|  |
| --- |
| ci6300 – INDIVIDUAL PROJECT |
| PROJECT REPORT |
| Daniel Carnovale |
|  |
| **K1336511** |
| **04/04/2017** |

|  |
| --- |
|  |

Table of Contents

[Introduction and Literature Review 3](#_Toc479084446)

[Methodology and Analysis (can be agile based) 3](#_Toc479084447)

[General: 3](#_Toc479084448)

[Details: 3](#_Toc479084449)

[Purchasing of the Domain Name 3](#_Toc479084450)

[Amazon 3](#_Toc479084451)

[Amazon EC2 4](#_Toc479084452)

[Console Access 4](#_Toc479084453)

[Physical Database Implementation 4](#_Toc479084454)

[Design 5](#_Toc479084455)

[Server hosting In House 5](#_Toc479084456)

[Choice of Amazon as a Cloud Provider 5](#_Toc479084457)

[Choice of Virtual Machine Hosting service 5](#_Toc479084458)

[Choice of Source Control (GitHub vs Dropbox) 5](#_Toc479084459)

[Local access to server 6](#_Toc479084460)

[Database (Technology, Management tool PHPPgAdmin) 6](#_Toc479084461)

[OS (Windows VS Linux) 6](#_Toc479084462)

[Choice of Linux distribution (Amazon Linux) 6](#_Toc479084463)

[Implementation 7](#_Toc479084464)

[Discussing How-To steps (Actual steps in appendix?): 7](#_Toc479084465)

[Testing & Evaluation 7](#_Toc479084466)

[Critical Review 7](#_Toc479084467)

[Appendix A 7](#_Toc479084468)

# Introduction and Literature Review

Cloud computing services have become more and more desirable as a means of hosting and managing projects for business or personal requirements, as opposed to in-house services. These two options can be combined however to further strengthen a project and I will be exploring this as a review of what I have applied to my “Unihood” project. Ultimately for this project a database and web app are required, and hosting and managing these within a virtualisation environment provides several benefits including automation of the re-construction of the entire system.

# Methodology and Analysis (can be agile based)

The methodology I applied led me to the use of forums, help pages on the web and advice from friends and family members to start investigating available technologies and procedures. Considered technologies include GitHub, MySQL, PuTTY, PHP, Python, Ruby / Perl, HTML5, CSS, Javascript, Bootstrap, Angular, JQuery UI, mySQL, postGreSQL, Ansible and Docker, some of which were utilised in the project. The Amazon services EC2, S3, EBS and Elastic IPs were all implemented into the project as utilisation of the wide range of Amazon technologies was the main direction I wanted to head into for this area. I intended to make full use of their “free tier” procedure / offer, though through easy mistakes regarding instance up-time this was unfortunately not achieved.

This was essentially an “agile” approach in that I chose to investigate tools and make quick assessments on which ones to deploy, without performing a very exhaustive research on the merits of each. I continued use of a tool or resource if it could be quickly and easily deployed to achieve an end, however when this did occur I made notes on how each such tool was deployed. This resulted in the recording of sufficient material to turn these notes into fully elaborated How-To’s.

## Purchasing of the Domain Name

Several titles for this project were considered far before any implementation was done, though once UniHood was decided and searched for to verify it was a unique name the next step was to “morph” it into a domain name format, and purchase it. GoDaddy was the main go-to here, so a GoDaddy accand the domain name of uni-hood.co.uk was bought for a very cheap price of one pound.

## Amazon

The aim to minimise the costs associated with the project was at the top of my mind when setting up a new Amazon account which would be linked to the Amazon Web Services. After some research, I discovered there was a “Free Tier” period that is included in the creation of a fresh account, so I could conveniently utilise this instead of my current Amazon account. To distinguish this from my main account, I felt it was necessary to link it to a fresh Gmail account also.

After linkage between the Gmail account and Amazon account was completed (through simply providing it as the email address during sign-up), I started the process of linkage between the Amazon account and the Amazon Web Services. It is a requirement of the AWS sign-up process to link this account as the services will be tied to it. The sign-up process does not provide you with full access to the AWS services right away, as it is seen as a kind of request form for which you will need to wait on a response from and/or keep in contact with Amazon themselves regards if and when they allow you to use these services.

