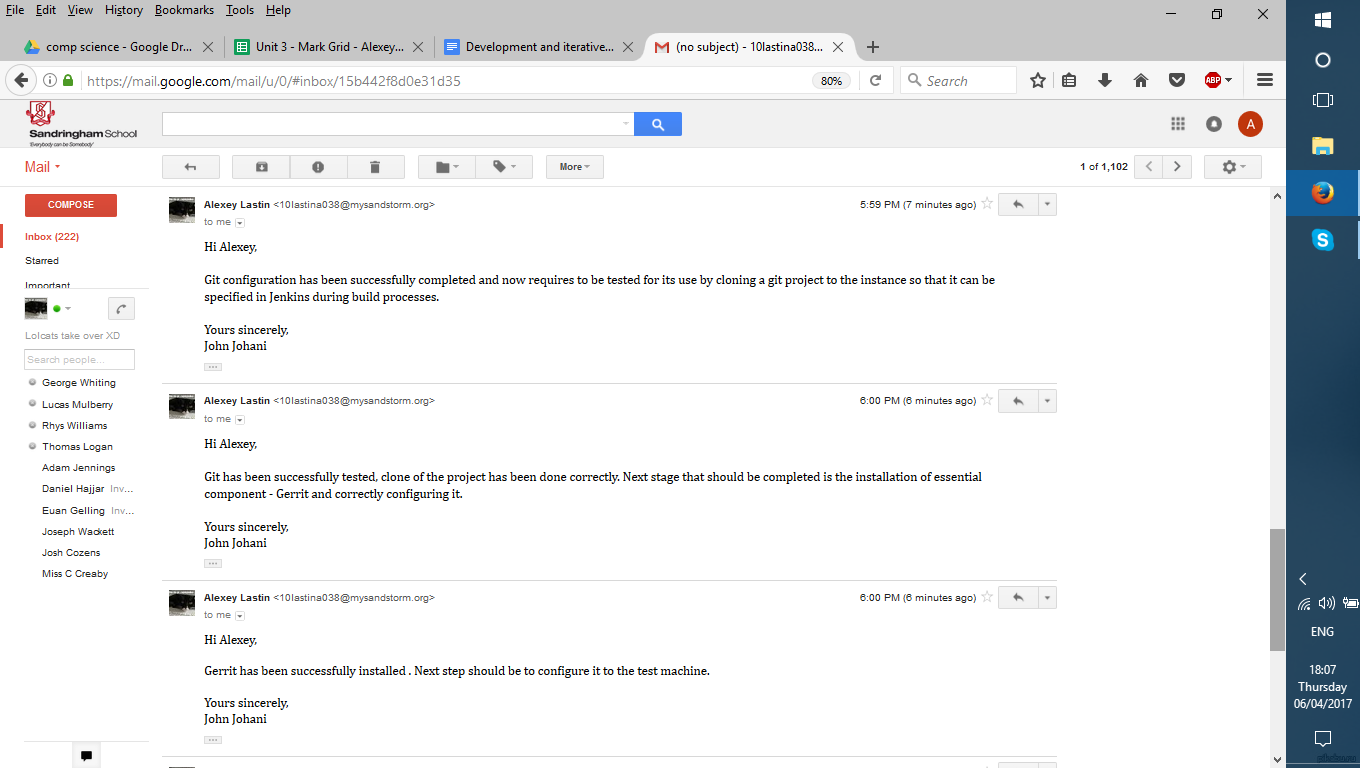
The next component that would have to be installed was Git. As Gerrit is a program that fundamentally requires code to be able to its functions and so Git is required to provide this code. (The repository with code has to be there for code review to be available).





**Prototype 7.5 Configuring -- GIT**

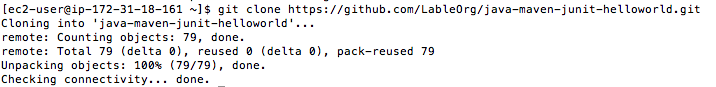
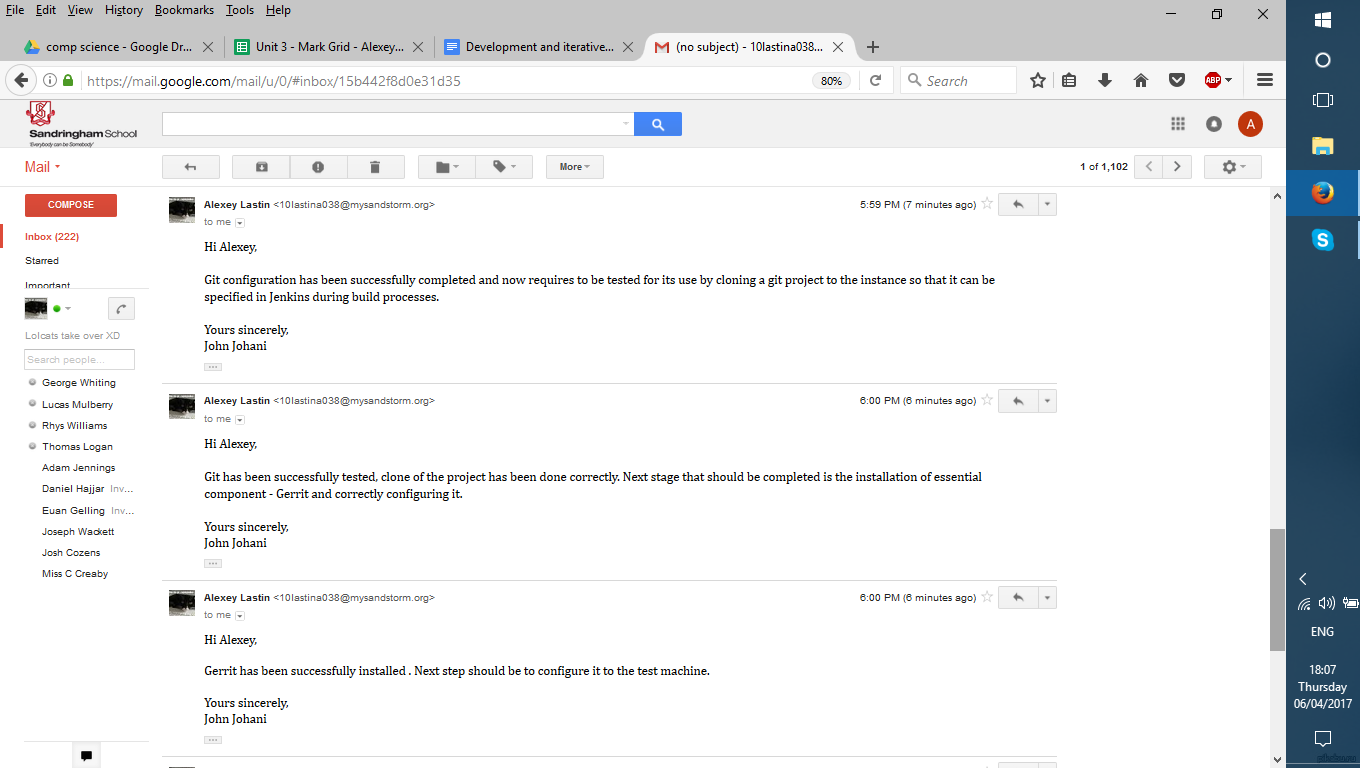
So firstly Git is to be downloaded and then installed. Then Git has to be configured so that it can be connected to via SSH to copy the required repository to the local machine. Generating public keys, which would allow the user to connect, does this. This would allow Gerrit to be used to check the code. To do this, Git has to be used to create a SSH key that would allow a user to connect - ‘ssh keygen’. Then save the key in a location of the users choosing and sign in as a ‘agent’ to through that ip address.



**Prototype 7.5.1 Testing -- GIT**

To test that git was installed and that I could clone the repository I copied a github project to my local computer. The result was successful.

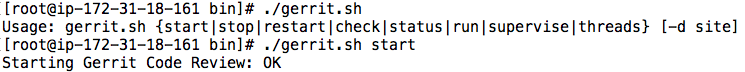
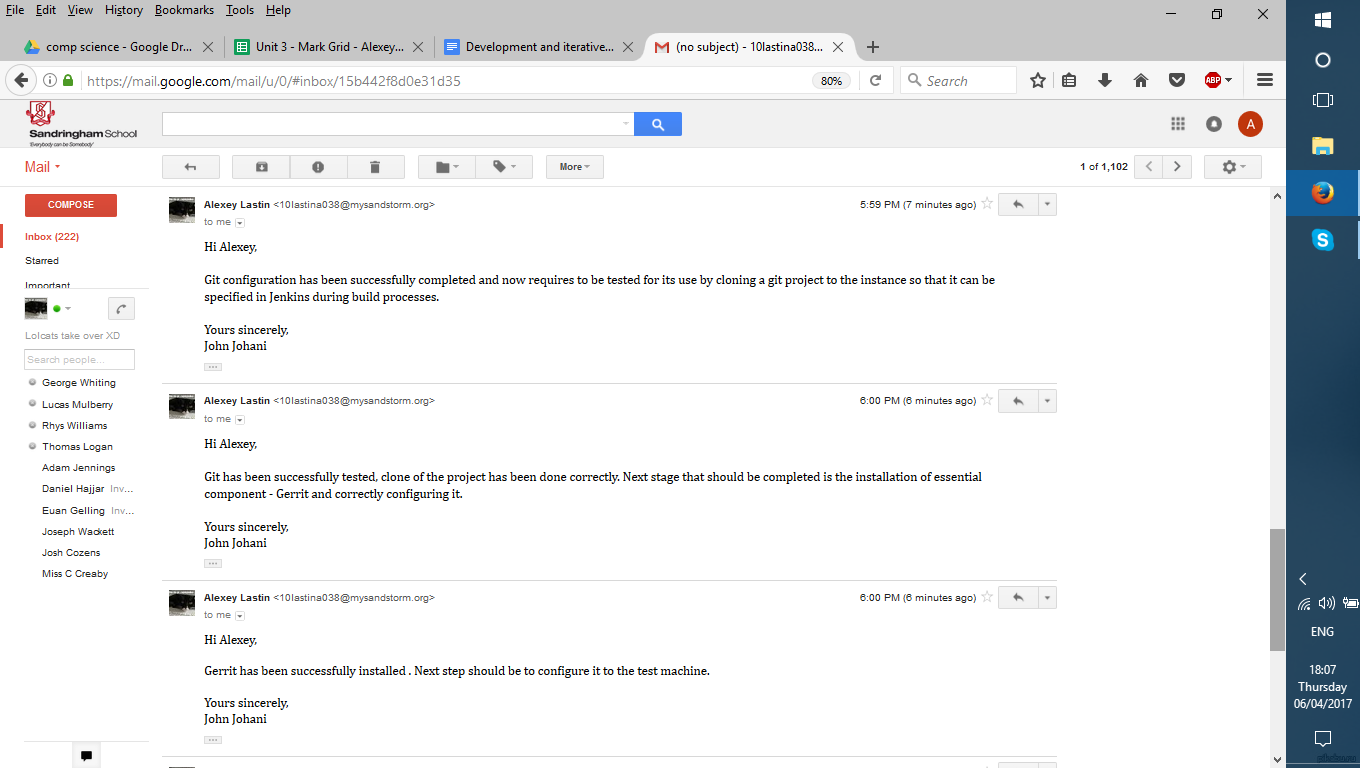
The test had shown that the program was capable of cloning the repository to the local machine.Disk partition:Users:alex:Desktop:Screen Shot 2017-01-22 at 19.46.11.png



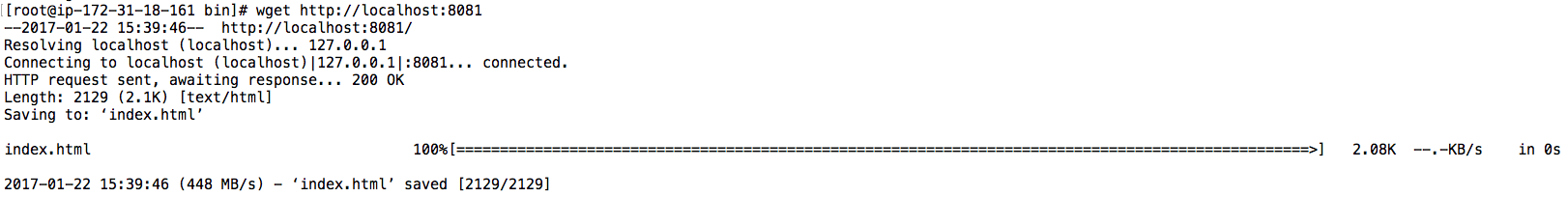
## Prototype 8 Installing -- Gerrit

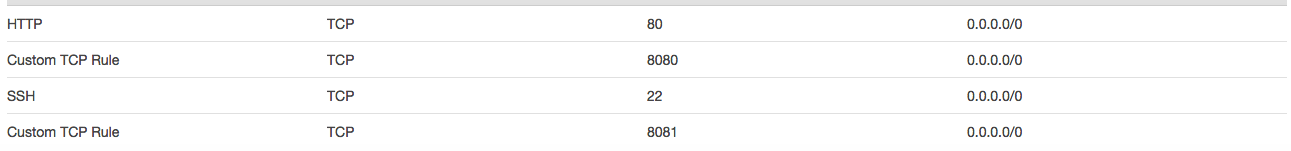
The next download that was required was gerrit. Here as I created a separate user to use gerrit to make it simpler to navigate between the files.

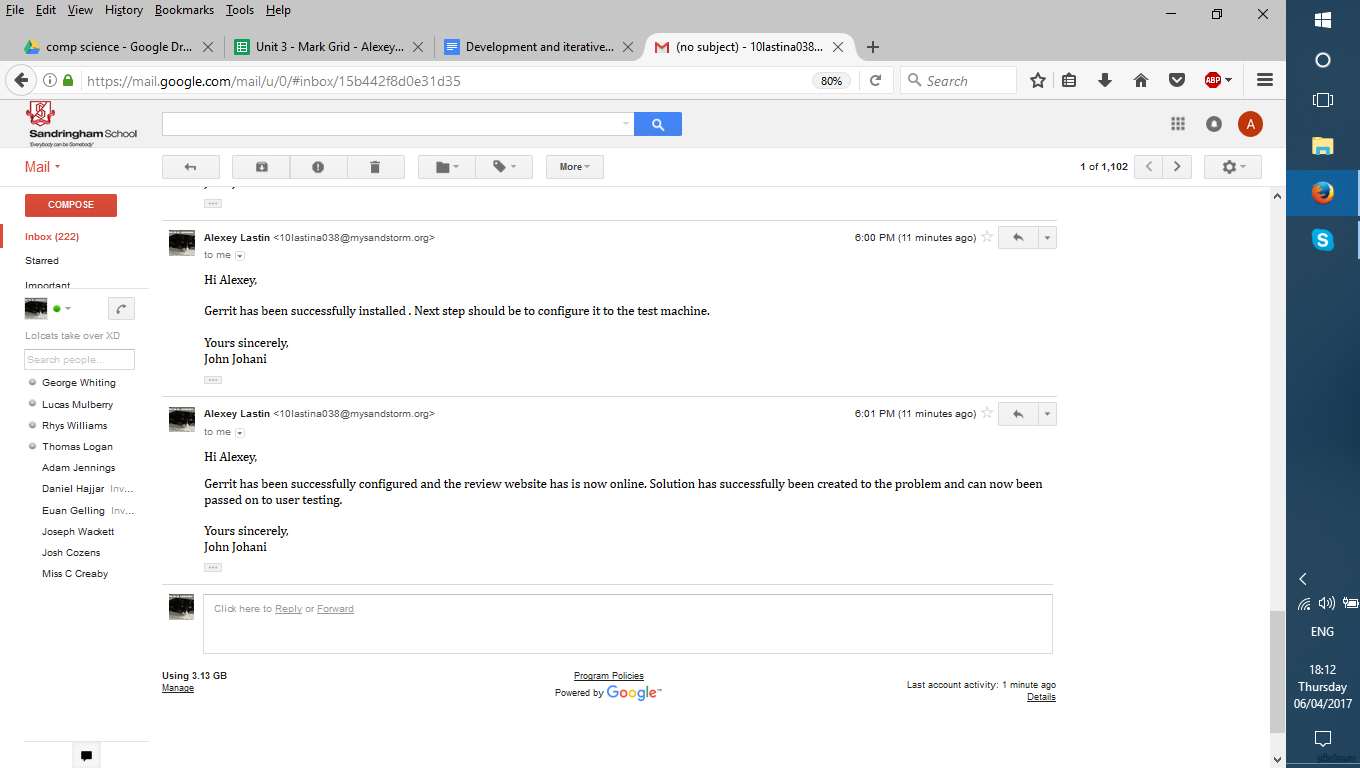
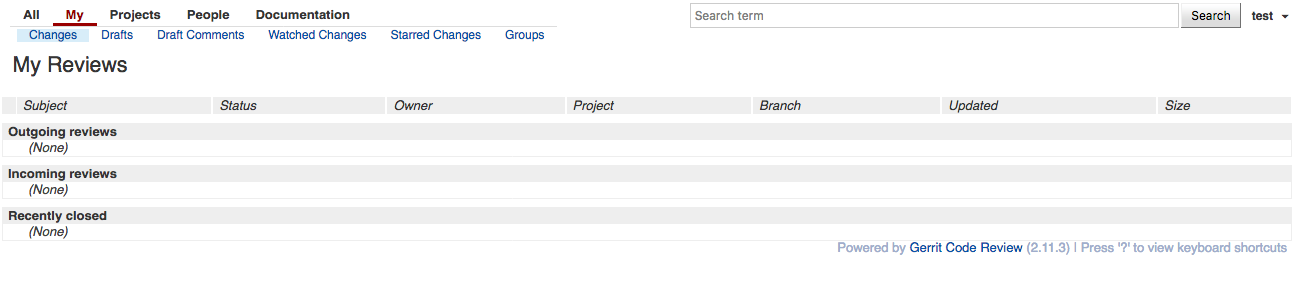


**Prototype 8.5 Configuring -- Gerrit**

After being downloaded Gerrit asked for the ip and port through which it was supposed to look for so that the website could be made. Having already put Jenkins at port 8080 I put Gerrit at the next one 8081.





The end result of this was also successful as I was able to get the site up and running. Hence making the Continuous Integration environment that would be ready for developer use.