This took longer than expected however, as there was some small issue between my bank (Barclays) accepting or managing the small transaction used to verify this linkage. I contacted Amazon support and started this as a small case, for which after a few message exchanges was resolved very quickly. I was very impressed with the speed and quality of support from Amazon, and it helped me proceed with little disruptions to the actual creation of the Unihood virtual machine. I have elaborated on this aspect in the How-To document under Amazon “Free Tier” Account Creation”.

It may take some weeks, though when the AWS are linked to your account this free tier program will then be available for use. It will greatly help minimise the costs when using (in this case) virtual machines. Options which are eligible for this are indicated clearly on each step of the instance creation process, so there is strong assurance that the right choice is being made cost-wise.

## Amazon EC2

At this stage it was necessary to choose an “AMI (Amazon Machine Image)” namely “Amazon Linux”. I realised under later experimentation this behaved very similar to the Centos 6.X version Operating Systems. I wanted to investigate how to manage large amounts of data as a side-objective in case I needed to utilise this later, which led me to the experimentation of EBS Volume creation, discussed further under the How-To Deliverable in the EBS Volumes section.

## Console Access

There is normally no “console” access as expected in stand-alone workstations or in VPlayer-type environments; alternate instructions were provided upon clicking the “connect” link for the instance. I needed to initiate a bash prompt connection onto the newly created instance instead as directed, discussed further under the How-To Deliverable in the Key Pair / Putty section.

The “StarUML” application was particularly useful here as it was much faster and easier than sketching the tables on paper in some respects, however each approach has their advantages. I was able to export this work as an image I could always refer to, while maintaining a top-down approach in that classes with no foreign key linkages were at the top of the diagram, and the rest below them. This made it very easy to start off with creation of the top classes and work downward, instead of having a “foreign key trap” of sorts in that a column depended on another column however that column depended on the one referencing it, or the existence of another table that could not yet be created due to a similar situation. Database design and physical implementation go hand-in-hand, in that it took me several iterations to perfect the structure of the physical database whilst changing the design diagram the whole time in sync. Once at a good stage however, it was ready to be utilised in the main web application later down the track.

## Physical Database Implementation

The PHPPgAdmin environment was my choice for the database to run under. Initially there was confusion regarding the postgres version to use (9,91,92,93, etc). In the end, I decided to go with the latest one (95) but later found serious compatibility problems with phppgadmin! A workaround was a quick edit to the “Connection” php service file, the link for which is in the Appendix.

Firstly however it was necessary to install the Postgres and Httpd packages on the virtual machine instance as well as some additional packages. The whole database management environment is included in these packages, so after starting them and appending /phpPgAdmin to the instance ip you will be presented with an interface to which implementation of the database diagram (from StarUML) can be applied. Creation of the database and tables through the interface is fairly straightforward, however logically you must pay attention to the usage of foreign keys and make sure they match the flow of the diagram. Once satisfied with the table creation and their related columns within the tables, test data can be inserted as rows and reviewed within a table’s overview page. You may encounter a scenario where only id’s are displayed instead of the actual piece of data they are referencing, so a more visual interface allowing foreign data to be recognised was desirable. My father’s open-source “Tables” facility for his current work database was adapted to account for my data instead, and foreign key values were correctly resolved to the values they linked to and visualised. All database data from the original phpPgAdmin environment, including the test data, can be exported to a “dump file” and this process is outlined in the How-To document under the “Backup/Restore Test Data” section. This file was then able to be imported into the new facility, as both environments were capable of recognising these SQL dump files.

# Design

## Server hosting In House

I chose the Amazon cloud-based environment due to it being a very well known company; offering great customer service and reliability, a free tier program and having a wide range of virtualisation tools and file storage available to its users. The EC2 and S3 services are the most relevant for this project, and when combined can immensely strengthen a virtual Amazon instance. As an additional benefit, through hosting the virtual server in the cloud, it is both accessible anywhere.

## Choice of Amazon as a Cloud Provider

Amazon already has a very high reputation of offering great Customer Service as well as being a massive marketing platform. After having used their S3 service in particular their storage seems very stable and offers a large amount of data to be stored so it seemed a strong candidate to make use of this as a cloud system for my project. Linking the storage facility with the virtualisation environment resulted in a very strong cloud-based solution.