**Development bibliography:**

**Main ec2 code to provision (link to source below)**

<http://search.cpan.org/dist/Net-Amazon-EC2/lib/Net/Amazon/EC2.pm>

**Cloud service**

<https://aws.amazon.com/>

**Gerrit –**

<https://www.gerritcodereview.com/>

**Jenkins –**

<https://jenkins.io/>

**Iterative Development Test Data**

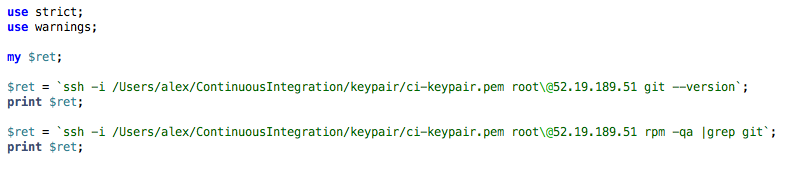
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **What testing** | **How testing** | **Test data** | **Good/Bad/ Boundary** | **Expected result** | **Actual result** | **Actions needed** |
| 1 | Instance creation | Using AWS website to manually set up instance | Input manually | If instance is created result is good | Good | Good | Create script to connect to instance |
| 2 | Instance creation | Using script to set up connection to AWS | Small script that connects to instance via SSH and then outputs a result | Result is good if instance is connected to or instance is not connected to | Good | Good | Create Instance automatically and output result |
| 3 | Create Instance automatically and output result | Create script that sets up connection | Script that starts instance and binds IP | IP is bound to instance automatically or IP not bound to instance | Bad | Good | Terminate instance |
| 4 | Terminate instance manually | AWS website to terminate instance | Manual input | Good if instance terminated or instance not terminated | Good | Good | Automatically terminate instance |
| 5 | Terminate instance automatically | Small script to terminate instance | Script to terminate instance | Instance terminated or instance is not terminated automatically | Good | Good | Disassociate address from instance automatically |
| 6 | Disassociate address from instance | Script to disassociate address | Script | If script disassociated, then good if not then bad | Good | Bad | Debug script for further testing |
| 7 | Disassociate address from instance | Script to dissociate address | Script | Same conditions as previous test | Good | Bad | Script isn’t working as needed – remove automatic start of instance |
| 8 | Use script to connect to instance and directly issue commands | Script that connects to instance and outputs results | Script to test connectivity and then output data | If script can connect and install Jenkins its good if not bad | Bad | Good Script connects and installs Jenkins | Script to make a local version of Jenkins that runs with local host |
| 9 | Script to make a local version of Jenkins that runs with local host | Script that installs Jenkins and then creates a local host at port 8080 | Script that test connectivity | If Jenkins webpage can be opened at port 8080 without a password, then | Good | Bad | Debug – password required still |
| 10 | Apache to test connectivity by opening a webpage | Script that installs and opens Apache | Script to test connectivity | If Apache script opens webpage, then good | Good | Good | Fix bug / incorrectness in main scripts |
| 11 | Debug – password required still | Script that opens Jenkins without password at port 8080 | Script to test connectivity | If Jenkins webpage opened at 8080 without password, then good | Good | Good - Jenkins webpage opened without password | Move onto installing other components -Maven |
| 12 | Install Maven from script | Script that downloads and installs maven | Script set up  Maven | If maven is downloaded and installed, then good | Good | Good – maven installed | Move onto installing next components – Nexus |
| 13 | Install Nexus from script | Script that downloads and installs Nexus | Script to set up Nexus | If Nexus is downloaded and installed, then good | Good | Good | Move onto Git |
| 14 | Install Git from script | Script that downloads and installs Git | Script to set up Git | If Git is downloaded and installed, then good | Good | Good | Configure Git so that it can is connected via SSH to Wikimedia |
| 15 | Connect instance to Wikimedia and output result | Generate SSH keys and connect to Wikimedia | Script to connect to Wikimedia and clone a repository from GitHub | If script connects to Wikimedia then good, better if GitHub repository is copied to local machine | Good | Very good | Clone git repository to local machine |
| 16 | Clone a git repository to local machine | Use git to clone repository | Command to clone repository to local machine | If command copies repository and saves locally it’s a success | Good | Good | Install gerrit |
| 17 | Download and install gerrit | Download gerrit and create review site | Script to download gerrit | If gerrit is downloaded then its good | Good | Good | CI environment complete |

# Testing: Evaluative

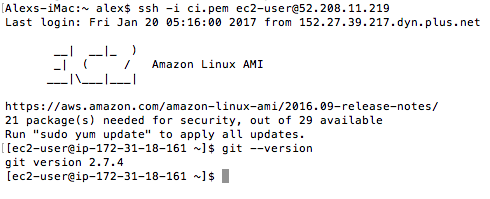
## Final Testing Evidence

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature name** | **Description** | **Priority** | **Evaluative conclusion** | **Refer to test data** |
| **Command line interface and parameters handling**  1 Parsing and passing command line parameters to main application | Basic functionality to pass parameters from user to the application | 1  Essential | Command line interface is implemented and work efficiently. Passing and parsing parameters to the main application work. | 1 and 2 |
| 2 Reading and handling configuration files from file system | Ability to read and use common configuration parameters which tend to stay the same for majority of the application runs | 1  Essential | Ability to use common config parameters is successful | 1 and 2 |
| **Network and connectivity**  1 Connectivity to Amazon EC2 environment | Ability to connect to the target environment where the applications will be deployed | 1  Essential | Connection to environment is secure and works correctly | 2 |
| 2 Create Amazon instance programmatically using Amazon API | Ability to automatically provision the servers on demand | 3  Minor | Partial completion of feature – automatic instance creation was successful however the termination doesn’t work correctly. | 2 |
| **Code to provision: Git** | | | |  |
| 1 Automate downloading of Git package from repository, available in the internet | The way of obtaining the binary package for source control system, essential for the solution | 1  Essential | Git package downloads correctly and installs Git from the internet | 3 |
| 2 Functionality to deploy and install Git app | Implementation of automated install of the source control system | 1- Essential | Deploying and installing Git works correctly | 3 |
| 3 Automated creation of Git repository | Creation of test / demo repository | 2  Major | Demo repository test successfully completed | 4 |
| **Code to provision: Maria DB** | | | |  |
| 1. Automate download of installation package from public repository | Obtaining the installation package from the internet automatically | 1 Essential | Automatic download of Maria DB is a failure. Stakeholder decide to remove feature from success criteria. | 5 |
| 2 Automate installation of Maria DB | Installation of database | 1 Essential | Automatic installation of Maria DB is a failure. Stakeholder decide to remove feature from success criteria. | 5 |
| **Code to provision: Gerrit** | | | |  |
| 1 Automate download of Gerrit | Obtaining the installation package from the internet automatically | 1 Essential | Automatic download of Gerrit is successful. | 8 |
| 2. Create a database in Maria DB to be used by Gerrit | Creation of data storage for Gerrit app | 1 Essential | Maria Db database creation was unsuccessful. | 5 |
| 3.Create accounts for Git/Gerrit in Maria DB | Setting up a test / demo / admin account for Git / Gerrit | 1 Essential | Git and Gerrit accounts in Maria Db unsuccessful. | 5 |
| 4.Create a test project in Gerrit | Demo project in Gerrit | 3 Minor | Demo project in Gerrit was successful | 8 |
| **Code to provision: Jenkins** | | | |  |
| 1 Automate downloading of the package | Obtaining the installation package from the internet automatically | 1 Essential | Automatic download and installation of Jenkins was successful | 6 |
| 2 Programmatically set up a Jenkins master instance | Set up of continuous build application | 1 essential | Master instance successfully created | 6 |
| 3  Create a test / demo project on Jenkins master | Test / demo project to visualise the results of set up | 3 Minor | Test demo project successful | 6 |
| **Code to provision: Jenkins slave** | | |  |  |
| 1  Automate installation of Jenkins slave | Provision Jenkins slave to implement distributed build functionality | 2 Major | Jenkins slave installation unsuccessful. | 5 |
| 2  Connect Jenkins slave to Jenkins master | Implementation of connectivity between master and slave instances of Jenkins | 2 Major | Connection between Jenkins master and Jenkins slave unsuccessful | 5 |
| **Code to provision: Maven** | | | |  |
| 1.Automate Maven installation | a. Install Java build tool so that it can be used in a continuous build process | 1 Essential | Automation of maven Installation is successful | 7 |
| 3.Update custom Maven configuration to match environment | Change the generic Maven configuration files to match the environment | 1 Essential | Feature has not been successfully created / configured | none |
| **Code to provision: Nexus** | | | |  |
| 1 Automate the download of free Nexus version from Sonatype repository | Getting the Nexus binaries from the vender | 1 Essential | Automated download and installation of nexus successful | 7 |
| 2.Automate Nexus installation | programmatically install the release repository app | 1 Essential | Automated download and installation of nexus successful | 7 |
| 3 Automate creation of test / demo Nexus repository | Setting up a demo release repository | 3 Minor | Automated creation of test demo unsuccessful | none |

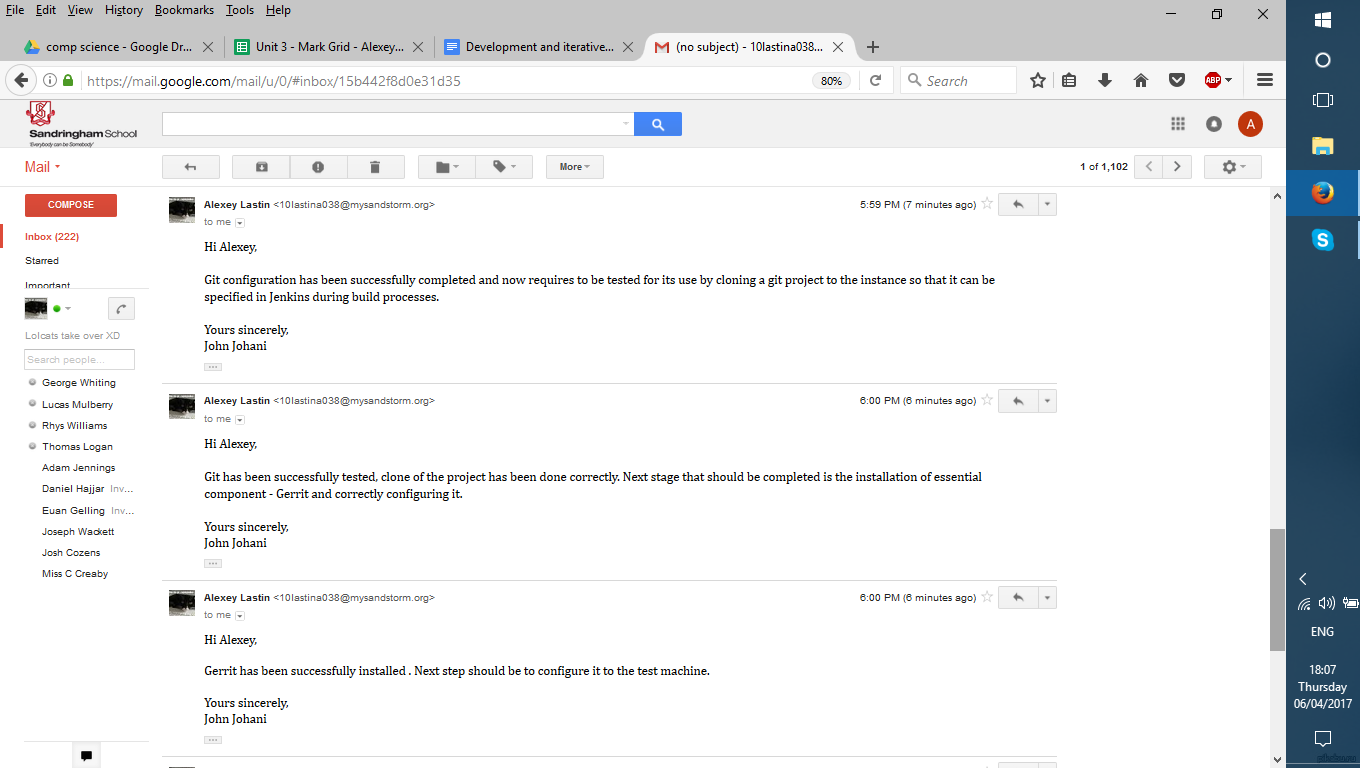
## Evidence of testing data:



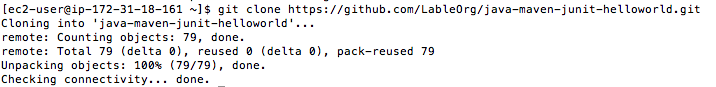
**Figure 1** - data input that tests automatic creation of connection and a test of installing key component GIT

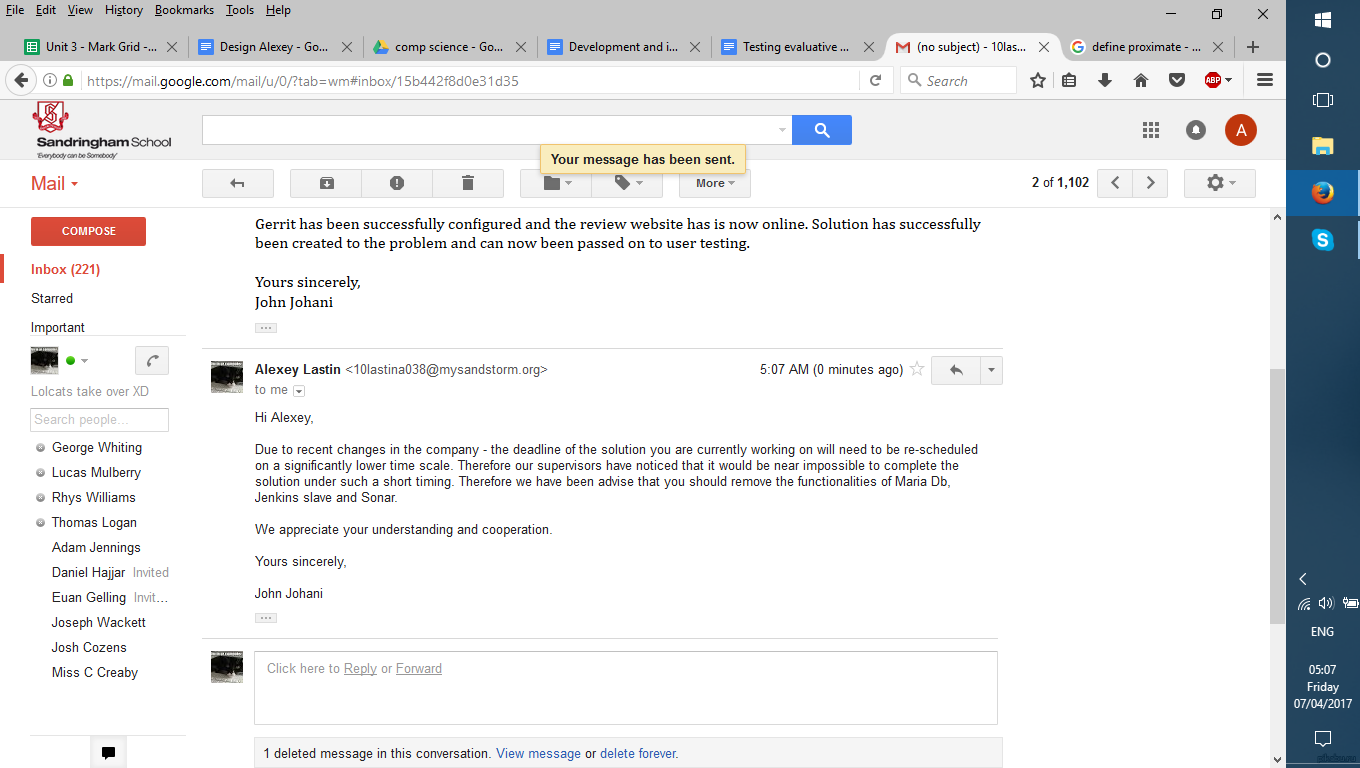


**Figure 2** - (the result of figure 1) testing of command line interface - passing and parsing of parameters to main application, also tests connectivity to AWS server.

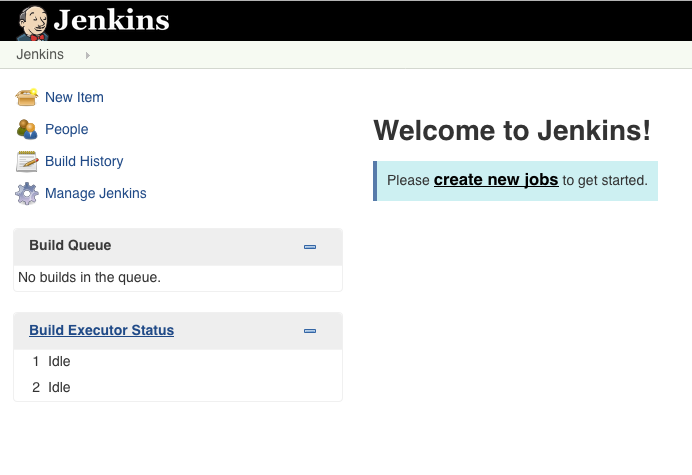
**Figure 3** - User confirmation that Git is installed, configured and now requires to be tested