## Choice of Virtual Machine Hosting service

I chose Amazon EC2 because I was already impressed with solely their Amazon Simple Storage Service (S3) at first, after having experimented with it for a while for general purposes, so had no doubt the EC2 service would serve just as well. It presented me with a very simple interface, quick response times, great support and so on and only had the slight issue with linkage of these services to my account however these were resolved very quickly.

Choice of Source Control (GitHub vs Dropbox)

I chose GitHub as a repository for all screenshots, source code, version control aspects and documentation (issues etc) needs. I manage it locally using the Windows application “TortoiseGit” which allows me to easily set up a new Git repo, and have access to many Git features though mainly I simply work off the local files, then commit and push them to the master repository link on GitHub. I also on the web interface manage “issues” which are small paragraphs of current bugs, reminders or just general notes regarding the project. I also have the opportunity to revert back to a past commission if something goes wrong in the current stage.

Dropbox on the other hand was a great repository for note taking also, and all important notes and even Putty files and similar were encrypted using the “Boxcryptor” app, which means that if my Dropbox account was compromised the encrypted files could only be read through an additional layer of security through this app. Locally, it’s very easy for me to login to the Boxcryptor app and view these files on a separate virtual drive safe from the eyes of the public.

## Local access to server

I decided to use the Windows Putty application because I am already very familiar with using it to connect to remote servers through the SSH (Secure Shell) protocol and by default it provides an excellent terminal interface to work off. Linux, being a terminal-based operating system itself, works very well with Putty and tips for connection through it are linked to from the connection window on the Amazon EC2 Management Console.

## Database (Technology, Management tool PHPPgAdmin)

## OS (Windows VS Linux)

## Choice of Linux distribution (Amazon Linux)

I needed to broaden my knowledge of the Linux world, and having the general view of Linux as a powerful, user-friendly and pro open-source platform persuaded me to select this as an AMI for the Amazon Instance.

## Security (Approach, Confidentiality, Privash, SSH Key Pair)

## StarUMLClass Diagram

After a significant level of experimentation with the “StarUML” application through my previous University years, I was able to quickly apply my knowledge to this project, and through the click-and-drag nature of the graphical environment I was able to quickly construct a basis for the actual database structure. I then was able to export this as an image file and commit & pushed to GitHub, so I could always refer to it whilst implementing the database.

## Backup techniques (test data, use PhpPgAdmin frequently to export dump files, considered Ansible and Terraform etc)

## Apache / PostGres configuration

## Docker consideration

## Encryption (BoxCryptor for Putty Keys) and linkage with Dropbox

## Domain name purchase

## How should I record information? Word .doc or GitHub .md files. Doc is such a standard, though for technical documentation I believe this should be in .md format.

## How to reliably reproduce runtime environment (research suggests that Docker is a strong solution to this, but as yet I have to learn Docker effectively)

## Choice of web-server (Apache, NginX ?) Apache because..

## Hack attempt prevention, disabling clear-text password login access.

## Similarities between Amazon Linux and Centos 6 (why Centos 6.x instructions should be followed)

## Installation of PostGres

## Regular yum update to enforce preventing of vulnerabilities regarding security issues

## Looked into SQL generation from UML but assumed it was very complex / couldn’t find anything / wouldn’t work for my version

## Benefit of public key usage with GitHub

## Considered enhancement of GitHub credentials

## Consideration of static (elastic) instead of dynamic ip

## Documentation of snapshots to accompany steps document

## Use of TortoiseGit to manage local Git repo

# Implementation

## Discussing How-To steps (Actual steps in appendix?):

Appendix A is the primary deliverable of this project work, and consists of a series of HOW-TO’s. These are designed to be simple, practical and readable instructions that allow a particular technical objective to be achieved, and yet do not assume a high existing level of expertise to understand and execute.

# Testing & Evaluation

* Succeeded in getting a web app up and running? (self assessment.. I reckon I did)
* (put the how-tos in front of somebody else! See if they can follow them !)

# Critical Review

Was my approach successful? Will anyone really benefit from my How-Tos?

Which design decisions would I do differently?

# Appendix A

<https://tech.enekochan.com/en/2014/04/11/fix-error-column-spclocation-does-not-exist-in-phppgadmin/>

(Append all How-To’s here, so they’re shown in contents section)