**Figure 4** - Git testing by deploying local Github project

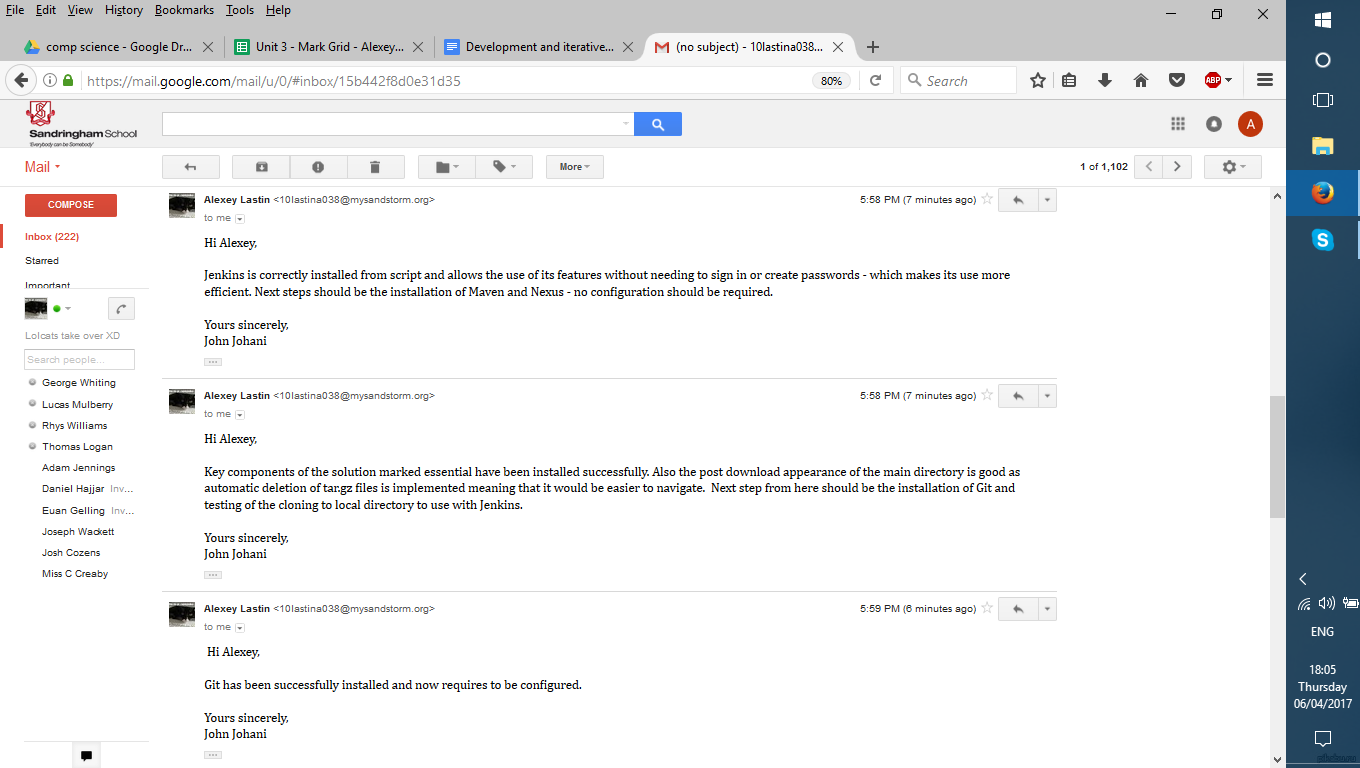




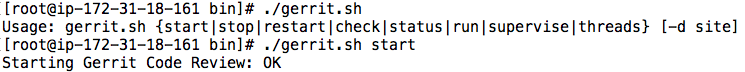
**Figure 5** - Maria Db is taken out from solution along with Sonar and jenkins slave.



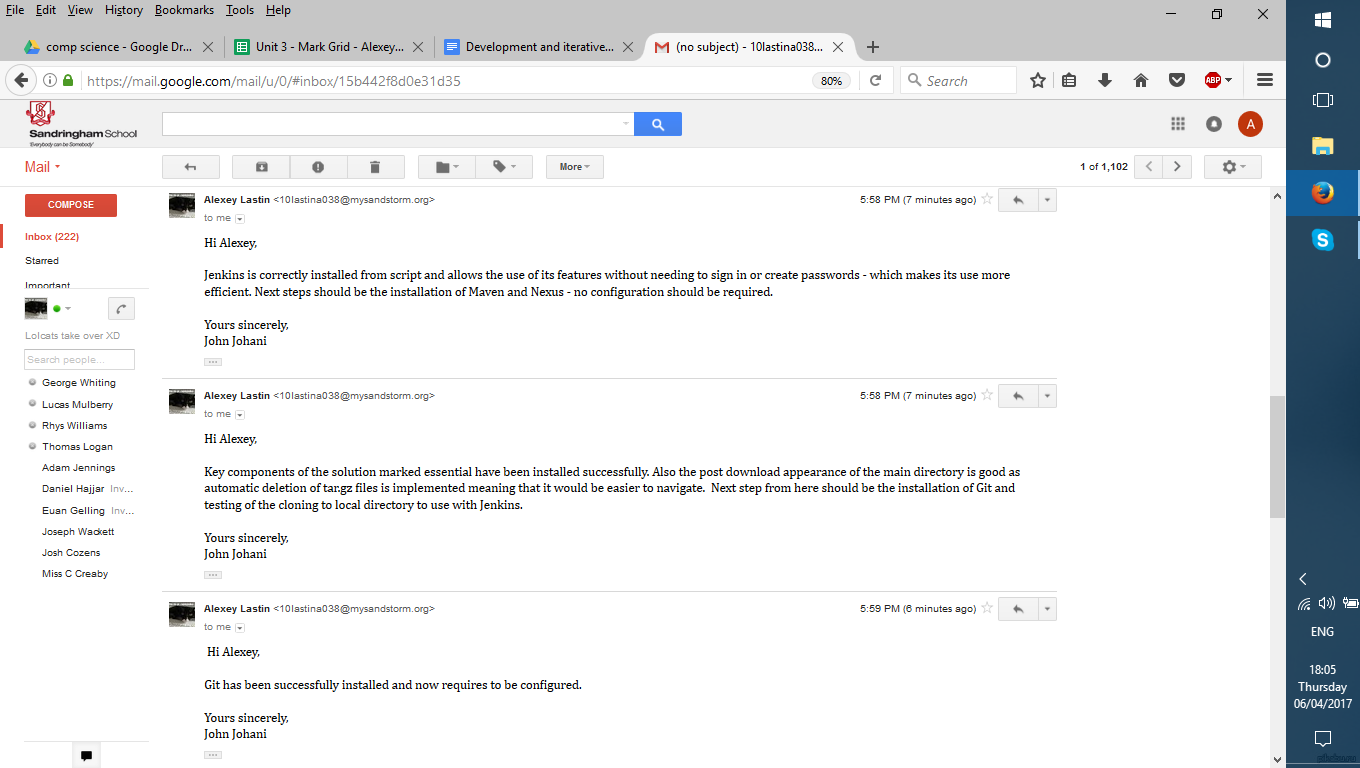
**Figure 6** - Jenkins testing and user review that it has gone successfully.

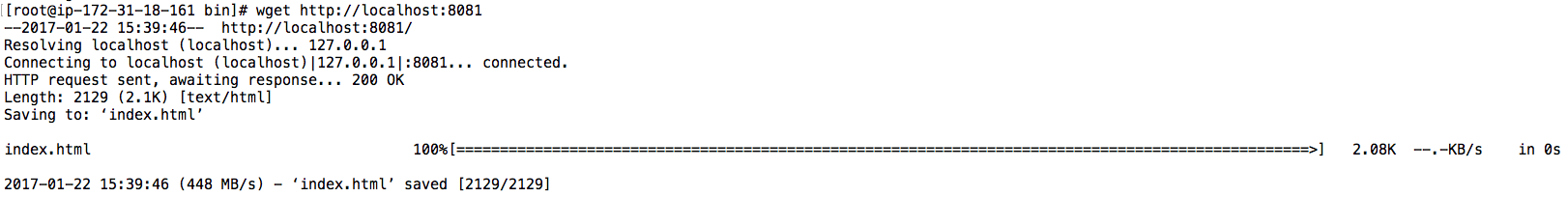


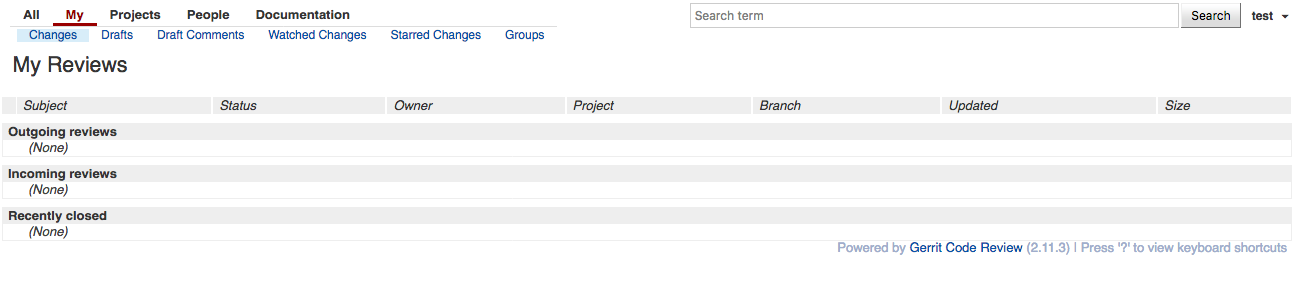
Disk partition:Users:alex:Desktop:Screen Shot 2017-01-20 at 13.32.08.png**Figure 7** - Maven and nexus installation and user feedback confirming it.



**Figure 8** - Gerrit installation and testing with final screenshot of working review site



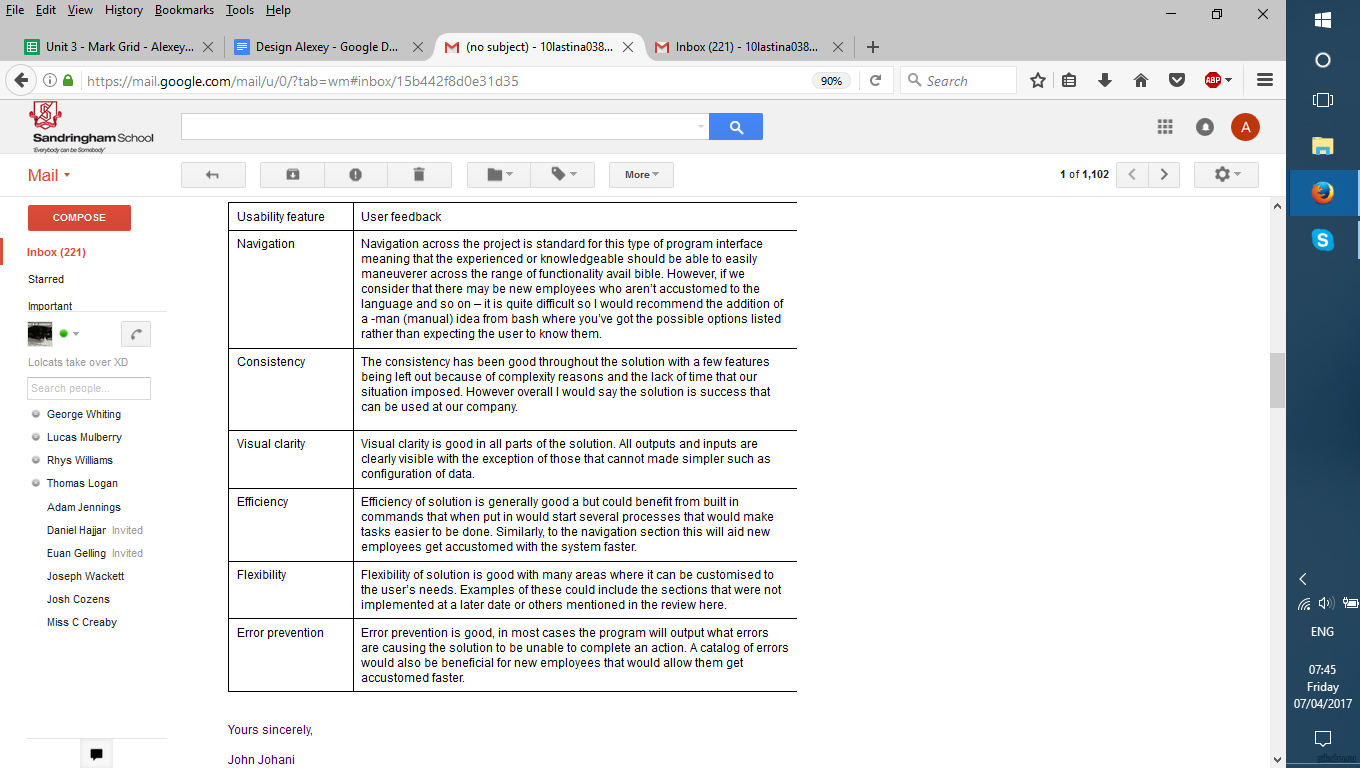


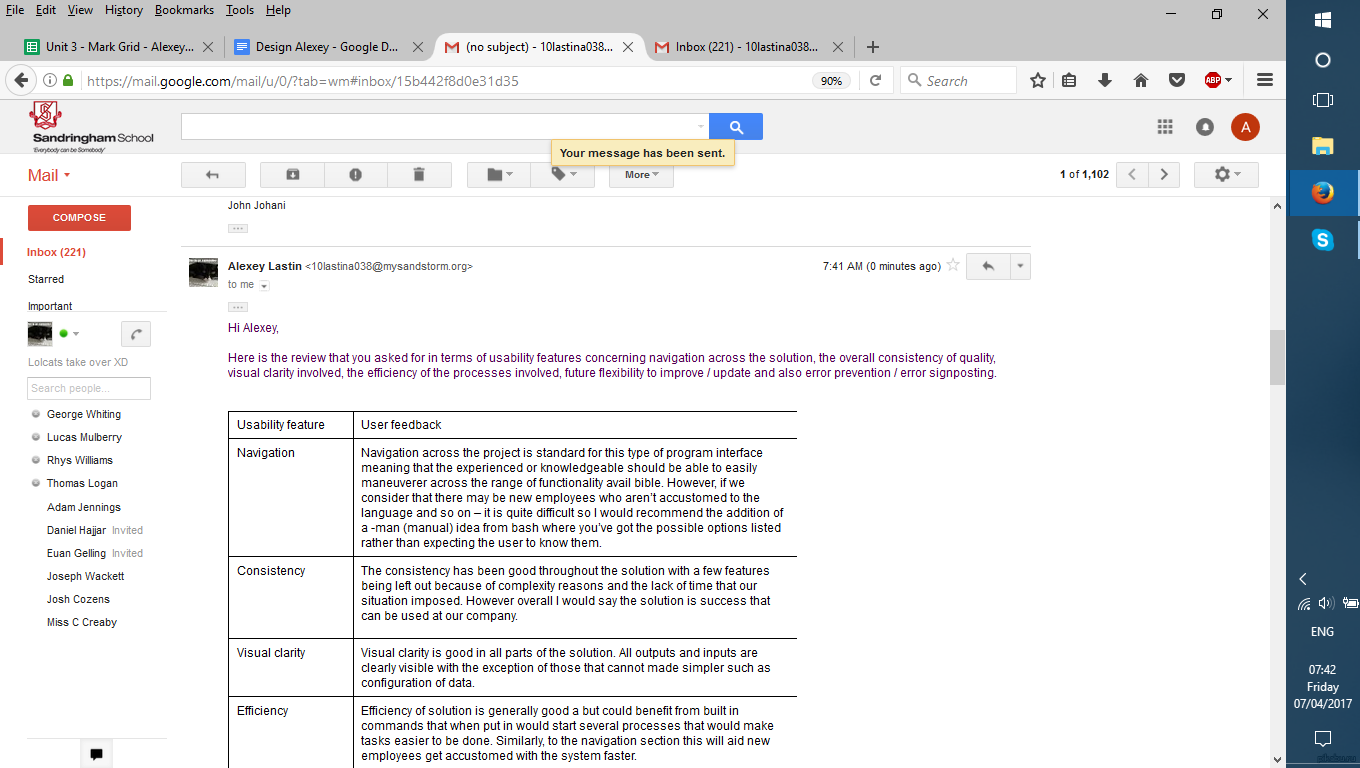


## Usability Testing (supported by evidence below)

|  |  |  |
| --- | --- | --- |
| Usability feature name | Description | User feedback |
| Navigation | This covers the ease of access to Jenkins server, Github, Command line interface, instance connection, Gerrit website and general navigation across the instance. | Navigation across solution is general good, however could benefit from the command line which outlines what actions are available to be done. |
| Consistency | This covers the all-round quality of the solution, so the different features being done correctly and efficiently. | Consistency across solution is quite good, however some features that have not been implemented have made it less consistent. |
| Visual clarity | This covers the ease of finding what the user wants in the program and is closely linked with navigation. This usability feature should be included in all parts of the solution i.e. command line interface – outputs shouldn’t be difficult to read; website should be simple and easy to understand. | Visual clarity is good in all parts of the solution. |
| Efficiency | This covers how quick, easy and simple specific processes can be done with the solution. These could include using Jenkins and Maven to set up a build, using Github to clone a repository to local machine. | Efficiency of solution is generally good a but could benefit from built in commands that when put in would start several processes that would make tasks easier to be done. |
| Flexibility | This covers how easy it would be to change the program should the need arise in the future. | Flexibility of solution is good with many areas where it can be customised to the user’s needs. |
| Error prevention | This covers what is done to take errors to a minimum and what makes troubleshooting them easier. | Error prevention is good, in most cases the program will output what errors are causing the solution to be unable to complete an action. |

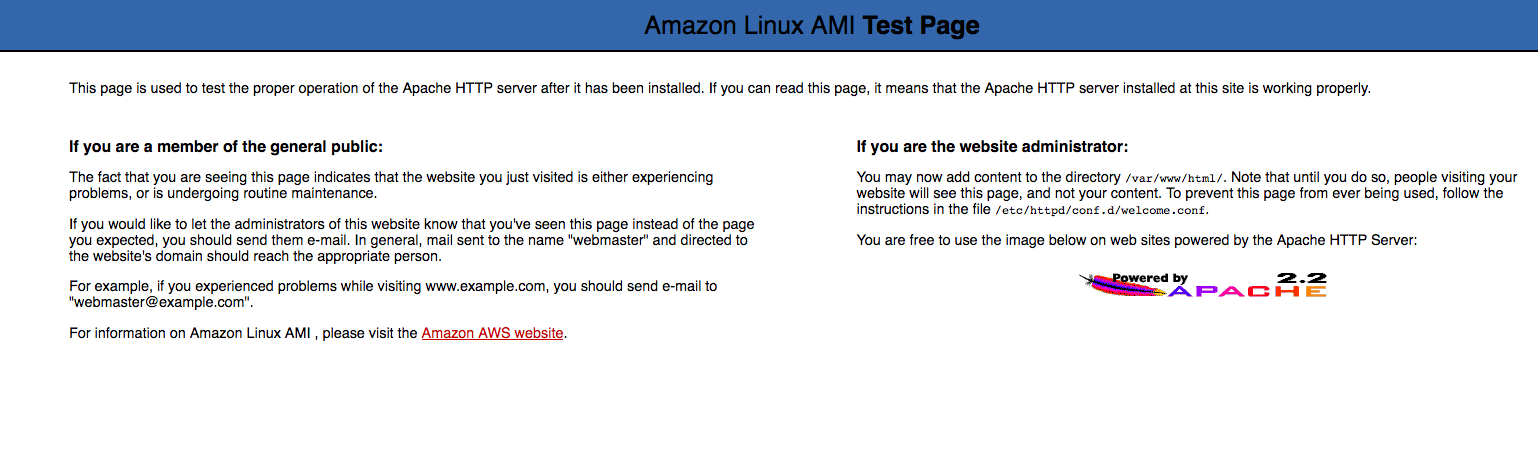
## User testimony for usability features:



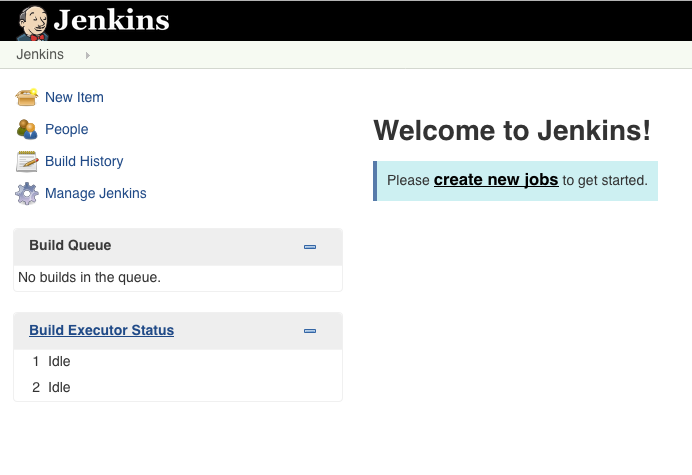


## Screenshots for usability

**1 - connection check**



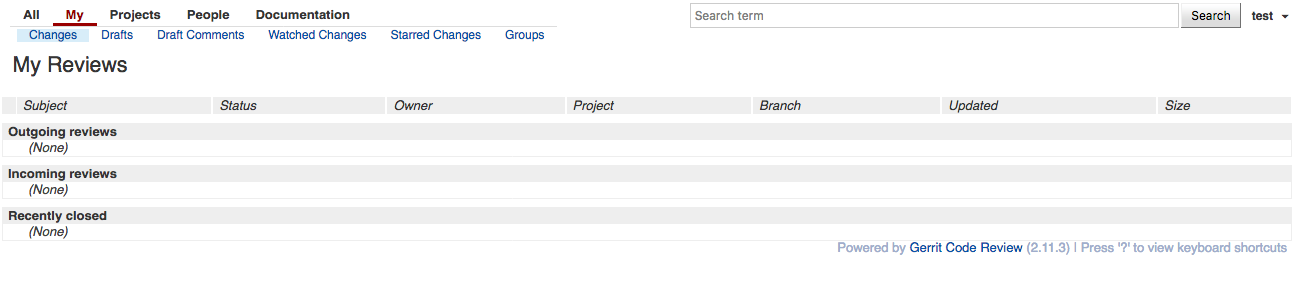
**2 - connection check jenkins**



**3 - files storage locations of environment**

Disk partition:Users:alex:Desktop:Screen Shot 2017-01-20 at 13.32.08.png

**4 - Gerrit**



# Evaluation

## Evaluate Success Criteria (Reference to evidence of testing data)

**Successful and Unsuccessful points of the final solution**

|  |  |
| --- | --- |
| **Successful** | **Unsuccessful** |
| Command line interface is implemented and work efficiently. Passing and parsing parameters to the main application work. | Partial completion of feature – automatic instance creation was successful however the termination doesn’t work correctly. |
| Ability to use common config parameters is successful | Automatic download of Maria DB is a failure. Stakeholder decide to remove feature from success criteria |
| Connection to environment is secure and works correctly | Automatic installation of Maria DB is a failure. Stakeholder decide to remove feature from success criteria. |
| Git package downloads correctly and installs Git from the internet | Maria Db database creation was unsuccessful. |
| Deploying and installing Git works correctly | Git and Gerrit accounts in Maria Db unsuccessful. |
| Demo repository test successfully completed | Jenkins slave installation unsuccessful. |
| Automatic download of Gerrit is successful. | Connection between Jenkins master and Jenkins slave unsuccessful |
| Demo project in Gerrit was successful | Feature has not been successfully created / configured |
| Automatic download and installation of Jenkins was successful | Automated creation of test demo unsuccessful |
| Master instance successfully created |  |
| Test demo project successful |  |
| Automation of maven Installation is successful |  |
| Automated download and installation of nexus successful |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature name** | **Description** | **Priority** | **Has criteria been achieved? Justification** |
| **Command line interface and parameters handling**  1 Parsing and passing command line parameters to main application | Basic functionality to pass parameters from user to the application | 1  Essential | Command line interface is implemented and work efficiently. Passing and parsing parameters to the main application work.  Hence the feature has been fully implemented.  ( |
| 2 Reading and handling configuration files from file system | Ability to read and use common configuration parameters which tend to stay the same for majority of the application runs | 1  Essential | Ability to use common configuration parameters was successful. Therefore, the feature has been fully completed. |
| **Network and connectivity**  1 Connectivity to Amazon EC2 environment | Ability to connect to the target environment where the applications will be deployed | 1  Essential | Connection to environment is secure and works correctly. Feature has been successfully implemented in full. |
| 2 Create Amazon instance programmatically using Amazon API | Ability to automatically provision the servers on demand | 3  Minor | Partial completion of feature – automatic instance creation was successful however the termination didn’t work correctly. |
| **Code to provision: Git** | | | |
| 1 Automate downloading of Git package from repository, available in the internet | The way of obtaining the binary package for source control system, essential for the solution | 1  Essential | Git package downloads correctly and installs Git from the internet. Therefore, the feature is fully successful. |
| 2 Functionality to deploy and install Git app | Implementation of automated install of the source control system | 1- Essential | Deploying and installing Git works correctly. Feature is fully successful. |
| 3 Automated creation of Git repository | Creation of test / demo repository | 2  Major | Demo repository test successfully completed  Feature is fully successful. |
| **Code to provision: Maria DB** | | | |
| 1. Automate download of installation package from public repository | Obtaining the installation package from the internet automatically | 1 Essential | Automatic download of Maria DB is a failure. Stakeholder decide to remove feature from success criteria. The feature has not been met as the feature hasn’t been included in the solution. |
| 2 Automate installation of Maria DB | Installation of database | 1 Essential | Automatic installation of Maria DB is a failure. Stakeholder decide to remove feature from success criteria. Automatic download of Maria DB is a failure. Stakeholder decide to remove feature from success criteria. The feature has not been met as the feature hasn’t been included in the solution. |
| **Code to provision: Gerrit** | | | |
| 1 Automate download of Gerrit | Obtaining the installation package from the internet automatically | 1 Essential | Automatic download of Gerrit is successful. Feature is a full success as the criteria is met. |
| 2. Create a database in Maria DB to be used by Gerrit | Creation of data storage for Gerrit app | 1 Essential | Maria Db database creation was unsuccessful. Criteria has not been met as feature not implemented. |
| 3.Create accounts for Git/Gerrit in Maria DB | Setting up a test / demo / admin account for Git / Gerrit | 1 Essential | Git and Gerrit accounts in Maria Db unsuccessful. Criteria has not been met as feature not implemented |
| 4.Create a test project in Gerrit | Demo project in Gerrit | 3 Minor | Demo project in Gerrit was successful. Criteria has been met fully. |
| **Code to provision: Jenkins** | | | |
| 1 Automate downloading of the package | Obtaining the installation package from the internet automatically | 1 Essential | Automatic download and installation of Jenkins was successful. The feature is a full success as the criteria is met. |
| 2 Programmatically set up a Jenkins master instance | Set up of continuous build application | 1 essential | Master instance successfully created. The feature is a full success as the criteria is met. |
| 3  Create a test / demo project on Jenkins master | Test / demo project to visualise the results of set up | 3 Minor | Test demo project successful. The feature is a full success as the criteria is met. |
| **Code to provision: Jenkins slave** | | |  |
| 1  Automate installation of Jenkins slave | Provision Jenkins slave to implement distributed build functionality | 2 Major | Jenkins slave installation unsuccessful. Criteria has not been met as feature not implemented |
| 2  Connect Jenkins slave to Jenkins master | Implementation of connectivity between master and slave instances of Jenkins | 2 Major | Connection between Jenkins master and Jenkins slave unsuccessful. Criteria has been partially met in that the master instance works, however a connection to slave instance has not been implemented. |
| **Code to provision: Maven** | | | |
| 1.Automate Maven installation | a. Install Java build tool so that it can be used in a continuous build process | 1 Essential | Automation of maven Installation is successful. Therefore, feature is fully successful as criteria is met. |
| 3.Update custom Maven configuration to match environment | Change the generic Maven configuration files to match the environment | 1 Essential | Feature has not been successfully created / configured. Criteria has not been met as feature not implemented |
| **Code to provision: Nexus** | | | |
| 1 Automate the download of free Nexus version from Sonatype repository | Getting the Nexus binaries from the vender | 1 Essential | Automated download and installation of nexus successful. Therefore, feature is fully successful as criteria is met. |
| 2.Automate Nexus installation | programmatically install the release repository app | 1 Essential | Automated download and installation of nexus successful. Feature is successful in full. |
| 3 Automate creation of test / demo Nexus repository | Setting up a demo release repository | 3 Minor | Automated creation of test demo unsuccessful. Feature is not met as criteria is not achieved. |

## Evaluate Usability Features (refer to user testimony and usability screenshots)

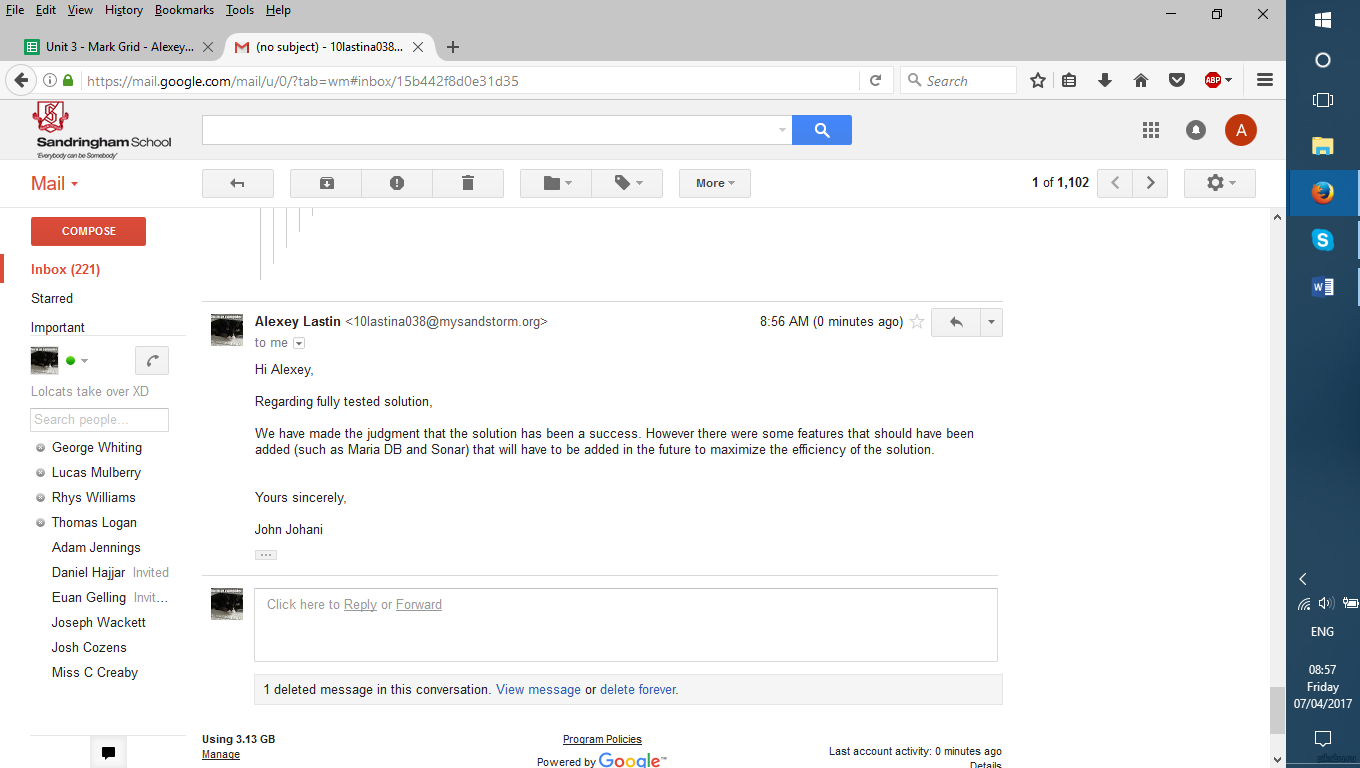
|  |  |  |  |
| --- | --- | --- | --- |
| Usability feature | Description | How successful was it? | Areas for improvement |
| Navigation | This covers the ease of access to Jenkins server, Github, Command line interface, instance connection, Gerrit website and general navigation across the instance. | Partial success | This could be improved by implementing a feature that shows what user options are available after each action that they complete. |
| Consistency | This covers the all-round quality of the solution, so the different features being done correctly and efficiently. | Partial success | This could be further improved by having implemented all the features on the success criteria, which would have allowed for a better consistency. |
| Visual clarity | This covers the ease of finding what the user wants in the program and is closely linked with navigation. This usability feature should be included in all parts of the solution i.e. command line interface – outputs shouldn’t be difficult to read; website should be simple and easy to understand. | Success | Visual clarity could benefit from a table of contents that could then allow the user to check what they want to do next after each command that they execute. There should be a |
| Efficiency | This covers how quick, easy and simple specific processes can be done with the solution. These could include using Jenkins and Maven to set up a build, using Github to clone a repository to local machine | Partial success | This could benefit from built in commands that when put in would start several processes that would make tasks easier to be done. |
| Flexibility | This covers how easy it would be to change the program should the need arise in the future. | Success | The solution can be easily changed |
| Error prevention | This covers what is done to take errors to a minimum and what makes troubleshooting them easier | Partial success | This could benefit from a more detailed explanation of what the error was and if so what actions could be taken to stop it from happening again. |

|  |  |  |
| --- | --- | --- |
| **Usability feature** | **What could be improved?** | **How would this be done?** |
| Navigation | New employees who aren’t accustomed to the language need a way of making it easier for them to learn how to use the software. | The addition of a -man (manual) idea from bash where you’ve got the possible options listed rather than expecting the user to know them all |
| Consistency | A few features have been left out because of complexity reasons and the lack of time that our situation imposed. These include Maria DB and Sonar installations and configurations. | During further maintenance and updates both features can be installed into the environment and configured to what the company wants. |
| Visual clarity | The text is all bunched together even though it has been separated to new lines or space in between put. | To improve this there should be a steady white line between all processing so that all information can be clearly understood. |
| Efficiency | Could benefit from built in commands that when put in would start several processes that would make tasks easier to be done. Similarly, to the navigation section this will aid new employees get accustomed with the system faster. | Simple built in commands such as ‘connect’ ‘disconnect’ can be introduced to aid the user when being used. |
| Flexibility | A catalogue could have been created in order to show main modifications related to Continuous integration that could be used. | Add captions to each module to detail what could edited and what the result of this action will be. |
| Error prevention | A catalogue of errors would also be beneficial for new employees that would allow them get accustomed faster | A catalogue can be made to store errors each time they occur rather than having the errors go after each line. This will also help visual clarity as the user will find the errors quickly. |

**Overall conclusion around the success of solution:**

Overall based on the end functionality of the solution is good. This is because the larger part of the success criteria has been successfully completed or has been completed to some extent. Also the solution was able to cover most of the essential features which therefore allows it to do its mission. However there were also some downsides of this solution in that although it had met most of the criteria individual essential features such as Maria DB and Sonar were to be dropped due to the complexity of the project.

From the user review below we learnt that they considered this project / solution a success, however would require to update it as there were some features that should have been included but were not implemented because of time and complexity reasons.



## Limitations & Maintenance

|  |  |  |  |
| --- | --- | --- | --- |
| **Limitation** | **Description** | **Justification** | **Possible ways to deal with limitations** |
| Dependant on AWS | The solution is dependent on Amazon Web Services to be used. This is because AWS is the hosting service that allows for the instance to be created. | This is a limitation because if AWS was to break or become unavailable then it would make the solution useless as it cannot be used. | Store backups on local computer in order to minimise damage caused. Potentially if this repeats regularly switch to another company that provides a similar service. |
| Requires the user to be quite knowledgeable in terms of commands to fully utilise | The solution is would be quite difficult to use for people with little experience with AWS and similar applications. | This is a limitation as it wouldn’t allow some people to use the program to its full capabilities or not being able to use the correct sequence of commands to output or start a process (i.e. build, initialisation) | Have catalogues that give details about what can be done at each point of the Continuous Integration process. |
| Automatic instance termination | The solution is supposed to be automatic in nature, however the instance termination doesn’t work | This means that a fundamental feature isn’t working correctly, and although it does stop the solution from working without it the solution loses out on efficiency | This can be thoroughly debugged and then made correctly. Therefore the automation should work. |
| Download and configuration of Maria DB | This was originally supposed to be in the original success criteria, however the client decided to drop it in order to save development times. | This means that an essential feature isn't being included in the final solution. On a practical note this makes it impossible to make version control a feature in the program. | The way to deal with this would be to install Maria Db, configure it and then connect Git and Gerrit. |
| Jenkins slave creation and connection to master instance | This was supposed to be a feature that was required by the solution in the initial success criteria that would allow multiple build processes to happen simultaneously (on different Jenkins servers) and provide different environments. | This was a major rate feature as it wasn’t essential for the solution to work, however this should be implemented as it is very beneficial to the efficiency when several build processes are run around the same time. | To implement this I would have to merge Jenkins master and slave nodes and configure them to the clients needs. Then establish a connection between the machines to have another version of Jenkins that will be availible to use |

|  |  |  |  |
| --- | --- | --- | --- |
| Maintenance issue | Potential improvements / developments | Limitations of these improvements | Potential improvements to further improve |
| Bug fixes | A potential improvement would be to install a filtering software that allows you to know where a problem has occurred. This would allow the user to be able to make maker regular checks so that the program is working correctly. | Thus, this is likely to bring slower output speeds after a command is put through, as it has to check if something has happened (mistake during the process) and then report it (comparing the result it gets from the computer to a list / database with the ‘human’ translations of these errors. | The categories should be separated into several groups and then examples of the errors inserted there. So when the error occurs only the general area of the error is outputted |
| AWS API change- sys change | A potential set of improvements would be to update the environment every time there is an update that has new content that can help make it more efficient. | However, the limitation to that is that it will be very time consuming and remaking the environment every AWS update that happens is inefficient. | This should be done rarely and only when a large update needs to take place. |
| New features - code to cope with its updates in AWS API | A potential set of improvements would be to update the environment every time there is an update that has new content that can help make it more efficient. | However, the limitation is that it would be a lengthy process to create brand new set of code for each update which would make updates slow, hence the environment would be inefficient as it should be used daily. | A template code should be created and then the others based around it. Also the updates should only happen once a month. |
| Updates to key libraries | This is very key to any solution designed with use of libraries because with each update they usually have more security and data that is useful for programs (i.e. EC2 library for perl). | However, the limitations of this could be that all software should be upgraded because one might not be able to be run on the other. Here this would mean more storage space is used and more processing power required. | Updates should be split into three types urgent, security and routine which the user has power to turn on and off. |
| Config of key features, (in a year ..) | Maintenance of key feature configurations is a very important way of ensuring that the features are operating correctly and that the user is has been using them right. | If the user isn’t doing them correctly, then the program could potentially be broken and won’t work as intended by the client. | To assist with this a user handbook should be made so that they can use the environment |
| Git hub | This is will require maintenance – because Git will only accept certain files and if these are not updated its likely that the file will not be used anymore - data on it would be useless. Hence it would be important to update the format. | A potential downside of this is converting and uploading the code is going to take a long time which would be inefficient. | This should be done every few weeks in order not to lose any data and efficiency. |
| Code handover key scripts | Over the years that the project may be operating it would be best to keep a constant record so that the next users will be able to understand what’s going on. | A potential limitation of this is again the time take to produce this will be long. | This could be shortened down to several key features that are crucial to be done and a smaller handbook on its general usage and flexibility. |